General Disclaimer

One or more of the Following Statements may affect this Document

- This document has been reproduced from the best copy furnished by the organizational source. It is being released in the interest of making available as much information as possible.

- This document may contain data, which exceeds the sheet parameters. It was furnished in this condition by the organizational source and is the best copy available.

- This document may contain tone-on-tone or color graphs, charts and/or pictures, which have been reproduced in black and white.

- This document is paginated as submitted by the original source.

- Portions of this document are not fully legible due to the historical nature of some of the material. However, it is the best reproduction available from the original submission.

Produced by the NASA Center for Aerospace Information (CASI)

(NASA-TM-84794) ORBIT DETERMINATION SOFTWARE DEVELOPMENT FOR MICROPROCESSOR BASED SYSTEMS: EVALUATION AND RECOMMENDATIONS (NASA) 95 p HC A05/MF A01 CSCL 09B G3/61 N82-29028 Unclas 28381

ORBIT DETERMINATION SOFTWARE DEVELOPMENT FOR MICROPROCESSOR BASED SYSTEMS: EVALUATION AND RECOMMENDATIONS

JULY 1980

National Aeronautics and Space Administration

Goddard Space Flight Center
Greenbelt Maryland 20771

SYSTEMS TECHNOLOGY LABORATORY SERIES STL-80-001

ORBIT DETERMINATION SOFTWARE DEVELOPMENT FOR MICROPROCESSOR BASED SYSTEMS: EVALUATION AND RECOMMENDATIONS

JULY 1980

Goddard Space Flight Center

FOREWORD

The Systems Technology Laboratory (STL) is a computational research facility located at the Goddard Space Flight Center of the National Aeronautics and Space Administration (NASA/GSFC). The STL was established in 1978 to conduct research in the area of flight dynamics systems development. The laboratory consists of a VAX-11/780 and a PDP-11/70 computer system, along with an image-processing device and some microprocessors. The operation of the Laboratory is managed by NASA/GSFC (Systems Development and Analysis Branch) and is supported by SYSTEX, Inc., Computer Sciences Corporation, and General Software Corporation.

The main goal of the STL is to investigate all aspects of systems development of flight dynamics systems (software, firmware, and hardware), with the intent of achieving system reliability while reducing total system costs. The flight dynamics systems include the following: (1) attitude determination and control, (2) orbit determination and control, (3) mission analysis, (4) software engineering, and (5) systems engineering. The activities, findings, and recommendations of the STL are recorded in the Systems Technology Laboratory Series, a continuing series of reports that includes this document. A version of this document was also issued as Computer Sciences Corporation document CSC/TM-80/6086.

The primary contributors to this document include

Charles M. Shenitz (Computer Sciences Corporation)

Other contributors include

Frank McGarry (Goddard Space Flight Center)
Keiji Tasaki (Goddard Space Flight Center)

Single copies of this document can be obtained by writing to

Keiji Tasaki
Code 582.1
NASA/GSFC
Greenbelt, Maryland 20771

ABSTRACT

This document is intended as a guide for National Aeronautics and Space Administration (NASA) management personnel who stand to benefit from the lessons learned in developing microprocessor-based flight dynamics software systems. The essential functional characteristics of microprocessors are presented. The relevant areas of system support software are examined, as are the distinguishing characteristics of flight dynamics software. Design examples are provided to illustrate the major points presented, and actual development experience obtained in this area by Computer Sciences Corporation personnel under contract to NASA Goddard Space Flight Center Code 582 is provided as evidence to support the conclusions reached.

TABLE OF CONTENTS

Section 1 - Introduction 1-1

Section 2 - Background 2-1

2.1 Computer Hardware Revolution Versus Software Evolution 2-1

2.1.1 Computer Hardware Revolution 2-2
2.1.2 Microprocessor Applications. 2-5
2.1.3 Growing Pains of a New Industry. . . . 2-6
2.1.4 Software Evolution 2-7

2.2 Orbit Applications. 2-8

2.2.1 Salient Characteristics of Orbit Determination Work 2-8
2.2.2 Early Project Plans for Microprocessor-Based Orbit and Attitude Systems . . 2-11
2.2.3 Demonstration Systems Developed. . . . 2-11

Section 3 - Modern Software Development Practices Applied to Microprocessors 3-1

3.1 General Software Development Considerations . . . 3-1

3.1.1 Source Languages and Structured Programming. 3-1
3.1.2 Source File Handling 3-4
3.1.3 Modularization 3-6
3.1.4 Floating-Point Capabilities. 3-8
3.1.5 Mathematical Software Library. 3-11
3.1.6 Minicomputer and Large Machine Support. 3-13

3.2 Microprocessor Software Development 3-14

Section 4 - Special Considerations in Developing Microcomputer Applications Software. . . 4-1

4.1 Recasting Algorithms. 4-2
4.2 Storage Considerations. 4-5
4.3 Network Configurations of Microprocessors . . . 4-7
4.4 Software Reliability: Fault Tolerance. 4-11

TABLE OF CONTENTS (Cont'd)

Section 5 - Microcomputer Hardware System Test Tools 5-1

5.1 Software Tools for Microcomputer Hardware Debugging 5-2
5.2 Hardware Aids for Hardware Diagnosis. 5-4

Section 6 - Microprocessor-Based Orbit Determination Configuration Examples 6-1

6.1 Microprocessor-Based Orbit Determination From Raw Data: A Ground System. 6-2
6.2 Microprocessor-Based Orbit Determination and Ephemeris Interpolation On Board. 6-6

Section 7 - Review of Microprocessor-Based Orbit Applications Development 7-1

7.1 IMP-16 and SM/PL Experience 7-1
7.2 INTEL 8080 Orbit Propagator Development Experience. 7-5

Section 8 - Conclusion 8-1

Appendix A - CSC Memorandum on IMP-16 and SM/PL System Bugs

Appendix B - GSFC Memorandum on a Particular SM/PL Bug

Glossary

References

TABLE OF CONTENTS (Cont'd)

Section 5 - Microcomputer Hardware System Test Tools 5-1

5.1 Software Tools for Microcomputer Hardware Debugging 5-2
5.2 Hardware Aids for Hardware Diagnosis. 5-4

Section 6 - Microprocessor-Based Orbit Determination Configuration Examples 6-1

6.1 Microprocessor-Based Orbit Determination From Raw Data: A Ground System. 6-2
6.2 Microprocessor-Based Orbit Determination and Ephemeris Interpolation On Board. 6-6

Section 7 - Review of Microprocessor-Based Orbit Applications Development 7-1

7.1 IMP-16 and SM/PL Experience 7-1
7.2 INTEL 8080 Orbit Propagator Development Experience. 7-5

Section 8 - Conclusion 8-1

Appendix A - CSC Memorandum on IMP-16 and SM/PL System Bugs

Appendix B - GSFC Memorandum on a Particular SM/PL Bug

Glossary

References

LIST OF ILLUSTRATIONS

Figure		
2-1	Early Plan for Utilization of Microprocessor Technology for Orbit and Attitude Applications.	2-12
4-1	Mailbox System of Interprocessor Communication	4-10
4-2	Hypercube Network Architecture.	4-12
6-1	Design for Ground Microprocessor-Based Orbit Determination System.	6-3
6-2	Basic Design of Spacecraft Orbit Determination and Ephemeris Interpolation Network	6-9
6-3	Expansion of Orbit Determination Processor Into Two Parallel Processors.	6-10
6-4	Approximate Timeline for Parallel Processing of Sequential Orbit Estimation. . . .	6-11

LIST OF TABLES

Table		
2-1	Integer Operation Execution Time Comparison	2-4

SECTION 1 - INTRODUCTION

Since October 1977, the Orbit Systems Operation of Computer Sciences Corporation (CSC) has been supporting the National Aeronautics and Space Administration's Goddard Space Flight Center (NASA/GSFC) in the development of flight dynamics software for implementation on microprocessor hardware. The main project has been the development of an orbit determination program tailored specifically for updating the orbit of the Solar Maximum Mission (SMM) spacecraft using satellite-to-satellite tracking (SST) data, with a Tracking Data and Relay Satellite (TDRS) as the relay satellite. The estimation scheme used in this program is the extended Kalman filter. The entire system was implemented on two interconnected National Semiconductor Corporation IMP-16 microprocessor systems with programmable read-only memory (PROM). The system was nicknamed "the shoebox" because of its portability and its diminutive size for a full computing system. The application software was developed on an IMP-16P development system with floppy disk, primarily using SM/PL, a structured, higher level language. The system terminal is the Texas Instruments Silent 700, a portable, "intelligent," nonimpact printing terminal. A related, but independent, system developed concurrently with the shoebox is the INTEL 8080 Orbit Propagator. It was developed on a Tektronix 8002 development system with a floppy-disk-based operating system, using a version of the FORTRAN language.

It is hoped that the experiences encountered during the development of these systems can serve as a guide in the decisionmaking processes for initiating microprocessor-based flight dynamics systems in the future. One purpose of this document is to encapsulate those experiences for such use. Another purpose is to present views of current microprocessor hardware, current software development practices, and

the special requirements for developing flight dynamics software in general and orbit determination software in particular and then fuse this information into meaningful guidelines for future projects.

It is assumed that readers of this document are not fully familiar with many aspects of software development, small machine hardware, flight dynamics analysis, and numerical analysis. Basic information and references are provided prior to the main discussions of the microprocessor experience gained thus far.

Section 2 of this document contains descriptions of the general trends in computer hardware and software to the present, as well as the background of the orbit determination work on microprocessors performed by CSC for GSFC. Section 3 provides an overview of modern software development practices, with a slant toward the applications of interest here. Section 4 lists many special considerations that usually must be made when developing scientific software for a small machine. Section 5 briefly describes special considerations involved in microprocessor hardware and software problem diagnosis. Section 6 describes the design of two possible microprocessor-based orbit determination systems. Section 7 reviews the experiences encountered in developing the orbit applications as particular examples of flight dynamics software on microprocessor-based hardware. Section 8 summarizes the guidelines derived in this document. Memoranda issued during this development project concerning SM/PL compiler bugs are reproduced in Appendixes A and B.

SECTION 2 - BACKGROUND

Before embarking upon a discussion of the hardware revolution that has brought about the microprocessor, it is necessary to provide a working definition of a microprocessor. A microprocessor is a computer central processing unit (CPU) that is contained on chips that are products of large-scale integration electronics technology. Usually the CPU is contained within one large-scale integration chip; the acceptance or rejection of a processor as a microprocessor based upon the number of chips comprised in the CPU is not of concern here. The term "microcomputer" is reserved for any configuration containing a CPU chip(s), memory, and interface circuits (i.e., a functional computing package based upon a microprocessor). Because the primary concern here is with orbit and attitude applications of microprocessor-based technology, the terms "microprocessor" and "microcomputer" are used interchangeably.

Section 2.1 discusses both the revolution in electronics that brought about the microprocessors of today and the revolution in software development techniques. Section 2.2 characterizes orbit applications and reviews the history of microprocessor orbit applications performed by CSC for GSFC.

2.1 COMPUTER HARDWARE REVOLUTION VERSUS SOFTWARE EVOLUTION

This section provides a brief summary of the dramatic change (truly a revolution) that has taken place in the brief lifetime (30-odd years) of the electronic digital computer. In this section, the point is made that the microprocessor is being groomed as the successor to the minicomputer while at the same time opening up new areas of application such as the home computer. Because the technology is rather new, signs

of growing pains exist, and these are pointed out. Finally, the progress made in software development methods is briefly examined to facilitate full comprehension of the need for considering software development techniques, even within a new, highly altered hardware environment.

2.1.1 COMPUTER HARDWARE REVOLUTION

The electronic digital computer originated shortly after World War II. Since that time, the electronics industry, the source of the computer's basic components, has itself made tremendous strides in the miniaturization of components. This trend has been accompanied by a general increase in speed (of device response) and a dramatic decrease in cost.

In the computer industry, the vacuum tube was replaced with semiconductor components (such as the transistor and the diode) in the mid-1950s. The compaction of discrete semiconductor devices onto small plates, or chips, marked the beginning of integrated circuit (IC) technology. The development of small-scale integration (SSI) and medium-scale integration (MSI) of circuits was reflected in the computer industry by the advent of production of minicomputers by the late 1960s. The next step in this progression was the achievement of large-scale integration (LSI) of circuits, which gave rise to the microprocessor.

The magnitude of change can be judged from the following: by the early 1970s, the minicomputer was smaller by a factor of 1000 or more than the machines of approximately 20 years before and, while being more powerful, cost approximately 1/100 as much as the older machines (Reference 1, page 1). Many of the first-generation minicomputers specified in Appendix A of Reference 1 are capable of performing only fixed-point multiplication or division of 16-bit arguments in 10 to 40 microseconds. Table 2-1 (from References 2 and 3)

shows that microprocessors have caught up with, and in many cases overtaken, these first-generation minicomputers. Although the results of a well-prepared series of benchmark tests (the accepted means of comparing different machines) are not presented here, the occurrence of a revolution in hardware can nonetheless be easily appreciated, in that an electronic board or caged set of boards may be able to supplant the minicomputer of 10 years ago. Although it is possible to conclude at this point that microcomputer hardware can replace first-generation minicomputers, caution should be exercised before committing resources to a microprocessor system for an area where it may not be suitable.

Another evolutionary (or, rather, revolutionary) path that can be traced for the development of the microprocessor is that stemming from combinatorial logic, or Boolean algebraic expression, implementations. These electronic arrays of logic eventually became programmable logic arrays (PLAs), many of them being reduced to an LSI chip. Their programmability lies in the capability of tailoring a mass-produced general logic array chip to a particular application. The chip would then act as a combinatorial logic, or gating process, for several binary inputs (usually simultaneous). From this alternative view of microprocessor development, the next logical step was to place on a chip full programming capabilities in the sense of a von Neumann machine, i.e., the standard digital computer as it is currently known. Programs could then be fixed in memory (in PROM, to be first discussed in Section 3.2) to provide control that could not have been achieved easily, if at all, with the hardwired gating processes of the PLAs. (For further details concerning PLAs and their relation to microprocessors, see Reference 4.)

ORIGINAL PAGE IS
OF POOR QUALITY

Table 2-1. Integer Operation Execution Time Comparison

OPERATION	EXECUTION TIME (MICROSECONDS)				
	LSI-11/03	Z-8000	8080A	Z-80A	PDP-11/45
ADD	1.72	2.25	9.61	5.25	2.86
MULTIPLY	24.52	17.50	–	–	5.64

7260-39

NOTES: 1. ALL OPERATIONS ARE 16-BIT (INTEGER).
2. THE DATA FOR THE LSI-11 WERE OBTAINED FROM REFERENCE 4, APPENDIX A. THE OTHER DATA WERE TAKEN FROM REFERENCE 3.
3. ONLY THE PDP-11/45 IS A MINICOMPUTER; THE OTHER PROCESSORS ARE MICROPROCESSORS.

