DESIGN STUDY OF AN AVIONICS NAVIGATION MICROCOMPUTER

William Lowell McCracken

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

DESIGN STUDY OF AN
AVIONICS NAVIGATION MICROCOMPUTER

by

William Lowell McCracken

June 1974

Thesis Advisor: U.R. Kodres

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Design Study of an Avionics Navigation Microcomputer		5. TYPE OF REPORT & PERIOD COVERED Engineer's Thesis; June 1974 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) William Lowell McCracken		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE June 1974 13. NUMBER OF PAGES
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

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

18. SUPPLEMENTARY NOTES

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

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

A microcomputer program to solve the complex task of air-borne navigation was developed to demonstrate the practicality of replacing constly general purpose digital computers with relatively inexpensive dedicated microcomputers on-board naval aircraft. The microcomputer program showed that microcomputers have sufficient speed and accuracy to solve the navigation problem. In order to overcome the microcomputer's

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |
(Page 1)

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

(20. ABSTRACT continued)

major deficiencies, speed and accuracy, special arithmetic subprograms based on table look-up were developed to trade inexpensive memory for more speed. An application of graph theory in the form of process graphs was made to facilitate the development and documentation of the navigation program. To aid in the testing of the microcomputer program, a Fortran simulation program was developed to confirm the results of an error bound analysis of the navigation program.

Design Study of an Avionics Navigation Microcomputer

by

William Lowell McCracken
Lieutenant, United States Navy
B.S., United States Naval Academy, 1967

Submitted in partial fulfillment of the
requirements for the degree of

AERONAUTICAL ENGINEER

from the

NAVAL POSTGRADUATE SCHOOL
June 1974

ABSTRACT

A microcomputer program to solve the complex task of air-borne navigation was developed to demonstrate the practicality of replacing costly general purpose digital computers with relatively inexpensive dedicated microcomputers on-board naval aircraft. The microcomputer program showed that microcomputers have sufficient speed and accuracy to solve the navigation problem. In order to overcome the microcomputer's major deficiencies, speed and accuracy, special arithmetic subprograms based on table look-up were developed to trade inexpensive memory for more speed. An application of graph theory in the form of process graphs was made to facilitate the development and documentation of the navigation program. To aid in the testing of the microcomputer program, a Fortran simulation program was developed to confirm the results of an error bound analysis of the navigation program.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

TABLE OF SYMBOLS

Symbols	
ACC	Accumulator
ALT	Altitude
ALTS	Smoothed Altitude
ALU	Arithmetic Logic Unit
BCD	Binary-Coded Decimal
BIT	Binary Digit
BYTE	Eight Binary Digits
CPU	Central Processor Unit
CROM	Control Read Only Memory
CY	Carry
DA	Drift Angle
DIM	Digital Input Multiplexer
DOM	Digital Output Multiplexer
DIP	Dual-Inline Package
DX	Smoothed Distance East/West
DY	Smoothed Distance North/South
DXD	Doppler Distance East/West
DYSD	Doppler Distance North/South
DXI	Inertial Distance East/West
DYI	Inertial Distance North/South
DXW	Air-Mass Distance East/West
DYW	Air-Mass Distance North/South
IC	Integrated Circuit
INS	Inertial Navigation System

I/O	Input/Output
IRO	Index Register Zero
IR1	Index Register One
IRA	Index Register Ten
IRB	Index Register Eleven
IRC	Index Register Twelve
IRD	Index Register Thirteen
IRE	Index Register Fourteen
IRF	Index Register Fifteen
G	Smoothing Factor
KLA	Latitude Conversion Constant
KLO	Longitude Conversion Constant
L	Least Significant Digit
LON	Computed Geographic Position Longitude
LAT	Computed Geographic Position Latitude
LSI	Large Scale Integration
LOF	Fix Longitude
LAT	Fix Latitude
M	Most Significant Digit
MOS	Metal-Oxide Semiconductor
NAV	Navigation
NIBBLE	Four Binary Digits
PROM	Programmable and Erasable Read Only Memory
R	Designated Index Register
RALU	Register and Arithmetic Logic Unit
RAM	Random Access Memory

ROM	Read Only Memory
SDD	Doppler Distance Across Heading
SHD	Doppler Distance Along Heading
SDX	System Bias Distance East/West
SDY	System Bias Distance North/South
SDXE	System Drift Error East/West
SDYE	System Drift Error North/South
SR	Shift Register
T	Time of One Navigation Cycle
TAS	Smoothed True Air-Speed
TASR	Rough True Air-Speed
TF	Time Between Each Fix
TH	True Heading
VAX	True Air-Speed East/West
VAY	True Air-Speed North/South
VDD	Doppler Velocity Across Heading
VHD	Doppler Velocity Along Heading
VGXI	Inertial Ground Velocity East/West
VGYI	Inertial Ground Velocity North/South
VGXW	Air-Mass Ground Velocity East/West
VGYW	Air-Mass Ground Velocity North/South
VSDX	System Bias Velocity East/West
VSDY	System Bias Velocity North/South
VWXR	Rough Wind Velocity East/West
VWYR	Rough Wind Velocity North/South

ACKNOWLEDGMENTS

I would like to thank the P3C Software Division at NADC, Warminster, for their assistance in supplying me with information on Avionics navigation during my experience tour there. I would also like to thank Intel Corp. for letting me use information from their MCS-4 User's Manual from which Figures 1 and 2 and Tables III, IV, and V were taken. I wish also to thank my thesis advisor, Dr. Uno R. Kodres, for his patience, guidance, and timely advice which made this report possible.

I. INTRODUCTION

Current naval aircraft are depending more and more on airborne digital computers. The digital computer has proven to be a great aid, providing the precision and speed required to perform many calculations. The digital computer is very versatile, because the same computer can be programmed to perform new or different tasks. The only limitations to the use of digital computers are those of cost and maintenance requirements.

The digital computers currently used by such naval aircarft as the P3C, A6E, A7E, and S3A are all large general purpose computers. These computers are very fast, flexible, but expensive. The large size, large power requirements, and great cost has limited each aircraft to one such computer. The complexity of these computers has made maintenance difficult. The large cost of each unit makes spares prohibitive, therefore one computer going down results in the operational loss of one aircraft.

It is advantageous to utilize several small distributed computers to meet all of the systems requirements. In order to minimize the probability of the entire system failing due to the failure of a single component in a critical computer circuit, completely separate computers could be used for the various system requirements with back up computers ready to fill in when needed. In addition to

reliability improvements, a distributed approach offers the possibility of using less complex equipment, flexibility in matching equipment to system requirements, and increased standardization. The limitation of this approach is the increased cost, weight, size and the complexity of system interconnections.

The creation of the microcomputer, using new developments in the Large Scale Integration (LSI) technology, has made the distributed computer system possible. The microcomputer is a complete general purpose computer on a set of four standard LSI chips. The LSI chip measures 200 mils by 200 mils, requires less than one watt of power and costs about $30. The processors Central Processing Unit (CPU), constructed on a single chip, is designed to be used in a multiprocessor environment. The limitations of a microcomputer are a limited instruction set and slow speed.

This thesis is a design study of the possibility of using the MCS-4 microcomputer as the Avionics Navigation Computer in a complex navigation system. Section II discusses what a microcomputer is, what microcomputers are currently available, and what is the system makeup of the MCS-4 microcomputer used in this report. Section III describes the navigatin systems currently available, what their advantages and limitations are, and the navigation system chosen for this report. The navigation equations used to compute the current position of the system's carrier are discussed in Section IV. The actual programming of the

microcomputer is discussed in detail in Section V. This includes the programming aids developed in this report, a program analysis of the requirements of this system, a detailed discussion of the executive routine and the subroutines used in this program, and an error bound analysis of the final program. Section VI describes the FORTRAN simulation program written to aid in the writing, debugging, and testing of the final navigation program. The conclusions of this report are summed up in Section VII. The listing of both the MCS-4 microcomputer navigation program and the FORTRAN simulation program are included in the Appendices.

II. MICROCOMPUTER

A microcomputer is a general-purpose digital computer constructed from a set of LSI chips. It has a complete instruction set and is capable of addressing sizeable memories. It can interface with a full complement of input and output devices.

The main component of a microcomputer is the microprocessor. The microprocessor is a CPU on a chip that interprets and executes instructions in a bit-parallel fashion. Included on the CPU chip are the Index registers, Arithmetic unit and Input/Output control logic.

The main feature that distinguishes a microprocessor from a general purpose or minicomputer CPU is that the entire CPU is on one chip. This has been made possibly by Large Scale Integration, where over 14,000 Metal-Oxide Semiconductor transistors can be put on one chip. A minicomputer CPU with the capability of one microprocessor would require over 100 TTL packages. The major advantages of microprocessors are low cost, low power requirement, and less complexity in system design. Figure 1 is a photomicrograph of the 4004 CPU chip used in the MCS-4 Microcomputer.

The microprocessor becomes a microcomputer when a control program, memory, and input/output circuits are added to the system. The control program is usually Metal-Masked on a Read-Only Memory (ROM) chip, however, a Programmable

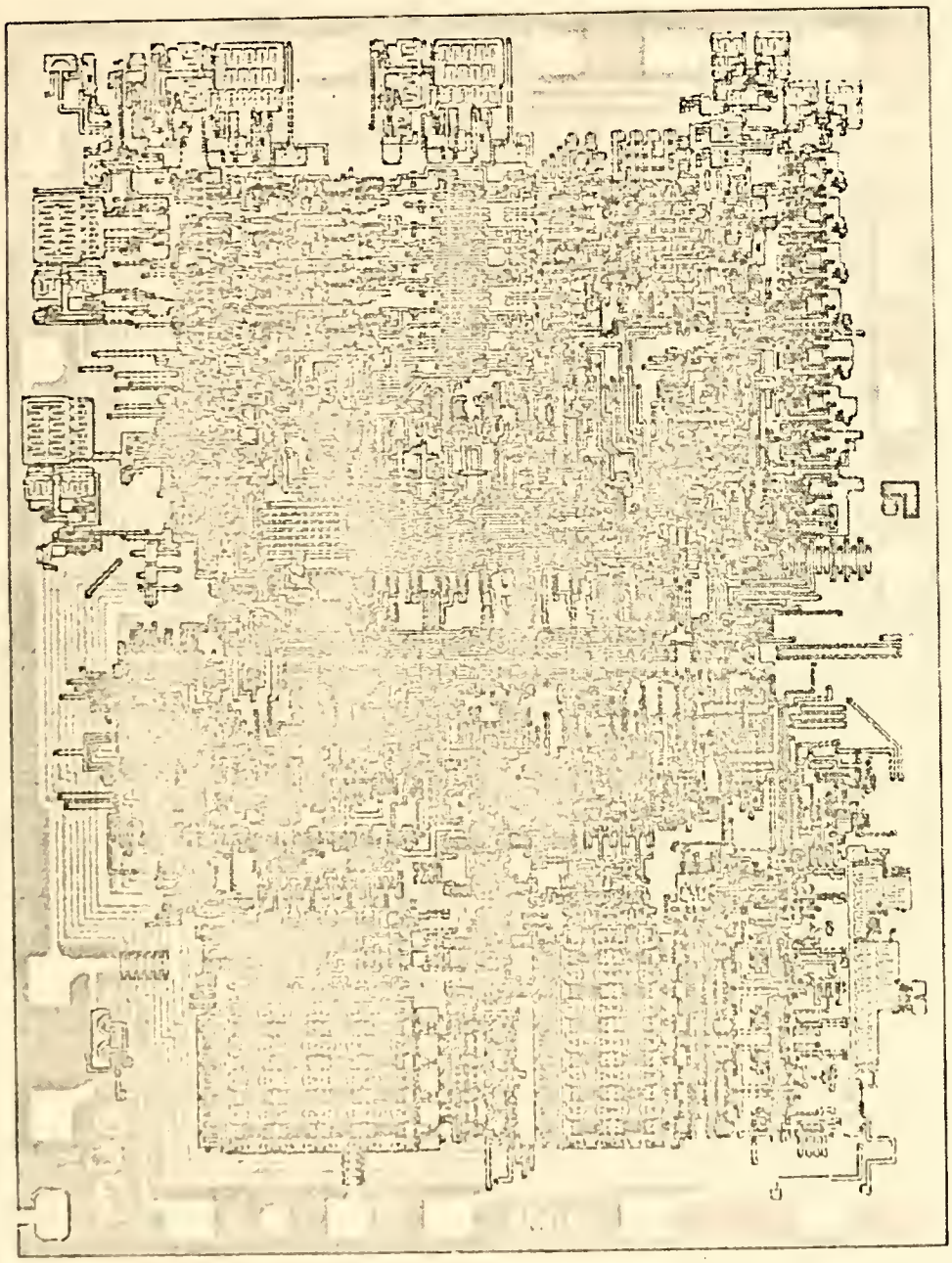


Figure 1. Photomicrograph of 4004 CPU

Read-Only Memory (PROM) may be used for initial system development. The storing of data or variables in a program is handled by the Random Access Memory (RAM) chip which is a read/write type memory. The input/output ports are incorporated on the RAM, ROM, or CPU chip or any combination of these chips. The total chip area of a typical microcomputer is 200 mils by 200 mils and costs less than $200.

Microcomputers have been used in specialized, single user data processing systems and as components in digital products. Microcomputers are currently being used in calculators, special purpose terminals, measurement systems, intelligent traffic controls, small business computers, and digital cash registers. Research is in progress to use microcomputers in intelligent peripherals, multiplexors and communications controllers, automotive control systems, and educational systems. Microcomputers can also be used in large computer systems to relieve the large central processor of much of the overhead associated with scheduling, text editing, and file management.

Microcomputers major application will be in the area of dedicated computations. They provide all the data processing power needed for these applications. These areas include computational tasks required by NTDS, TSC, and most current naval aircraft.

The following is a list of advantages of microcomputers:

1. Reduced costs due to reduction in number of logic card types.

2. Self test capability.
3. Equipment modularity.
4. Equipment commonality.
5. Ease in design changes through microprogram.
6. Reduced logistic support.
7. Standardization of peripheral interfaces.
8. Multiprocessor capability.

The total savings in cost, power required, and size, together with the flexibility inherent in the system, make the microcomputer a powerful tool for system design of dedicated computational tasks.

A. COMPARISON OF MICROCOMPUTER WITH GENERAL PURPOSE COMPUTER

The general purpose computer and microcomputer represent the two extremes on the computer scale. The general purpose computer is large in size, very expensive, requires much power, and is very fast. The microcomputer is small, inexpensive, requires little power, and is relatively slow in comparison. The general purpose computer is built using Integrated Circuits (I.C. Technology), while the microcomputer is built from Large Scale Integration Techniques (LSI Technology). The great cost savings of LSI Technology in computer design can best be shown by comparing the cost of designing and manufacturing a 3000 gate logic unit using I.C. techniques versus the LSI techniques.

The cost of manufacturing a computer system can be expressed by the following formula:

$$CS = \sum_{i=1}^{n} [(CM(i) + CT(i))V(i) + CG(i)]$$

where

CS ≡ Cost of System

CM(i) ≡ Cost of Manufacturing the i-th Replaceable Module

CT(i) ≡ Cost of Testing the i-th Replaceable Module

V(i) ≡ Number of Modules used in the system of the i-th type

CG(i) ≡ Cost of generating the i-th Replaceable Module, including design, layout, test condition generation, etc.

The cost of manufacturing the computer system from I.C. circuits was estimated using figures from people knowledgeable in this area, however, documentation of these figures was not available.

The cost of manufacturing and the cost of testing an individual I.C. module was assumed to be independent of the type of I.C. module because of standard means of manufacturing and testing these units. This assumption simplifies the "cost of system" equation as follows:

$$CM(i) \cong \overline{CM}$$

$$CT(i) \cong \overline{CT}$$

$$CS = (\overline{CM} + \overline{CT})\ V + \sum_{i=1}^{n} CG(i),$$

where

$$V = \sum_{i=1}^{n} V(i).$$

I.C. technology does not permit highly complex functional circuitry to be placed on one module. The typical module consisting of sixteen pins contains eight gates. A 3000 gate logic pattern would therefore require V = 360 modules. The technology typically used by computer manufacturers permits 60 modules to be assembled on one panel which provides all the interconnections between the modules. The system would then require six panels at an estimated cost of $100 per panel.

In determining the number of distinct modules used, it was assumed that about one-half of the modules used in the system are duplicates of previously designed modules, n = 180. The cost of designing one new module was estimated to be $250. The total cost of one system was computed as follows:

$\overline{CM}$ = $20

$\overline{CT}$ = $ 5

V = 360

CG(i) = $250

$$n = 180$$

$$CS = (\overline{CM} + \overline{CT}) + CG(i) + 600$$

$$CS = 9{,}000 + 45{,}000 + 600 = 54{,}600$$

To produce this unit in large numbers, the cost of manufacturing and testing the modules and panels is multiplied by the number of units produced while the cost of designing remains unchanged. The cost of 1000 units could be predicted as follows:

$$CS(1000) = 9000(1000) + 45{,}000 + 600(1000)$$

Total Cost of 1000 units = $9,645,000

Next the cost of developing and manufacturing a 3000 gate logic unit using LSI technology was investigated. The cost estimates for this analysis were obtained from a trip to Intel Corp., Santa Clara, California. The LSI design cost of the 8008 CPU chip used in the MCS-8 Microcomputer was used as an indication of the cost associated with manufacturing the required logic unit by LSI technology.

The entire design and layout of the 8008 CPU was done exclusively by hand and took a total of four man-years. The approximate cost of developing this chip is estimated at $100,000. The total manufacturing and testing costs of

the chip is $15. Since all the logic gates are on one chip, there are no other costs associated with the unit. The cost of one system is therefore:

$$CS = (15)\ (1) + 100,000 + 0$$

$$CS = \$100,015$$

Again, the cost of producing the system in large quantities, the design cost remains fixed while the manufacturing and testing costs are multiplied by the number of units. The cost of 1000 units using LSI technology is as follows:

$$CS(1000) = (15)(1000) + 100,000$$

Total cost of 1000 units = $115,000

It was concluded from these calculations that LSI technology could produce the system in quantity about 80 times cheaper than by I.C. technology.

It was concluded from this analysis that the microcomputer has a tremendous cost advantage over general purpose computers in systems produced in large quantities with a need for a dedicated computer. The object of this thesis is to demonstrate that the microcomputer also has the speed and computational power to handle complex tasks.

B. COMPARISON OF MICROCOMPUTER WITH MINICOMPUTER

The minicomputer was developed to meet the need for dedicated systems to handle data processing requirements such as communications control, data acquisition, and small business accounting. The microcomputer is proving itself capable of handling these tasks with improved price/performance, compactness, and reliability.

The greatest advantage the minicomputer has over the microcomputer is speed. The LSI chips used for microcomputers are made using Metal Oxide Semiconductor, MOS, technology. MOS technology allows smaller size of individual transistors, and logical circuits together with low power consumption. The electrical properties of MOS circuitry, however, make it slower than the Bipolar circuitry used in minicomputers. In order to increase the computational speed of microcomputers, many microcomputer makers are switching from the slower P-channel MOS devices to the much faster N-channel MOS devices. By using microprogramming techniques such as pipelining, the computational speed of microcomputers can become as fast or faster than minicomputers.

The next advantage the minicomputer has over the microcomputer is in the instruction set size. The first generation microcomputers were limited to between 40 to 60 instructions while the minicomputers had instruction sets in the 100-120 range. The gap here is also closing with the second generation microcomputers having instruction sets of 50-100 instructions.

The last major advantage of the minicomputer is the existing software developed over the past several years to assist program development. With the development of high level programming languages (PL/M for the Intel 8008 and 8080), the microcomputers are quickly eliminating this advantage of the minicomputers.

The microcomputer offers better price/performance, lower power consumption, smaller size and higher reliability than minicomputers. Although a single microcomputer can not match the power of a minicomputer, several microcomputers can be combined to share the workload at a cost still less than a minicomputer. An added advantage of multi-microcomputers is processor reliability which can be increased through the use of back-up processors, providing a self-test and repair capability.

Table I is a summary of the comparison of microcomputers and minicomputers.

C. CURRENTLY AVAILABLE MICROCOMPUTERS

The number of microcomputers being designed and currently available is increasing rapidly. This section covers eight microcomputers that are currently available or will be in the near future. Table II is a summation of the capabilities and support available for each microcomputer.

	MINICOMPUTER	MICROCOMPUTER
CPU Cost	$1500	$30
Instruction Speed	2-5 msec	10-20 msec
Execution Speed (Memory to Memory Add)	5-20 msec	15-60 msec
Instruction Set	100-150	50-100
Price/Instruction Ratio	$80-$300	$40
Registers	1-30	16-100
Price/Register Ratio	$300	$15
Memory Size	64K	64K
Performance/Price $\left(\frac{\text{Word Length}}{\text{Add Time} \times \text{Price}}\right)$	200 $\left(\frac{\text{Bits}}{\text{Sec-Dollar}}\right)$	100 $\left(\frac{\text{Bits}}{\text{Sec-Dollar}}\right)$
Power Consumption	4 Watts	1 Watt
Reliability (CPU)	Less (Due to 100 TTL Packages)	Greater (Due to One LSI Chip)

TABLE I. MICROCOMPUTER VERSUS MINICOMPUTER

	INTEL MCS-4	INTEL MCS-8	INTEL 8080	AMI 7300	FAIRCHILD PPS-25	ROCKWELL PPS-4	SIGNETICS PIP	NATIONAL IMP-16
Instr. Word (Bits)	8	8	8	8	12	8	8	8
Number Instr.	45	48	74	512	95	56	64	42
Memory Size (Words)	4K	16K	65K	4K	6.6K	4K	32K	65K
Subroutine Nesting	3	7	Unlimit	32	Unlimit	Unlimit	8	Unlimit
Execution Time (μs)	10.8	7.5	2	4	62.5	20	4	3
CPU Cost ($)	30	60	NA	300	NA	150	100	500
No. of Registers	17	8	9	49	7	4	7	23
No. of I/O's	64/128	8/24	256/256	65K	98/16	200/196	256/256	65K
Technology	P-Chan	P-Chan	N-Chan	P-Chan	P-Chan	P-Chan	N-Chan	P-Chan
No. CPU Chips	1	1	1	2	7	1	1	5
Status	Distrib.	Distrib.	Distrib.	Proto	mfg.	mfg.	proto	Distrib.
Software Aids	Excel.	Excel.	Excel.	Fair	Fair	Fair	Fair	Good

TABLE II. SUMMARY OF AVAILABLE MICROCOMPUTERS

1. Intel MCS-4

The MCS-4 is a four-bit parallel processor with a fixed instruction set. The 4004 CPU is a P-Channel MOS LSI chip mounted on a 16-pin package. A four-bit data bus connects the CPU with up to 16 ROMs and 16 RAMS. The instruction cycle is 10.8 microseconds.