2.1.2 MICROPROCESSOR APPLICATIONS

A good starting point for surveying the range of current microprocessor applications is a review of minicomputer applications by the early 1970s. First, the minicomputer was used in applications where the computing power of a big machine would be wasted. Second, the minicomputer entered areas where a large machine could be replaced by one or more minicomputers with a substantial cost reduction realized. Finally, the physical limitations (power, space, cooling) of the placement of a large machine could be ignored with the new, downsized hardware. Thus, typical minicomputer applications in the early 1970s included the following (Reference 1, page 7):

- Process control--chemical plant, experimental laboratory
- Device control--typesetting machine, optical character scanner
- Data transfer control--data communication network
- Problem solving--scientific problems, business data processing, students' homework problems

A quick survey today would show that microprocessors are fast becoming the heirs to minicomputers in the applications cited above (Reference 5, pages 159 and 215). Also, from the gains that microprocessors have made in achieving extreme cost reductions and minimal environmental supports, many new areas have opened up for the microprocessor that were not suitable for the minicomputer, the most radical being that of the general home market. The impact of this new market being opened is that a major portion of microcomputer hardware and software is being geared for novice use. The professional in search of microprocessor-based systems to support computationally complex applications (i.e., applications that

require the implementation of a variety of mathematical functions, such as flight dynamics) must be extremely cautious, because the field is much more limited in hardware suitable for such applications.

2.1.3 GROWING PAINS OF A NEW INDUSTRY

The release of new products in any field cannot be expected to be entirely free of kinks and bugs. Time and widespread usage must still be considered the ultimate tests of a product's reliability. A brief recounting of bugs found in new 16-bit microprocessors is offered below as an example of the existence of such developmental problems.

In the March 1979 issue of BYTE magazine (Reference 3), a preview was given of the Zilog Z-8000 microprocessor chip. In addition to its commendable specifications of speed, already presented in Table 2-1, the following features were announced:

- Memory management hardware (e.g., storage write protection)
- Powerful, multimode instruction set
- Possible system bus structure oriented toward high-performance network systems
- Disk-based development system with higher level languages

The article's author called the Z-8000 "the best thing yet for personal [this author's emphasis] applications which involve number crunching."

Just 1 year later in the same journal (Reference 6), the following facts were disclosed. The preproduction samples of the Z-8000 reportedly did not execute all instructions correctly. In addition, various peripheral devices and chips (e.g., the memory management chip) were unavailable, as was

software from the manufacturer or other vendors, at the time of writing. The preproduction samples of the Motorola 68000 (16-bit) microprocessor also exhibited symptoms of improper instruction execution and incapability of executing at the maximum rated speed.

2.1.4 SOFTWARE EVOLUTION

This section provides a brief discussion of the overall change in software development facilities, or system support software. A closer view of specific facilities and the implications of their presence or absence in the development of computationally complex software is presented in subsequent sections. In contrasting the progress made in machine hardware with the changes in software development methods, it must be borne in mind that the advances in hardware are restricted to one area of technology, whereas the activity of developing software applications and system support software involves working with new areas of logic, human-to-machine interfaces, and human-to-human interfaces.

The problems of understanding and handling the new areas of logic and interfacing have only relatively recently been partially identified and attacked. Therefore, the changes in software have been much less dramatic than those in machine hardware. In addition, the computer applications being undertaken have become so ambitious and complex as to merit the descriptions of "nearly impossible" and "utterly absurd" (Reference 7, pages 252 through 253), thus aggravating the already wide disparity between the states of the art of hardware and software technologies.

It is not surprising that by the early 1970s it was recognized that there was a widening gap in the cost of software versus the cost of hardware. Studies showed that in some large-scale computer applications, approximately 75 percent

of the cost was attributed to software development, and the remainder was expended in hardware purchases. It was predicted that by 1985 the software/hardware cost ratio would reach 90/10 (Reference 8). The point to be made here is that when choosing a software development configuration for a given application, the availability of proper software development facilities (discussed in Section 3) should not be sacrificed in order to achieve only a savings in hardware, because the penalties incurred (lack of reliability due to rushed system testing, programmer morale deterioration, accelerated obsolescence) by delays encountered during system development will probably far exceed the hardware savings.

2.2 ORBIT APPLICATIONS

With a cursory view of the hardware revolution that gave rise to the microprocessor having been provided in Section 2.1, this section examines the application of the new technology to flight dynamics problems. Section 2.2.1 specifies the characteristics of flight dynamics problems in general and the orbit determination problem in particular that are of primary importance in dictating the choice of computing systems for their solution. Section 2.2.2 presents the original overall plans that were proposed in Code 582 of GSFC for a microprocessor-based network for orbit and attitude simulation and determination. Section 2.2.3 provides an evaluation of the CSC microprocessor orbit determination software development activities performed for GSFC.

2.2.1 SALIENT CHARACTERISTICS OF ORBIT DETERMINATION WORK

Several characteristics of orbit determination work set it apart from many other problems, scientific or otherwise, that have been attacked with microprocessor-based technology.

These characteristics, many of which are applicable to flight dynamics problems other than orbit determination, are as follows:

1. The computer's work (in this case, number crunching) is basically CPU bound.

2. A high-precision representation of numbers is required.

3. Large data structures must be used during some computations. The use of peripherals here would be too slow, implying that storage in memory is necessary.

4. There is little parallelism in the usual division of the problem into its mathematical components, although a small degree of parallelism is pointed out in the design example of Section 6. The steps of force evaluation, integration, observation modeling, and filtering are basically sequential.

5. Parallelism of specific operations in the form of vector or array processing is of a small dimension as compared to weather prediction or diffusion problems (numerical solution of partial differential equations). Nevertheless, special hardware units for Kalman filtering problems have been built (Reference 9, pages 94 through 102 and 172 through 178) for large machines. A special high-speed vector processor suitable for the LSI-11 microcomputer has been built and used (Reference 10). There is no reason in principle why microprocessor-based systems for orbit determination work could not profit from such hardware as long as (a) modern software development methods could still be used in the implementation process and (b) the hardware is flexible enough to accept dimension as an input parameter

(i.e., tying the modeled state to a specific dimension may be too restrictive).

6. As a restatement of the first characteristic, the real-time input/output requirements are usually minimal. The input of an orbit determination system may come solely at the beginning of a run (post-processing tracking data), in infrequent bursts of large blocks (upload set of tracking data received at a ground site), or at a steady, low rate (Global Positioning System (GPS) data). This is vastly different from the demands of an inflight data acquisition system constantly taking up to 400 readings per second of a signal strength level (Reference 11) and performing some statistical analysis. The latter system is essentially input/output bound (the data were stored on a cassette, and periodic printouts were made), and the design described in the reference reflects this characteristic. The demands of an onboard attitude determination for sensor sampling, on the other hand, would probably be sufficiently low that input/output would not be the main consideration.

7. The development of orbit determination software (or any software that performs a great deal of processing) requires some special consideration in the development facilities chosen. The very nature of this type of software dictates the capability of supplying the programmer/analyst/tester with a large amount of formatted output when needed. Current microprocessor-based development systems may have the following deficiencies: (a) limited input/output capability (no operating system, as with the IMP-16 development system, or only formatted octal dumps and traces, as with the NASA Standard

Spacecraft Computer-1 (NSSC-1) simulation support facilities) and/or (b) limited printing facilities (no fast, reliable printer--an expensive component not usually found in, or compatible with, microcomputer systems).

2.2.2 EARLY PROJECT PLANS FOR MICROPROCESSOR-BASED ORBIT AND ATTITUDE SYSTEMS

An overall plan involving a minicomputer and a network of several microprocessors for performing a large-scale simulation of attitude sensing, orbit determination, attitude determination, and ground control commands was proposed in Code 582 of GSFC around 1976 (Reference 12). A graphic presentation of part of this proposal is given in Figure 2-1.

The assumed advantages of having such a facility based mainly on microprocessor hardware were

- Low-cost hardware
- Independent, modular development
- Capacity for expansion with additional hardware modules

The first step in developing this simulation facility was the design and implementation of the IMP-16 Orbit Determination System described in Section 2.2.3. This constituted the arrival of a new application for microcomputers in the Code 580 environment.

2.2.3 DEMONSTRATION SYSTEMS DEVELOPED

The orbit applications software systems addressed in this document are as follows:

- IMP-16 Orbit Determination System (ODS) (composed of two functionally independent CPUs: a data base CPU and a computational CPU)
- INTEL 8080 Orbit Propagator

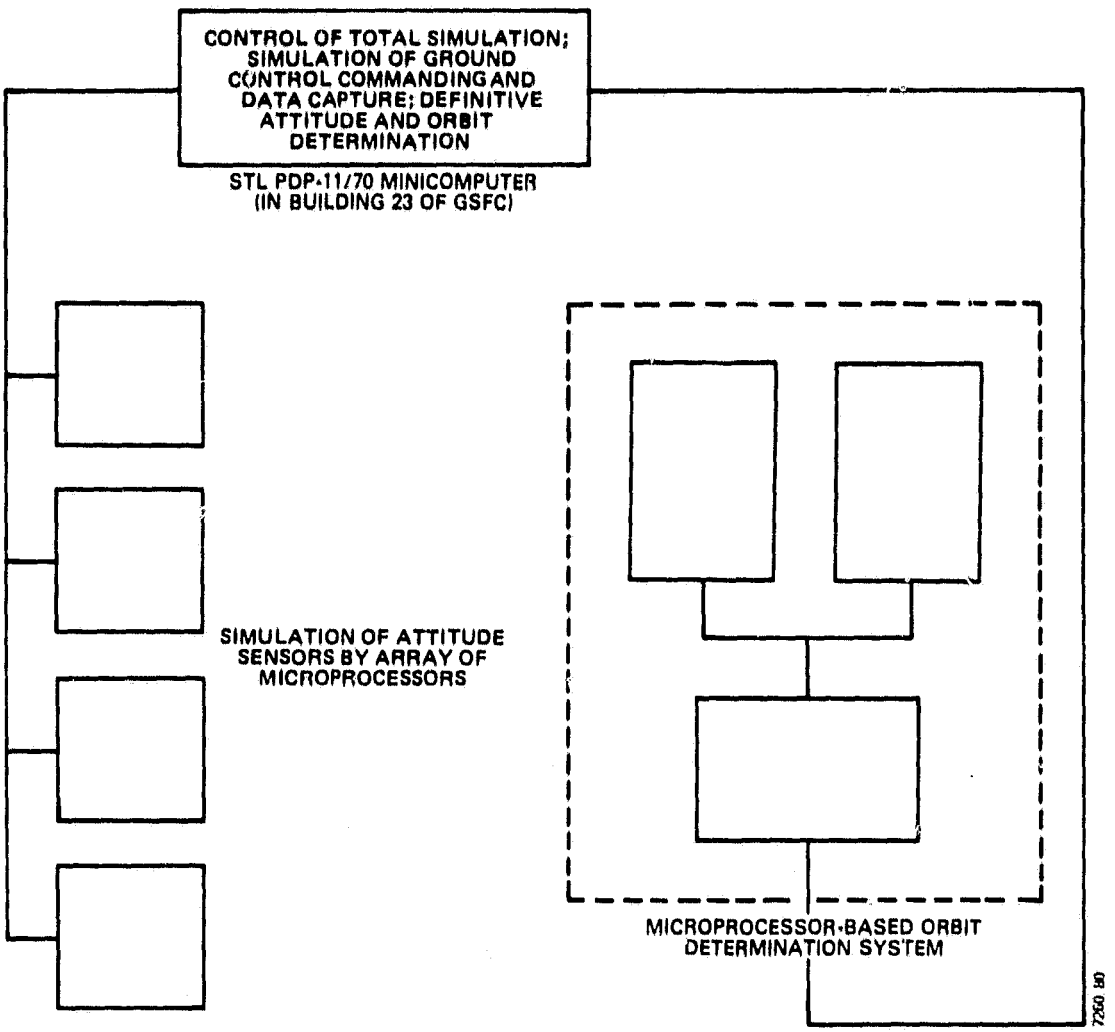


Figure 2-1. Early Plan for Utilization of Microprocessor Technology for Orbit and Attitude Applications

- IMP-16 Tracking Data Preprocessor (under development; contained in the IMP-16 ODS data base CPU)

From these software development efforts, several important lessons were learned. These lessons are based on the difficulties encountered, many of which are specified in Sections 2.2.1, 3, and 7.

The defining document of the IMP-16 ODS is Reference 13. A paper summarizing the project's results as of October 1979 was given at the 1979 Flight Mechanics/Estimation Theory Symposium at GSFC (Reference 14). The paper's observations and conclusions are summarized below.

The advantages of the IMP-16 ODS are as follows:

1. Low-cost hardware--The two IMP-16 microprocessors plus the required memory cost approximately $3000.
2. Portability--The hardware system is totally portable. The hardware with firmware (software in PROM) can be taken to any ground site where tracking data is available (for the appropriate spacecraft), and processing can then be begun (after certain initial conditions have been input).
3. Modularity--More memory or more processors can be added to the system in order to meet new project requirements.

The disadvantages encountered during the development of the system included the following (for further details, see Section 7.1):

1. Lack of a reliable compiler
2. Lack of facilities for link-editing and relocating separately compiled modules
3. Lack of facilities for transferring code developed in random-access memory (RAM) to PROM code without modification

4. Lack of manufacturer-supported hardware diagnostics and field engineering personnel (although personnel of the GSFC Microprocessor Development Facility in Building 23 responded admirably well to all hardware problems)

5. Lack of software diagnostic tools that could separate software problems from hardware problems

6. Lack of an operating system in general

7. Lack of precision in the floating-point numbers (less than 10 digits)

The overall results have

- Demonstrated the use of microprocessors for sophisticated computational models
- Demonstrated the need for a fuller complement of system software in order to aid development
- Demonstrated the efficacy of dividing a sophisticated problem among microprocessors
- Demonstrated that numerical accuracy--not speed or memory requirements--is the main problem in orbit determination with the IMP-16 microprocessor and its accompanying floating-point package
- Confirmed the correctness of the approach of conducting studies on a mainframe in order to choose proper models and use of precision

The INTEL 8080 Orbit Propagator (Reference 15) development experience was for the most part different than that for the IMP-16 ODS. An operating system with file management (for floppy disks) aided development. Accuracy was not a problem, in that the equivalent of IBM REAL*8 was available (precision

of more than 16 digits). The primary difficulties encountered with the project were the following (see Section 7.2 for further details):

- Lack of standard FORTRAN statements
- Lack of mathematical subroutine library
- Presence of a few compiler bugs and undocumented limitations
- Slowness of the floating-point software (the INTEL 8080 is an 8-bit machine)

However, the overall results of the effort were that the INTEL propagator results very closely matched the results of test cases run on the IBM S/360-95. The conversion from the development version on a Tektronix system to the final PROM chip version was handled very efficiently by GSFC Microprocessor Development Facility personnel.

SECTION 3 - MODERN SOFTWARE DEVELOPMENT PRACTICES APPLIED TO MICROPROCESSORS

Only in approximately the past 12 years has the discipline of software development been receiving the attention it requires. The basic system support software (i.e., assemblers, compilers, linkage editors, and operating systems) has been available for some time. However, the actual processes of designing, coding, and testing programs, as well as the coordination of the efforts of many people performing these tasks, have only recently been held up for close scrutiny and discussion. The overall movements that encompass these investigations can be considered now to be the areas of software engineering (development practices and management) and software science (metrics and formalized axioms and proofs). The aim of this section is not to survey the aforementioned fields in general, but rather to extract from them methods that should be used for microprocessor-based software development for flight dynamics applications.