The instruction set used by the MCS-4 consits of 45 instructions grouped into three categories: Machine, Input/output, and accumulator. System development aids include a cross assembler available on the IBM 360/67 located at the Naval Postgraduate School. The SIMQ4-02 Hardware Prototyping System with Assembling, Programmable-ROM Programming, and Debugging capability is also available at the NPS microcomputer lab.

The MCS-4 Microcomputer was chosen as the navigation microcomputer in this thesis. The MCS-4 is covered in more depth in Section II.D.

2. Intel MCS-8

The MCS-8 uses the 8008 single CPU chip. The 8008 is an eight-bit fixed instruction set parallel processor mounted in an 18-pin package. The 8008 executes a single instruction in 20 microseconds. The 8008 is capable of addressing 16K bytes of memory. An eight-bit data bus interfaces the processor with memory. A total of 48 instructions are available broken into four groups: Instruction

Register, Accumulator, Program Counter and stack control, and Input/Output.

System Development aids include a cross assembler available on the IBM 360/67 at the Naval Postgraduate School. The SIM8-01 Prototyping System with programmable and erasable ROMs is available in the NPS microcomputer lab for development and check-out of microprograms. The development of the PL/M Higher Level Language patterned after PL/I for the MCS-8 greatly facilitates the programming task. The PL/M compiler is also available on the IBM 360/67 at NPS. Lastly the INTELLEC-8 system, available in the NPS microcomputer lab, makes available to the programmer a resident software monitor, assembler, PROM programmer, and text editing capability to aid in the development of microprograms for the MCS-8.

3. Intel 8080

The Intel 8080 is the first of the second generation microcomputers. The 8080 derives more speed and capability by using the more efficient N-Channel MOS. The 8080 CPU is a single-chip eight-bit parallel processor in a 40 pin package. The 8080 contains six eight-bit data registers, an eight-bit accumulator, three eight-bit temporary registers, four testable flags, and an eight-bit arithmetic/logic unit. The execution time for one instruction is 2 microseconds.

The 8080 can directly access 64K bytes of memory. A separate 16-bit address bus is provided as well as ten

control lines that indicate CPU and I/O bus status. Up to 256 I/O devices can be directly addressed. Multiprocessor capability is designed into the MCS-8080.

The 8080 is software compatible with the 8008 microprocessor. The 8080 instruction set contains the 48 instructions of the 8008 plus 26 new instructions for a total of 74. The 8080 is capable of unlimited subroutine nesting.

System development aids include a cross assembler, INTELLEC 8080 simulator, and the PL/M higher level programming language.

4. AMI 7300

The AMI 7300 is an eight-bit fully parallel, bus-oriented processor. The processor consists of two chips, the Micro instruction ROM chip (MIR) and the registers-adder logic unit chip (RALU). Both chips are P-Channel MOS and each is packaged in a 40-pin dual-inline package (DIP).

The MIR contains a mask-programmable 512 word x 22 bit microinstruction ROM and a programmable instruction mapping array, allowing up to 50 microprogram locations to be predefined for macroinstruction decoding. This allows the instruction set to be tailored to suit a particular application. A hardware address stack and loop counter allows subroutine nesting to seven levels.

The RALU contains 48 registers which may be utilized as one or two first-in/last-out stacks or as one or two files of general registers. The eight-bit adder/subtractor performs over 30 arithmetic and logical operations. The

instruction set consists of three basic formats: Register control, literal and branching. The processor can address up to 64 K of memory.

Software development aids include a cross assembler and instruction simulator.

5. Fairchild PPS-25

The PPS-25 is a BCD oriented 25-digit serial/parallel processor. The system is best suited to decimal applications such as calculators, keyboard/printer interface, and vending machines. Four level subroutine nesting and three way conditional branching are provided. The system can be microprogrammed with a custom instruction set to best meet the needs of the user.

Program storage consists of up to 26 ROMs, each capable of storing 256 twelve-bit words. Seven general purpose 25-digit registers are provided and an external interrupt capability is included.

The 3805 Arithmetic Chip includes the Adder/Subtractor plus a 25-digit register. The instructions are located in the 3810 ROM. A total of 30 arithmetic/logic and 16 I/O instructions are available with the standard set. The I/O format permits expansion to 63 I/O instructions.

Software development aids include a cross assembler and instruction simulator.

6. National Semiconductor GPCP/IMP-16

The GPCP sixteen-bit microcomputer processor consists of five MOS LSI chips, each mounted in a 40-pin DIP. The

five chips consist of four RALU chips and one CROM (Control Read Only Memory). Each RALU chip is a four-bit slice of CPU with its own registers, ALU logic, and I/O data lines. Multiple CROMs may be used to increase the size of the instruction set.

The RALU consists of a four-bit program counter, four-bit memory data register, four-bit memory address register, four-bit accumulator, a pushdown stack, data multiplexer, and four-bit arithmetic and logic unit. The system may be expanded to 32 bits by adding four more RALU chips.

The CROM contains the control instructions for the RALU chips. The CROM is broken into two parts, an instruction ROM containing 100 twenty-three-bit words and an address control ROM consisting of 12 programmable ten-bit words.

The IMP-16 is a sixteen-bit microcomputer developed to use the GPCP microprocessor. The CROM provided with the system contain a 43 word instruction set. The instruction set may be expanded to meet the system designer's specific needs. Communication between the RALU chips and the CROM chip is over a sixteen-bit data bus and a four-bit control bus. This requires sixteen extra TTL packages excluding memory and timing.

Software development aids include a cross assembler and prototyping system with resident monitor, assemblers, and linking loader.

7. North American Rockwell PPS-4

The PPS-4 microcomputer consists of a set of six MOS LSI chips. The CPU is a four-bit single chip processor mounted in a 42-pin package. The other five chips support the CPU and consist of: A 256 x 4-bit RAM, 1024 x 8-bit RAM, an I/O buffer, and a two-phase clock generator.

The CPU can drive up to 4K bytes of ROM and 4K bytes of RAM over its 12-bit parallel address bus. The basic instruction set contains 50 instructions with an execution time of five microseconds.

Twenty-one multiplexed lines interconnect the CPU with ROM, RAM, and I/O circuits. These lines are functionally grouped into twelve parallel address lines, eight parallel data lines, and one write command and I/O enable line. The ROM has two chip-select inputs and the RAM has one chip-select input, which may be directly controlled by discrete outputs from the CPU or I/O circuits to expand on memory without the need for auxiliary circuits. Each I/O chip can handle up to 12 inputs and 12 outputs with a total system capability of 16 I/O chips.

Software development aids include a cross assembler and simulator available on a national time-sharing network.

8. Signetics PIP

The programmable integrated processor (PIP) is a single-chip eight-bit CPU made with N-channel MOS technology. This second generation CPU is packaged in a 40-pin DIP. The

address logic, control memory, and ALU are organized around an eight-bit bidirectional data bus. There are fifteen address lines for handling external memory and I/O circuitry. The PIP instruction set contains over 64 instructions with an execute time of less than ten microseconds for the most complex instruction.

The PIP chip can be broken into four parts, the address logic, the RALU, and the control section. The address logic section handles all instructions and includes a return address stack that allows subroutine nesting to eight levels. The RALU section contains four 8-bit general purpose registers and executes all arithmetic, boolean, compare, and rotate operations. The control section manages operation of all external control lines, decodes all instructions, and coordinates the activities of all other internal circuitry.

Software development aids include a cross assembler and instruction simulator.

9. Summary

Microcomputers have the capability of replacing both special function logic modules and large computational machines. Certain microcomputers are more suited for one application than for others. In selecting a microcomputer, such parameters as data word length and type, instruction power, and interface structure must be considered. Table II is a summary of the microcomputers discussed in the section.

The rapid developments in the LSI technology should bring about many new improvements in microcomputers such as wider word lengths, larger memory capacity, and more flexible and convenient I/O interfacing. Improvements in LSI structure will make possible single chip microcomputers with the CPU, I/O, and memory all on one chip. Work on a Bipolar LSI processor could develop into a microcomputer with more speed and memory than most present minicomputers.

D. THE MCS-4 MICROCOMPUTER

The MCS-4 was the first microcomputer made. Compared to other microcomputers, it is slow with a limited instruction set. The most complex function available on the MCS-4 is a four-bit add. The MCS-4 is also the least expensive microcomputer. The structure of the MCS-4 is similar to a general purpose computer, making it compatible with the requirements of a navigation computer.

The MCS-4 was chosen as the microcomputer in this design study for two major reasons. The first reason is that the MCS-4 is the least powerful and hence serves as a lower bound of the microcomputers. To prove that the MCS-4 is capable of handling the required navigational computations, would in itself prove microcomputers capable of handling complex tasks. The second major reason for choosing the MCS-4 microcomputer is that it is available, has been tested, and has the required software aids to complete a design study.

1. MCS-4 System Description

The MCS-4 microcomputer is built up from a standard set of off-the-shelf chips. The only custom part is the ROM chip which stores the specific program defined by the user and requires a metal mask option for each new program.

The MCS-4 consists of four chip types, each packaged in a conventional 16-pin DIP:

(1.) A Central Processor Unit Chip-CPU-4004

(2.) A Read Only Memory Chip-ROM-4001

(3.) A Random Access Memory Chip-RAM-4002

(4.) A Shift Register Chip-SR-4003

The CPU contains the control unit and the arithmetic unit. The ROM stores the program and data tables, the RAM stores input data and variables, and the Shift Register is used in conjunction with I/O devices to effectively increase the number of I/O lines.

A complete microcomputer can be built using only a single CPU chip and a single ROM, and the only external circuitry required is a two phase clock. The CPU is capable of driving a system up to 16 ROMs (4K bytes), 16 RAMs (640 bytes) and 128 I/O lines, with no additional interfacing circuitry. The CPU communicates with the RAM's and ROM's by means of a four-line data bus. This single data bus is used for all information flow between the chips except for control signals which are sent to RAM and ROM over five additional lines. Figure 2 shows the MCS-4 System Interconnection.

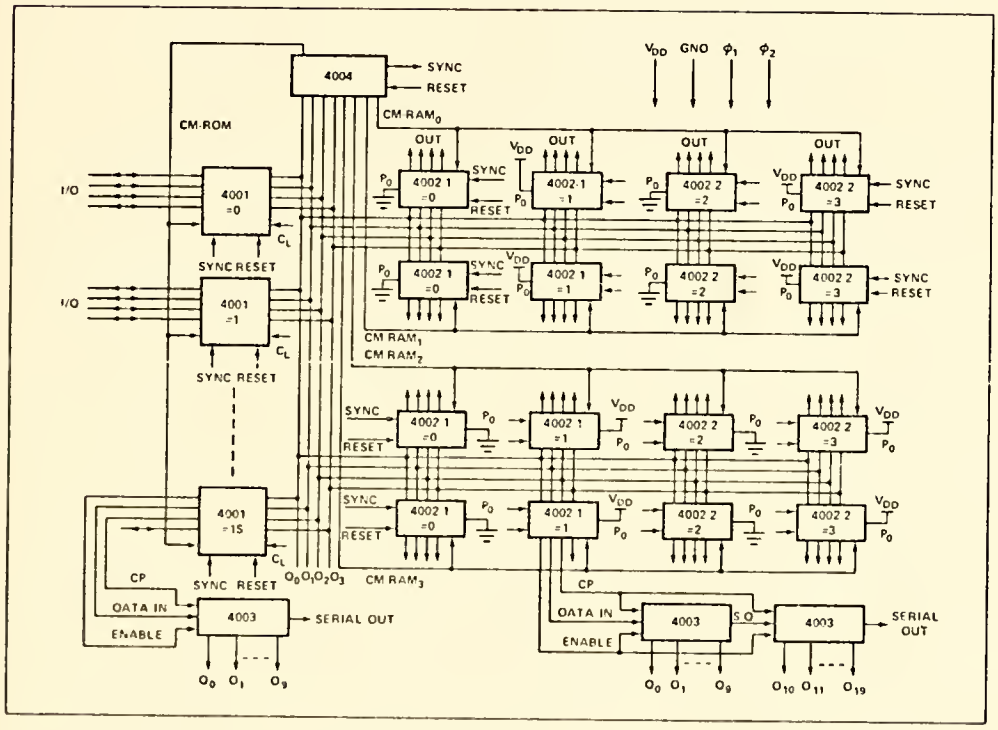


Figure 2. MCS-4 System Interconnection

The MCS-4 uses a 10.8 microsecond instruction cycle. The cycle is broken into eight steps. In the first three steps, the CPU sends the memory address to the ROM in three 4-bit nibbles. The ROM then sends back 8 bits of instruction in two 4-bit nibbles during steps 4 and 5. The instruction is then interpreted and executed by the CPU during the last three steps.

a. 4004 CPU

The heart of the MCS-4 microcomputer is the 4004 CPU. The 4004 CPU contains the following functional blocks:

(1.) Address Register and Address Incrementer

(2.) Index Register

(3.) 4-bit adder

(4.) Instruction Register, Decoder and Control

(5.) Peripheral Circuitry

The Address Register is a RAM array of 4 x 12 bits. One level is used to store the current instruction address, leaving three levels to store the addresses of nested subroutines. As each byte of address is sent onto the data bus, the address is incremented by a 4-bit carry look-ahead circuit. The incremented address is then transferred back into the address register.

The Index Register is a RAM array of 16 x 4 bits and has two modes of operation. In one mode of operation the index register provides sixteen directly addressable storage locations. In the second mode, the index registers

provide eight pairs of addressable storage locations for addressing RAM and ROM as well as for storing data fetched from ROM. The index registers can thus provide 64 bits of RAM to a minimum MCS-4 system of one CPU and one ROM.

The 4-bit Adder is of the ripple-through carry type. The output of the adder is transferred to the accumulator and carry flip-flop. The accumulator is provided with a shifter to implement rotate right and rotate left instructions. The accumulator also communicates with the command control register, the condition flip-flop and the 4-bit internal data bus. The condition logic allows the execution of conditional instructions based on the contents of the accumulator, index registers, or the status of the control lines.

The Instruction Register is an 8-bit register which is loaded with the two 4-bit nibbles of instruction read from the ROM. The instructions are decoded in the instruction decoder and appropriately gated with timing signals to provide the control signals for the various functional blocks.

The peripheral circuitry consists of the 4-bit internal data bus, the timing and SYNC generator, one ROM command control and the four RAM command control output buffers, and the reset flip-flop.

b. 4001 ROM

The 4001 is a 2048-bit metal mask programmable ROM. The 4001 performs two functions. As a ROM, its first function is to store 256 x 8-bit words of program or data tables. The second function of the 4001 is to act as a vehicle of communication between the data bus lines and peripheral devices through the 4-bit input-output port located on each chip.

c. 4002 RAM

The 4002 also performs two functions. As a RAM it stores 320 bits arranged as four registers of twenty 4-bit characters each. As a vehicle of communication with peripheral devices, it is provided with 4 output lines and associated control logic to perform output operations.

d. 4003-SR

The 4003 is a 10-bit serial-in/parallel-out serial-out shift register. Its function is to increase the number of output lines to interface with I/O devices.

2. Instruction Set

The MCS-4 Instruction Set consists of a total of 45 instructions grouped into three sets: Machine Instructions, Input/Output and RAM Instructions, and Accumulator Group Instructions.

The Machine Instructions are the Housekeeping instructions of the MCS-4. They consist of two types of instructions: one-word instructions which are 8-bits wide

and require one instruction cycle, and two-word instructions which are 16-bits wide and require two instruction cycles. Table III is a list of the Machine Instructions.

The Input/Output Instructions are used to transfer data between the Accumulator and RAM. These instructions are also used to transfer data between the Accumulator and the I/O ports located on the ROMs and RAMs. Table IV is a list of the Input/Output Instructions.

The Accumulator Instructions are used to perform bit-by-bit manipulation of the data in the Accumulator. Table V is a list of the Accumulator Instructions.

3. System Development Aids

The program written for the MCS-4 is metal-masked into the ROMs. Before doing this, the program needs to be tested to insure that it functions correctly in all situations. To aid in testing programs written for the MCS-4, there are available three system development aids. These aids include an assembler and interpreter, a complete hardware prototyping system, and a resident software monitor.

The MCS-4 Assembler and Interpreter is an ALGOL-W program which can be used to test and debug programs for the MCS-4 microcomputer. The system consists of an assembler which allows symbolic programming of the MCS-4 ROM and an interpreter which simulates the actions of the MCS-4. The program provides extensive diagnostic facilities for monitoring the actions of the MCS-4 program.

[Those instructions preceded by an asterisk (*) are 2 word instructions that occupy 2 successive locations in ROM]

MACHINE INSTRUCTIONS (Logic 1 = Low Voltage = Negative Voltage; Logic 0 = High Voltage = Ground)

MNEMONIC	OPR $D_3 D_2 D_1 D_0$	OPA $D_3 D_2 D_1 D_0$	DESCRIPTION OF OPERATION
NOP	0 0 0 0	0 0 0 0	No operation.
*JCN	0 0 0 1 $A_2 A_2 A_2 A_2$	$C_1 C_2 C_3 C_4$ $A_1 A_1 A_1 A_1$	Jump to ROM address $A_2 A_2 A_2 A_2$, $A_1 A_1 A_1 A_1$ (within the same ROM that contains this JCN instruction) if condition $C_1 C_2 C_3 C_4$[1] is true, otherwise skip (go to the next instruction in sequence).
*FIM	0 0 1 0 $D_2 D_2 D_2 D_2$	R R R 0 $O_1 D_1 D_1 D_1$	Fetch Immediate (direct) from ROM Data D_2, D_1 to index register pair location RRR.[2]
SRC	0 0 1 0	R R R 1	Send register control. Send the address (contents of index register pair RRR) to ROM and RAM at X_2 and X_3 time in the Instruction Cycle.
FIN	0 0 1 1	R R R 0	Fetch indirect from ROM. Send contents of index register pair location 0 out as an address. Data fetched is placed into register pair location RRR.
JIN	0 0 1 1	R R R 1	Jump indirect. Send contents of register pair RRR out as an address at A_1 and A_2 time in the Instruction Cycle.
*JUN	0 1 0 0 $A_2 A_2 A_2 A_2$	$A_3 A_3 A_3 A_3$ $A_1 A_1 A_1 A_1$	Jump unconditional to ROM address A_3, A_2, A_1.
*JMS	0 1 0 1 $A_2 A_2 A_2 A_2$	$A_3 A_3 A_3 A_3$ $A_1 A_1 A_1 A_1$	Jump to subroutine ROM address A_3, A_2, A_1, save old address. (Up 1 level in stack.)
INC	0 1 1 0	R R R R	Increment contents of register RRRR.[3]
*ISZ	0 1 1 1 $A_2 A_2 A_2 A_2$	R R R R $A_1 A_1 A_1 A_1$	Increment contents of register RRRR. Go to ROM address A_2, A_1 (within the same ROM that contains this ISZ instruction) if result ≠ 0, otherwise skip (go to the next instruction in sequence).
AOD	1 0 0 0	R R R R	Add contents of register RRRR to accumulator with carry.
SUB	1 0 0 1	R R R R	Subtract contents of register RRRR to accumulator with borrow.
LO	1 0 1 0	R R R R	Load contents of register RRRR to accumulator.
XCH	1 0 1 1	R R R R	Exchange contents of index register RRRR and accumulator.
BBL	1 1 0 0	D D D D	Branch back (down 1 level in stack) and load data DDDD to accumulator.
LDM	1 1 0 1	D D D D	Load data DDDD to accumulator.

Table III. MCS-4 Machine Instructions

INPUT/OUTPUT AND RAM INSTRUCTIONS

(The RAM's and ROM's operated on in the I/O and RAM instructions have been previously selected by the last SRC instruction executed.)

MNEMONIC	OPR $D_3\ D_2\ O_1\ D_0$	OPA $O_3\ D_2\ O_1\ D_0$	OESCRIPTION OF OPERATION
WRM	1 1 1 0	0 0 0 0	Write the contents of the accumulator into the previously selected RAM main memory character.
WMP	1 1 1 0	0 0 0 1	Write the contents of the accumulator into the previously selected RAM output port. (Output Lines)
WRR	1 1 1 0	0 0 1 0	Write the contents of the accumulator into the previously selected ROM output port. (I/O Lines)
WPM	1 1 1 0	0 0 1 1	Write the contents of the accumulator into the previously selected half byte of read/write program memory (for use with 4008/4009 only)
WR0[4]	1 1 1 0	0 1 0 0	Write the contents of the accumulator into the previously selected RAM status character 0
WR1[4]	1 1 1 0	0 1 0 1	Write the contents of the accumulator into the previously selected RAM status character 1.
WR2[4]	1 1 1 0	0 1 1 0	Write the contents of the accumulator into the previously selected RAM status character 2
WR3[4]	1 1 1 0	0 1 1 1	Write the contents of the accumulator into the previously selected RAM status character 3
SBM	1 1 1 0	1 0 0 0	Subtract the previously selected RAM main memory character from accumulator with borrow
RDM	1 1 1 0	1 0 0 1	Read the previously selected RAM main memory character into the accumulator.
ROR	1 1 1 0	1 0 1 0	Read the contents of the previously selected ROM input port into the accumulator. (I/O Lines)
AOM	1 1 1 0	1 0 1 1	Add the previously selected RAM main memory character to accumulator with carry.
RD0[4]	1 1 1 0	1 1 0 0	Read the previously selected RAM status character 0 into accumulator.
RO1[4]	1 1 1 0	1 1 0 1	Read the previously selected RAM status character 1 into accumulator.
RD2[4]	1 1 1 0	1 1 1 0	Read the previously selected RAM status character 2 into accumulator.
RO3[4]	1 1 1 0	1 1 1 1	Read the previously selected RAM status character 3 into accumulator.

Table IV. MCS-4 Input/Output Instructions

ACCUMULATOR GROUP INSTRUCTIONS

CLB	1 1 1 1	0 0 0 0	Clear both. (Accumulator and carry)
CLC	1 1 1 1	0 0 0 1	Clear carry.
IAC	1 1 1 1	0 0 1 0	Increment accumulator.
CMC	1 1 1 1	0 0 1 1	Complement carry.
CMA	1 1 1 1	0 1 0 0	Complement accumulator.
RAL	1 1 1 1	0 1 0 1	Rotate left. (Accumulator and carry)
RAR	1 1 1 1	0 1 1 0	Rotate right. (Accumulator and carry)
TCC	1 1 1 1	0 1 1 1	Transmit carry to accumulator and clear carry.
DAC	1 1 1 1	1 0 0 0	Decrement accumulator
TCS	1 1 1 1	1 0 0 1	Transfer carry subtract and clear carry.
STC	1 1 1 1	1 0 1 0	Set carry.
DAA	1 1 1 1	1 0 1 1	Decimal adjust accumulator.
KBP	1 1 1 1	1 1 0 0	Keyboard process. Converts the contents of the accumulator from a one out of four code to a binary code.
DCL	1 1 1 1	1 1 0 1	Designate command line.

NOTES: (1)The condition code is assigned as follows

$C_1 = 1$ Invert jump condition
$C_1 = 0$ Not invert jump condition
$C_2 = 1$ Jump if accumulator is zero
$C_3 = 1$ Jump if carry/link is a 1
$C_4 = 1$ Jump if test signal is a 0

(2)RRR is the address of 1 of 8 index register pairs in the CPU.

(3)RRRR is the address of 1 of 16 index registers in the CPU.

(4)Each RAM chip has 4 registers, each with twenty 4-bit characters subdivided into 16 main memory characters and 4 status characters. Chip number, RAM register and main memory character are addressed by an SRC instruction. For the selected chip and register, however, status character locations are selected by the instruction code (OPA).

Table V. MCS-4 Accumulator Instructions

There are two hardware prototyping systems available. Each system uses the electrically programmable and erasable ROM (PROM). The 1701 or 1702 PROM is programmed on the prototyping system and simulates the exact action of a programmed ROM. The SIM4-02 prototyping board consists of a CPU and is capable of programming and testing up to sixteen 1701 or 1702 PROMs. The 4004 CPU also controls sixteen RAMs together with TTL simulations of eight ROM output ports and eight ROM input ports. The SIM4-01 is designed as a prototype for small systems. This board contains provisions for up to four 1701s or four 1702s. It also provides up to four RAM output ports, four ROM ourput ports and four ROM input ports. For small quantity systems where the cost of designing a metal-masked ROM can not be justified, the 1701 or 1702 PROM can be used instead.

The recently announced INTELLEC-4 system makes available to the programmer a resident software monitor, assembler, PROM programming, and text editing capability.

4. Costs

The cost of developing a system that uses the MCS-4 Microcomputer can be divided into two areas: Hardware and software. The hardware costs have been shown to be small for the microcomputer. This is a list of the MCS-4 hardware costs:

	(1)	(over 100)	(over 1000)
CPU	$60	$30 each	$15 each
ROM	$60	$15 each	$ 5 each
RAM	$30	$15 each	$ 5 each

The software costs of programming the microcomputer are not really known. The assembler type language used to program the MCS-4 requires the programmer to keep track of the contents of all registers as the program steps through its set of instructions. Even one bit, such as the carry bit, can cause large errors to be produced if not accounted for. All indexing and transferring of data between the processor and the assigned locations in memory must be written into the program.

The task of writing a small program on the MCS-4 is fairly straight forward, however, as the computational complexity grows, the task of writing the program increases rapidly.

The sections following this discuss the problems encountered in programming the MCS-4 to handle the computational tasks of a large navigation system. Aids in software development were investigated and are also covered.

III. NAVIGATION SYSTEM

Air Navigation is the process of directing the movement of an aircraft from an initial point to a desired final point. A coordinate reference frame must be established and the initial point and the final point must be located in the reference coordinates. A Navigation System must provide timely coordinate measurement and computation of desired aircraft position.

The prime navigation problem years ago was to reach the desired destination. Today many aircraft wish to use the same airspace at the same time. This problem has required separation standards to be put into effect. Aircraft are assigned a block of space, measured by lateral, longitudinal, and vertical dimensions, which moves at the speed indicated in the pilot's flight plan. It is the pilot's responsibility to remain within this block of space. As more aircraft wish to use the same airspace, the separation standards will have to be reduced. The desired navigation system is one that gives a continuous, real time indication of where the aircraft is located. Navigation systems must be required to have greater accuracy, greater automation, and simplicity in operation and display.

The need for a navigation system to be reliable is a necessity. Equipment failure must not endanger the aircraft.

The reliability of a navigation system can not be measured solely by the fault-free operation of one piece of equipment. Reliability is a function of the operation of the total navigation system. In case of partial equipment failure, the navigation system must automatically switch to an alternate source of information. Therefore, decision making circuitry must also be a part of the navigation system to insure a continuous flow of accurate navigation information.

The following is a review of current navigation systems available. From this information, the possibility of integrating the MCS-4 microcomputer into a total navigation system will be investigated.

A. SELF-CONTAINED NAVIGATION SYSTEM

Only automated navigation systems can satisfy the requirement for continuous and accurate navigation information. The need is for a pictorial type of display which will give the pilot an accurate and immediate indication of the aircraft's present position. Economic constraints have encouraged that this navigation system be standard throughout the military and have a close commonality with civil aviation needs. Many of the military and civil aircraft in use today have space provided for this type of system.

The navigation systems in greatest use today are radio navigation systems. In order to keep the cost and complexity of airborne radio navigation equipment down, a comprehensive system of large and costly radio stations must be kept on

the ground. Some of the major systems in use today are listed in Table VI. When such a system is created, it is almost impossible to abolish it. The high initial cost of setting up a new system and the large number of aircraft equipped for and dependent on the system in use, make it hard for new systems based on new technology to become established.

Radio navigation systems have many limitations. Some of these limitations are due to the short distance of ground wave propagation, sky wave contamination, atmospheric noise, multipath effects, and site error which can result in ambiguous position fixes. Errors in radio navigation signals can not be predicted because they are a function of daily and seasonal changes in environment, temperature, ionosphere location, and local weather. Even the placement and type of radio used in the system is affected by political factors. These limitations, together with frequency interference between radio stations and the susceptability of radio navigation systems to jamming, all support the desirability of aircraft to have a self-contained navigation system.

Military aircraft navigation systems require world-wide flexibility with the ability to navigate without reliance on ground-based aids or the use of equipment susceptible to jamming. Civil aircraft that fly over oceanic or desert routes require a self-contained navigation system as its primary source of position fixing. The development of

	SYSTEM	FREQUENCY	TYPE	COVERAGE
1.	Decca Navigator	70-130 kHz	Hyperbolic Fix/Phase Difference	Short Range
2.	Loran A	2 MHz	Hyperbolic Fix/Pulse	Long Range
3.	Loran C	100 kHz	Hyperbolic Fix/Pulse	Very Long Range
4.	Omega	10-14 kHz	Hyperbolic Fix/Phase Difference	World Coverage
5.	Automatic Direction Finder ADF	300 kHz - 3 MHz	Bearing Only	Long Range
6.	VHF Omnidirectional Range VOR	108-118 MHz	Bearing Only	Short Range
7.	Tacan	960-1215 MHz	Bearing and Range	Short Range
8.	Distance-Measuring Equipment DME	960-1215 MHz	Range Only	Short Range

TABLE VI. RADIO NAVIGATION SYSTEMS

hovercraft that must cross large bodies of water without land based navigation aids and the need for cruise missiles that can independently maneuver over hostile territor all point to the need for self-contained navigation systems.

B. AIR DATA SYSTEM

An air-data system consists of aerodynamic and thermodynamic sensors inputting to a central air-data computer. The sensors measure the characteristics of the air surrounding the vehicle and input them to the computer. The computer calculates flight parameters such as true airspeed, free-stream outside-air temperature, and Mach number. The sensors required in this system are angle-of-attack vane, static pressure source, pitot tube, and total temperature probe.

In order to use the air-data computations for dead-reckoning, the attitude and heading of the vehicle must be supplied from some external source onboard. This can be in the form of a simple directional and vertical gyroscope or a stable-platform configuration such as an Inertial Navigation System. This information together with information on the velocity and direction of the wind allows the navigator to determine the position of the vehicle by extrapolating from a previously known fixed position.

C. DOPPLER NAVIGATION SYSTEM

A Doppler Navigation system radiates a pattern of beams to the surface of the earth and receives the reflection of

this energy back. The difference between the frequency of the signals transmitted and the frequency of the signals received is called the Doppler Effect and can be used to compute the vehicle's velocity along each beam. The frequency shift in each of the beams is detected and used by the system computer to calculate the distance traveled along and across the vehicle's true heading.

The advantages of the Doppler Navigation System, as listed by Kayton and Fried [Ref. 21], are as follows:

1.) It provides continuous velocity and position with respect to the ground.

2.) It is completely self contained.

3.) Its average-velocity information is extremely accurate.

4.) Doppler-Radar information is obtainable anywhere on earth, including over oceans.

5.) It is an all-weather system.

6.) Doppler Radars are amenable to high-reliability all-solid-state design because of their low radiated power.

7.) The Doppler Navigator does not require preflight alignment or warmup.

8.) It radiates at microwave frequencies.

The disadvantages of the Doppler Navigation System are:

1.) It is dependent on an external direction sensor for azimuth information.

2.) The position information derived from a Doppler Navigator degrades as the distance traveled increases.

3.) The short-term or instantaneous velocity information is not as accurate as the average velocity.

D. INERTIAL NAVIGATION SYSTEM

Airborne inertial navigation systems, hereafter referred to as INS, operate on the principle that every time the vehicle changes speed or direction it is said to experience an acceleration. By measuring this acceleration, the velocity and distance traveled by the aircraft can be found by integrations.

The INS is a gyro-stabilized platform with accelerometers mounted so that accelerations of the vehicle are measured in the north/south, east/west and vertical directions. The total system is mounted on gimbals to allow the vehicle to rotate without disturbing the attitude of the stable platform.

The INS must be kep tangent to the earth's surface at all times or the accelerometers will experience an acceleration error due to gravity. This problem is solved by constructing a theoretical pendulum in the system with its bob at the center of the earth and its other end on the surface of the earth. The aligning of the INS before flight will bring this pendulum to rest making it impossible to be set in motion by any force at the surface of the earth. Any small error in the alignment causes a sinusoidal error, called the Schuler Pendulum effect to be introduced in all inertials. This effect is periodic with a period of 84 minutes.

The Inertial Navigation System has the following advantages as listed by Kayton and Fried [Ref. 21]:

1.) Its indications of position and velocity are instantaneous and continuous.

2.) It is completely self-contained.

3.) It is nonradiating and nonjammable.

4.) Navigation information is obtainable anywhere on the earth.

5.) It is an all-weather system.

6.) Navigation information is independent of the vehicle's maneuvers.

7.) It directly provides outputs of position, ground-speed, and vertical position.

8.) It is the most accurate source of roll, pitch, and attitude of the vehicle.

The Inertial has the following disadvantages:

1.) The position and velocity information degrades with time.

2.) The equipment is expensive and relatively difficult to maintain and service.

3.) The INS must be initially aligned.

E. POSITION FIXING SYSTEMS

Position fixing is the determination of the vehicle's position without reference to any former position. Short range radio systems such as ADF, TACAN, DECCA, VOR, and LORAN A are not considered applicable for long range navigation.

1. Long Range Radio Navigation Systems

The two primary long range Radio Navigation Systems in use today are LORAN C and OMEGA. LORAN C and OMEGA are

both hyperbolic fixing systems. The vehicle's fix position is obtained by measuring the relative distance from two or more stations of known location.

a. The LORAN C Network

The LORAN C network is comprised of master ground station and at least two associated slave ground stations. Pulsed transmissions radiated from the master station are received at the slave stations; each slave station then transmits similar groups of pulses, synchronized accurately with the signals received from the master station. Receivers on-board the vehicle measure the time differences between the master and slave's transmissions which allows an automatic fix computation.

The following are advantages and capabilities of LORAN C:

1.) Currently operational with minaturized receivers developed.

2.) Range up to 1500 nm.

3.) Continuous fixing provided.

4.) Passive at the receivers.

5.) All weather capability

6.) Accurate position fix to 200 ft.

7.) Total cost of on-board receiving equipment approximately $10,000.

The disadvantages of LORAN C are:

1.) Position fix in hyperbolic coordinates.

2.) Dependent on fixed ground stations.

3.) Susceptible to atmospheric noise and sky-wave contamination.

4.) Susceptible to jamming.

5.) Total world coverage not available.

b. The OMEGA System

OMEGA is an earth reference hyperbolic phase-matching navigation system operating at 10.2, 11.33, and 13.6 kHz. The system is designed to provide world-wide coverage with accuracies better than one nautical mile. The position of the vehicle is determined by measuring the relative phase of the signals transmitted from two or more stations.

OMEGA has the following advantages:

1.) Range 8000 nm.

2.) Coverage is global with just eight stations.

3.) Continuous fixing available.

4.) Passive receiver on the vehicle.

5.) All weather capability.

The disadvantages of OMEGA are:

1.) Position fix is in hyperbolic coordinates.

2.) Dependent on fixed ground stations.

3.) Susceptible to Ionospheric disturbances.

4.) Susceptible to jamming.

2. The Satellite Navigation System

Satellite Navigation is a method of fixing the vehicle's position from data obtained from an artificial

earth satellite. The position of an artificial satellite can be predicted by the orbit it is traveling. A relative fix can be made from the satellite by measuring the elevation angle, the azimuth angle, range to the satellite, or the rate of change of altitude, azimuth or range. If a single variable is measured, successive measurements are required.

There are three basic types of satellite navigation: Doppler Systems, which measure rate of change of slant range; angle measuring systems, which measure elevation angle; and ranging systems, which measure the slant range.

Satellite Navigation Systems have the following advantages:

1.) All-weather position fixing.

2.) Communication service to the vehicle while in flight.

3.) Worldwide service.

4.) Capable of accuracies up to 200 ft.

5.) Can provide vehicle with velocity and heading information as well as position information.

6.) Passive at the receiver.

The disadvantages of a Satellite Navigation System are as follows:

1.) Special receiving equipment required including antennas with vertical patterns.

2.) The satellite orbit must be predicted and continuously up-dated.

3.) The satellite's signal may be susceptible to jamming.

4.) The satellite signal is susceptible to ionospheric refraction.

5.) Navigation errors are introduced by drifts in the frequencies of the satellite transmitters.

3. The Terrain Mapping System

A Terrain Mapping System operates on the principle that the geographic location of any place on the land surface of the earth is uniquely defined by the vertical contours or topography of the surrounding area. The Terrain Mapping System measured the vertical contour of the terrain along its flight path, using a radar altimeter to measure clearance above the terrain and a reference altitude of the vehicle. By subtracting instantaneous radar measured altitude from the reference altitude, the Terrain Mapping System determines the terrain contour. The system then searches its computer memory to find a stored terrain contour, whose coordinates are known, which closely matches the measured one. This serves to fix the vehicle's position.

The advantages of the Terrain Mapping System are as follows:

1.) It is an all-weather system.

2.) It is completely self-contained.

3.) It is inherently resistant to jamming.

4.) It is accurate up to 50 ft.

5.) Accuracy demands imposed on the radar altimeter are not excessive.

6.) It is unaffected by man-made changes in topography.

The Terrain Mapping System has the following disadvantages:

1.) The terrain contour to be flown over must be previously mapped.

2.) To increase the accuracy of the system, there must be an increase in the computer memory capacity.

3.) A dead-reckoning system must also be carried so the Terrain Mapping System knows where to search for a contour match.

4.) It can not be used to obtain position fixes over the ocean or large bodies of water.

F. THE INTEGRATED SYSTEM

The basic parameters of the navigation systems currently available have been reviewed. In order to select the proper long range navigation system for military use, a minimum set of system requirements was established:

Range	3000 nm.
Accuracy	5 nm.
Velocity	0 - 2000 kts.
Coverage	Worldwide
Weather Restriction	All-weather capability
Flexibility	Capable of self-testing and switching mode when needed
Equipment	Non-jammable, completely self-contained

A review of the previously mentioned navigation systems indicates that no single system available is capable or flexible enough to meet the system requirements. The Doppler Navigation System is self-contained and provides very accurate average velocity measurements, however, the system can not sense short term velocity fluctuations and must depend on an external heading source. The Inertial Navigation

System is self-contained and provides excellent short-term position, velocity, and heading information. The Inertial Navigation System, however, has a long-term error build-up due to the inherent Schuler Cycle in the system.

In order to meet the system requirements, an integrated-navigation-system was chosen. An integrated-navigation-system feeds the outputs of several navigation systems into a central computer which then provides a single more accurate output. The central computer provides decision making ability to test the accuracy of the individual navigation systems. By inputting fixed positions to the central computer at regular intervals, the computer can determine system drift parameters and provide the proper system bias velocity.

The navigation system developed in this paper is an Inertial/Doppler system integrated by an MCS-4 microcomputer. The microcomputer combines the short-term accuracy of the Inertial with the long-term accuracy of the Doppler to obtain the most probable position of the vehicle. The micro-computer is programmed to provide decision making flexibility to ensure the outputs remain accurate during partial system failure. The wind influencing the vehicle is continually computed and updated in the MCS-4 memory so that the computer can automatically switch to an air-data mode of operation in case of Inertial and Doppler failure.

The navigation equations used to compute the vehicle's position from the inputted data are covered in the next section.

IV. NAVIGATION EQUATIONS

The navigation equations necessary to calculate the position of the vehicle relative to the earth are presented in this section. The mathematical model presented takes the dead-reckoning outputs of the Inertial, Doppler, and Air-Mass Systems and extrapolates the present position of the vehicle from the last known position. Direct position data can be inputted into the algorithm and is used to update the position of the vehicle. The system drift error is computed by comparing the known position with the dead-reckoned computed position.

In describing the equations in detail, a graphical picture of each equation's function is included. The graphical picture represents each equation as a black box with specified inputs which produce the desired outputs. A complete description of graph theory as applied to this paper is given in Section V-A.

A. THE EARTH MODEL AND THE COORDINATE REFERENCE FRAME

The first step in developing the Navigation equations is to prepare a mathematical model of the earth. The earth can be approximated as a sphere of radius R, the nominal equatorial radius. However, this approximation would produce an unacceptable error as great as 3 miles for every 1,000 nm. traveled. This is due to the flattening of the earth at the poles and the bulging of the earth at the equator.

A more exact model of the earth is the reference spheroid where the earth is assumed to be an ellipsoid of revolution with the semi-major axis R, the nominal equatorial radius, and the semi-minor axis of radius P, the polar radius. Accuracy to within 30 ft. for every 1,000 nm. traveled is possible with this model. The determination of angels, however, in this system is very difficult.

The model chosen in this system is a combination of the two previously stated models. All angles are determined from the spherical model thus allowing the convenient latitude/longitude coordinate reference frame to be used to map each point on the real earth to the earth model. The reference spheroid is then used to determine arc lengths that more closely represent the actual shape of the earth. The arc lengths are applied to the determined latitude and longitude in the form of conversion constants. This method of computation is very simple and has a maximum error of 300 ft. for every 1000 nm. traveled.

B. DERIVATION OF CONVERSION CONSTANTS

The international nautical mile equals 6076.1033 ft. One minute of arc length measured at the equator equals 6087.08 ft. The longitude conversion constant, KLO, is used to relate nautical measurement to the actual geographical measurement:

$$KLO = \frac{\text{feet/degree}}{\text{feet/nautical mile}}$$

$$KLO = \frac{6087.08 \text{ feet per minute/60 minutes per degree}}{6076.1033 \text{ feet/1 nautical mile}}$$

$$KLO = 60.1084 \text{ nautical miles/degree}$$

The average length of a minute of arc length on a meridian is 6076.82 ft. To determine the latitude conversion constant, KLA, which is also used to relate nautical measurement to the actual geographical measurement, the following equation is used:

$$KLA = \frac{\text{feet/degree}}{\text{feet/nautical mile}}$$

$$KLA = \frac{6076.82 \text{ feet per minute/60 minutes per degree}}{6076.1033 \text{ feet/1 nautical mile}}$$

$$KLA = 60.0071 \text{ nautical miles/degree}$$

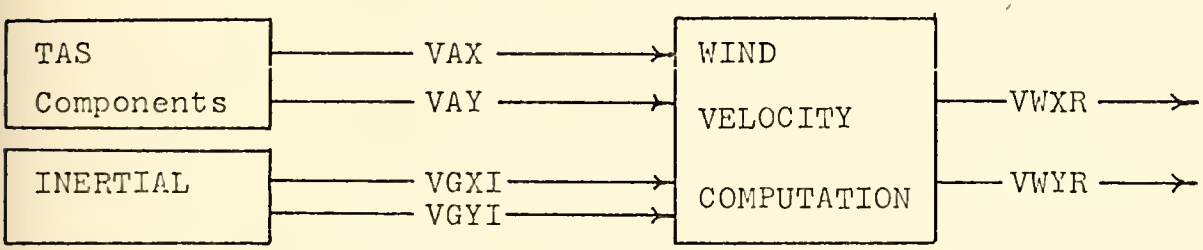
The distance traveled in nautical miles, east-west and north-south, is divided by the conversion constants resulting in distance traveled measurements in degrees, longitude and latitude.

C. WIND CALCULATIONS

The wind effecting the motion of the vehicle can be obtained by comparing the north-south and east-west vector

components of the true airspeed, as computed by the air-mass system, with the vector components of ground speed, as computed by the inertial system. The calculated wind components must be continually updated so the current wind components stored in the computers memory are the most current. In case of failure of the Inertial and Doppler Systems, the computer will be capable of computing the dead-reckoning position from the outputs of the air-mass system and the last stored wind components.

The wind velocity components are computed from the following equations:

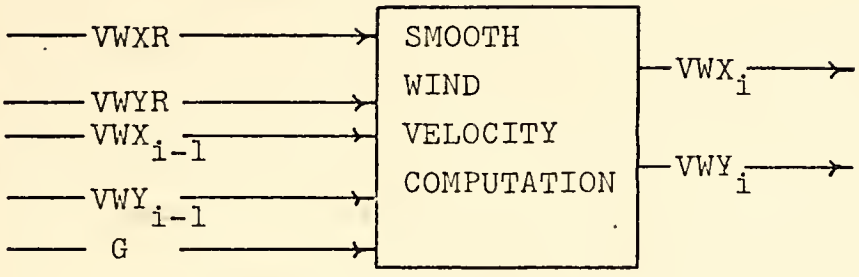


$$VWXR = VGXI - VAX$$

$$VWYR = VGYI - VAY$$

The wind velocity components computed are subject to gusts, maneuvers, and other short term disturbances. In order to decrease the effects of random fluctuations in the wind velocity, the new computed wind velocity is averaged with the previously computed wind velocity in a smoothing

routine. The smooth wind velocity is derived from the following equations:



$$VWX = G\ VWX + (1-G)\ VWXR$$

$$VWY = G\ VWY + (1-G)\ VWYR$$

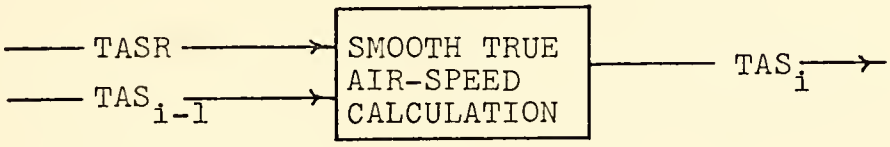
D. DEAD-RECKONING CALCULATION

Dead-reckoning is a means of navigation in the absence of position fixes. The vehicle's position can be estimated by measurements of the groundspeed components. The groundspeed components are integrated over a given time interval to give the distance traveled in that time interval. The estimated position of the vehicle is the summation of the distance increments traveled from the last known fixed position.