3.1 GENERAL SOFTWARE DEVELOPMENT CONSIDERATIONS

In general, each of the following subsections briefly introduces one aspect of software development, provides appropriate reference(s) for further discussion, demonstrates how the absence or improper implementation of this method can hinder or cripple the development of computationally complex software, and relates the software method or facility to developing a microprocessor-based system.

3.1.1 SOURCE LANGUAGES AND STRUCTURED PROGRAMMING

A primary concern in initiating a software project is the choice of programming language(s). Assembly language has its advantages in usually being an efficient means (i.e., requiring minimal core and machine time) of handling complicated and frequent input/output as well as byte and/or

bit manipulation. However, as noted in Section 2.2.1, these operations were not of primary concern in the GSFC projects, whereas numerical computation was. Problem-oriented (higher level) languages enable a user to address a problem directly, rather than being burdened with the many considerations involved in utilizing the machine's hardware, as assembly language programming requires.

As stated previously, the average programmer production (for given categories of programs) in terms of the number of debugged source language statements written is approximately constant over several projects. When the power (i.e., the economy of expression) of a higher level language such as FORTRAN or PL/I is taken into account, the resultant mean productivity in terms of object code produced can be as much as five times greater with higher level language programming than with assembly language (Reference 16, page 94). Assuming that the compiler (language translator) for the higher level language is not grossly inefficient (or bug ridden), a scientific programming language such as FORTRAN or PL/I (or an adaptation thereof) should be considered as the natural choice for building flight dynamics software.

One possible objection that can be made to choosing a higher level language when using a new development system (see Section 3.2) is that the assembler is in principle a less complex piece of system software than the compiler, and hence assembly language can be used with much greater confidence than the compiler language. If this is the case, the development system itself, rather than the use of a higher level language, should be rejected. Examples of bugs exhibited in microprocessor-based compilers used by CSC thus far in orbit determination work are provided in Section 7.

Another idea used in choosing a primary programming language is structured programming. A full discussion of this topic is beyond the scope of this document. It is sufficient here to state that structured programming is basically the principle that a program can be synthesized from, and decomposed into, the following basic units of structure:

- Alternation (binary or multiple-case decisions)
- Iteration (e.g., the DO-loop)
- Sequencing (straight-line, sequential code)

The benefits of using structured programming principles include

1. A consistent methodology of design, coding, and testing
2. A clearer presentation of the relations between problem, design, and code
3. A disciplined method of exercising statement sequencing and control

The third benefit listed above is particularly important, since an almost incomprehensible maze of control transfers can be created by unsound use of the GO TO and other control statements. In fact, studies of various large programs coded in FORTRAN have shown that errors within statements belonging to the category of sequencing and control account for a minority (approximately one-third) of the errors detected during early stages of testing but approximately one-half of the errors detected during the later stages (system testing) (Reference 8). Since it is almost universally accepted that a bug detected in the later stages of testing is more expensive to correct than if it were detected in the early stages, the discipline of structured programming is certainly appealing in its concentration on controlling the genesis of these costly errors.

It should be noted that the initial efforts in developing theories of structured programming were primarily aimed at banning the use of GO TO statements (Reference 17). Since that time, most of the work in structured programming has emphasized the presence of structure rather than the total prohibition of GO TO statement use. This is considered here to be the correct approach.

As noted in Section 2.1.4, any tool that can help close the gap between the respective costs of hardware and software must be seriously considered. Thus, consideration should be given to choosing a structured programming language, i.e., one in which the necessary elements of structure are part of the language. Although standard FORTRAN does not include enough proper structures for implementing structured programming directly, structured programming can be used with FORTRAN if the developer (1) designs the program with proper structures down to the final step of coding and then uses the basic language elements for implementing the structures or (2) obtains or builds a preprocessor that will accept (nonstandard) elements of structure in FORTRAN code (Reference 18).

3.1.2 SOURCE FILE HANDLING

A software development system must provide reliable and powerful aids for helping the programmer implement code for programs of any degree of complexity. If the programmer must expend great effort to change, store, and retest the code, then his/her productivity must necessarily suffer. The development system must offer facilities that assume the many bookkeeping tasks associated with developing software. The following points should be noted:

1. A powerful and reliable text editor, the most frequently used software development aid (Reference 19),

should be available. The set of editing commands for the IMP-16 text editor was comprehensive enough; however, the execution times for some commands (e.g., relocating a block of consecutive statements) were so slow that a poor implementation could be suspected, since the execution for numerous floating-point operations was not unreasonably slow.

2. A large and reliable storage medium must be present. It was not until task personnel were well into the IMP-16 project that the proper way of initializing new floppy disks (i.e., by copying the contents of an existing, working floppy disk onto the new one) was learned and then practiced. Until that time, disk errors severely hampered development efforts. In addition, it was found that the disk drives of the two National Semiconductor IMP-16 development systems were incompatible under certain operations (see Appendix A, item 1). Finally, the single-side, single-density floppy disks used provided a little more than 150K (16-bit) words in 616 sectors; and when the entire system was present on the initial part of the disk, only approximately 55 percent of that capacity was available for application program storage. Thus, keeping more than one version of routines during development involved keeping track of multiple volumes of floppy disks, when the desired number of disks were available for storage.

3. A filing system for (mounted) storage media should be present. In this way, the bookkeeping involved in keeping track of source, object, and load modules is assumed by the operating system, thus freeing the programmer for more fully utilizing his/her

technical skills. The Tektronix 8004 development system used for the development of the INTEL Orbit Propagator had a reliable file management system. However, the IMP-16 development system had none, and programmers were forced to keep track of all modules stored on floppy disk throughout the development and test phases. In addition to being a major distraction from the programming effort itself, this led to errors of misplaced updated versions and overwritten files (especially when a compilation or load module output exceeded estimated bounds). The point here is that the facilities and practices of software development used on big machines should not be overlooked when developing sizable software on small machines.

4. If possible, the capacities of the development aids should be ascertained before a commitment to their use is made. It was found that the source code editors of the Tektronix and IMP-16 development systems and the SM/PL compiler of the IMP-16 system could no longer accept the large files being input. The files became very large due to the lack of linking facilities in the systems. This overflow (of tables and work areas) necessitated segmenting the software, sometimes with a loss of time due to the efforts needed for new interfacing requirements. Thus, larger capacity system programs would have been more desirable (of course, the constraint of 16K bytes of RAM in the IMP-16 and a similar constraint in the Tektronix system were the main causes of this shortcoming).

3.1.3 MODULARIZATION

The principle of modularization is that a software program or system should be broken down into several smaller,

manageable parts. In this way, the system will likely be easier to code, test, and maintain than if it were composed of very large segments of code. Considerations of module size, individual module structure or purpose(s), and inter-module relationships are discussed in Chapter 3 of Reference 7 and throughout Reference 20. The development of software for orbit determination demands an approach using modularization; any other approach makes the verification of numerical accuracy and the tracking down of computational bugs in such a complex process almost impossible.

The developer of orbit determination or any other computationally complex software should seek the following facilities in a development system:

- It should be easy to handle several separate source files with successive versions. This relates to the considerations of a filing system and reliable storage media discussed in Section 3.1.2.
- The system should allow for unit testing of individual modules followed by easy integration of the modules into a larger component or the entire system. The resetting of compilation parameters should not be required for the integration. To achieve this, the compiler should generate relocatable code, and a linkage editor should be available for correctly linking the individual modules.
- Global data references (such as COMMON in FORTRAN and EXTERNAL in PL/I), as well as external program references, should be allowed. Although these facilities are not part of the concept of modular programming (in fact, some persons believe that they violate modularity (Reference 20, pages 37 through 43)), they are of extreme practical importance.

The compilers and loaders used for the microprocessor software development projects described in Section 2.2.3 lacked the facilities for generating relocatable code and for directly linking several modules. The unexpected effects encountered in working around these problems are described in Section 7 and Appendix A.

3.1.4 FLOATING-POINT CAPABILITIES

The developer of software that involves much noninteger computation is faced essentially with two choices: fixed-point or floating-point formats and computations.[1] The choice of fixed-point computation has the following advantages:

1. The basic hardware integer arithmetic operations can be used directly for their given precisions (i.e., single-precision integer arithmetic will serve for single-precision fixed-point arithmetic).
2. The available machine words are used for significant scaled digits only; the exponent, or scaling factor, is not held in physical storage.

The discipline of coding fixed-point operations is aided by the existence of a convenient notation (Reference 21, Section 6). Nevertheless, the system designer must bear in mind the following disadvantages that fixed-point arithmetic entails:

1. It is time consuming to program, especially when there are large numbers of variables and operations to keep track of, since the possible range of values for every quantity must be accounted for.

---

[1]The set of arithmetic capabilities is only one criterion for choosing a language; e.g., the language may offer floating-point arithmetic capabilities and yet not allow certain operations, such as shifting.

2. It is a tedious and error-prone process due to the number of shifts and rescalings that must be accounted for while coding (for example, see comments at the beginning and generally throughout the listing of the NSSC Orbit Representation Program in the appendix of Reference 21).

3. For a process as complex as orbit determination, some parameters may vary over several orders of magnitude. An example of this is the atmospheric density parameter used in drag routines, especially if the orbit is highly eccentric and has a low perigee. The ensuing problem here is that with a fixed scale factor, the quantity loses many significant digits (i.e., takes on many high-order zeros) in going from its largest allowed value to its smallest. One possible solution, which entails much overhead in planning and coding, is dividing the problem (the orbit, here) into more than one region, each region having its own scale factor for atmospheric drag.

4. Some statistical quantities found in estimation problems such as orbit determination can vary over several orders of magnitude, entailing a loss of significance as discussed in item 3 above. The previously suggested idea of sectioning the problem may no longer work here--the variation of the quantities may not be that readily predictable. For example, the conservative, a priori state error covariances used at the beginning of a filter run may decrease, at some unpredictable rate, by several orders of magnitude during the run.

5. Fixed-point arithmetic may be implemented directly with a precision as great as that of the hardware integer operations available on the machine. For

extended precision, the use of extended-precision software routines is required, thus slowing execution time. If these routines are not already part of the development system, effort must be spent in developing them.

Most of the disadvantages associated with programming fixed-point operations do not exist with floating-point arithmetic. Rather, the penalty paid is in execution time, especially if the floating-point operations are implemented in software on the microprocessor. The results of the past projects considered here are presented in Section 7.

In choosing a floating-point facility, whether hardware or software, the following points should be considered:

1. The precision of the floating-point numbers used in the implementation of the operations should satisfy the requirements of the project at hand. The IMP-16 system floating-point software package provided 32 bits for the twos-complement representation of the mantissa, or fractional part, of each floating-point number, which is equivalent to slightly more than nine digits. Although the objectives of the IMP-16 ODS were met for one orbit and could be extrapolated to a second orbit (Reference 22), difficulties were encountered along the way. The accuracy requirements imposed were not very stringent; highly precise orbit determination work appears to be possible only when precision on the order of IBM or DEC PDP FORTRAN REAL*8 (approximately 16.7 decimal digits) is available.

2. The penalty paid in speed may be excessive when floating-point operations are implemented in software. This is especially true of REAL*8 (8 bytes

for fraction and exponent) being software simulated on an 8-bit machine (see Section 7.2).

3. A new floating-point system, be it hardware or software, if not developed with care and for sophisticated users, may lack provisions for guard digits. Guard digits are extra digits (bits) on either side of the fraction (mantissa) within the arithmetic registers; their purpose is to ensure the fullest possible accuracy for the given significance for each operation (Reference 23). In particular, software-implemented operations are almost guaranteed not to have the (two) necessary guard digits on the right-hand side, since this need does not arise in the integer (or fraction) arithmetic operations upon which the software implementation is based.

3.1.5 MATHEMATICAL SOFTWARE LIBRARY

Persons who have worked with FORTRAN or any other higher level language on large machines and, in recent years, on minicomputers have been able to take for granted the existence of software mathematical routines that are directly callable from the language. The functions included in this category are nominally the square root, exponential, logarithm, trigonometric, hyperbolic, and miscellaneous arithmetic functions.

The situation with almost all microprocessor-based systems is quite the contrary: few, if any, functions are provided. (This circumstance is closely connected with the lack of capabilities of generating relocatable code and performing full link-editing, as discussed in Section 3.1.3.) The development efforts for the IMP-16 and INTEL applications, as well as a previous NSSC project (Reference 21), included time

spent in coding and testing most of the necessary routines. Although good references are available from which algorithms of the desired precision can be expeditiously extracted (for example, References 24, 25, and 26), this "reinvention of the wheel" does require

- An expenditure of time away from the main work at hand
- Methods and facilities possibly beyond the bounds of a microprocessor system for proper validation of the new routine(s) (Reference 27, Part V)
- An implementation that is not optimal--For example, the INTEL Orbit Propagator's square-root routine, coded in FORTRAN along with the rest of the system, required numerical tests on the argument in order to choose a reasonably good initial value ("first guess") for the standard Newton-Raphson iteration scheme. The apparent alternatives were either to allow the method to make so many iterations that computation would be excessively slow and accuracy probably lost or to code the routine or part thereof (near optimally) in the INTEL assembly language, with which the project programmers were not familiar.

Thus, the prior existence of a well-tested mathematical software library is desirable for avoiding a loss of time in the development of orbit determination or similar software. As an example, the LSI-11 series of microcomputers support the full FORTRAN-IV PLUS compiler and its mathematical subroutines library (along with microcoded floating-point implementation of REAL*4 and REAL*8 operations), which has already been well tested by a large community of users over the span of a few years.

3.1.6 MINICOMPUTER AND LARGE MACHINE SUPPORT

Microcomputer development systems of today are hard pressed to meet the demands for reliable, powerful system software and peripherals, as specified in the preceding sections. An alternative that has been available in certain systems for some time is the following. A language translator (compiler or assembler) and a compatible linkage editor are supported by a minicomputer or a large machine. Then, the final code output (the load module) is that which is appropriate for the microprocessor. This final code can then be downloaded to the microprocessor or possibly executed by a simulator for the microprocessor (for preliminary testing). In this way, the developer benefits from the reliability and convenience present in the larger system. One important aspect of minicomputer and large-machine support of microprocessor software development is the system test library and support programs, which allow for the orderly storing of benchmark test case input and output data and for the execution of test run result analysis and comparison programs.

This methodology of microprocessor software development certainly appears to solve many of the previously described problems. However, since such facilities will often cross boundaries between different manufacturers, they should be suspected of having some heretofore undiscovered bugs or, at least, erroneous interpretations of language specifications. Also, differences in hardware (e.g., storage size) must be considered. An almost ideal example of minicomputer support of microprocessors is the LSI-11 microprocessors, which are supported by the software available for the DEC PDP-11 family of minicomputers, of which they are downward-compatible members.