1. Air Mass Mode

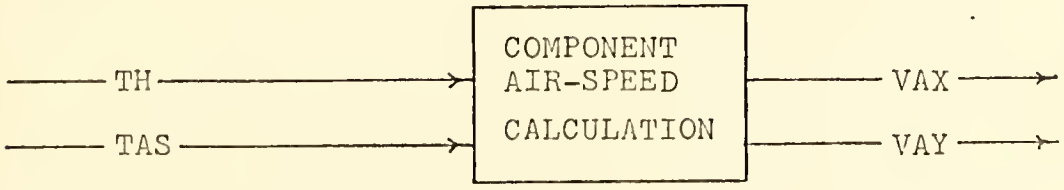
The air mass mode of operation is a back up for the two primary dead-reckoning navigation systems, the Inertial and Doppler. The components of airspeed, true heading, and stored wind information are combined to output distance increments in the north-south and east-west directions.

The true air-speed inputs are influenced by random fluctuations in the Pitot-Static System. In order not to influence the navigation cycle by these fluctuations, the inputted true air-speed is averaged with the previously stored true air-speed. The smooth true air-speed is derived from the following equation:



$$TAS = \frac{TAS + TASR}{2}$$

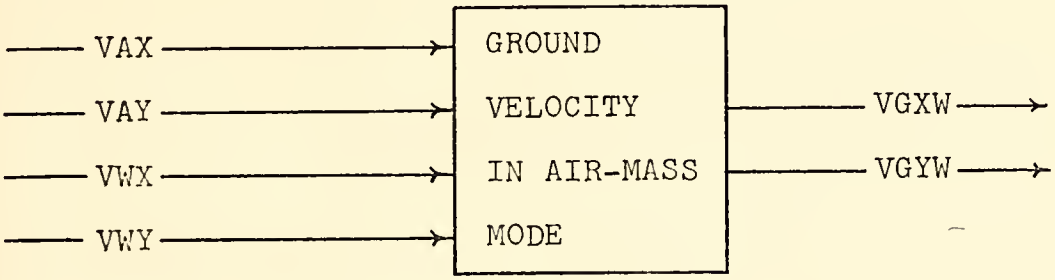
The north-south and east-west components of air-speed are calculated using the previously calculated true-air-speed and the true heading inputted from the INS. In case of INS failure, magnetic heading may be inputted from the flux valve together with the magnetic variations applicable set in by hand. The air-speed components are calculated as follows:



$$VAX = TAS \sin (TH)$$

$$VAY = TAS \cos (TH)$$

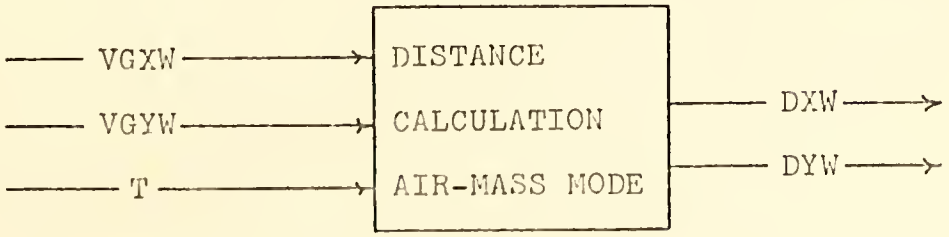
The air-speed components represent the velocity and direction of the vehicle through the air-mass. The ground track of the vehicle can be found by adding vectorally the motion of the air-mass, wind velocity, to the air-speed as shown in Figure 3. The equations for the ground velocity in the air-mass mode are as follows:



$$VGXW = VWY + VAX$$

$$VGYW = VWY + VAY$$

Lastly, the ground distance traveled by the vehicle during one navigation cycle of the computer can be determined by integrating the ground velocity over the time interval of the navigation cycle.



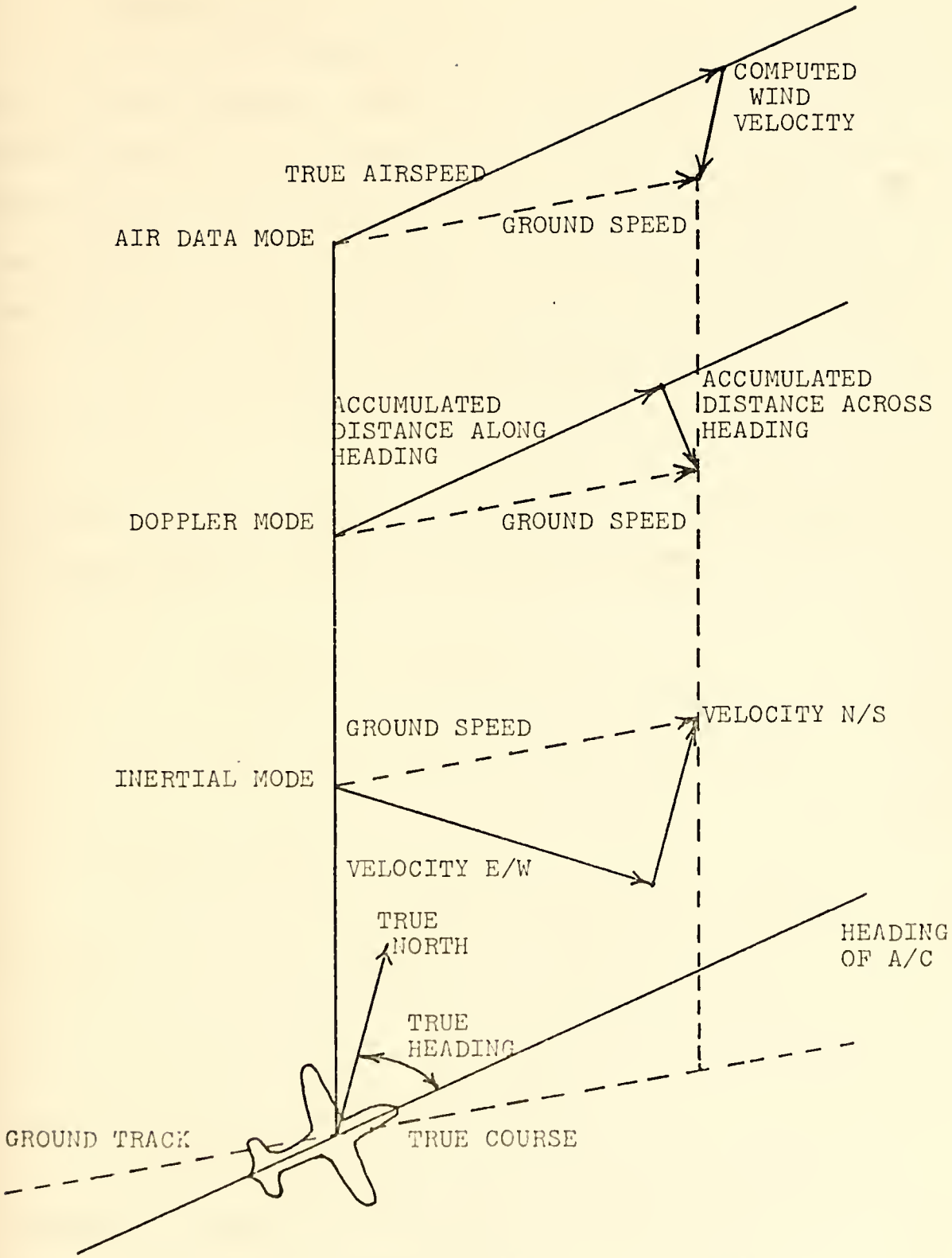
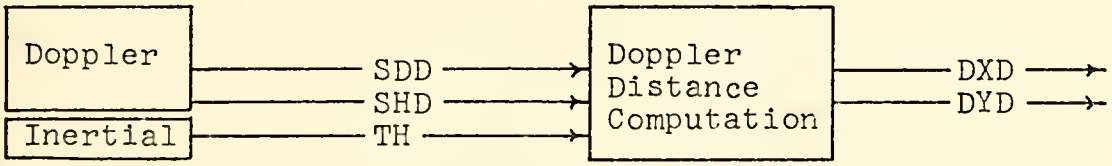


Figure 3. NAVIGATION VECTOR DIAGRAM

2. Doppler-Mode

The Doppler System obtains very accurate average velocity measurements and converts them to distance measurements along and across the vehicle's true heading. These measurements along with true heading, from an external source, can be used to calculate the distance traveled by the vehicle as shown in Figure 3.

The dead reckoning position increments computed from the doppler are derived from the following equations:



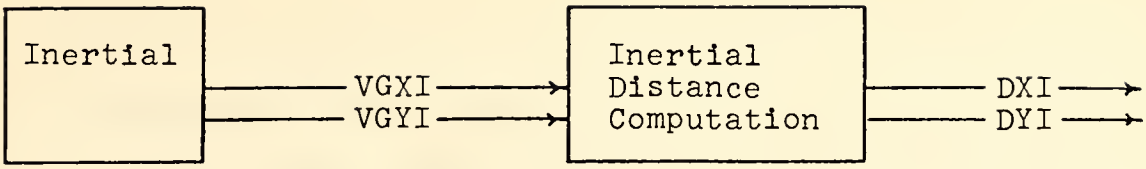
$$DXD = SDD \times COS\ (TH) + SHD \times (TH)$$

$$DYD = SHD \times COS\ (TH) - SDD \times SIN\ (TH)$$

3. Inertial Mode

The Inertial Navigation System, (INS), provides excellent short-term velocity and heading information. The outputs of the INS are the velocity components of the vehicle north-south and east-west. The ground track of the vehicle can be determined as shown in Figure 3.

The dead reckoning position increments computed from the inertial are derived from the following equations:

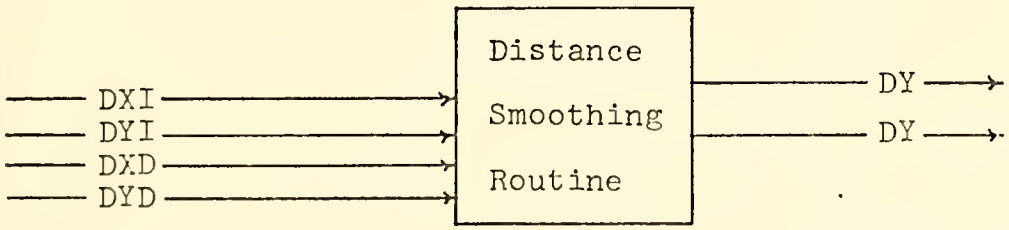


$$DXI = T \cdot VGXI$$

$$DYI = T \cdot VGYI$$

4. Integrated System Mode

To take advantage of the extremely accurate average-distance measurement of the Doppler and the precise short-term distance measurement of the Inertial, the distances measurement by each is averaged in a smoothing routine to give the most accurate dead-reckoning distance measurements possible. The smoothing factor, G, used in the smoothing equation is set depending on the accuracy of each system.



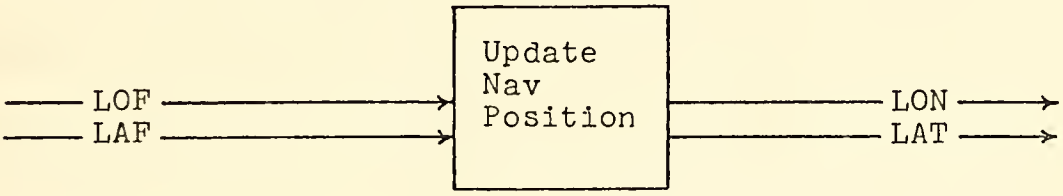
$$DX = G \times DXI + (1-G) \cdot DXD$$

$$DY = G \times DYI + (1-G) \cdot DYD$$

E. POSITION CALCULATION

1. Geographic Fix Position

Any time the vehicle's position is fixed, the updated fixed position will be loaded into the computer's memory along with the time the fix was taken. The dead-reckoning position coordinates at the time of the fix will be up-dated to the new position coordinates and the computer will continue to dead-reckon the vehicle's position from the last inputted fix.

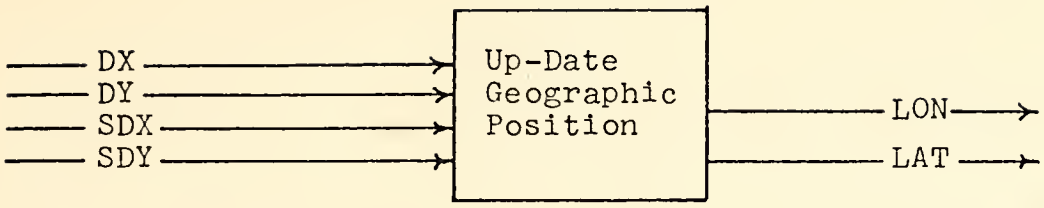


$$LON = LOF$$

$$LAT = LAF$$

2. Up-Date Geographic Position

The initial position of the vehicle is loaded into the computer memory before launch. The geographic position is up-dated each time a fix is taken. The geographic position between fixes is updated by dead-reckoning distance computations from the Doppler and Inertial. The zero subscript indicates the vehicle's previous geographic position. The computed system drift increments are also inputted to account for system drift errors in the dead-reckoning computation.



$$LAT = LAT_o + \frac{DY+SDY}{KLA}$$

$$LON = LON_o + \frac{DX + SDX}{KLO \times COS(LAT)}$$

3. System Drift Error

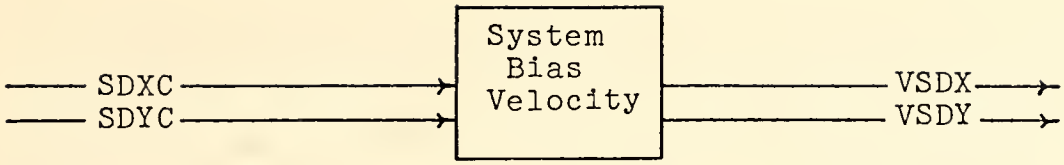
Each time a new position fix is taken, the fixed position is compared with the calculated dead-reckoned position. From this information, the amount of drift in the dead-reckoning computations between fixes is determined.

$$SDXC = (LOF-LON) \times KLO \times COS\ (LAT)$$

$$SDYC = (LAF-LAT) \times KLA$$

4. System Bias Velocity

The system drift error computed from the previous fix is used to update the system bias velocity. The system bias velocity is used to cancel the system drift error.

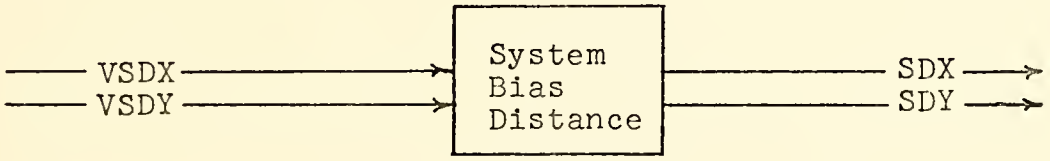


$$VSDX = VSDX + (SDXC \times TF)$$

$$VSDY = VSDY + (SDYC \times TF)$$

5. System Bias Distance

The system bias distance is computed by integrating the system bias velocity over the time of one nav cycle. The system bias distance is applied to the up-date geographic position to cancel the effects of the system drift.



$$SDX = VSDX \times T$$

$$SDY = VSDY \times T$$

The navigation program combines these equations as described in the Navigation Functional Flow Chart, Figure 5.

V. PROGRAMMING THE MICROCOMPUTER FOR NAVIGATION

The Navigation equations discussed in Section IV described how to process the outputs of the Inertial, Doppler, and Air-Mass Systems in order to calculate the current position of the vehicle. In order to make a working system, the Navigation equations had to be programmed into the software of the MCS-4 microcomputer.

The program was designed to incorporate the following principles and requirements:

1.) The total program had to be written in the assembly language of the MCS-4 microcomputer.

2.) The total run time of one navigation cycle could not exceed 200 milliseconds.

3.) The design of the program should be modular to facilitate the addition or deletion of new code.

4.) The program should consist of an executive routine that calls upon various subroutines as indicated on a process graph of the navigation program.

5.) The design allows for substitution or addition of new functions, routines, and/or programs in a straight forward manner.

6.) For each built-in function, a single subroutine is called to perform the calculation.

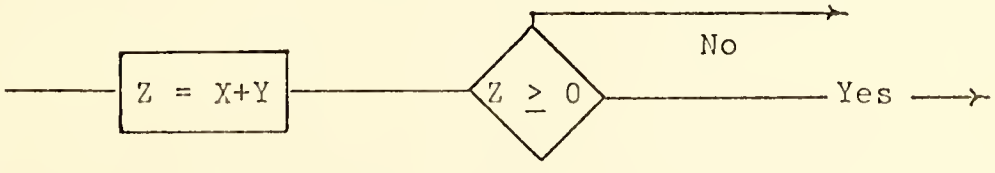
The instruction set of the MCS-4 microcomputer, as described in Section II.C., contains only addition and subtraction of two four digit binary numbers as the fundamental arithmetic operation. In order to make the detailed programming easier, a so-called "process graph" method was used to develop the program.

A. GRAPH THEORY

Programming the navigation equations would be an easy task in a higher level language such as FORTRAN. To program the same equations in an assembly language requires a detailed knowledge of the current contents of all index registers and memory locations. The status of even one single bit such as the carry bit can not be left unaccounted. The requirements of keeping minute details in mind led to the development of a graphical means of representing the microcomputer program.

Graph theory provides a simple and powerful tool for constructing mathematical models of discrete arrangements of objects. The process graph consists of vertices which are pairwise connected by a directed line.

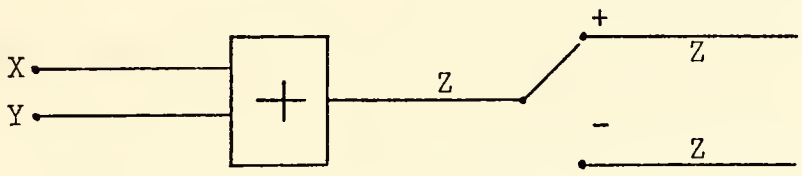
The best means to illustrate the concept of graph theory applied to microprogramming is through an example. Consider the problem of adding two numbers together which reside at given locations in memory and whose sum is to be located in a specified location in memory depending on its sign. This process can be described by the following flowchart:



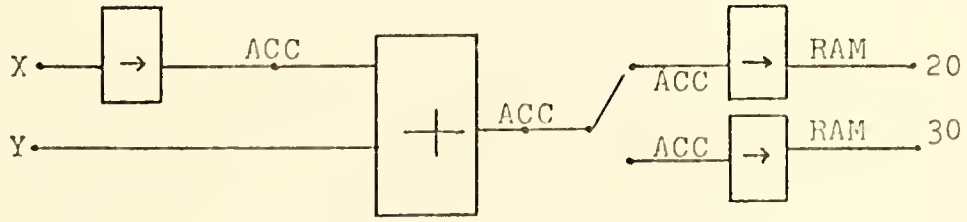
This notation does not give much information about how the processor is to accomplish this process, how much memory is

required by the program, the input and output variables, and how much time is needed to carry out the process.

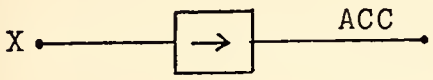
The same example written as a process graph could be as follows:



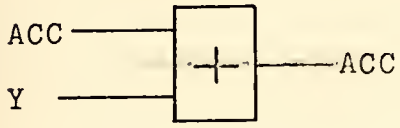
The process graph indicates that the variables X and Y are inputs to the function add and Z is the output of that function. A test is made of the output, Z, to decide where Z will be located. The amount of time and memory involved in this computation can be easily established by referencing the known memory space and time required for each operation involved. The depth to which the programmer draws the required graph for his computation is a function of the capability of the operations he has available to him together with his own preferences. The same example discussed could be graphed in more depth, as follows:



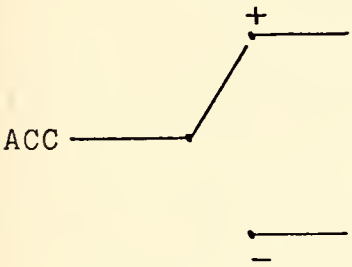
Another programmer reading this graph only needs to know the following in order to write the required program:



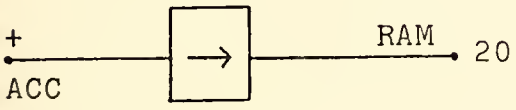
Load the variable X into the accumulator



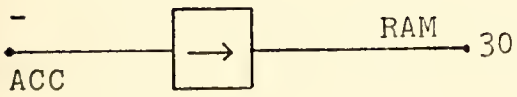
Add the variable Y to the contents of the accumulator



Test the sign of the contents of the accumulator



If the sign of the accumulator is positive, store the contents of the accumulator into RAM location (2,0)



If the sign of the accumulator is negative, store the contents of the accumulator into RAM location (3,0).

The detailed process graphs used in developing the navigation program were a great aid in the design, analysis and documentation of the program. The following is a list of

the problems that could be analysed directly from the process graph:

1.) What portions of the problem are sequential.

2.) What portions of the problem may be processed in parallel by independently operating processors.

3.) Where and how intercommunication may take place when a multiplicity of processors is used.

4.) What capacity must the processor possess before it can be successfully used to solve the problem.