Another aspect of larger machine support of software development for a microprocessor system does not involve the

interfacing of software between different machines. This is simply the strategy of utilizing the larger machine for performing preliminary studies or for producing benchmark or truth run results against which the microprocessor results can be compared. This methodology was used in designing and developing the IMP-16 Orbit Determination Program (Reference 13, Section 2).

3.2 MICROPROCESSOR SOFTWARE DEVELOPMENT

The previous subsections of Section 3 have described various development methods, facilities, and support software that are applicable to developing software for most electronic digital computers. This subsection notes some basic distinctive features of microprocessor software development.

A distinction is eventually made in the microprocessor software development process between fixed executable code (with fixed program constants) and variable, "erasable" program areas. The fixed code and constant areas are designated in this document as PROM (programmable read-only memory). PROM content remains intact between machine power-down and power-up actions. The erasable work area is conceptually the same type of main memory that is familiar to users of larger machines. This storage, which does not hold its value after a machine power-down operation, is usually designated as RAM (random-access memory).

The microprocessor programmer must invariably consider two machines for his/her project. One machine may be a microcomputer that is the development system. This usually contains some bulk storage medium (such as paper tape, cassette tape, or floppy disk) and system software (such as an editor and compiler) with which applications software can be developed. The second machine is the microcomputer in which the final software will be installed. This is referred to as the

target system. It usually consists of the processor(s), peripheral interfaces, and memory areas (RAM and PROM) that are required for the specific application.

The software development process is almost always carried out at the microcomputer development system itself in an interactive fashion. There is usually no modem connection (i.e., telephone line hookup) available. Compilations and test runs are made by the single user of the development system at his/her command, rather than proceeding from a work queue of several jobs such as would be found in a batch computing system. The discipline of microcomputer software development in which the development system is a minicomputer has already been discussed in Section 3.1.6.

SECTION 4 - SPECIAL CONSIDERATIONS IN DEVELOPING MICROCOMPUTER APPLICATIONS SOFTWARE

As mentioned in earlier sections of this document, the digital computer solution of flight dynamics problems was confined for many years to the domain of large computers that offered the computational power and input/output facilities required. Only with the advent of powerful minicomputers with the appropriate floating-point and general mathematical facilities were attempts made to solve orbit determination problems on minicomputers. The current microprocessor revolution in computer hardware is opening a new era in further miniaturized, specialized, and localized implementation of computational systems of all types, and orbit determination is one of the areas being explored in this regard.

From past experience in scaling down from a large computer system to a minicomputer, many lessons can be drawn about adapting to a new, smaller computational environment. As stated in Section 3, for performing orbit determination work, the facilities of the development system and final (target) system should resemble those found in larger systems, if heavy penalties in development time, reliability, and accuracy of results are to be avoided. However, some concessions must be made for reduced computational power, storage, and input/output facilities of the target microprocessor system, and these concessions must be carefully specified and designed at the beginning of the project.

Section 4.1 discusses considerations that must be given to the algorithms used. Section 4.2 discusses some design concessions made in past projects for file handling and storage in a minicomputer or microprocessor-based system.

Section 4.3 discusses the concept of using network or multiprocessor configurations of microprocessors for increasing throughput (i.e., all system work accomplished from the given user load). Section 4.4 briefly discusses a technique for increasing software fault tolerance, which is particularly applicable to onboard, real-time systems.

4.1 RECASTING ALGORITHMS

In designing computationally complex software for implementation on a microprocessor-based system, the analyst must reconsider the appropriateness of both standard and new algorithms for use in his/her project. Potential numerical problems must still be considered. Briefly, such problems include

- Accuracy of numerical operations and, in particular, the subtraction of nearly equal numbers (Reference 28, Chapter 1)
- Poor conditioning in linear algebraic systems (Reference 29, page 150)
- Poor conditioning of polynomial roots (Reference 28, pages 22 through 24)
- Numerical instability such as that caused by parasitic solutions in the Milne method of numerical integration (References 29, pages 121 and 122)

The expected, or typical, accuracy of a computationally complex algorithm may be impossible to determine directly (a priori). Existing reports of results obtained from the use of the algorithm should be considered. An example of such a report is the following. A simulation of an interplanetary trajectory was made, which involved a 30-day arc of over 600 points (observations) and which used several Kalman filtering implementations (algorithms).

The results were accurate to within 10 digits at best. Arithmetic of 18 digits (double precision) on a UNIVAC 1108 was used (Reference 30, page 12). From reports such as this, the minimum precision needed to achieve a desired accuracy of the final computations can be determined.

The following examples illustrate decisions made in the past in view of the limitations on computations of the systems developed:

- Microprocessors are inherently much slower than their larger computational counterparts due to the limitations of the technologies (material composition) of their LSI circuitry. Thus, algorithms that are less expensive computationally should be chosen whenever possible. The modified Harris-Priester atmospheric density model was chosen over the Jacchia-Roberts model for the IMP-16 Orbit Determination Program for this reason.
- The order, degree, or scope of models should be limited. The geopotential field for the IMP-16 project was restricted to a 6-by-6 expansion maximum, with the default being set as 4 by 4. Preliminary analytical studies carried out on the IBM S/360-95 helped determine a reasonable, minimum size of the model for this microprocessor implementation (Reference 13, Section 2).
- For the INTEL 8080 Orbit Propagator, execution time was reduced by saving the sum of the perturbing forces from the prediction step of the Adams predictor-corrector and adding this sum (a vector sum) to the recomputed central body force at each subsequent correction step (Reference 15, pages A-13 and A-15 through A-17). In this manner, only

the dominant term of the sum of forces on the spacecraft is adjusted during the small correction steps, and this computation is trivial compared to the full acceleration computations otherwise performed. Since the orbit propagator is intended for orbits of the geosynchronous class, this omission of the variation of the noncentral body forces through the iterated correction steps is certainly justified, and the saving in execution time is significant.

The following examples present algorithms that were intended for the NSSC-1 on SMM. The operations described are all fixed point; single-precision operations are hardware implemented, and double-precision operations are software implemented.

- Considerations of the magnitudes of quantities can show that a predicted change of state can only be of such small magnitude that the high-order term of the old state (in double precision) cannot be affected, except possibly by a carry from subtraction. In that case, the changes can be computed in single precision and added to the low-order words of the respective components of the old state (Reference 31, pages 12 through 16). The costly operations in double precision are then avoided.

- Division by two or small powers thereof, even in single precision, can be avoided by performing the appropriate number of shifts (Reference 31, page 15). Since a check for positive signs and correct relative placement of significant digits (bits) of the operands was necessary before dividing, the substitution of a shift was an economizing and simplifying action.

- Double-precision (software) division was not direct. It was implemented as a Newton-Raphson iterative scheme (Reference 31, pages 38 through 40). This costly operation was avoided in the normalization of the attitude quaternion by performing a Taylor series expansion of the reciprocal square root of the sum of the squares of the quaternion components. Consideration of the magnitude of this sum and required accuracy implied that only a trivial series of operations need be performed (Reference 31, pages 17 and 18).

The choice of a system with floating-point facilities and precision of the caliber of those found on large machines should be made for orbit determination applications in order to obviate the need for manipulations such as those described in the preceding examples.

4.2 STORAGE CONSIDERATIONS

In almost all cases, the memory (core) and peripherals for an operational microcomputer system will be less extensive than the facilities for a larger machine. A 16-bit word size usually dictates a maximum of 64K bytes of memory (RAM plus PROM), although a system with multiple banks of memory and a paging register can overcome this limit (e.g., the NSSC-1 minicomputer has this page register facility). Thus, economization of storage is required. However, the designer of a microprocessor-based orbit determination system is then faced with several problems, since, as pointed out in Section 2.2.1, orbit determination algorithms require relatively large data structures, especially if an iterative solution (differential correction) for a state is being performed.

The designer of an onboard orbit determination system faces a more acute problem. The resources of an onboard computer are even more constrained in that there are limitations on

space and power and prohibitions against both angular momentum changes and currents not controlled by the attitude control system. Therefore, in general, the only storage available is RAM and PROM memory, making the memory constraint problem worse.

The following examples illustrate the principle of planning carefully for storage for some orbit applications programs:

- The ephemeris representation program implemented on the NSSC-1 minicomputer for use on board SMM was designed for accepting an upload data set that would include single-word residuals for ephemeris point components. The program computed the remainder (double-word part) of the ephemeris points by evaluating a Fourier power series for each ephemeris point component. The entire package eliminates the need for storing an entire ephemeris onboard (for 3 days) while satisfying the need for onboard, real-time ephemeris data (Reference 21).
- A feasibility report was submitted by CSC concerning the transfer of an existing estimation executive, the ENTREE program, from the CDC Cyber 176 to the LSI 11/23 microcomputer (Reference 32). It pointed out that the following considerations of storage should be made:
  - The frequent, lengthy reports originally output by the program on the large machine would have to be greatly reduced, due to the limited storage for buffers and to the less powerful printing facilities.
  - The previously external (disk or type) file of observation data would have to reside in memory, and the size would therefore be reduced.

- The large COMMON blocks would have to be trimmed.

Another important aspect of planning the storage is the following. There is an increase of between 30 and 50 percent in programming effort (relative cost per instruction) when 85 percent of the machine's speed and memory capacity is utilized (Reference 8). Thus, the proper use of space must be a primary aspect of the design of an orbit determination system implemented on a microprocessor.

4.3 NETWORK CONFIGURATIONS OF MICROPROCESSORS

The concept of computers working together in a cooperative manner to meet user needs is not new. This principle was used in some large systems at first; however, only with the advent of the minicomputer was new flexibility found in building systems with more than one processor. Systems with one large computer and one or more minicomputers (e.g., working as front-end processors to the large machines) soon appeared. Networks with minicomputers alone were also considered and developed. The different types of configurations included single-bus, multibus, multiported memories, and private memories, among others. Reference 33 provides a basic discussion of the advantages and possible disadvantages of a minicomputer network, as well as the distinctions between different types of configurations.

The arrival of microprocessor technology implies that the substitution of microprocessors for minicomputers in many types of networks is now feasible. The advantages of "stacked" microprocessors include the following (Reference 5, pages 208 through 210):

- Microprocessor hardware is low in cost.
- Hardware can be specialized for each application (e.g., peripheral controllers, computational processors, executive and message routine controllers).

- Software can be truly modularized and, thus, specialized on a processor-by-processor basis (e.g., interrupt-driven processing versus sequential).
- A system can be easily upgraded or expanded.

The previously cited reference also emphasizes configurations in which most of the memory is shared among the processors rather than being private memory (i.e., unique storage for each processor). The advantage of such a configuration is that microprocessor chips are relatively slow and inexpensive compared to available memories, which are faster and more expensive. The point is that adding processors (and, thus, more capabilities) is inexpensive; adding memory units currently is not. However, the disadvantages of such systems are as follows:

- The proposed switching matrix for giving access of the several ports of memory to each microprocessor in the system and resolving conflicts therein is a complex and expensive piece of hardware known as a crossbar switch; by its very nature this piece of equipment is much more appropriate for large-machine multiprocessor systems (Reference 34, page 129; Reference 35, page 34).
- For orbit determination applications, the need for manipulating large arrays and the general lack of peripheral activities (i.e., the system is CPU bound) would probably result in too-frequent memory conflicts (see Section 2.2.1).

As with minicomputer systems, microprocessors in a system can be connected to one another by various configurations of buses and by individual and shared memory. Private memory avoids the problems and delays of resolving conflicts

that arise with shared memory. A good system configuration that includes both kinds of memory and uses the "mailbox" system of interprocessor communication is the following (Figure 4-1 and Reference 34):

1. Processor 1 is the coordinator for this system; all other processors communicate with it but do not communicate directly among themselves.

2. Processor 1 can send a message to processor 2 by placing the message in the beginning of the memory area designated as box 12.

3. Processor 2 periodically checks box 12 to determine whether it has a message from processor 1 to pick up and, if so, does so.

4. Processor 3 can send a message to processor 2 by placing in box 31 a message for processor 1 to forward the original message to processor 2. Processor 2 receives the intended message in its box 12.

5. Additional shared memory is used for transferring large blocks of data among processors.

The steps specified above are only the most basic definitions of the system's operations; the details of initializing the message areas, of confirming the receipt and/or execution of the messages (which can be commands), and of the protocols used are not discussed here. This basic scheme may be modified by specifying that a sending processor interrupt the intended receiving processor of a deposited message so as to alert the latter to pick up the message. This modification has the advantage of speeding up the overall message routing process without requiring the coordination and conflict resolution of conventional interrupt-driven input/output. In a system with N microprocessors, direct communication