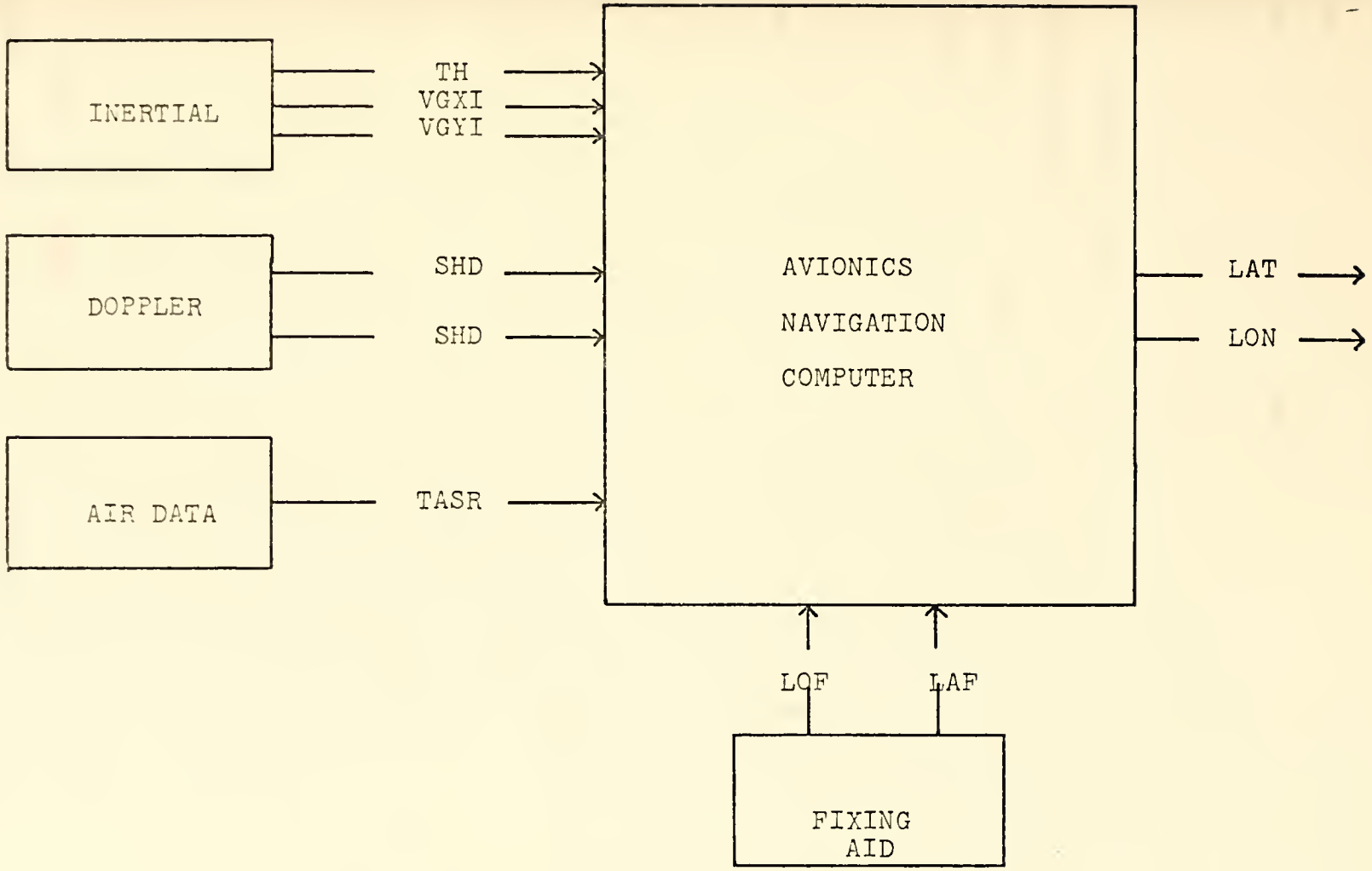
5.) What is the maximum allowable time for the operation of each subprogram.

6.) What is the memory space requirement for each subroutine.

B. DEVELOPMENT OF NAVIGATION PROCESS GRAPHS

The MCS-4 microcomputer was first thought of as a black box taking inputs from the Inertial, Doppler, Air-Mass, and Position Fixing Systems and outputting the vehicles current position in latitude and longitude. The graphical representation of this system is shown in Figure 4.

Figure 4 represents the total navigation system. The Inertial, Doppler, Air-Mass, and Position Fixing Systems provide input to the system as described in Section IV. The microcomputer program was required to complete the system. The programming was accomplished by continually breaking down the functional operations into smaller and smaller parts until the operations were simple enough to be directly written in the MCS-4 machine language. The functional process



NAVIGATION SYSTEM BLOCK DIAGRAM

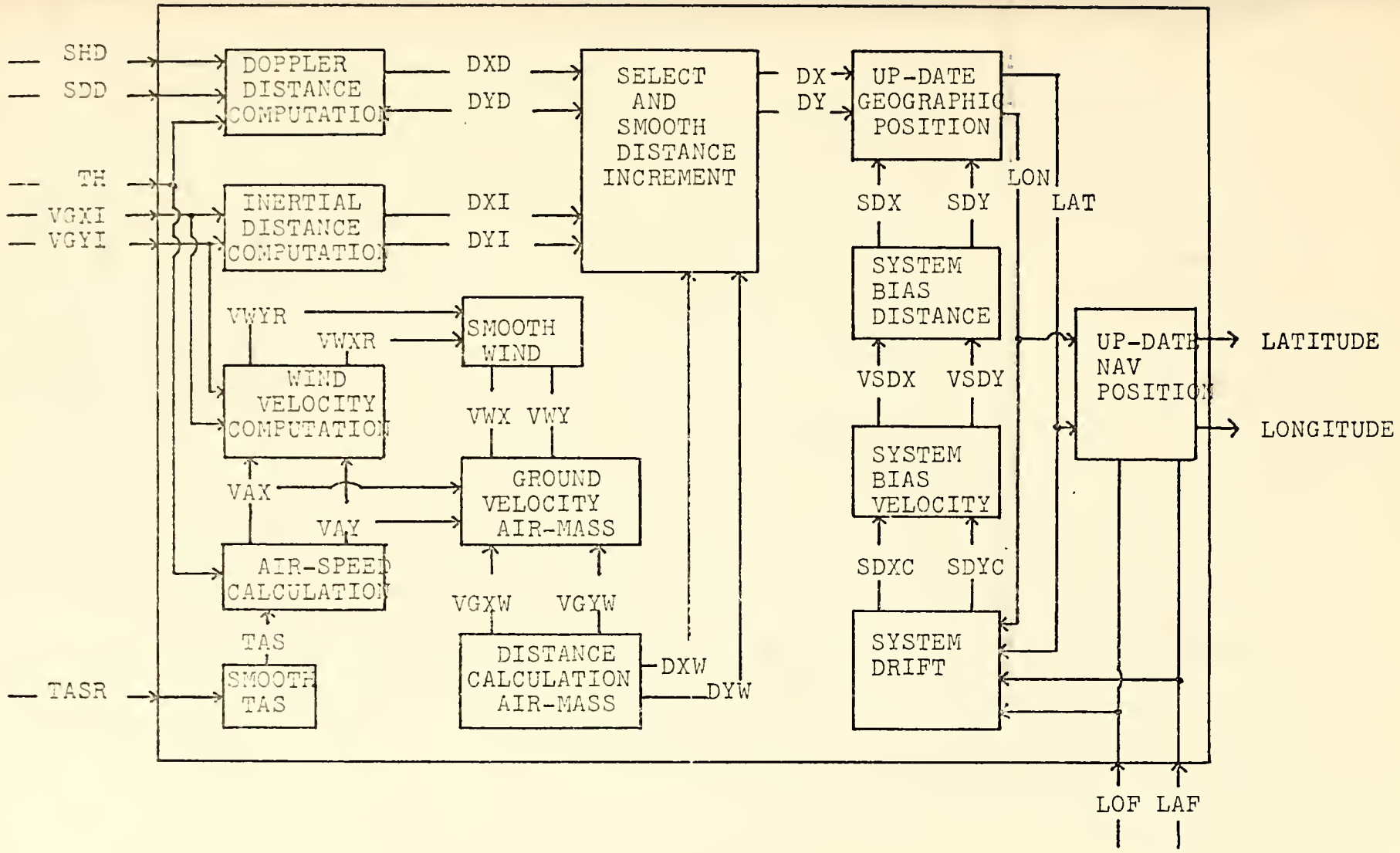
Figure 4

graph, Figure 5, represents how the microcomputer program was initially broken down into the fundamental navigation equations as described in Section V.

The functional process graph was analyzed to define the flow of the input variables and to determine the feasibility of a multi-processor system in order to shorten the required computational time. It was noted that the program could be broken into two parts, the calculation of the distance increments, and the calculation of the latitude and longitude from the distance increments. The analysis demonstrated that the computation time could be nearly halved by having two microcomputers working simultaneously to produce the desired outputs. The system was designed to have one microcomputer receive the given inputs and compute the distance increments traveled, while at the same time, the second microcomputer computes the latitude and longitude from the previously calculated distance increments.

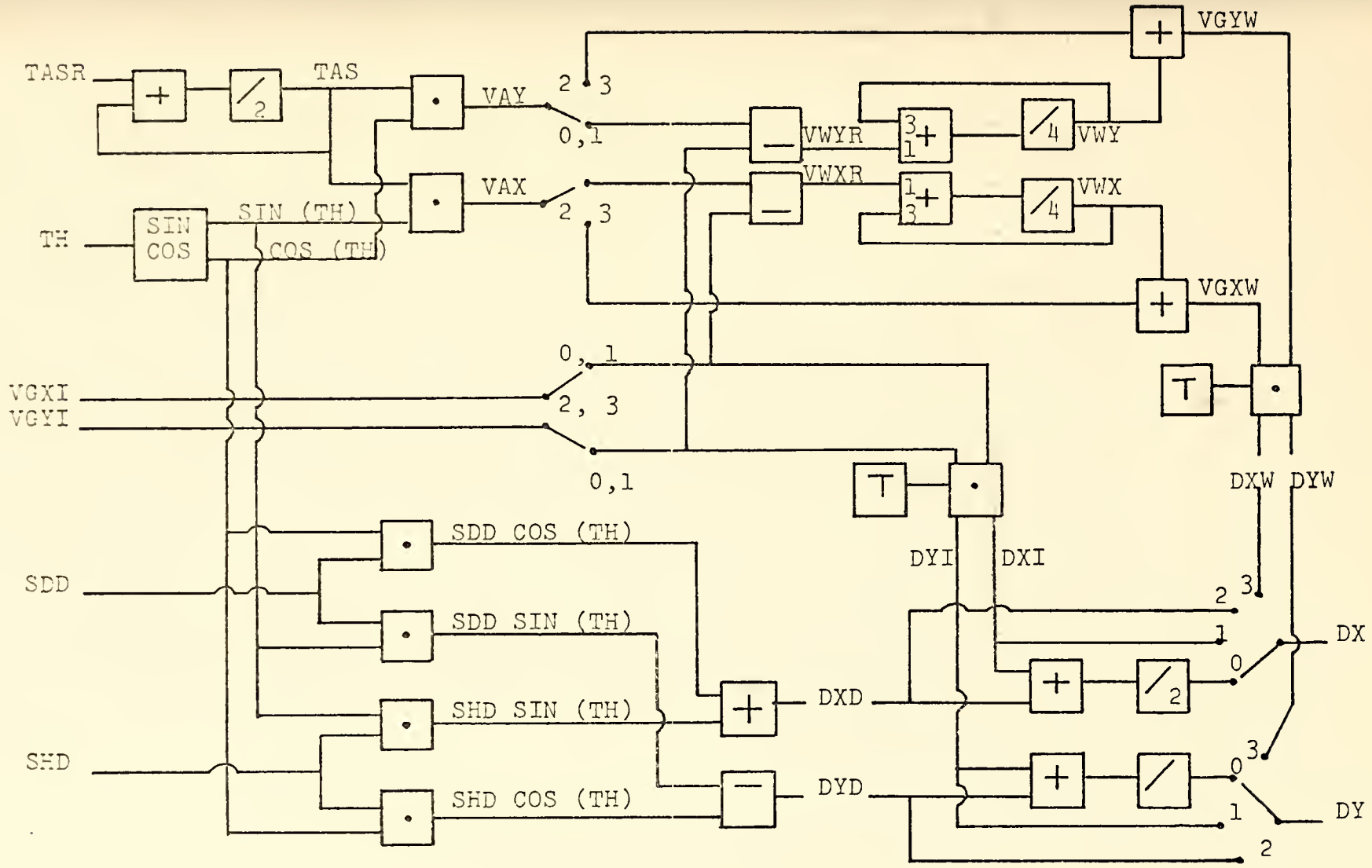
The functional process graph was then broken down into the operational process graphs for each microcomputer, Figure 6, and Figure 7. The operational process graphs represented the desired program for each microcomputer. The program described by the operational process graph, Figure 6, was written to investigate the time and effort required to develop the required software.

The program was written in a modular form as required. Each operation represented in the operational process graph



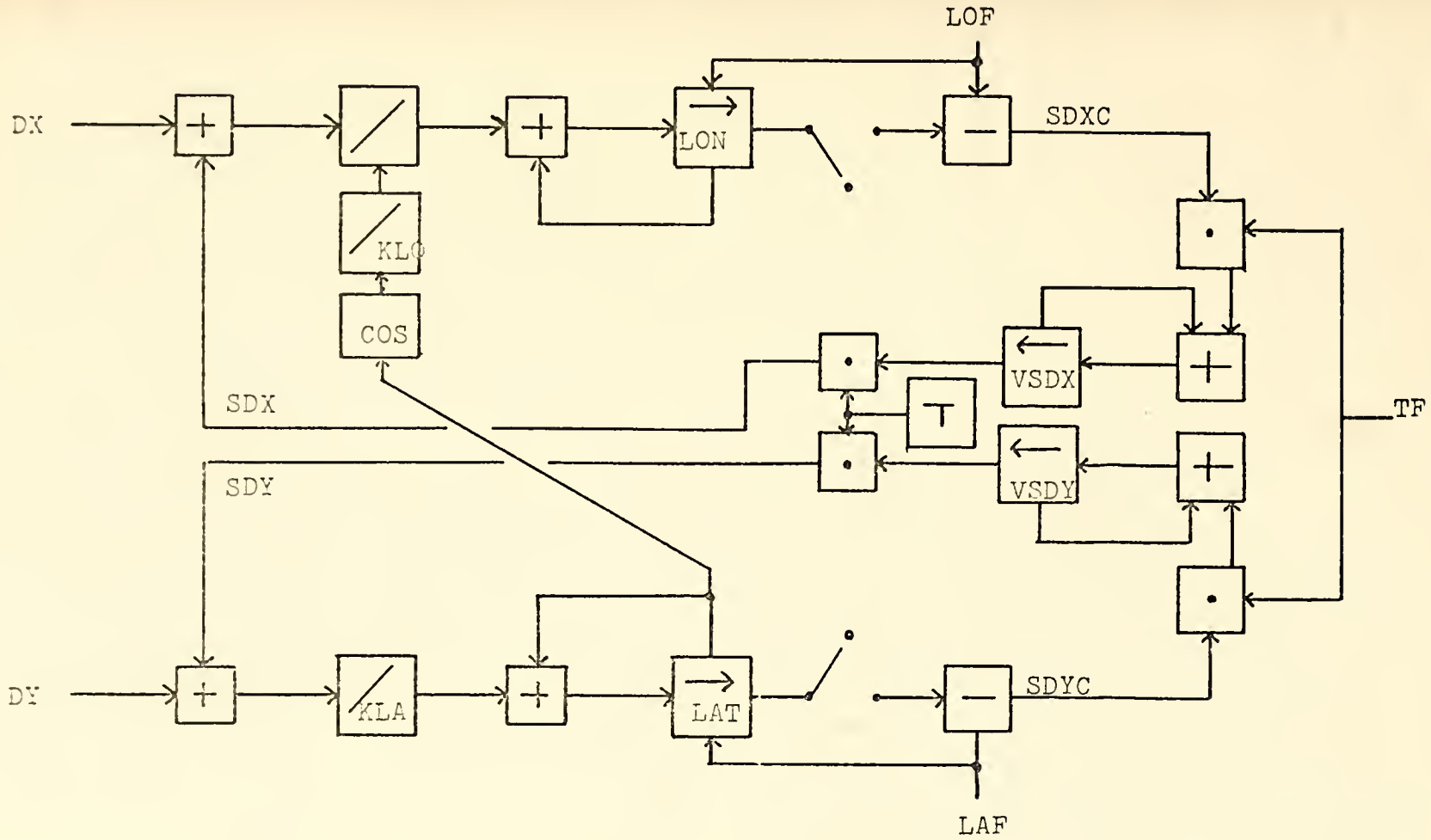
NAVIGATION FUNCTIONAL FLOW CHART

Figure 5



OPERATIONAL PROCESS GRAPH

Figure 6



NAVIGATION POSITION PROCESS GRAPH

Figure 7

was written as a separate subroutine. An executive routine was then developed to call each subroutine in the proper order and account for each variable as required by the operational process graph.

C. PROGRAM ANALYSIS

A program analysis was developed to define those problem areas that had to be solved before programming the microcomputer. The areas that were investigated were computational speed, memory space available, and accuracy required.

The operational process graph, Figure 6, represented the tasks to be accomplished. The type of operations and number of operations that were required are listed as follows:

Operations	Times Called
Multiply	12
Cosine	1
Sine	1
Addition	12
Subtraction	3
Division by two	7
Total	36

The operational process graph was used to determine the speed limitations on each operation programmed. The total navigation cycle was limited to 200 milliseconds. Since the number of operations required to be performed differed depending on what sensors were operational, the critical path of

the operational process graph was determined. The critical path occurred when the Inertial and Doppler were both operational, Figure 8. The operations involved in the critical path were ten multiplies, Cosine and Sine calculations, ten additions, three subtractions, and seven divisions by two. The total computational time for the critical path was limited to 200 milliseconds.

The critical operations which had to be developed were the Multiply, Cosine, and Sine subroutines. Investigating previous work on the MCS-4 indicated that previously programmed multiply, Cosine, and Sine routines were requiring 50 msec., 650 msec., and 750 msec. respectively. In order to program the microcomputer for navigation, the development of the following routines were required with the following constraints:

Multiply	less than 200 msec.
Cosine	less than 200 msec.
Sine	less than 200 msec.

The time used by each of these subroutines determined the number of processors required to meet the 200 millisecond time constraint.

The amount of memory available to program the navigation routine was a function of the 4004 CPU and the number of microprocessors used. One 4004 CPU can directly drive sixteen ROMs and sixteen RAMs. The number of instructions in

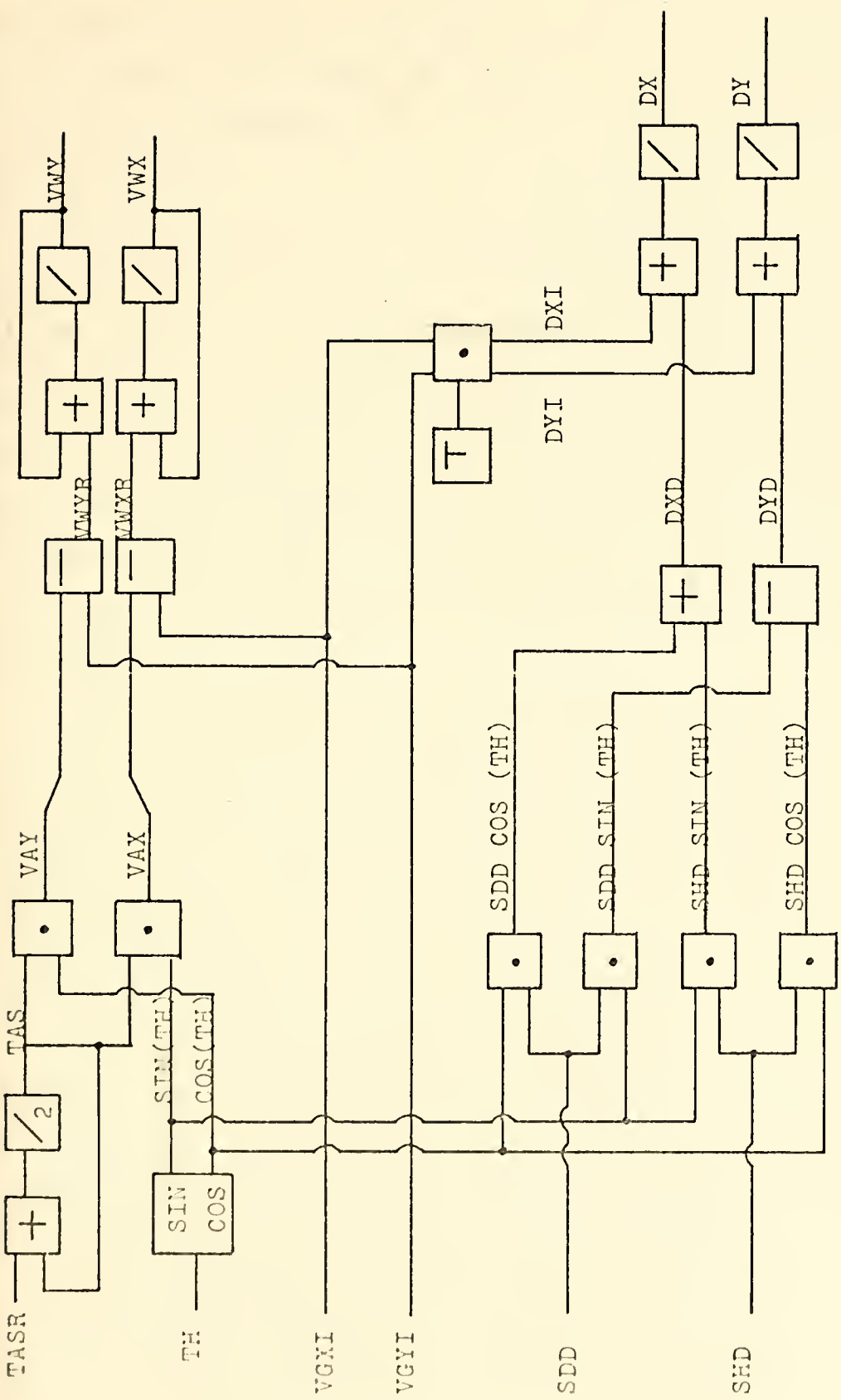


Figure 8. OPERATIONAL CRITICAL PATH

the navigation routine was limited by the space available in the ROMs. Since each ROM could hold 256 instructions, the program was limited to 4,096 instructions per microprocessor.

The time required to sequentially execute every instruction in the sixteen ROMs would be 44 mlliseconds. Since the program was required to be written with an executive routine and a set of subroutines that would repeat the same set of instructions several times, the limiting time constraint would be reached before using up the available ROM space in one microprocessor. It was determined from this analysis that ROM space would not be a limiting factor in writing the navigation program.

The amount of memory space available to store the values of each variable was determined by the space available in the RAMs. Since one 4004 CPU could drive 5120 bits of RAM, the navigation program was limited to 320 variables of 16 bits for each microprocessor. The number of variables required was determined from the operational process graph, Figure 6, where each line connecting a pair of vertices represents one variable. There were 54 lines indicating that a maximum of 54 variables were required plus those variables used in any single operation. By overlaying variables in the same RAM memory space, the memory space requirement was reduced. It was determined from this analysis that RAM space available would not be a limiting factor.

The required accuracy of the navigation program was a function of the accuracy of the input variables. The accuracies of Inertial, Doppler, and Air-Mass systems used onboard the P3C aircraft were used in this analysis as representing the state-of-the-art systems in naval aircraft today. The specifications for these systems are as follows:

System	Designation	Accuracy
Inertial	ASN-84	± 1.5 knots RMS
True Heading	ASN-84	± 9 ARC-MIN RMS
Doppler	APN-187	± 1.0 knots RMS
True Air Speed	Pitot-Static	± 2.0 knots RMS

The accuracy of the navigation program was a function of the accuracy of input data as well as the bit size assigned to each variable. The limited accuracy of the input data permitted each variable to be no greater than 16 bits. This allowed each variable to be represented by four hexadecimal-digits with the first bit assigned as the sign bit. The hexadecimal point for speed measurements was fixed so that there is one hexadecimal digit to the right of the decimal point. This allowed the accuracy of the speed inputs to be within ± .0625 knots. The range on the inputs due to a 16 bit variable limitation was ± 2047.99 knots. The accuracy requirement was not considered a major limitation in the program analysis.

The program analysis pointed out the major areas that had to be demonstrated feasible before programming of the

microcomputer for navigation was initiated. The program analysis was used to set the design goals at each step and the process graph proved to be a major tool in performing the program analysis.

D. SUBROUTINES

A large number of the mathematical operations required in the navigation program are repeated many times. In order to decrease the total programming effort and also decrease the memory-capacity requirements, many of the operations required were written as subroutines. These routines consist of a series of instructions dedicated to a specific task. The subroutines developed for the navigation program were divided into two groups, those involving complex mathematical operations and those involving more common functional operations. Each subroutine was written in a general form to permit its recurring use by the executive program.

1. The Multiplication Routine

The major limitations of the MCS-4 microcomputer to be overcome were the limited instruction set and slow speed of calculation. The most powerful arithmetic instruction available was the single addition two hex-digits. The multiplication routine was written to increase the capability of the MCS-4 microcomputer in order to satisfy the requirements of the navigation program.

A multiplication routine had been written involving multiplication by a series of additions. The program required

only fifty instructions, however, the computational time was 40 milliseconds. Since the navigation program required a minimum of ten multiplications, this method was unsatisfactory.

One of the advantages the MCS-4 microcomputer has is its inexpensive memory. It was decided to investigate a different way of programming the microcomputer that would take advantage of available memory. It was discovered that memory space could be traded for speed by a table look-up. The multiplication subroutine was written using a table look-up scheme.

a. Capability of Subroutine

The multiplication subroutine was designed to take a four hex-digit number, X, and multiply it by a four hex-digit number, Y, resulting in a four hex-digit chopped number. The input X is loaded in Index Registers R8 → RB with the least significant digit, Xo, loaded in R8. The Input Y is loaded in Index Registers RC → RF with the least significant digit, Yo, loaded in RC. The product will appear in Index Registers RC → RF with the least significant digit located in RC.

b. Description of the Routine

The process was done by a table look-up scheme similar to longhand multiplication. An example of this method follows:

			(X_3	X_2	X_1	X_0)
		TIMES	(Y_3	Y_2	Y_1	Y_0)
			X_3Y_0	X_2Y_0	X_1Y_0	X_0Y_0
		X_3Y_1	X_2Y_1	X_1Y_1	X_0Y_1	
	X_3Y_2	X_2Y_2	X_1Y_2	X_0Y_2		
X_3Y_3	X_2Y_3	X_1Y_3	X_0Y_3			

The table of values used in the multiplication routine consits of a 16 x 16 matrix of product values. Each product value is an exact value of a multiplication of two single hex-digit numbers. The row that would normally contain the products of a zero multiplication is used for instructions within the ROM containing the table. A test for a zero input is made in the body of the subroutine. Each product value within the table consists of a two hex-digit number with the second value being the least significant digit. During the execution of the program, each digit in the two hex-digit number is loaded in a separate index register. In this write up, an "M" stands for the most significant digit, an "L" stands for the least significant digit, and a "CY" stands for a carry. The longhand multiplication with the portion chopped in this program can be shown as follows:

							X_3	X_2	X_1	X_0	
							Y_3	Y_2	Y_1	Y_0	
						M_{30}	—	—	—	—	
				M_{31}	L_{31}	M_{21}	—	—	—		
		M_{32}	L_{32}	M_{22}	L_{22}	M_{12}	—	—			
	M_{33}	L_{33}	M_{23}	L_{23}	M_{13}	L_{13}	M_{03}	—			
	CY_4	CY_3		CY_2		CY_1					
Z_3		Z_2		Z_1		Z_0					

A numerical example of this procedure in decimal is as follows:

							9	7	5	3	
							2	4	6	8	
						7	—	—	—	—	
					54	4	—	—	—		
				36	28	2	—	—			
			18	14	10	0	—				
			6	10	13						
		2	4	0	5						

c. Development of the Program

The method used in this procedure becomes very complex because of the large number of separate hex-digits involved and the small number of index registers available to store each digit. This problem becomes more complex since each addition of two hex-digits creates a possible carry. A solution to this problem is to make a process graph that simulates the multiplication process.

The process graph for the multiplication routine is shown in Figure 9. The symbols used have the following meaning:

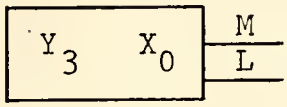
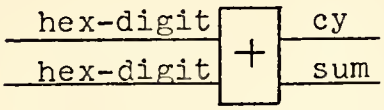


table value of product of the digits X_0 times Y_3



addition

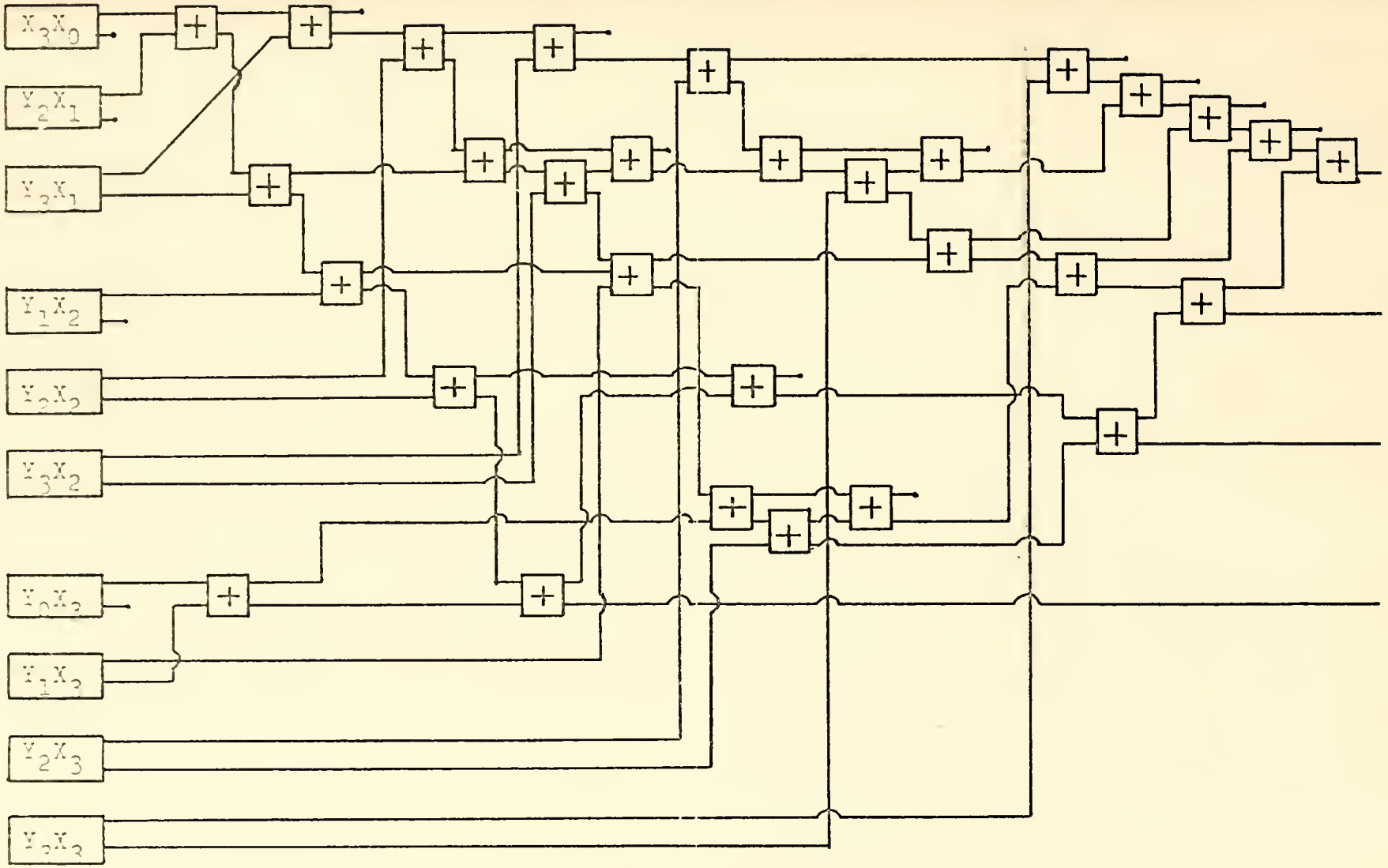
The following examples is used to show the flow of the hex-digits throughout the routine.

Input: X = 1 2 3 4 Hex
Y = 5 6 7 8 Hex

Calculation by hand:

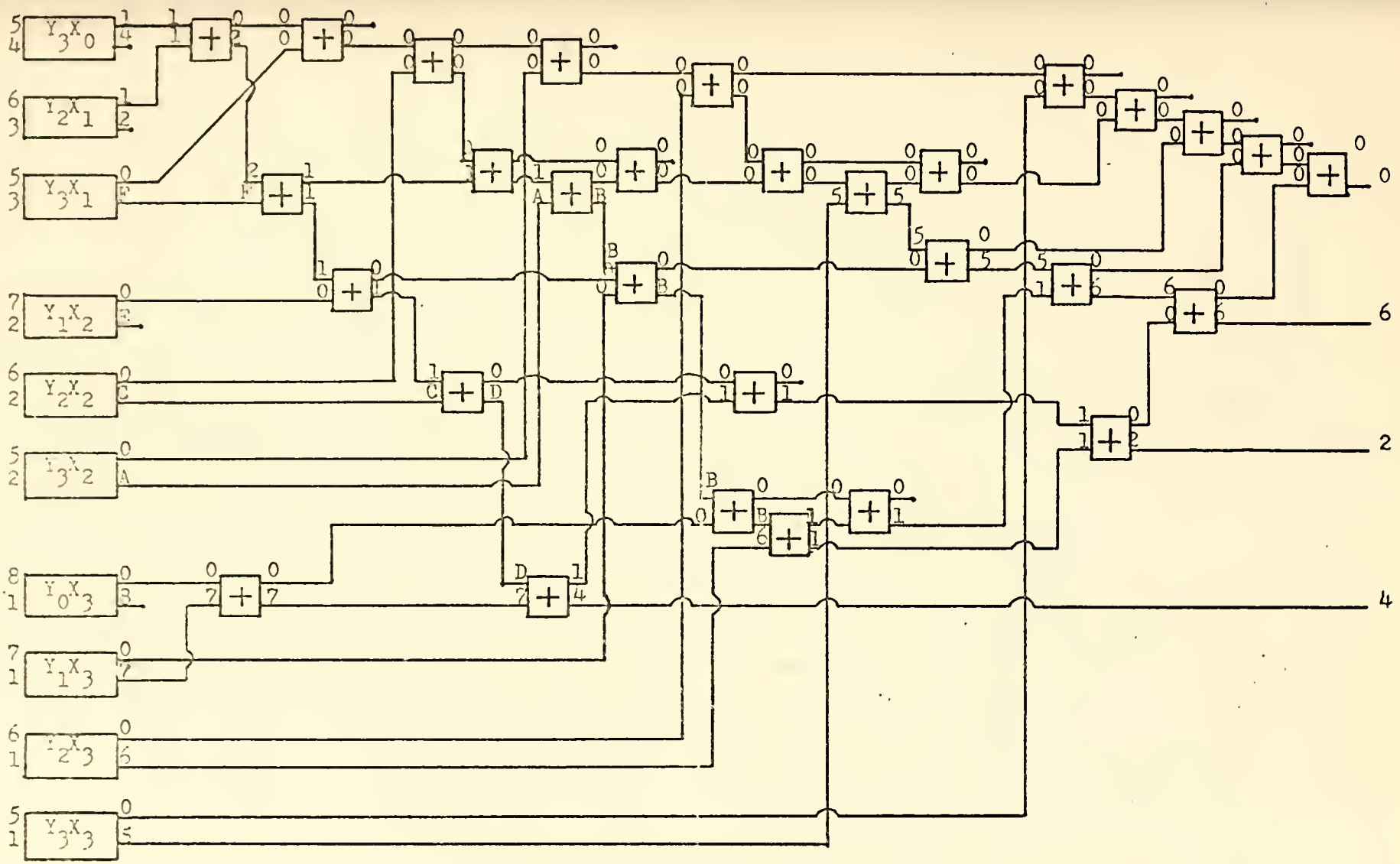
```
                                  5 6 7 8
                                  1 2 3 4
                                --------
                              0 8
                            0 7 0 E
                          0 6 0 C 1 2
                        0 5 0 A 0 F 1 4
                      ---------------------
0     1     2     2
  CY    CY    CY      CY
-----------------
0     6     2     4
```

Figure 10 shows the complete flow of the numbers throughout the routine. The value obtained agress with that computed by hand and can be used as a check of the process graph.



PROCESS GRAPH FOR MULTIPLICATION ROUTINE

Figure 9



SAMPLE MULTIPLY USING PROCESS GRAPH

Figure 10.

d. Constructing the Program

The process graph was used to assign and keep track of the Index Registers being used (Fig. 11).

The subroutine first stores all the values in the Index Registers into RAM, R2 and R3 are used to address values in the RAM. R0 and R1 are used to fetch values from the multiplication table. R4 → RF are the working registers and are assigned as shown in Figure 11.

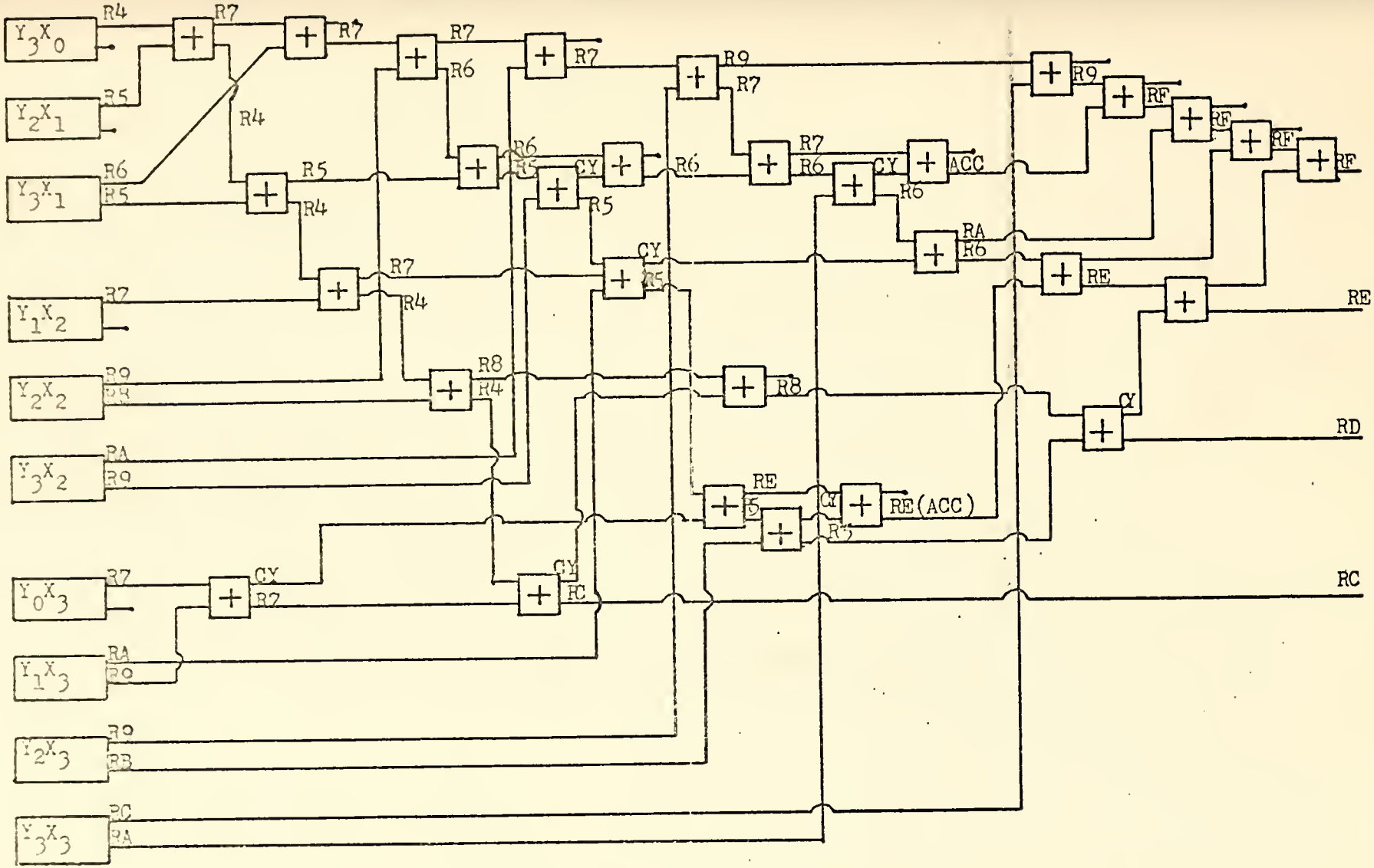
e. Expansion of Multiplication Routine

The multiplication routine was written to give a truncated product of two positive numbers accurate to four significant digits. The multiplication routine which handles both positive and negative numbers is shown in the process graph, Figure 12.

Each number inputted into the multiplication is tested for its sign. The sign bit is then shifted out to normalize the number to four significant digits. The previously described multiplication routine is then executed. The resulting number is shifted back to its proper form and the required sign bit is set.

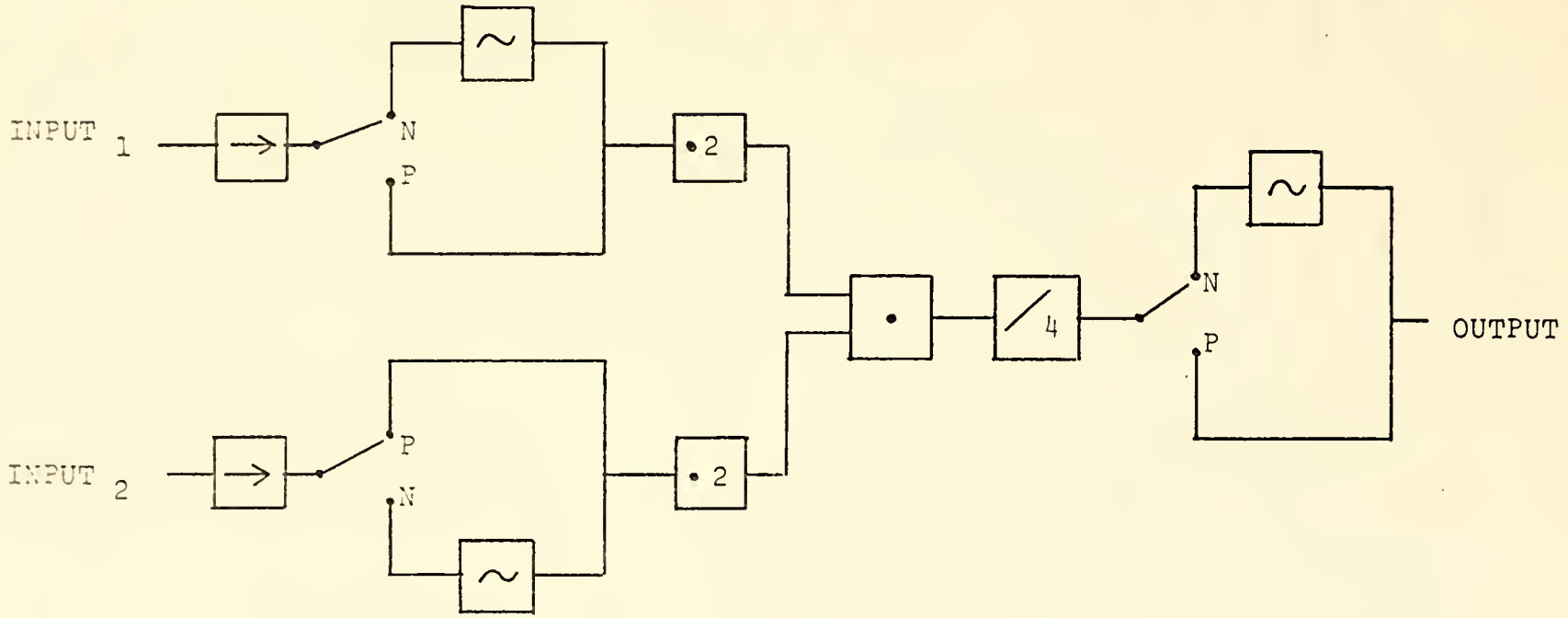
f. Summary

The multiplication routine developed required the memory space of three ROMs and computed its result in five milliseconds. A complete listing of the multiplication routine is found in Appendix C.



MULTIPLY INDEX REGISTER ASSIGNMENT

Figure 11.



EXPANDED MULTIPLY PROCESS GRAPH

Figure 12

2. The Cosine Routine

The development of a Cosine routine for the microcomputer which could compute sufficiently fast was one of the major programming tasks. Cosine Routines written for general purpose computers are usually written as series approximations in order to save memory space. The object of the routine written for this program was to increase the speed of calculation.

Two Cosine routines previously programmed on the MCS-4 Microcomputer were investigated. The first was a Chebyshev approximation routine which required 750 milliseconds to compute the Cosine. The second routine investigated was a Cordic approximation which required 350 milliseconds. Both methods were too time consuming for this project.