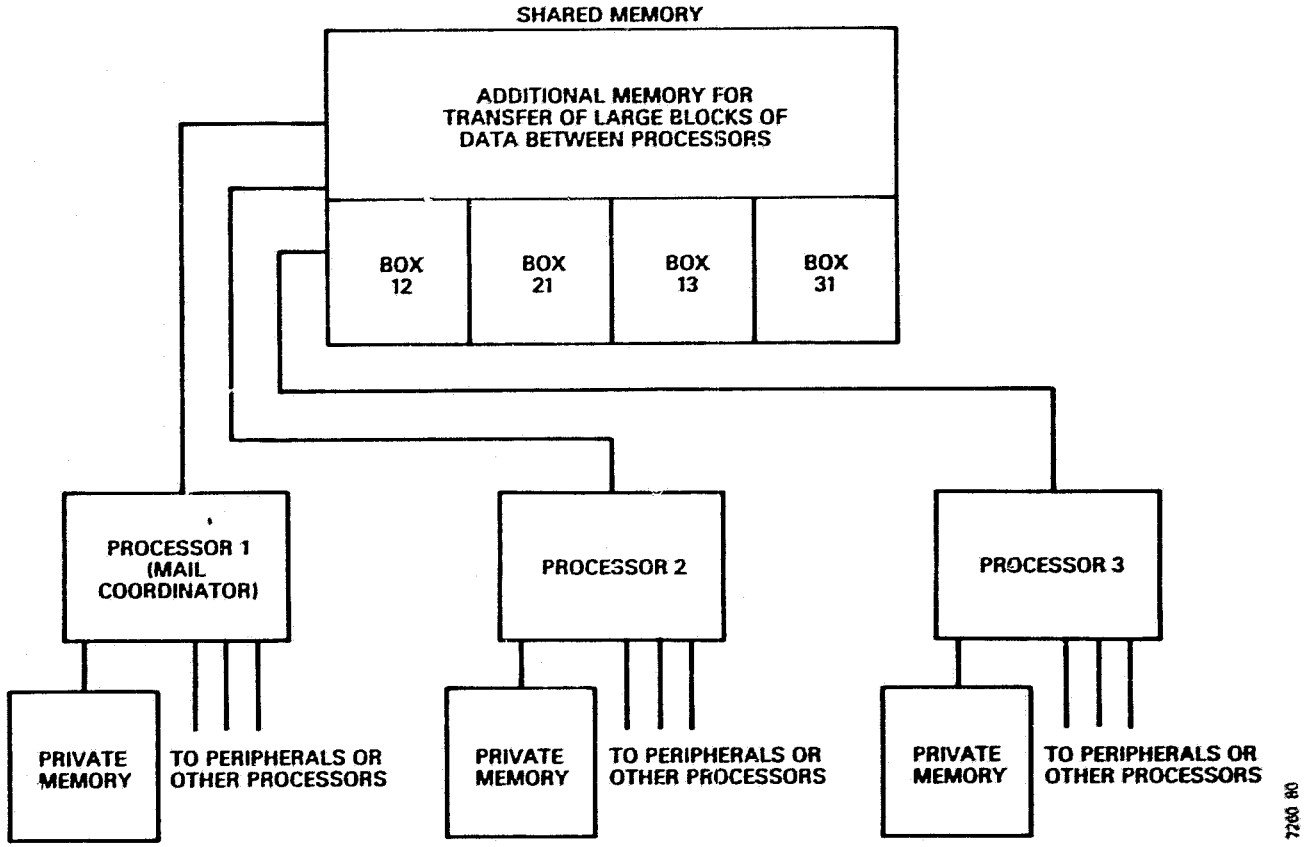


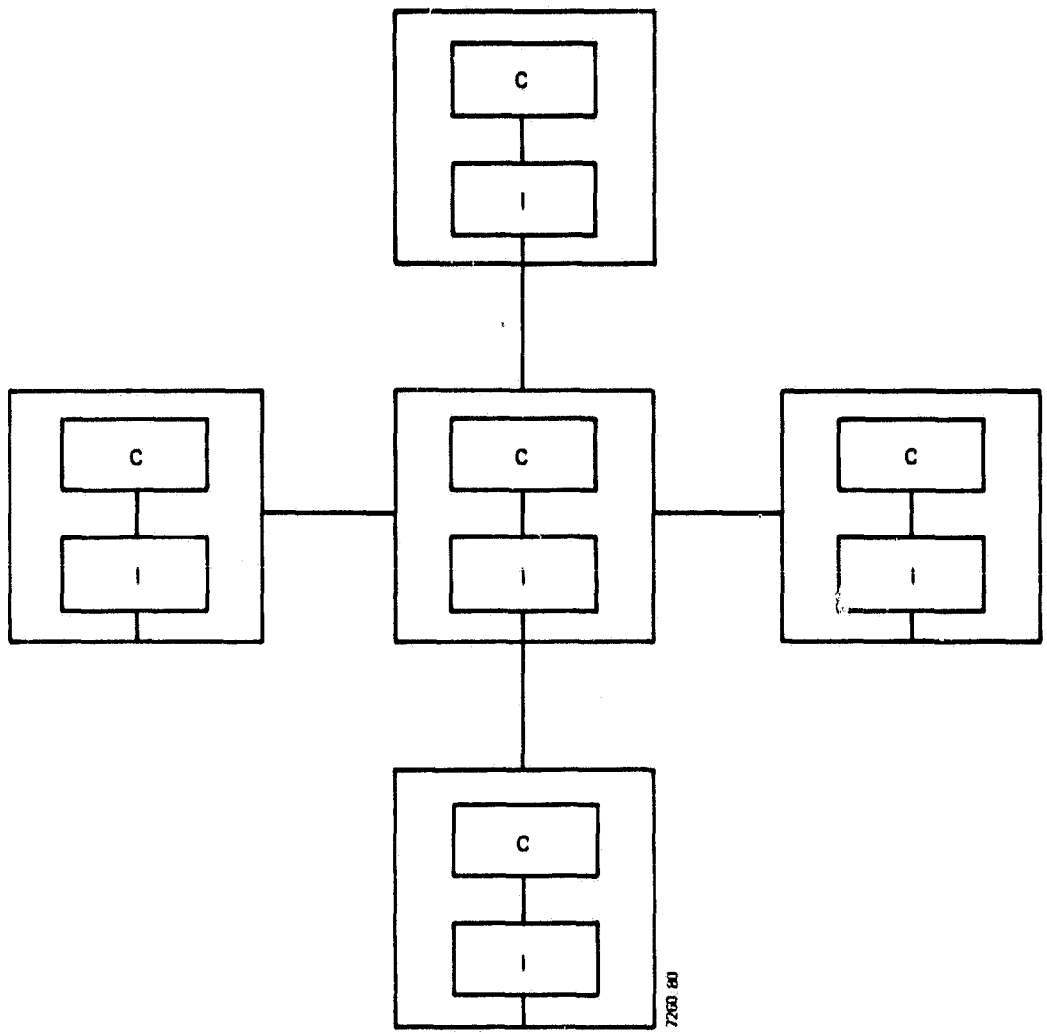
Figure 4-1. Mailbox System of Interprocessor Communication

among all processors would involve the use of N(N-1) boxes and the polling (or, alternatively, interrupt sensing) by each processor of the possible N-1 sources of messages. This compares unfavorably with the previously described mailbox scheme with a designated controller or coordinator, in which there are only 2(N-1) boxes and each processor (other than the controller) need only poll, or sense, messages from the controller (in addition to any peripherals it may control or any other system with which it may communicate). In a system in which N is small, this is not significant (see the microprocessor orbit determination design examples in Section 6).

Another microprocessor network configuration is the "hypercube," shown in Figure 4-2 (Reference 36, pages 218 and 219). Each node, or "cube" of the network consists of two microprocessors. One of the microprocessors performs the computational functions that are designated for that node. The other microprocessor serves as the message handler, or interface, to the network. This configuration is especially powerful when used for solving a large problem that can be divided into several parts that must communicate with each other in nontrivial ways (e.g., large data flow, special formatting, format checking). An example of the use of this configuration in an onboard microprocessor-based orbit determination system is provided in Section 6.2.

4.4 SOFTWARE RELIABILITY: FAULT TOLERANCE

A basic goal in developing any piece of software is to make it as reliable as possible. That is, the software should produce the correct set of results when it performs its function(s) on the input data, and it should specifically indicate when it cannot handle the input. This reliability is especially desirable for computer applications in remote



LEGEND: I = INTERFACE PROCESSOR OF THE NODE
C = COMPUTATIONAL PROCESSOR OF THE NODE

Figure 4-2. Hypercube Network Architecture

environments where human intervention in the processing is unfeasible or impossible.

Large machines and their associated operating systems have for some time featured the detection of faults, or violations, of hardware or software standards during execution. A typical hardware fault is integer addition overflow. In the past, software faults of interest have typically been in the realm of language and semantics; however, they are not of concern here. (An example of such a software fault, for which a test is optionally available in PL/I, is a subscript range violation.)

The main subjects of interest here are software faults that are violations of user-specified (high-level) functions and the methods of handling them in a fashion that is independent of human intervention and operating system actions. The divergence of the Newton-Raphson method of root seeking is an example of this kind of software fault. If a system is to have a second chance for continuing its processing uninterrupted after encountering a fault, the appropriate contingency handling methods must be designed and implemented in the system. The following discussion on a methodology for handling software faults is taken from Reference 38.

Designing software fault tolerance into a system has the following advantages:

- The system will be more reliable (i.e., protect itself against committing errors in its final results).
- The system will be closer to being autonomous.
- The extra cost involved in fully testing the system for complete reliability of a powerful, highly optimized implementation of a function may be saved by providing a backup module utilizing a more

standard, reliable, but less efficient algorithm. For example, the bisection method of root finding can serve as a backup for the Newton-Raphson method.

To keep the system within the prescribed bounds of storage, decisions must be made as to which modules (or functions) should be protected from software faults. The following principles can aid in eliminating some functions from such consideration:

- Closed-loop functioning (i.e., control with feedback) is less susceptible to failure, in general, than most other applications, since it can usually correct its own errors using the feedback.
- Housekeeping activities such as telemetry formatting are usually not critical to spacecraft survival, and thus the only protection needed is the prevention of those of their actions that adversely impact essential application programs.

Software fault tolerance can be implemented by use of the recovery block concept. This is a software structure that consists of a primary function module, an acceptance module, and a backup module. As previously mentioned, the primary module can be an optimal or near-optimal implementation of the desired function with some risk of failure (in production of correct output) involved, and it may be untested for many suspected borderline cases of input. The backup module is very reliable, although not as efficient as the primary. The acceptance module consists of a test or a series of tests to determine whether correct results were produced by the primary module and, if necessary, by the backup module. For example, the acceptance test for the previously specified root-finding routines can simply be the evaluation of the expression whose root is being sought by substituting the

alleged root produced by the primary or backup module in the expression. The resulting number can be compared with a small, predetermined tolerance.

Reference 37 also mentions other kinds of tests that can be performed:

- Real-time input data magnitudes and rates of change can be checked; and if these variables can take values over a large range, the magnitude of the differences between consecutive values (instances) can be checked for being within a prescribed range.
- Parameters that can vary in an arbitrary (rather than smooth) fashion can be transmitted between modules (or between the ground and the spacecraft) in a coded fashion (with redundancy, parity, or other errors checks).
- "Reasonableness" tests can be performed by cross-checking data and looking for consistency (e.g., a high gyro rate on a spacecraft implies a high vehicle spin rate; a check of optical sensor data can be made by the onboard system to confirm this).

Thus, the recovery block technique can be used for making software--in particular, onboard software--more impervious to software faults. A high-level, ambitious example of this, suggested in GSFC Code 582, is the implementation of a Kalman filter as the primary module for orbit estimation and of a batch least-squares algorithm as the backup. The criteria for acceptance of the results (i.e., detecting the convergence of the spacecraft's orbit state) are not well defined at this time.

SECTION 5 - MICROCOMPUTER HARDWARE SYSTEM TEST TOOLS

This section introduces some diagnostic aids, both hardware and software, that are required for developing and maintaining microprocessor hardware systems. Although it is not customary for programmers to become involved with hardware functioning and testing, the need for such involvement arises when working with microprocessor systems in general. The usual areas of involvement in hardware maintenance include

- Initial software diagnosis of hardware problems--Such diagnosis is mandatory, because a hardware specialist cannot be requested until some indication exists that the system problems being encountered are due to hardware malfunctioning and not to software errors.
- Maintenance of replaceable parts of the hardware system (e.g., chips, boards, sockets)
- Overall development of the hardware system and maintenance of components that are basically not modular (e.g., peripherals, power supply)

The extent of the hardware support available to the hardware and software system developers depends upon the following:

- Existence of an operating system to aid in the running of diagnostic programs and in the printing of results
- Existence of a package of hardware testing utility programs
- Maintenance support available from the equipment manufacturer or vendor
- Extent to which the overall system is made up of different manufacturers' equipment

For all hardware support that cannot be obtained from the available system software and the manufacturer's field engineering staff, a plan must exist for both developing the software tools and ensuring that hardware expertise is readily available. Section 5.1 describes some software tools that can aid in locating hardware problems. The programming staff must be prepared to use and possibly develop such software tools. Section 5.2 discusses basic electronic diagnostic tools for hardware maintenance.

5.1 SOFTWARE TOOLS FOR MICROCOMPUTER HARDWARE DEBUGGING

The fundamentals of diagnosing microprocessor hardware anomalies must be understood by microprocessor software developers. The primary application of this knowledge is the testing of RAM and PROM to verify that assigned data elements are being stored correctly. This is usually the first test or series of tests used to decide whether program execution anomalies are being caused by hardware or software errors. These tests are performed by using the utility programs described below.

The RAM test may consist of simply storing an input pattern in successive memory locations, reading location contents back, and comparing this data with the original pattern (held in a register). A failure to match the data read back with the original pattern halts the test routine (i.e., sends it back to the system monitor). A dump of the register's contents then informs the user as to which memory location is at fault. (A listing of this basic program for IMP-16 microprocessors is provided as item 3 of Appendix A). This test can be enhanced to proceed through all memory and display all locations that fail the pattern test, sequentially trying

different test patterns and continuing to loop with all patterns through all RAM until the user halts the process.

Memory problems caused by the coincidence of several factors (i.e., content-dependent faults) often defy straightforward diagnosis. One possible approach is a repeated permutation of test patterns performed in one segment of memory at a time. This type of test is described in Reference 38.

For PROM, the basic software diagnostic tests are concerned with (1) the correctness of the code burned into the PROM chips and (2) the correct sensing of PROM code placed into the PROM board sockets. A set of diagnostic utility programs may consist of the following:

- A pattern comparison program for checking the microprocessor receiving the PROM--A PROM chip having a preset test pattern is inserted in the microprocessor PROM socket, and its address locations are examined. Incorrect pattern readings with this test indicate problems with address lines, memory registers, PROM sockets, or the PROM board(s) in general. For example, the wrong voltage may be being applied to the board(s).
- A comparison program for checking the user-generated PROM chips against the binary code image placed in RAM--The original binary code should be kept on a nonvolatile storage medium (besides PROM) such as floppy disk to be available for use in checking the PROM chips. The dropping of a bit (i.e., change in bit state) was observed on one occasion while developing the IMP-16 shoebox system.

Other utility programs should be available for initial diagnosis of hardware problems. For example, a utility that continuously tests the interrupt line(s) or input/output

lines can be used for either confirming or refuting the existence of malfunctions in that hardware.

5.2 HARDWARE AIDS FOR HARDWARE DIAGNOSIS

Certain devices can be considered essential for system development and problem diagnosis in the field. Following the use of the software methods described in Section 5.1, these devices can be used for further attempts to solve the hardware/software error conflict that often occurs when working with a new, unreliable, or overburdened (real-time) system. The tools are the following:

- Oscilloscope
- Voltmeter
- Logic analyzer
- Hardware module tester system
- In-circuit emulator

An oscilloscope is useful for indicating the presence or absence of a level (i.e., constant signal) for a given flag or latch (external input/output register). For example, use of this device can determine whether interrupts are enabled or disabled in the microcomputer during program execution. A voltmeter can be used to ascertain whether the correct voltage is being applied in a given circuit (e.g., to a board or chip).

A logic analyzer is generally used for displaying the contents of registers, memory locations, or data lines as binary (or octal or hexadecimal) data. Some analyzers include optional features that enable the user to display the sequence of instructions executed, up to a certain limit, with the analyzer possibly halting on the condition that the microprocessor being tested reaches a specific address. A machine-level snapshot of program execution is thus obtained.

A hardware module tester system is a set of prepared (PROM) test programs that can check hardware modules individually by benchmarking them against identical test modules. An in-circuit emulator is an aid for testing and debugging programs on the machine language level (possibly with a display of the corresponding Assembly Language mnemonics) by means of real-time register displays and memory inspection and altering capabilities. In some units, intended PROM code can be loaded into a RAM unit that simulates PROM addressing. Additional information on these hardware tools is provided in Reference 5, pages 250 and 260, and Reference 19, pages 38 through 41.

Two other tools that are related to in-circuit emulators are front panel stepping and display controls and real-time debugging monitors. Although these aids are mainly used for software debugging, because they allow the user or developer to work at the machine level and can be implemented in hardware, their inclusion here is relevant. This type of program fault diagnosis is usually reserved for use when higher level (problem-oriented) debugging techniques fail.