The procedure developed in this project was a table look-up, linear interpolation routine. The Newton Divided-Difference Interpolating Polynomial was used because of its simplicity. The size of the table required and the accuracy of the results are both functions of the degree of the Polynomial used. For simplicity and speed, a first-order divided-difference table was used which resulted in a linear interpolation of the form:

$$F(X) = F(X_0) + (X - X_0)\, F[X_1, X_0]$$

where

$$X = \theta$$

$$F(X) = \cos \theta$$

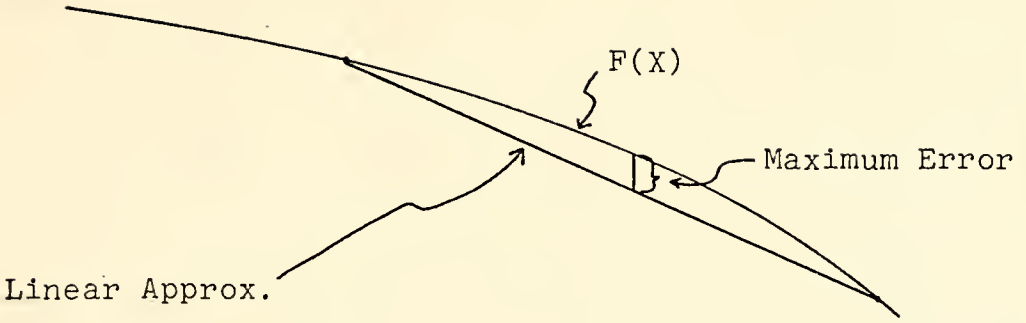
$$F[X_1, X_c] = \frac{F(X_1) - F(X_0)}{X_1 - X_0}$$

The table consisted of all values of $F(X_i)$ and $F[X_{i+1}, X_i]$ for $F(X) = \cos \theta$, $0 \leq \theta \leq 1.88$ radians in hexadecimal. The increment used for each value in the table was .08 radians hexadecimal. The table was constructed from a Fortran program which used a decimal increment of $\frac{1}{.03125}$, equivalent to .08 hexadecimal, and outputted the desire table values in hexadecimal.

a. Table Length

The size of the table loaded into the program was a function of the required accuracy of the Cosine routine. The data supplied to the program from the navigation devices was accurate to three significant figures.

Different size tables were constructed and tested for accuracy. Since the interpolation was linear, the largest error occurred at the mid-point between each table value.



The table size was adjusted until the maximum error was within three units in the fourth significant figure, thus guaranteeing three significant figure accuracy.

The table values were loaded sequentially into ROM number three. The first entry was the value of $F(X_0)$ followed in order by $F(X_1,X_0)$, $F(X_1)$, ..., $F(X_n)$,$F(X_{n+1},X_n)$. The remaining part of the ROM was used for the interpolating routine.

b. Description of the Program

The Cosine routine was designed to be called from outside of the ROM. The input registers RC, RD, RE, RF with RF the least significant figure.

Examples:

		RC	RD	RE	RF
θ = 1.520	would be loaded	0	2	5	1
θ = 0.748	would be loaded	8	4	7	0

The description of the programs function is best shown by an example. Let the input be $\theta = 0.74$. Therefore

RD	RD	RE	RF
0000	0100	0111	0000
0	4	7	0

The first object of the program was to find the required values $(F(X_i),F(X_{i+1},X_i))$ in the table. The ROM can be thought of as a 16 by 16 matrix of 8 bit words. Each byte therefore will hold two significant figures of each table value. Since each table value has four significant figures and two table values were required for each computation, only four bytes needed to be retrieved from the table. If the four required bytes for each computation are thought of as a unit, the table could be thought of as rows of units with four units in each row. The increment in radian value therefore between each row is .20 radians hexadecimal. The required row in the table is easily found by dividing the inputted angle by two.

$$\frac{0740}{2} = 03A0 \qquad \text{(hexadecimal)}$$

The required row in this example is row three. To find the required unit in row three, the remainder of the inputted angle is divided by four.

$$\frac{00A0}{4} = 0028$$

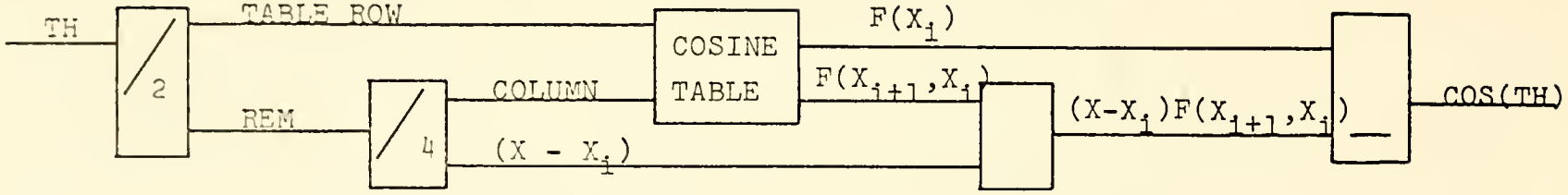
The required unit in this example is unit two. The final remainder of the inputted angle, 0008, is equal to the difference $(X - X_i)$.

In summation, the location of $F[X_{i+1},X_i]$ in the table was found by simple divisions by two and four performed on the input and the difference. Divisions by powers of two can be obtained by shifting operations.

After the desired table values and difference were found, the multiplication routine was called which multiplied $(X - X_i)$ times $F[X_{i+1},X_i]$ and stored the result in index registers RC to RF. The final computation took the value in RC to RF and subtracted it from $F(X_1)$ and stored the result in index register RC, RD, RE, and RF to be returned to the executive program. The step by step procedure of the Cosine routine is shown in the Cosine process graph, Figure 13.

c. Expansion of Routine

The table used in the Cosine Routine was also the table required by a Sine routine. It was noted that this routine could also be used to find the sin θ for $0 \leq \theta \leq 90$ by subtracting the input θ from 90 (1.88 rads-θ rads) and using the same routine since $\cos(90 - \theta) = \sin(\theta)$.



COSINE PROCESS GRAPH

Figure 13

d. Summary

The memory space required by the Cosine routine was one ROM plus the space taken up by the subroutines called by the Cosine routine. The main part of the Cosine routine contains only 46 instructions. The time required to execute the Cosine routine was basically the time required to execute the multiplication. The Cosine routine required only a total of 5.17 milliseconds. This computational speed represents a 70 fold decrease in the computational time to compute the Cosine by previously available routine. By table look-up schemes, it was proven that the computational speed of the microcomputer could be competitive with that of a general purpose computer. The feasibility of using the MCS-4 as a navigation computer was also proven by demonstrating its ability to find the Cosine and Sine at competitive speeds and within the memory limits and accuracy limits required.

3. Common Routines

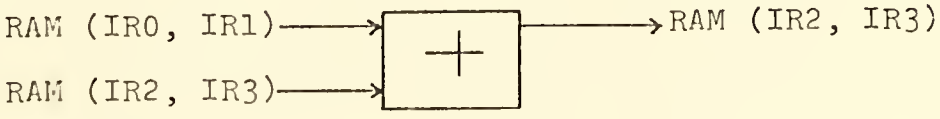
The common subroutines were written to do the basic housekeeping operations such as storing data, simple arithmetic, shift operations, and transfer of data between RAM and IR. These subroutines were called by the executive routine and the multiply and Cosines routines to aid in the data handling. The functions handled by the common subroutines were those best suited for the MCS-4 and therefore could be written in a straight forward way requiring little

speed or memory space. The functions of the common routines were broken into three groups: Arithmetic, Shifting, and Data Handling.

a. Arithmetic

The arithmetic routines handled the simple additions and subtractions required in the navigation program. These routines were handled well in the MCS-4 by the 4-bit ripple-through carry type adder incorporated in the 4004 CPU. This allowed direct addition or subtraction of two hexdigits in either the accumulator or RAM. There were two addition routines, two subtraction routines, and two special purpose arithmetic routines written.

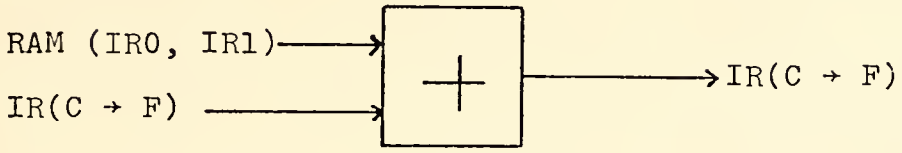
The two addition routines written were ADDRAM and ADDRAMIR. ADDRAM adds the contents of the RAM addressed by IR0 and IR1 to the contents of the RAM addressed by IR2 and IR3 and stores the results in the RAM addressed by IR2 and IR3. The contents of the RAM addressed by IR0 and IR1 remains unchanged.



ADDRAM occupies 12 words of ROM and takes 288.8 microseconds of computation time.

ADDRAMIR adds the contents of the RAM addressed by IR0 and IR1 to the contents of IRC thru IRF and stores

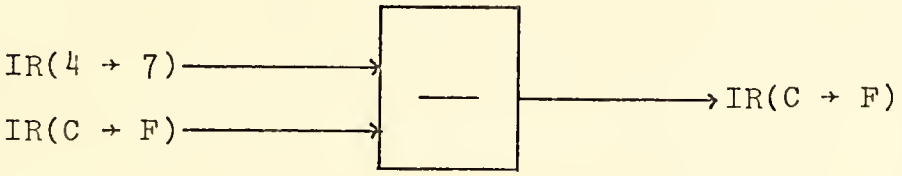
the results in IRC thru IRF. The contents of the RAM addressed by IR0 and IR1 remains unchanged.



ADDRAMIR occupies 22 words of ROM and takes 237.6 microseconds of computation time.

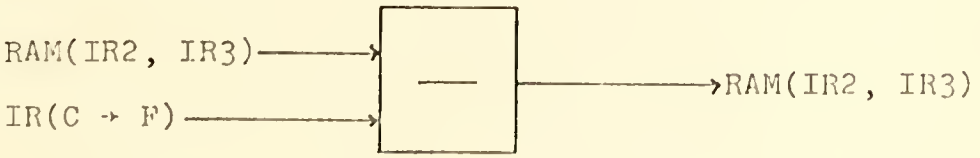
The two subtraction routines written for the navigation program are SUBIR and SUBRAMIR.

SUBIR subtracts the contents of IRC thru IRF from the contents of IR4 thru IR7 and stores the result in IRC thru IRF. The contents of IR4 thru IR7 remains unchanged.



SUBIR occupies 17 words of ROM and takes 182.6 microseconds of computational time.

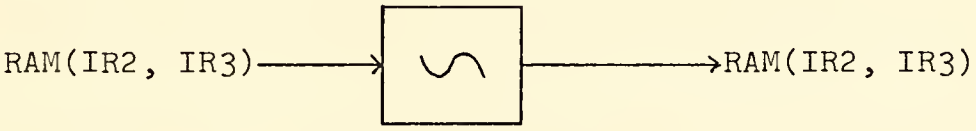
SUBRAMIR subtracts the contents of IRC thru IRF from the contents of the RAM addressed by IR2 and IR3 and stores the result in the RAM addressed by IR2 and IR3. The contents of IRC thru IRF remains unchanged.



SUBRAMIR occupies 32 bytes of ROM and takes 345.6 microseconds of computational time.

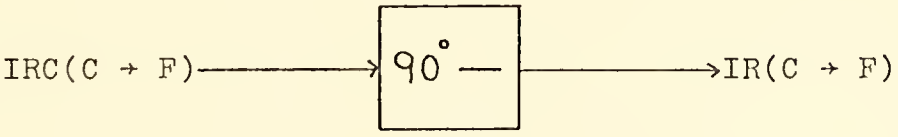
The two special arithmetic routines written for the navigation program are COMPLEMENT and COMANGLE.

COMPLEMENT takes a four character hex number addressed by IR2 and IR3 in RAM, takes the ones complement of that number, and stores the result back into the location in RAM addressed by IR2 and IR3.



COMPLEMENT occupies 28 bytes of ROM and takes 302.4 microseconds of computational time.

COMANGLE takes the angle loaded in IRC thru IRF, subtracts it from 90 in hex radians, and stores the result back into IRC thru IRF.



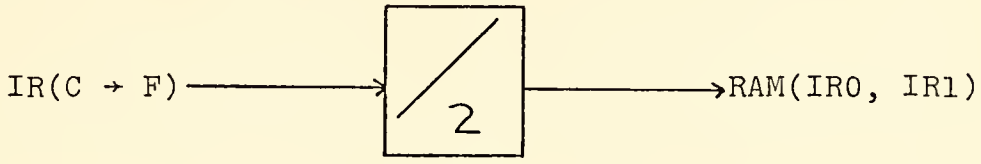
COMANGLE occupies 18 bytes of ROM and takes 194.4 microseconds of computational time.

b. Shifting

Multiplication and division of hex-digits by a multiple of two was accomplished by shifting the variable either left or right the required number of bits. To take

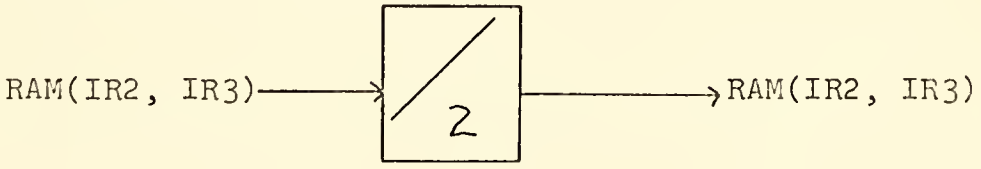
advantage of this capability, two subroutines were written for the navigation program, DIV2IR and DIV2.

DIV2IR divides the contents of IRC thru IRF by two and stores the result in the RAM addressed by IR0 and IR1. The division is accomplished by shifting the variable right one bit. The original contents of IRC thru IRF are unaffected.



DIV2IR occupies 24 bytes of ROM and takes 259.2 microseconds of computational time.

DIV2 divides the contents of the RAM addressed by IR2 and IR3 by two and stores the result in the RAM addressed by IR2 and IR3.



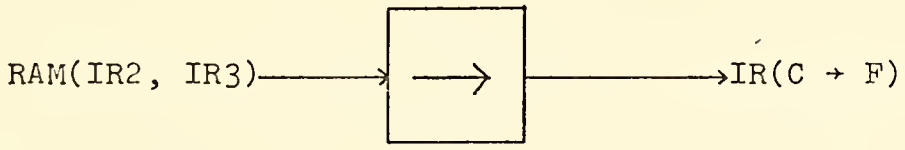
DIV2 occupies 33 bytes of ROM and takes 356.4 microseconds of computational time.

c. Data Handling

The transfer of data within the navigation program was accomplished by the data handling routines. These routines saved much memory space in the Executive Routine and the Multiply and Cosine Routines by grouping these tasks

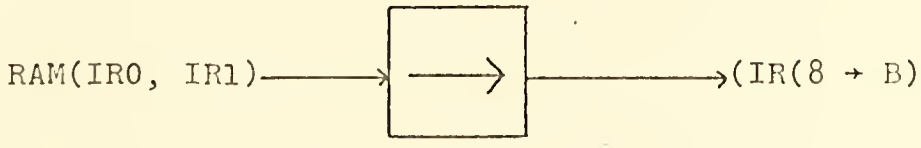
into separate subroutine calls. There were five data handling routines written for the Navigation program. Three routines, RAMIRC, RAMIR8, and IRRAMC, were written to transfer data between the IRs and RAM. TRANRAM was written to transfer data between different locations in RAM. TIME was a special routine written to load the proper time interval of one navigation cycle into the IRs. All five routines were written to handle data of four hex-digit size.

RAMIRC transfers the contents of the RAM addressed by IR2 and IR3 into IRC thru IRF. The original contents of RAM are unaffected.



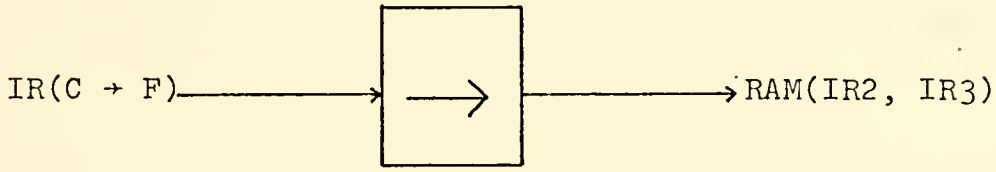
RAMIRC occupies 16 bytes of ROM and takes 172.8 microseconds of computational time.

RAMIR8 transfers the contents of the RAM addressed by IR0 and IR1 into IR8 thru IRB. The original contents of RAM are unaffected.



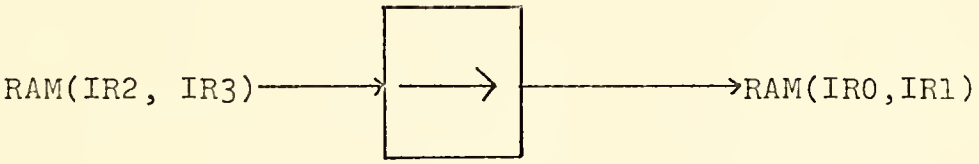
RAMIR8 occupies 16 bytes of ROM and takes 172.8 microseconds of computational time.

IRRAMC transfers the contents of the IRC thru IRF into the RAM addressed by IR2 and IR3. The original contents of IRC thru IRF are unaffected.



IRRAMC occupies 16 bytes of ROM and takes 172.8 microseconds of computational time.

TRANRAM transfers the contents of RAM addressed by IR2 and IR3 into RAM addressed by IR0 and IR1. The original contents of RAM addressed by IR2 and IR3 are unaffected.



TRANRAM occupies 8 bytes of ROM and takes 86.4 microseconds of computational time.

TIME is a special subroutine written to load the time interval of one navigation cycle into IR8 thru IRB. The time interval used in this program was 200 milliseconds. To input a different time interval into the navigation program, a new TIME routine is substituted into the program.

TIME occupies 9 bytes of ROM and takes 97.2 microseconds of computational time.

c. Summary

Thirteen Common Subroutines were written for the Navigation program. Each routine was designed to handle the particular needs of the Navigation program. All thirteen routines fit in one ROM. A complete listing of the Common Subroutines is in Appendix A.

4. Summary of Subroutines

Each operation defined in the operational process graph, Figure 6, was successfully programmed within the memory and time constraint of the program analysis. Each subroutine was written in a modular form allowing for easy addition or subtraction of new code. The memory size, computational time, and capability of each subroutine is listed in Table VII.

E. EXECUTIVE ROUTINE

The Executive Routine was written to call up the subroutines in the order described by the operational process graph, Figure 6. The Executive Routine established the priorities of each function and was designed to make all the decisions in the execution of the Navigation program. The variables used by the Executive Routine were all stored in RAM. The designated location of each variable in RAM is shown in Table VIII.

Subroutine	Length (8-bit words)	Time (μsec)	Purpose
MULT	743	5000	[IR(8→B)]·[IR(C→F)]→IR(C→F)
COS	255	5170	COS[IR(C→F)] → IR(C→F)
ADDRAM	12	289	RAM(0,1) + RAM(2,3)→RAM(2,3)
ADDRAMIR	22	238	RAM(0,1) + IR(C→F)→IR(C→F)
SUBIR	17	183	IR(4-7)-IR(C→F)→IR(C→F)
SUBRAMIR	32	346	RAM(2,3)-IR(C→F)→RAM(2,3)
COMPLEMENT	28	302	~[RAM(2,3)]→RAM(2,3)
COMANGLE	18	194	(90°)-IR(C-F)→IR(C→F)
DIV2IR	24	259	[IR(C→F)]/2→RAM(0,1)
DIV2	33	356	[RAM(2,3)]/2→RAM(2,3)
RAMIRC	16	173	RAM(2,3) → IR(C→F)
RAMIR8	16	173	RAM(0,1) → IR(8→B)
IRRAMC	16	173	IR(C→F) → RAM(2,3)
TRANRAM	8	86	RAM(2,3) → RAM(0,1)
TIME	9	97	(Nav Cycle Time)→IR(8→B)

Table VII NAVIGATION SUBROUTINES

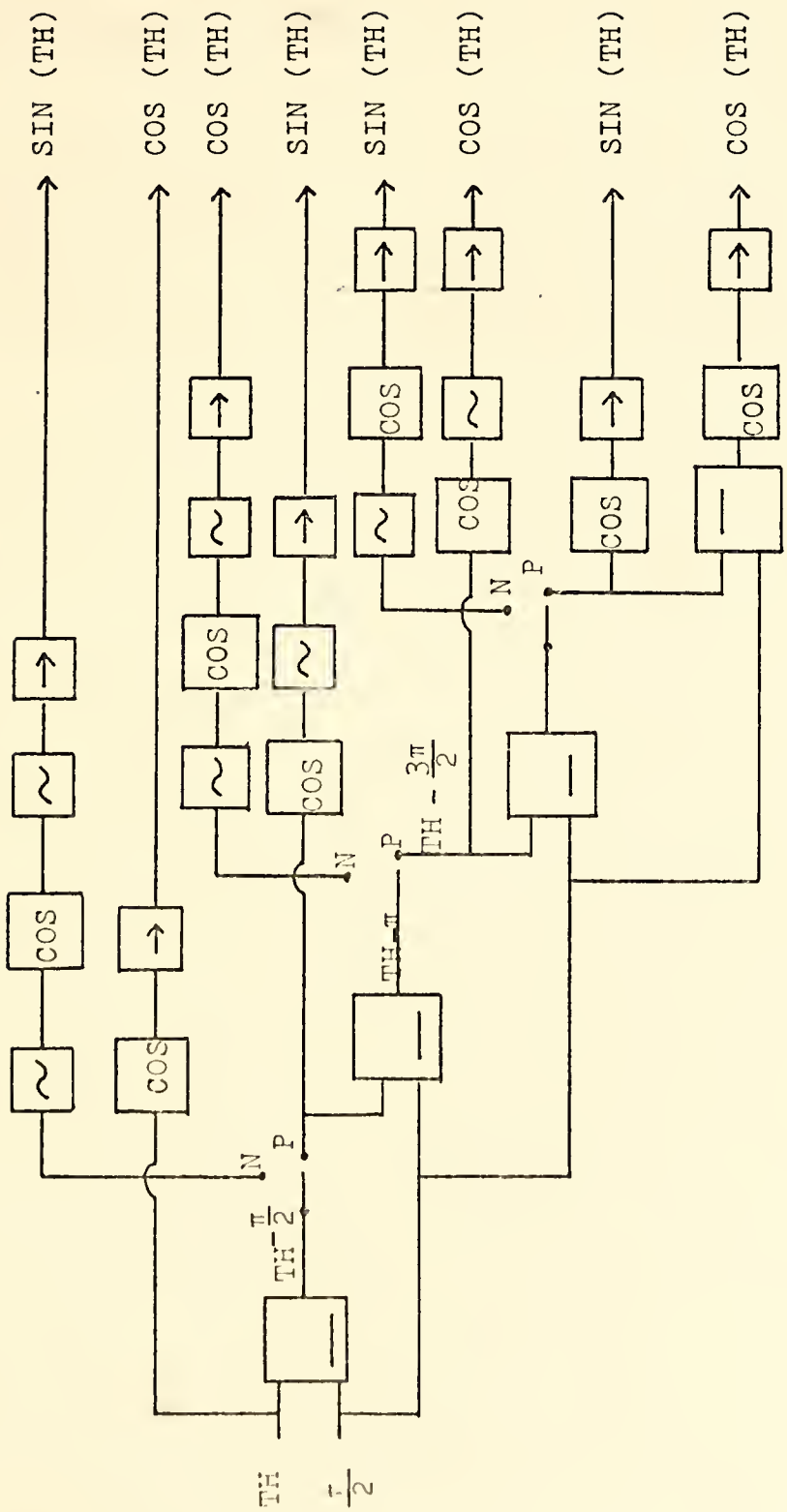
		0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
R	0	FLAGI		FLAGD		FLAGC					DX				TAS		
A	1										DY				COS(TH)		
M	2		TASR												VAX		
0	3		VWX				VWY				TH-π/2				VAY		
R	4		VGXI				VGYI				TH				SIN(TH)		
A	5		SHD				SDD								TH		
M	6		SDD SIN(TH)				SDD COS(TH)								MULT		
1	7														DX		
R	8														DYD		
A	9																
M	A	SIGN				INDEX REGISTER STORAGE FOR MULT											
2	B																

Table VIII. DESIGNATION OF RAM VARIABLES LOCATIONS

The first operation of the Executive Routine was to compute the proper Cosine and Sine of the current true heading. Before calling the COSINE subroutine, the Executive Routine determined which quadrant the true heading was in and set up the input angle in the proper form so that the Sine or Cosine could be computed by the COSINE subroutine. After calling the COSINE subroutine, the Executive Routine stored the output with the proper sign in its location in RAM. A graphical display of the decisions made by the Executive Routine in computing the Cosine and Sine of the true heading is illustrated in Figure 14.