A microcomputer front panel may be very much like that of most minicomputers, where functions such as single-stepping through a program (binary code) and displaying and altering both registers and memory locations may be performed. The front panel functions are implemented in hardware (including firmware) and are initiated by physically actuating dials, levers, and other devices. The real-time debugging monitor may offer the aforementioned front panel functions (and possibly additional ones) by software implementation. The user/tester can interactively exercise control over these activities. Additional details about such monitors are contained in Reference 7, pages 298 through 325. All of the

lower level software debugging techniques described above are extensions of the classical techniques of core dumps and execution tracing (Reference 7, pages 294 through 297).

SECTION 6 - MICROPROCESSOR-BASED ORBIT DETERMINATION CONFIGURATION EXAMPLES

This section provides two configuration design examples that illustrate the principles and facts presented in previous sections. Section 6.1 presents a ground microprocessor-based orbit determination system that accepts raw tracking data from a tracking data network and, via a sequential estimation technique, solves for a designated user (target) satellite state. Section 6.2 presents a suggested hardware configuration and top-level software design for an automated onboard orbit determination processor that supplies other spacecraft processors with current spacecraft position and velocity information on demand for data annotation. For both systems, only the most basic requirements are presented; the discussions here should not be considered replacements for complete requirements analyses, functional specifications, and design specifications.

The reader is urged to refer to Section 4.3 when necessary in order to review the ideas presented there concerning configurations of cooperating microprocessors. While studying the design examples below, the main principle of Section 3 should be borne in mind: the hardware used to implement the proposed configurations must have associated with it software development facilities that are comparable to the advanced, high-level facilities available on large systems; otherwise, the development of the reliable, computationally complex orbit determination software desired will be too costly, if not impossible, to attain.

6.1 MICROPROCESSOR-BASED ORBIT DETERMINATION FROM RAW DATA: A GROUND SYSTEM

The first example is based upon the requirements for a system built around the IMP-16 shoebox orbit determination system. (A prototype of this system was built for testing in the NASA/GSFC Code 580 environment. This prototype system, although very similar to the example presented here, has a different system architecture. Additional details are provided in References 39 and 13.) System specifications are as follows:

- The system will use sequential estimation for solving for the orbital state of a particular, low-altitude (drag-perturbed) satellite.
- The system will be placed online with a particular tracking data network and will be able to preprocess two-way TDRS System (TDRSS) raw data for tracking the satellite of interest.
- The system will have a leased-line connection with a mainframe, which will supply it with relay satellite ephemeris data at a relatively low data rate.
- A printer will be available for system reports.
- The user will be able to control the system's operations from a provided terminal without degrading the major processing of data therein. Only a temporary suspension of printer activities during dialogues between the user and the system via the terminal will be tolerated.

From these minimal specifications, the microprocessor network configuration depicted in Figure 6-1 can be offered as the

ORIGINAL PAGE IS
OF POOR QUALITY

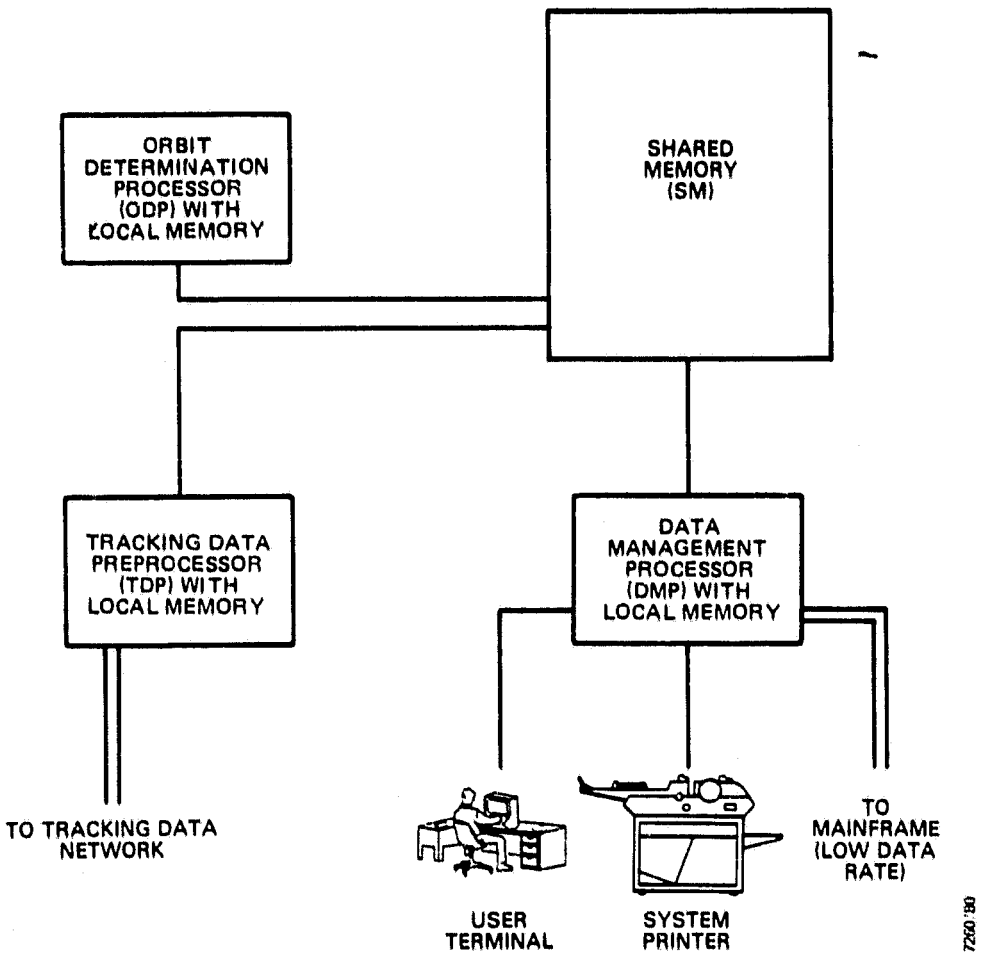


Figure 6-1. Design for Ground Microprocessor-Based Orbit Determination System

initial system design. The following characteristics should be noted:

- The three processors communicate via a mailbox system utilizing shared memory (SM) (see Section 4.3). This method of interprocessor communication need not be augmented by interrupts, since such communication occurs on a relatively infrequent basis here.
- The actual hardware connection between each processor and SM may be with a single system bus (a harness of shared communication lines with arbitration hardware for resolving conflicts). The interfaces for the tracking data network and the mainframe are separate.
- The tracking data preprocessor (TDP) alone is obligated to meet the demands of the tracking data network.
- The data management processor (DMP) is responsible for the following:
  - Handling user input from the terminal
  - Producing reports at the printer
  - Handling all alarm conditions and requests from other processors (i.e., serving as the traffic controller for the mailbox system)
  - Accepting relay ephemeris data from the mainframe
  - Performing interpolation for the relay ephemeris in anticipation of its need by the orbit determination processor (ODP)

The facts that the leased-line rate is low, that terminal input/output is allowed to preempt printer output, and that

all interprocessor communication is by the mailbox system without interrupts involving the processors imply that the DMP should not be subject to race conditions (congestion of transactions) in its input/output. The executive of the DMP will obtain at least one observation (range and range-rate pair with its associated time tag) in anticipation of the request from the ODP for that. This will allow the DMP to perform interpolation on the relay satellite orbit while the ODP is processing the previous observation data. Decisions must be made to ensure that the sending of output to the printer does not tie up the DMP; for example, once a line has been placed in a buffer for printing, it can be sent to the printer by direct-memory access, or cycle-steal access, rather than by single-word output instructions under processor control. This will allow the DMP to attend to its other tasks. Of course, factors such as available buffer area (Section 4.2) and printer capacity (Section 2.2.1) must be considered in designing the output reports. An example of the use of cycle-steal and single-word input/output (in a spacecraft minicomputer) is provided in Reference 40, pages 1-13 and 1-14.

The ODP is responsible for producing the corrected orbit using a sequential filter to process the acquired TDRSS data. A typical cycle of interprocessor action within the system is the following:

1. The ODP places a request in the mailbox for the DMP to provide a new, time-tagged range and range-rate pair with corresponding relay ephemeris information.

2. The DMP has presumably already received the time-tag for the next observation pair from the TDP (via the "mail") and has performed the interpolation for the relay position and velocity at that time. The

DMP assembles all this data, together with the range and range-rate measurements, in a buffer in SM and places a message in the box for the ODP, telling it where it can pick up the observation data.

3. The ODP periodically checks its mailbox (between calls to major modules). It finds the response from the DMP and reads in the observation data. It finishes this transaction by placing a message in its (outgoing) box for the DMP, acknowledging receipt of the data.

4. The DMP picks up the message while scanning its boxes. It can now work on setting up the next set of observation data in anticipation of the next request from the ODP.

The system described above should be realizable with microprocessor hardware. The demonstration of the IMP-16 shoebox system (without the preprocessor) showed the basic feasibility of such a system. The features of DMA hardware for a separate printer and the mailbox system rather than interrupt hardware for interprocessor communication should help eliminate communication problems that arise in system development and operation.

6.2 MICROPROCESSOR-BASED ORBIT DETERMINATION AND EPHEMERIS INTERPOLATION ON BOARD

This section uses the hypercube architecture described in Section 4.3 to meet a set of hypothetical requirements for a spacecraft mission. The following objectives must be met:

- Orbit determination will be done on board with a sequential estimator.
- The system will process one-way and two-way raw TDRSS data.

- The system will accept upload data passed through the main spacecraft, including
  - User (self) spacecraft epoch element set
  - Two-way raw TDRSS observation
  - Observation related data (e.g., relay satellite ephemerides)
  - Commands controlling the orbit determination work
- The system will provide other processors in the spacecraft with the satellite's position and velocity by interpolation from the most recent orbit determination solution and ephemeris data generated on board.
- Interpolation requests and responses will be carried over one channel;[1] all other communication between the orbit system and the outside will take place over a second channel, which will be linked to the main spacecraft computer.
- In addition to upload data, the data and commands from the main spacecraft computer channel will include the following:
  - Command initialization
  - Spacecraft clock time
  - Request for status information
  - One-way raw TDRSS data received by the spacecraft

[1]The term "channel" is used here to indicate a basic set of communication lines between two processors, possibly including handshaking (control) lines.

The basic performance requirements such as orbit determination accuracy, orbit propagation and interpolation accuracy, expected interpolation request load and maximum allowable response times, and restart capabilities (e.g., after a maneuver) are not specified here.

The basic hardware configuration is shown in Figure 6-2. One cube, or node, is dedicated to ephemeris interpolation; the second is concerned with sequentially estimating the orbit; and the third is the tracking data preprocessor. The ODP--specifically, processor B2--can be replaced by two processors that will work essentially in parallel on the sequential orbit estimation. This setup is shown in Figure 6-3, and the approximate timeline schedule of the processors' computations appears in Figure 6-4.

Areas for further study relative to this configuration design include the following:

- Basic timing relationships between different processors in the system must be well modeled and well tested.
- For propagating across data gaps, processor B2a may be used for orbit propagation, while processor B2b is used for covariance propagation.
- The interpolator, node A, nominally receives ephemeris data from node B but does not output any data to node B other than relay coordinates for observation modeling. Use of the interpolator to supply node B with interpolated user spacecraft coordinates for state transition matrix evaluations may be desirable, especially when propagating across data gaps in the manner previously described. This scheme has the advantage of keeping the covariance

ORIGINAL PAGE IS
OF POOR QUALITY

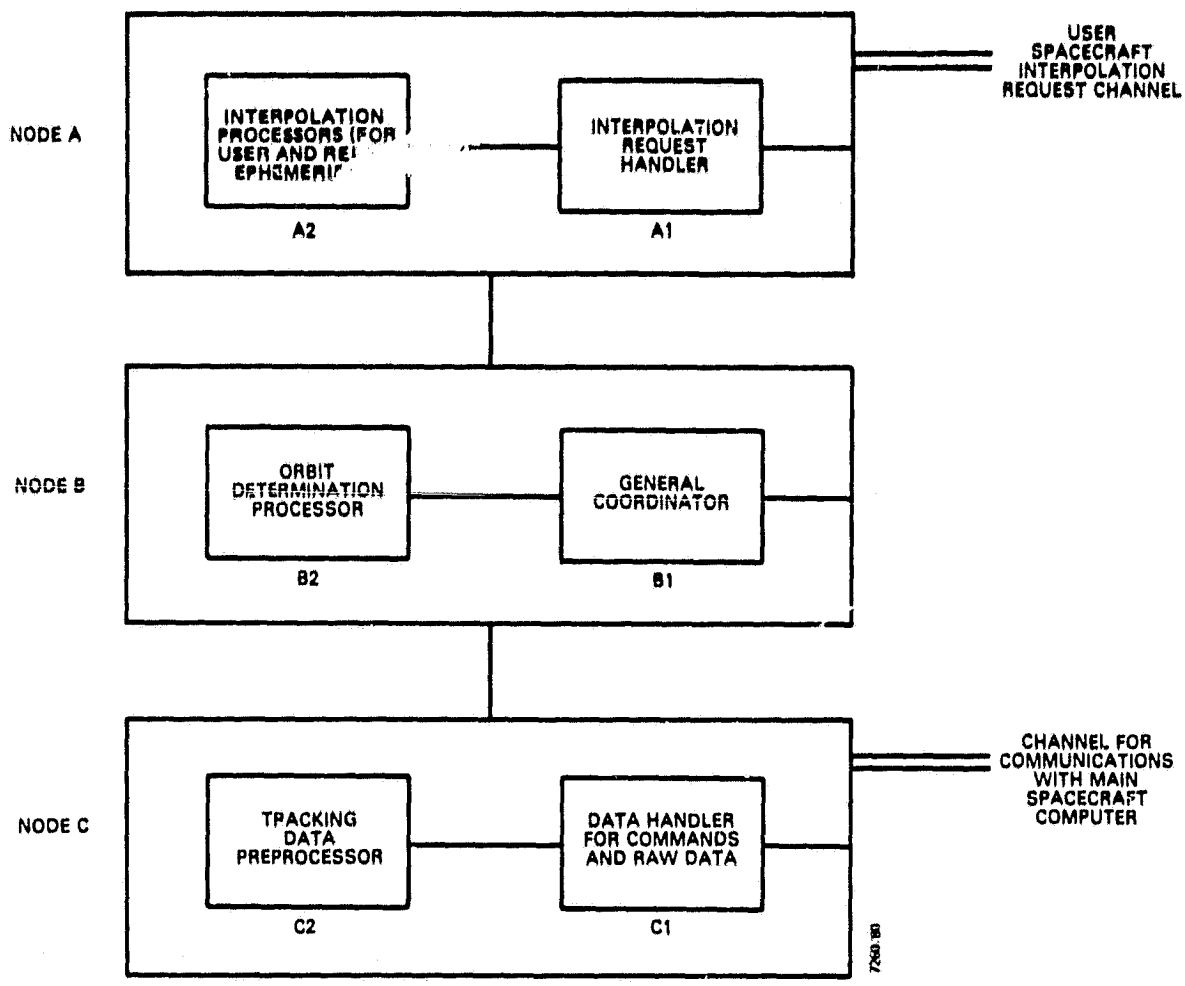


Figure 6-2. Basic Design of Spacecraft Orbit Determination and Ephemeris Interpolation Network

ORIGINAL PAGE IS
OF POOR QUALITY

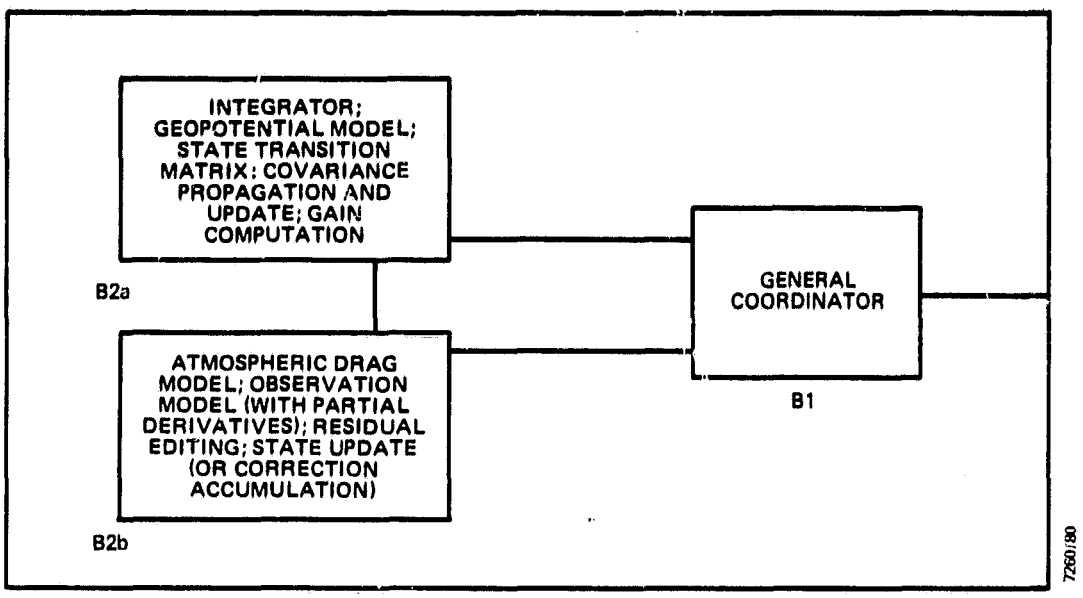


Figure 6-3. Expansion of Orbit Determination Processor Into Two Parallel Processors

ORIGINAL PAGE IS
OF POOR QUALITY

Figure 6-4. Approximate Timeline for Parallel Processing of Sequential Orbit Estimation

propagation independent of state integration. The very nature of the state transition matrix, the standard algorithms for its computation, and its applications (for range-rate partial derivatives and covariance propagation) would probably make tolerable the loss of accuracy through the use of interpolation as compared to high-order integration and full force modeling. The increase in interprocessor communications and the possible loss of accuracy must be weighed.

The hypercube configuration design has the following advantages:

1. The major functions can be computed in parallel.
2. Enhancements are relatively easy to implement (e.g., more TDRSS observation types may be added to the observation model and preprocessor).
3. Communication among this system's processors and with the rest of the spacecraft is effectively isolated from the computational processes of the system.
4. The microprocessor hardware can be specialized to meet functional requirements. Processors A2, B2 (B2a and B2b), and possibly C2 should have floating-point hardware and should be more powerful (i.e., faster) models; the remaining processors can be slower and do not require floating-point capabilities.

The potential problem areas for the development and operation of this system are as follows:

1. The space and power requirements must not exceed spacecraft limitations.

2. Designing the overall system communications will require careful analysis of possible execution flows (e.g., the "communications processors" of each node will probably assume important executive functions).

3. The overall system testing must be done in a rather complex simulated environment; the design, implementation, and use of this simulator may require more effort than the development of the software for the individual processors.

SECTION 7 - REVIEW OF MICROPROCESSOR-BASED ORBIT APPLICATIONS DEVELOPMENT

The initial efforts by CSC in developing orbit propagation and orbit determination systems based upon microprocessor hardware are described primarily in Section 2.2.3, with references to these activities in several other sections and in Appendixes A and B. (The appendixes contain reprints of memoranda describing system bugs found in the IMP-16 system.) This section focuses on the major system flaws encountered, especially those that illustrate the points made in Sections 3 and 5 concerning microprocessor software and hardware, respectively. Section 7.1 summarizes the IMP-16 development experience; Section 7.2 discusses the INTEL 8080 Orbit Propagator development experience.

7.1 IMP-16 AND SM/PL EXPERIENCE

Several major deficiencies in the IMP-16 development system severely impacted software development. These were

- Unreliability of the compiler
- Lack of a file system and an operating system in general
- Dependence on a program-controlled, serial (bit) communication between processors
- Inability to link external modules and to directly download code from the development machine to the target machine
- Lack of facilities for transforming RAM code into PROM code on an efficient, reliable basis and for checking the PROM code efficiently

The unreliability of the compiler was one of the most insidious problems. It eroded the programmers' confidence in the end product of their source code, and it forced them to change their approach toward the application at hand from a problem-oriented (higher level) approach to a machine-level approach. For programs of the length and complexity of those required for solving orbit determination problems, this represents a considerable impediment to progress.

The lack of a file system and an operating system in general imposed upon the task programmers the considerable burden of the bookkeeping for keeping track of the code under development. In addition, there was no system protection against the destruction of developed code on the floppy disks by programmers' errors or system program anomalies. Therefore, the programmers had to spend time rearranging and restoring code on the disk in order to compensate for, or to prevent, the overwriting of existing code.

A final frustrating implication of the lack of an operating system was felt during the unit testing and system testing activities. The development of orbit determination software requires much intermediate output, as discussed in item 5 of Section 2.2.1, to locate and fix bugs affecting the numerous computations present. Without an operating system, the input/output facilities are quite primitive. The task programmers had to obtain intermediate computational results by either (1) dumping the areas of memory containing the quantities of interest one number at a time or (2) modifying several sections of code, recompiling, and reloading. The latter option involved working at a level that was much lower than that achieved with standard input/output code written with formatting statements; the former method was extremely tedious and time consuming. The time required for almost all testing was thus significantly lengthened.

The dependence upon a program-controlled, serial communication between processors accounted for much time spent in trying to (dynamically) separate the computations performed by the dual IMP-16 shoebox processors from the communications taking place between them. The use of parallel transmission between processors implemented with minimal program control would have helped make the separation between computations and communications distinct.

The final two deficiencies listed represent related problem areas. The lack of relocatable code and automated external reference-resolving facilities implied that a transfer of code (consisting of many modules) from RAM to PROM had to be accompanied by corresponding changes in the external references in several places in the source code. It was not until late in the task that a method (base page jump table) suitable for organizing this process was conceived and implemented.

The greatest obstacle presented CSC task personnel in performing the burning of PROM chips was the difference in RAM and PROM locations in the development machine and the target machine in which the PROM resides. A new utility based on the current but inadequate system loader had to be developed for facilitating this process (see Reference 41). The lack of sufficient RAM in both sides (for both IMP-16 microprocessors) of the original shoebox and in the development system implied that the entire system could not be tested in RAM before being transferred to PROM. The unit tests and tests combining RAM and PROM code exacerbated the manual external reference resolving problems, since interrelated module addresses changed very often. Finally, the PROM code in each side of the shoebox could not be verified directly for correctness against the binary image on floppy disk. Instead, a complicated process of transferring the code from the disk

to the development system and then to the shoebox in separate segments (due to the restricted RAM space) for comparison testing with the PROM code was necessary. The PROM chips (26 in all, for applications) could be tested only on an individual basis in the PROM reading/burning system. Future system builders who rely on PROM code for their final system should be wary of any development system in which the transformation from RAM to PROM is not straightforward.

Certain features for PROM development would remedy many of the problems experienced during the IMP-16 PROM development phase. These are as follows:

- The PROM space addressing in the development system should be identical to that in the target system. This would be extremely beneficial for systems that have a limited capability for generating relocatable code. Advantages of the equivalent PROM address spaces include the following:
  - The transition from a working RAM version to the final PROM version via mixed PROM/PAM versions could be handled more smoothly.
  - The final PROM development version would be the final target version; no address translation would be required.
- The RAM space should be sufficiently large that a major portion of, if not the entire, intended system could be tested at once in RAM. This would also allow for testing the completed PROM build against its entire binary image in RAM.
- The compiler and (linking) loader should be compatible and should allow for producing a PROM version load module in a straightforward manner.

- The capability to read and burn several PROM chips simultaneously should be available. This would greatly accelerate the processes of producing, verifying, and modifying PROM code.
- A test package with PROM chips containing test patterns should be available for performing quick tests of the accuracy of the programmer's PROM chips and of the PROM system itself (see Section 5.1).

7.2 INTEL 8080 ORBIT PROPAGATOR DEVELOPMENT EXPERIENCE

The INTEL 8080 Orbit Propagator was developed by CSC on a Tektronix 8002 development system. The final tested RAM code was downloaded to an INTEL 8080 in-circuit emulator, which simulated PROM for the INTEL microprocessor. After this test, code was burned into PROM and successfully tested. These final tests were carried out by GSFC Microprocessor Development Facility personnel.

The Tektronix development had the following positive features:

1. Disk file system
2. Operating system with reasonably flexible input/output facilities
3. FORTRAN language with REAL*8 floating-point capabilities (precision of approximated 16.7 digits)

The following drawbacks impeded development:

1. The FORTRAN language lacked some standard language features such as the COMMON, DATA, and EQUIVALENCE statements. Thus, large arrays of constant data, such as the spherical harmonic coefficients, had to be initialized and handled in an awkward, tedious, and inefficient manner.

2. No facilities existed for linking separately compiled subroutines. Thus, all the orbit propagator code (15 subroutines) had to be recompiled and relinked each time the code was modified.

3. The compiler and loader operations were slow--recompiling and reloading the entire propagator without printing the listing took approximately 15 minutes.

4. No mathematical subroutine library was available on the system. Thus, time had to be spend in developing the required routines (see Section 3.1.5, especially item 3).

5. The available floating-point operations were implemented entirely by system software. The execution of code was thus exceeding slow. The program performed only two integration steps in approximately 40 seconds; using an Adams-Bashforth, Adams-Moulton sixth-order predictor-corrector with a Runge-Kutta fourth-order starter, lunar and solar gravitational perturbations and a 4-by-4 geopotential field were modeled. The propagation rate was approximately doubled when the third-body effects were removed. Since the propagator was built for modeling orbits of the geosynchronous type, step sizes of 430 seconds were nominally taken, which implies that propagating one orbit of such a satellite with all effects modeled required approximately 1 hour of computation.

SECTION 8 - CONCLUSION

The previous sections of this document have derived guidelines for aiding in the decisionmaking processes involved in the selection of hardware for building microprocessor-based flight dynamics systems. The fields of hardware and software development in general and specifically as related to microprocessor systems have been presented. The specific requirements for orbit determination systems and similar computationally complex software have also been reviewed. Two hypothetical orbit determination problems of interest have been stated, and possible solution designs have been offered. Finally, the experiences encountered in developing orbit systems for GSFC have been summarized, with the focus being on major system deficiencies and their impact on the development process.

Thus, the main themes underlying the presentations in this document are as follows:

- The use of microprocessors for the solution of flight dynamics problems appears to be feasible. The completed systems described herein demonstrate this.
- The development systems used for developing orbit determination software (and flight dynamics software in general) must offer system support software that is appropriate for the problem; otherwise, the development of such systems will be prone to long delays, and the final systems may be much less reliable than desired (due primarily to the difficulty of modification and maintenance) (Sections 3, 4.1, 4.2, and 7).
- The hardware components of the system should be testable by supplied hardware diagnostic programs

so as to ease the burden of hardware maintenance that is unduly placed on the microcomputer programmer/ analyst (Section 5.1).

- The transfer of developed RAM code to PROM and the methods of verifying the correctness of the PROM code should be as straightforward and efficient as possible, as described at the end of Section 7.1.

The DEC LSI-11 microcomputer is a prime example of a system that comes close to meeting the criteria specified above. The software development can be done on any member of the completely compatible, well-tested, and well-maintained family of PDP-11 minicomputers with all the required development tools present. Floating-point microcode and maintenance personnel from the manufacturer are available.

However, several precautions must be taken. The requirements for space and power must still be weighed, especially for spacecraft systems, and especially in view of the fact that the LSI-11 processor is implemented on three chips rather than on a single chip. Another point to be considered is the fact that there is currently little experience or history for use in evaluating the processes of downloading developed code from the supporting minicomputer to the LSI-11 and of burning the final code into PROM. Thus, these activities must be carefully planned and monitored.

The main conclusion that should be drawn from this document is that state-of-the-art microprocessor technology can be and must be exploited for flight dynamics applications. It is hoped that the experiences and guidelines presented in this document will serve effectively in the selection process of microprocessor hardware and the accompanying software development systems for the applications of interest.

APPENDIX A - CSC MEMORANDUM ON IMP-16 AND SM/PL SYSTEM BUGS

ORIGINAL PAGE IS
OF POOR QUALITY

CSC COMPUTER SCIENCES CORPORATION

INTEROFFICE CORRESPONDENCE

to Task 885 File; Distribution from C. Shenitz CM date July 3, 1978

subject SYSTEM HARDWARE AND SOFTWARE BUGS

In this memorandum, we present a list of bugs encountered in the IMP-16 development systems prior to and during the implementation phase of the orbit determination system. The aims of this presentation are the following:

- To inform the GSFC ATR of the time-consuming system debugging that has been required thus far.
- To warn the future users of these system bugs and to supply work-arounds wherever possible.

In the sections that follow, the hardware bugs are presented first. The bugs encountered in the manufacturer's system software, as well as in the independently vended SM/PL compiler, are detailed next.

1. Faulty Disk Drives

The 4th floor disk drives are incompatible with those on the second floor. A frequent symptom of this was the disk errors encountered in the loader scratch area when switching from one set of drives to the other. Furthermore, it appears that just reading from a disk with the 4th floor apparatus adversely affects the disk later when reading from it with the second floor drives. Thus, it is strongly recommended that use of the 4th floor system be avoided entirely until the disk drives are refurbished.

2. New Floppy Disks

A new floppy disk should be informally initialized by copying sector 0 from an operational disk into sector 0 of the new disk. Then, the sectors of the new disk should be written on consecutively from 1 to 267. Failure to due so may result in disk errors.

3. Memory Faults

The IMP memory has been known to go bad on occasions. When other debugging efforts fail, the memory can be checked by a simple program to store, load, and

ORIGINAL PAGE IS
OF POOR QUALITY

compare a test quantity in consecutive memory locations. The simple program is as follows:

Object code	Source code		
3481	RCPY	3,0	
A300	ST	0, (3)	; Store pattern
D300	SUB	0, (3)	; Compare with original
4B01	AISZ	3, 1	; Prepare for next location
11FB	BOC	1, .-4	; Continue if location is good
2409	JMP	@09	; Jump to monitor, if location
			; is bad

Register 1 contains the pattern used for testing.
Register 3 points to the first location to be used for testing.

4. Missing Chips

If the IMP development system is used by a variety of users, it is possible that the wrong CROM may be in the machine, in place of the arithmetic (double-precision) CROM needed for floating-point applications. This possibility should be checked if previously tested floating-point routines fail to execute.

5. Base Page Relocation

It was recently discovered that the loader automatically relocates base page storage by 10 hex (added to object module's base-page origin) without informing the user. Thus, the user who will be linking an SM/PL program object module must specify during compilation that the base-page locations start at a low enough address such that location E0 or higher is not reached. This will protect the SM/PL pointers in F0 through FF which are used by SM/PL-compiled modules.

6. Missing External References

The loader does not flag missing (unresolved) external references. It outputs a load module, as if everything was correctly specified. Thus, the programmer must check that all external references are resolved. This applies in particular to external procedures declared in SM/PL programs by the pseudo-function ASMPROC.

7. Order of Pointers In Calls

The SM/PL compiler will not generate the correct object code for the statement

```
CALL ADDF (.A (0), .B (0), .A (0));
```

whereas it will correctly compile

```
CALL ADDF (.B (0), .A (0), .A (0));
```

The user must avoid non-consecutive occurrences of pointers (dot operator) to the same variable in a subroutine call.

8. Too Many Blanks

Although the SM/PL compiler is supposed to accept source code in free format (with respect to the placement of blanks), the following occurred:

```
IF A<B THEN

CALL SUB (.Z (0));
```

was flagged as bad syntax after 'CALL', while

```
IF A<B THEN CALL SUB (.Z (0));
```

was accepted.

9. Integer Multiplication and Division

If an integer multiplication (division) is used in an SM/PL program whose corresponding load module is called by another (independently compiled) SM/PL program the main (first entered) SM/PL program must also contain an integer multiplication (division). If this rule is not adhered to, a base-page pointer (location F7) will not be initialized. This pointer is required for the above operations.

10. Erroneous Maps

The map generated by an SM/PL compilation will occasionally contain (obvious) erroneous information. The specific storage length may be incorrect (but the allocation is correct as seen by examining the next variable's location). Often misspellings occur. However, it does not appear that the corresponding object and load modules are defective.

APPENDIX B - GSFC MEMORANDUM ON A PARTICULAR SM/PL BUG

ORIGINAL PAGE IS
OF POOR QUALITY

R. K. Schwenk
8/15/78

SM/PL Errors Caused by Faulty Addressing:

The usual way of accessing data is to use indirection via the nearby pointer table. For instance, to load the contents of I into RØ, the compiler generates:

```
LD RØ, = I
```

Often, however, the compiler chooses to exploit the current contents of R2 to access the desired data. Suppose that I and J have consecutive addresses. Then another way of loading RØ with I is to perform:

```
LD RØ, OFF (R2)
```

provided that the address of J is in R2. Unfortunately, the compiler does not sufficiently account for transfers of control when it determines the current contents of R2.

Examples:

(1) Consider the nested do-loops:

```
DO I=1 TO N;
   DO J=1 TO N;
        .                    where I and J have, say,
        :                    consecutive addresses

   END;
END;
```

The object code generated will be:

```
        LI RØ, X'0001
(#DØØ2) ST RØ, I
        LD R2, =N
        SKG RØ, ØØØ (R2)
        JMP     .+X'ØØØ2
        JMP     #NOØ2
        LI  RØ, X'ØØØ1
(#DØØ3) ST RØ, J
        LD R2, =N
        SKG RØ, ØØØ (R2)
        JMP     .+X'ØØØ2
  (**)  JMP     #NDØ3

              .
              :
              .
```

ORIGINAL PAGE IS
OF POOR QUALITY

```
        LI RØ, X'ØØØ1
        LD R2, =J
        ADD RØ,ØØØ (R2)
        JMP    #DOØ3
(#NDO3) LI RØ, X'ØØØ1
  (*)ADD RØ,OFF (R2)
        JMP  #DOØ2
(#NDØ2) . . .
```

The second ADD statement (*) would be correct if the address of J were in R2. However, this statement is only accessed from the JMP #NDØ3(**). Thus, the contents of R2 will be the address of N, and an error occurs. This error can be avoided by placing an assignment statement between the END statements.

(2) Consider a do-loop of the form:

```
DO I=K+1 TO N;
      :
END;
```

where I and K have consecutive addresses, say.

The compiler generates:

```
        LI RØ, XØØØ1
        LD R2, =K
        ADD RØ, ØØØ (R2)
(#DOØ2  ST RØ, OFF (R2)
        LD R2, =N
        SKG RØ,Ø,R2
        JMP  .+XØØØ2
        JMP  #NDØ2
             :
        LI RØ, X'ØØØ1
  (*)   LD R2, =I
        ADD RØ, ØØØ (R2)
        JMP  #DOØ2
 (#NDØ2)  . . .
```

The store statement (#DOØ2) will not work after the first time through the loop. The compiler assumes that R2 contains the address of K while the incrementation of I has caused the address of I to be in R2 (see (*)). This error will not occur with the simpler DO statement:

```
KP1 = K+1;
DO I = KP1 TO N;
```

GLOSSARY

CPU	central processing unit
CSC	Computer Sciences Corporation
DMP	data management processor
GPS	Global Positioning System
GSFC	Goddard Space Flight Center
IC	integrated circuit
LSI	large-scale integration
MSI	medium-scale integration
NASA	National Aeronautics and Space Administration
NSSC	NASA Standard Spacecraft
ODP	orbit determination processor
ODS	Orbit Determination System
PLA	programmable logic array
PROM	programmable read-only memory
RAM	random-access memory
SM	shared memory
SMM	Solar Maximum Mission
SSI	small-scale integration
SST	satellite-to-satellite tracking
TDP	tracking data preprocessor
TDRS	Tracking Data and Relay Satellite
TDRSS	TDRS System

REFERENCES

1. Auerbach Publishers, Auerbach on Minicomputers. New York: Petrocelli Books, 1974

2. Digital Equipment Corporation, Microcomputer Processor Handbook, 1979

3. I. Rampil, "Preview of the Z-8000," BYTE, March 1979, vol. 4, no. 3, pp. 80-91

4. S. Berman, "Microprocessors vs. Programmable Logical Arrays," in Microprocessor Basics (M. Elphick, editor). Rochelle Park, New Jersey: Hayden Book Company, Inc., 1977

5. C. Sippl, Microcomputer Handbook. New York: Petrocelli/ Charter, 1977

6. "BYTE News" (column), BYTE, March 1980, vol. 5, no. 3, pp. 108-112

7. E. Yourdan, Techniques of Program Structure and Design. Englewood Cliffs, New Jersey: Prentice-Hall, Inc., 1975

8. B. Boehm, "Software and Its Impact: A Quantitative Assessment," Datamation, May 1973, vol. 19, no. 5, pp. 48-59

9. K. Thurber and H. Buijs, Large Scale Computer Architecture. Rochelle Park, New Jersey: Hayden Book Company, Inc., 1976

10. J. Berube, "Description of a High-Speed Vector Processor," in Minicomputers and Large Scale Computations (P. Lykos, editor). Washington, D.C.: American Chemical Society, 1977

11. T. Harr and R. Phillips, "Interrupts Call the Shots in Scheme Using Two Microprocessors," Electronics, February 14, 1980, vol. 52, no. 4, pp. 166-170

12. K. K. Tasaki, National Aeronautics and Space Administration, Goddard Space Flight Center, private communication

13. Computer Sciences Corporation, CSC/SD-78/6131, System Description of the IMP-16C Microprocessor Orbit Determination Program, C. Shenitz, C. Rabbin, G. Snyder, and C. Goorevich, October 1978

14. K. Tasaki and C. Goorevich, "Shoebox Orbit Determination System for SMM - Preliminary Results" (paper presented at Flight Mechanics/Estimation Theory Symposium, October 1978), NASA Conference Publication 2082, April 1979

15. Computer Sciences Corporation, CSC/TM-79/6080, INTEL 8080 Orbit Propagation Program System Description and User's Guide, C. Rabbin, February 1979

16. F. Brooks, The Mythical Man-Month. Reading, Massachusetts: Addison-Wesley Publishing Company, 1975

17. E. Dijkstra, "Go-To Statement Considered Harmful" (letter to the editor), Communications of the ACM, March 1978

18. Computer Sciences Corporation, CSC/SD-78/6128, Structured FORTRAN Preprocessor (SFORT) 11/70 User's Guide, D. Wilson, September 1978

19. C. Magers, "Managing Software Development in Microprocessor Projects," IEEE Computer, June 1978, vol. 11, no. 6, pp. 34-42

20. G. Myers, Reliable Software Through Composite Design. New York: Petrocelli/Charter, 1975

21. Computer Sciences Corporation, CSC/SD-77/6078, System Description of the Orbit Representation Program, C. Shenitz, July 1977

22. --, CSC/TM-79/6208, Evaluation of the IMP-16 Microprocessor Orbit Determination System Filter, C. Shenitz, September 1979

23. M. Payne, "Sources of Error in Floating Point Arithmetic" (paper presented at Flight Mechanics/Estimation Theory Symposium, October 1976), NASA Conference Publication No. 2023, September 1977

24. A. Ralston and H. Wilf, Mathematical Methods for Digital Computers, Volume 1. New York: John Wiley and Sons, Inc., 1960

25. J. Hart et al., Computer Approximations. New York: John Wiley and Sons, Inc., 1968

26. Control Data Corporation, Control Data 6000 Computer System Programming, System/Library Functions, A Study of Mathematical Approximations

27. F. Hetzel, Program Test Methods. Englewood Cliffs, New Jersey: Prentice-Hall, 1973

28. A. Cohen et al., Numerical Analysis. New York: John Wiley and Sons, Inc., 1973

29. C. Gerald, Applied Numerical Analysis. Reading, Massachusetts: Addison-Wesley Publishing Company, 1970

30. Jet Propulsion Laboratory, Technical Memorandum 33-771 (NASA-CRT-148278), A Numerical Comparison of Discrete Kalman Filtering Algorithms: An Orbit Determination Case Study, C. Thornton and G. Bierman, June 1976

31. A. Demott, Preliminary Study of On-Board Attitude Control for Multimission Modular Spacecraft (MMS) (prepared by Computer Sciences Corporation for Goddard Space Flight Center under contract NAS 5-11999, Task 558; not issued as a formal document), February 1976

32. Computer Sciences Corporation, CSC/TM-79/6272, A Feasibility Study for a General Microprocessor-Based Orbit Determination System, A. Shell and C. Shenitz, October 1979

33. W. Riley, "Minicomputer Networks - A Challenge to Maxicomputers?", in A Practical Guide to Minicomputer Applications (F. Coury, editor). New York: IEEE Press, 1972

34. D. Chang, "Multiprocessing With Microprocessors," in Microprocessor Basics (M. Elphick, editor). Rochelle Park, New Jersey: Hayden Book Co., Inc., 1977

35. Comtre Corporation, Multiprocessors and Parallel Processing (P. Enslow, editor). New York: John Wiley and Sons, Inc.) 1974

36. G. Rao, Microprocessors and Microcomputer Systems. New York: Van Nostrand Reinhold Company, 1978

37. H. Hecht, "Fault-Tolerant Software for Real-Time Applications," ACM Computing Surveys, December 1976, vol. 8, no. 4, pp. 391-407

38. D. Kinzer, "A Memory Pattern Sensitivity Test," BYTE, October 1978, vol. 3, no. 10, pp. 12-16

39. Computer Sciences Corporation, CSC/TM-78/6360, IMP-16 Tracking Data Preprocessor Design, L. Lu and C. Shenitz, December 1978

40. National Aeronautics and Space Administration, Goddard Space Flight Center, S-700-55, Multimission Modular Spacecraft (MMS) Onboard Computer (OBC) Flight Executive Definition, A. Merwarth, March 1976

41. Computer Sciences Corporation, "SM/PL PROM Burning Utility (IMPLDR)" (interoffice correspondence, Task 866 file), C. Rabbin, November 28, 1978

STL BIBLIOGRAPHY

Systems Technology Laboratory, STL-78-001, System Description of the IMP-16C Microprocessor Orbit Determination Program, C. Shenitz, C. Rabbin, and G. Snyder, October 1978

--, STL-78-002, Landsat/NAVPAC System Description and User's Guide, S. R. Waligora, December 1978

--, STL-79-001, Intel 8080 Orbit Propagation Program System Description and User's Guide, C. Rabbin, April 1979

--, STL-79-002, Evaluation of the IMP-16 Microprocessor Orbit Determination System Filter, C. Shenitz, September 1979

--, STL-79-003, Multimission Modular Spacecraft Ground Support Software System (MMS/GSSS) MODCOMP Device and MAX IV Dependency Study, T. Weldon and M. McClellan, December 1979

--, STL-80-001, Orbit Determination Software Development for Microprocessor-Based Systems: Evaluation and Recommendations, C. M. Shenitz, July 1980

--, STL-80-002, The Two-Way TDRSS Observation Model for the LSI-11/23 Microcomputer, C. E. Goorevich, July 1980

--, STL-80-003, Automated Orbit Determination System (AODS) Requirements Definition and Analysis, S. R. Waligora, C. E. Goorevich, J. Teles, and R. S. Pajerski, September 1980

--, STL-80-004, Algorithms for Autonomous Star Identification, P. Gambardella, October 1980

--, STL-80-005, Autonomous Onboard Attitude Determination System Specifications and Requirements, M. D. Shuster, S. N. Ray, and L. Gunshol, December 1980

--, STL-81-001, Systems Technology Laboratory (STL) Library Methods and Procedures, W. J. Decker and P. D. Merwarth, September 1981

--, STL-81-002, Mathematical Specifications of the Onboard Navigation Package (ONPAC) Simulator (Revision 1), J. E. Dunham, February 1981

--, STL-81-003, Systems Technology Laboratory (STL) Compendium of Utilities, W. J. Decker, E. J. Smith, W. A. Taylor, P. D. Merwarth and M. E. Stark, July 1981

--, STL-81-004, Automated Orbit Determination System (AODS) Environment Simulator for Prototype Testing (ADEPT) System Description, S. R. Waligora, J. E. Fry, Jr., B. J. Prusiewicz, and G. N. Klitsch, August 1981

--, STL-81-005, Preliminary Automated Orbit Determination System (AODS)/AODS Environment Simulator for Prototype Testing (ADEPT) User's Guide, S. R. Waligora, Y. Ong, J. E. Fry, Jr. and B. J. Prusiewicz, September 1981

--, STL-82-001, GPSPAC/Landsat-D Interface (GLI) System Description, J. B. Dunham, H. M. Sielski, and W. T. Wallace, April 1982

--, STL-82-002, GPSPAC/Landsat-D Interface (GLI) System User's Guide, H. M. Sielski, J. B. Dunham, and P. D. Merwarth, March 1982

--, STL-82-003, Autonomous Attitude Determination System (AADS), Volume 1: System Description, K. A. Saralkar, Y. G. Frenkel, G. N. Klitsch, K. Liu and E. Lefferts, April 1982

--, STL-82-007, System Description for the Global Positioning Subsystem Experiment Package (GPSPAC) Experiment Data Preprocessor (GEDAP), P. D. Merwarth and J. F. Cook, June 1982