Next the Executive Routine smoothed the current TAS indication with the previous TAS computed. The smoothed TAS was then used to compute VAX and VAY as described in the Navigation equations and illustrated by the operational process graph.

After computing VAX and VAY, the Executive Routine tested to see if the INS was good. If the INS was good, the Executive Routine computed the wind acting on the vehicle and the distance increments traveled as measured by the INS. If the INS was down, the Executive Routine computed the distance increments traveled as measured by the AIR-MASS system using the last computed wind components. Next the Doppler System was tested to see if it was up or down. If the Doppler was up, the distance increments as measured by the Doppler were computed.



TRUE HEADING PROCESS GRAPH

Figure 14

The distance increments outputted from the Navigation Program were selected by the Executive Routine in the following priority according to the following conditions:

Priority	Conditions
1. INS and Doppler Smoothed	INS up and Doppler Up
2. INS	INS up and Doppler down
3. Doppler	INS down and Doppler up
4. Air-Mass	INS down and Doppler down

The Executive Routine was loaded into two ROMs with space left for addition of new code. A complete listing of the Exeuctive Routine is in Appendix B.

F. ERROR ANALYSIS

Microcomputers have a limited arithmetic capability. The basic arithmetic operation of the MCS-4 is a four bit addition. More complex arithmetic processes had to be reduced to this basic operation. It was therefore very important to avoid unnecessary precision throughout the calculations. Since the inputs into the Navigation program came from instruments whose precision is limited to three hexadecimal digits, the choice of four hexadecimal arithmetic was considered to be sufficiently accurate. The following error bound analysis was performed to show that the input errors dominate the total error.

The starting point for the error analysis is the operational process graph, Figure 6. It was apparent from the

operational process graph that the outputs DX and DY are symmetric, therefore an analysis of only the computations for DX were made. A process graph that involved only the operations which have an influence on DX was constructed, as shown in Figure 15, from the operational process graph, Figure 6.

The errors corresponding to each operation in Figure were designated e_1, e_2, e_3, ..., e_8. The initial errors of the inputted data were expressed as e(TH), e(VGXI), e(SDD), and e(SHD). Due to the smallness of the errors, the products of errors were considered negligible when compared to the linear terms. The propagation error was derived as discussed in Chapter 2 of Reference 28 in the Bibliography.

There were two means by which each operation contributed to the error propagation.

1.) Transmitting the errors which were inputted into the operation.

2.) Adding an error of its own, which is due to the rounding or truncating process which limits the number of digits carried to the next operation.

The transmitted errors were calculated by calculating the differentials of the expression.

$$d(x+y) = dx + dy$$

$$d(xy) = y\,dx + x\,dy$$

$$d\left(\frac{x}{y}\right) = \frac{y\,dx - x\,dy}{y^2}$$

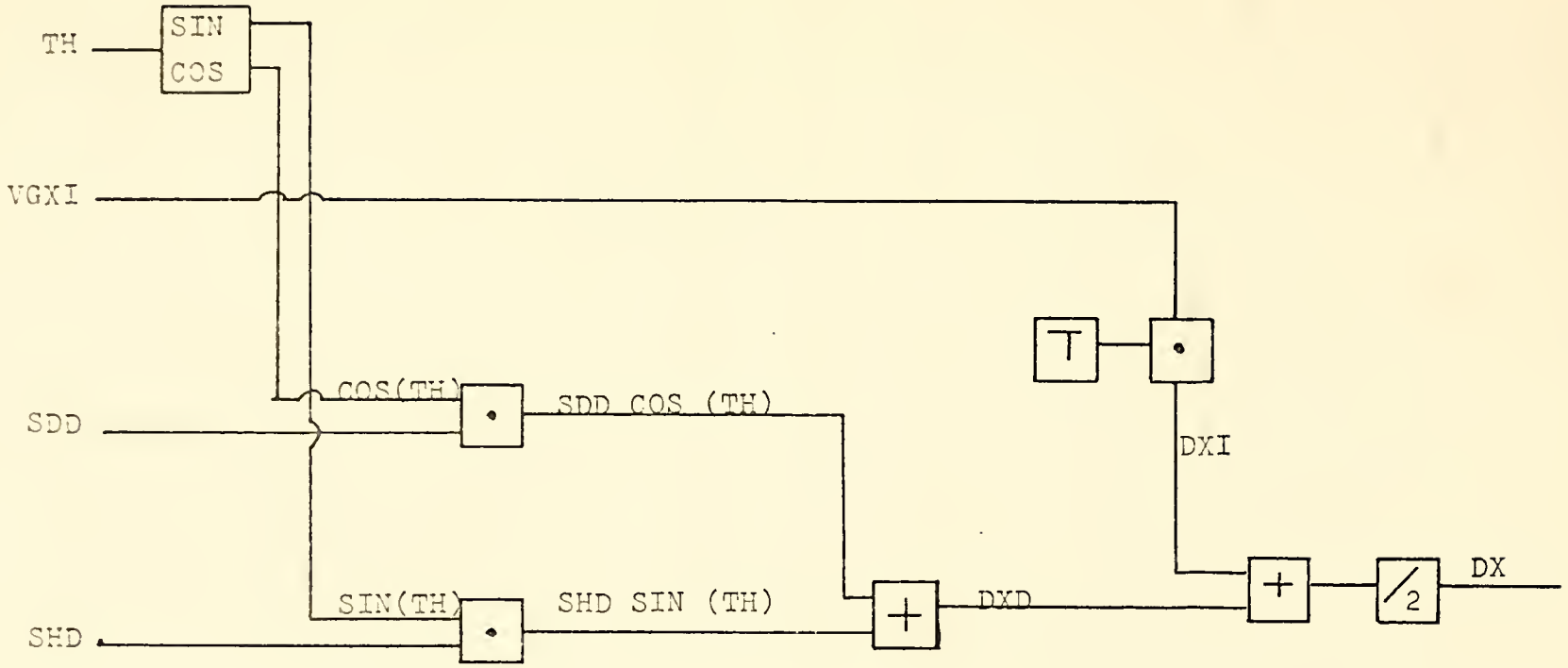


Figure 15. PROCESS GRAPH OF DX

$$d(\sin x) = \cos x \, dx$$

$$d(\cos x) = - \sin x \, dx$$

The rounding or truncating errors were simply added to the transmitted error and thereafter propagated through the remainder of the calculation.

The error after the first operation in Figure 15 is given by:

$$e(\cos(TH)) = - (\sin(TH)e(TH) + e_1$$

This error is further transmitted by operation three.

$$e(\cos(TH)SDD) = SDD(-\sin(TH)e(TH) + e_1) + \cos(TH)e(SDD) + e_3$$

Similarly, the error after the second and fourth operation is given by:

$$e(\sin(TH)SHD) = SHD(\cos(TH)e(TH) + e_2) + \sin(TH)e(SHD) + e_4$$

This gives the total error after operation six as:

$$e(\cos(TH)SDD) + e(\sin(TH)SHD) + e_6$$

In a similar fashion, the error after operation five is:

$$e(T\cdot VGXI) = VGXI\cdot e(T) + T\cdot e(VGXI) + e_5$$

Combining the above, the total error is expressed as:

$$e(DX) = \frac{1}{2}\{e(T\cdot VGXI) + [e(\cos(TH)SDD)+e(\sin(TH))SHD + e_6]+e_7\}+e_8$$

The following is obtained by expanding the expression for e(Dx):

$$\begin{aligned} e(Dx) = \frac{1}{2}\{&VGXI\cdot e(T)+e(VGXI)\cdot T+e_5+SDD(-\sin(TH)e(TH)+e_1) \\ &+ \cos(TH)\cdot e(SDD)+e_3+SHD(\cos(TH)e(TH)+e_2) \\ &+ \sin(TH)e(SHD)+e_4+e_6+e_7\}e_8 \end{aligned}$$

The error bound for e(Dx) is computed by using the triangular inequalities.

$$\begin{aligned} |e(Dx)| \leq \frac{1}{2}\{&|VGXI|\cdot|e(T)| + |T|\cdot|e(VGXI)| + |SDD|[|\sin(TH)||e(TH)| \\ &+ |e_1|] + |\cos(TH)||e(SDD)|+ |SHD|[|\cos(TH)||e(TH)| \\ &+ |e_2|] + |\sin(TH)||e(SHD)| + |e_5| + |e_3| + |e_4| \\ &+ |e_6| + |e_7|\} + |e_8| \end{aligned}$$

The time increment T is controlled by the programmer and hence may be chosen to be a power of sixteen, therefore the inherent error e(T) = 0. The addition process carried out in operations six and seven is carried out exactly in the range of values permitted, therefore $e_6 = 0$ and $e_7 = 0$. This reduces the error bound expression for the total propagated error to:

$$e(Dx) \leq \frac{1}{2}\{|T| \cdot |e(VGXI)| + |SDD|[|\sin(TH)||e(TH)| + |e_1|]$$
$$+ |\cos(TH)||e(SDD)| + |SHD|[|\cos(TH)||e(TH)| + |e_2|]$$
$$+ |\sin(TH)||e(SHD)| + |e_5| + |e_3| + |e_4|\} + |e_8|$$

The error bound for each operation in Figure 15 due to roundoff and truncation was found to be:

Operation	Error Bound (Decimal)	Corres. Error
Cosine	3×10^{-5}	$\lvert e_1 \rvert$
Sine	3×10^{-5}	$\lvert e_2 \rvert$
Multiplication	3×10^{-6}	$\lvert e_3 \rvert$, $\lvert e_4 \rvert$, $\lvert e_5 \rvert$
Division	8×10^{-6}	$\lvert e_8 \rvert$
Subtraction	8×10^{-6}	
Addition	0	$\lvert e_6 \rvert$, $\lvert e_7 \rvert$

The error bound for the inputs was obtained from published sources. Since an actual maximum error for each system could not be found, the (3σ) value, 99.7% CEP, was used. The

systems onboard the P3C naval aircraft were used as representing the current "state-of-the-art" systems in operational use today.

System	Error (30)	Corresponding Error
Inertial Navigation System	5.4 NM/HR	|e(VGXI)|
Inertial True Heading	.5 degrees	|e(TH)|
Doppler Along Heading	1.8	|e(SHD)|
Doppler Across Heading	3.6	|e(SDD)|

The error bound for the Doppler is a function of velocity. In order to compute the error bound for the system, an example was used. The error bound analysis was calculated for an aircraft with a velocity of 400 knots along the true heading and a velocity of 50 knots across the true heading. The navigation cycle time used was 150 milliseconds. The true heading of the vehicle was 045 degrees. The inputs and error bounds inputted into the system were calculated as follows:

TH = 045 degrees = 0.7854 radians

e(TH) = .5 degrees = 0.0087266

VGXI = 3.8.2 knots

e(VGXI) = 5.4 knots

TIME = .15 sec = .0000416 HR

SHD = (400 knots)(.0000416 HR) = .01664 nm.

e(SHD) = (.01664 nm)(.018) = .0002995 nm.

SDD = (50 knots)(.0000416 HR) = .00208 nm.

e(SDD) = (.00208 nm)(.036) = .0000748 nm.

Although the values calculated above are dimensionally correct, they are not in the proper form for inserting into the computer. The MCS-4 microcomputer used in this system works with a 16-bit fixed point variable. In order not to lose significant digits in each number, the inputs are all normalized. The fixed point number used in the calculation is in the form of (._ _ _ _), therefore the value of VGXI used in the computations is .3182. The time variable is normalized to .4160 which results in SHD equalling .1664. Now that the inputs are dimensionally correct and in the proper format, they can be substituted into the error bound equation as follows:

$$|e(Dx)| \leq \tfrac{1}{2}\{(.416)(.0054) + (.0208)[(SIN(.7854))(.0087266)$$

$$+ .00003] + (COS(.7854))(.000748) + (.1664)$$

$$[(COS(.7854))(.0087266) + .00003] + (SIN(.7854))$$

$$(.002995) + .00003 + .00003 + .00003\} + .000008$$

$$|e(Dx)| \leq .0030798$$

This is the maximum error bound for the total navigation system. The maximum error bound equation was broken into two parts, one part indicated the maximum propagated errors due to the inputs and the other indicated the maximum error developed by the navigation program.

The error bound for the inputs alone is calculated as follows:

$$|e(Dx)|_{input} \leq \frac{1}{2}\{|T||e(VGXI)| + |SDD|[|SIN\ (TH)||e(TH)|]$$

$$+ |COS\ (TH)||e(SDD)| + |SHD|[|COS\ (TH)||e(TH)|]$$

$$+ |SIN\ (TH)||e(SHD)|\}$$

$$|e(Dx)|_{input} \leq \frac{1}{2}\{(.416)(.0054) + (.208)(.70711)(.0087266)$$

$$+ 70711)(.000748) + (.1664)(.70711)(.0087266)$$

$$+ (.70711)(.002995)\}$$

$$|e(Dx)|_{input} = .003024$$

Next the maximum error due to the calculations in the navigation program alone were computed.

$$|e(Dx)|_{prog} \leq \frac{1}{2}\{(.0208)(.00003) + (.1664)(.00003) + .00003$$

$$+ .00003 + .00003\} + .000008$$

$$|e(Dx)|_{prog} \leq .0000557$$

The results of the error bound analysis indicated that the maximum error created in the navigation microcomputers computations was only 1.8 per cent of the total maximum error. It was concluded from this analysis that the accuracy of the MCS-4 navigation program using a 16-bit fixed word data length was well within the limits required for the navigation problem.

The best way to check the results of this error bound analysis would be to fly the system in an actual aircraft. Since an aircraft was not available for this purpose, a detailed FORTRAN simulation program was written to test the functions of the navigation program. Section VI discusses the FORTRAN simulation program and the results obtained from examples tested on this program.

G. SUMMARY OF PROGRAM

The navigation program written in this thesis for the MCS-4 microcomputer takes the outputs of the Inertial navigation system, Doppler navigation system, and Air-Data

system and computes the vehicle's distance traveled in the east/west and north/south directions. The total program consists of 1768 instruction words on seven 4001 ROM chips broken into an executive routine and fifteen subroutines. Two 4002 RAM chips are required to store the data and variables used in this program. The total computational time required for one navigational cycle is between 36 and 80 milliseconds depending on the navigational mode used. The computational error developed by the navigation program from error bound analysis represents only .1 per cent of the total error. The program uses a sixteen bit fixed point variable which allows it to accept inputs up to 2047.99 knots with an accuracy of ± .0625 kts. The total cost of the one CPU chip, seven ROMs, and two RAMs used by the system is $95.00.

A complete listing of the navigation program written in the MCS-4 assembler language is found in Appendices A, B, and C.

VI. FORTRAN SIMULATION PROGRAM

The development of the MCS-4 Microcomputer navigation program was a very time-consuming process that involved the construction and testing of many subroutines. The testing of the total program in its final form was limited by the capability of modeling the inputs to the navigation program in a machine level language. Hand calculated examples were tested and used to debug the completed program, it was decided to write a detailed bit-by-bit digital simulation of the navigation program in FORTRAN on the IBM 360/67 computer. The FORTRAN simulation was written so that major changes in word length and scaling could be tested without requiring major rewriting of the microcomputer program.

Parameter changes were made to the Inertial, Doppler, and Air-Mass inputs without incurring the added cost and time of developing an actual system. The FORTRAN simulation was a comprehensive and flexible means of testing the microcomputer program through a series of realistic and unrealistic tasks. A complete listing of the FORTRAN simulation program is found in Appendix D.

A. DEVELOPMENT

The FORTRAN simulation program was developed as an exact simulation of the navigation program developed for the MCS-4 microcomputer. The program was written by utilizing the

process graphs developed in Section V. The program includes a parallel solution of the navigation equations utilizing the FORTRAN routines available on the IBM 360/67 computer. The results of the solution by the MCS-4 simulation and the FORTRAN solution provided a comparison of the 16-bit fixed point solution of the MCS-4 microcomputer program and the 24-bit floating point routines on the IBM general purpose computer.

The FORTRAN Simulation provided the programmer with the ability to input many variables. The outputs of the Inertial, Doppler, and Air-Mass systems were modeled to include system failures and input errors. The outputs of the simulation indicated how well the programmed microcomputer could handle changing variables as compared to a programmed general purpose computer.

The main body of the simulation program was written as a subroutine in order that systems programmed or simulated in FORTRAN could easily be tested with the navigation routine. The main body of the simulation program declares the index registers, random access memory locations, and read only memory locations as dimensioned arrays. The numbering and size of each array corresponds exactly with the addressing and size of these areas within the MCS-4. The main body of the simulation program prepares the inputs and initial conditions for the execution of a simulated navigation cycle. The hexadecimal Cosine table stored in the MCS-4 ROM is

computed and stored in the ROM array by the main body of the simulation program. The simulation begins when the subroutine NAV is called with the initial inputs.

Subroutine NAV is a FORTRAN simulation of the MCS-4 navigation executive program. The inputted variables to NAV are 32-bit floating point decimal numbers. Each variable is converted to a 16 bit fixed point hexadecimal number by the subroutine CONVRT, Appendix D. After each input is in the proper form, subroutine NAV stores each variable in the RAM array at the same location corresponding to the RAM addresses in the MCS-4, Table VIII. After the inputs have been stored in the proper locations and in the proper format, subroutine NAV calls a FORTRAN simulation of each subroutine called by the MCS-4 navigation executive routine. Each subroutine is executed in the same order as in the MCS-4 executive program. The 16-bit hexadecimal results are printed to provide a check with the results of the actual MCS-4 microcomputer navigation program. The NAV subroutine lastly calls subroutine CONVD, Appendix D, which converts the 16 bit hexadecimal results into floating point decimal results which are then compared with the solution of the navigation equations by floating point FORTRAN calculations. A complete listing of subroutine NAV is found in Appendix D.

Each common subroutine programmed on the MCS-4 microcomputer was duplicated in the FORTRAN simulation. A complete description of each subroutine and its graphical

representation can be found in Section V.D.3. A complete listing of the FORTRAN simulation of each of the common subroutines is found in Appendix D.

The multiplication and Cosine subroutines used in the FORTRAN simulation were developed from the multiplication process graph, Figure 9, and the Cosine Process graph, Figure 13. The Cosine routine uses the Cosine table stored by the main body of the simulation program. The multiplication table used by the simulated multiply routine was computed individually for each pair of hexadecimal digits called by the multiply routine.

The last subroutine written for the simulation program, subroutine PRINT, was written to incorporate some of the features of the MCS-4 Assembler and Interpreter. Subroutine PRINT, Appendix D, was developed to allow the programmer to dump the contents of the index registers and RAM locations at any point during the execution of the simulation.

A complete listing of the FORTRAN simulation program is in Appendix D.

B. APPLICATION

The FORTRAN simulation was first used to optimize the navigation program. The first navigation program written on the MCS-4 had been designed to handle only positive numbers within the true heading limitations of zero to ninety degrees. Before expanding the program to handle all quadrants

and both positive and negative inputs, different ways to implement these changes were investigated. The Fortran simulation allowed numerous techniques to be attempted with the advantages and limitations of each readily apparent in the output. The Fortran simulation program was easy to change, whereas the MCS-4 program would have required major program changes to implement each technique. After the Fortran program had been modified to do the required tasks, the MCS-4 navigation program was modified to incorporate these required changes. The true heading was used in four separate quadrants of ninety degrees each with the Executive routine testing the true heading input and computing the Cosine and Sine as required. The sign of each variable was incorporated into the program by using a ones complement scheme with the first bit of the 16-bit word becoming the sign bit. The required changes to each subroutine was first tested in the simulation program before being programmed into the navigation routine.

The second use of the simulation program was to debug the MCS-4 program in its final form. The simulation program was run through a series of examples. The simulated MCS-4 program results were compared with the Fortran computed results to insure the correctness of the simulation program. After the simulation program had been completely debugged, the actual MCS-4 program was run with the same inputs as the simulation program. By dumping the contents of the registers

and RAM memory at key points in the actual program and comparing these values with those computed and printed out by the simulation program, the sections of the actual program in error were easily identified. After changing a section in error, the actual program and simulation program were run to see if the correction was correct and also see if it affected any of the other sections. This method of checking and rechecking proved to be an excellent tool for debugging microcomputer programs. The only problem encountered in this method was the required time for outputs to be printed from the IBM 360/67.

The third use of the simulation program was to check the results of the error bound analysis. The same inputs used in the error analysis were inputted into the simulation. Simulation runs were made using correct inputs in order to compare the results of the MCS-4 calculation with those of the Fortran program in order to determine the errors due to the navigation program. Simulation runs were made for each navigation mode of operation and for each true heading quadrant. Next the same runs were made with the inputs at maximum errors. The results of these runs and the comparison of the errors developed are summarized in Table IX.

The results, summarized in Table IX, confirmed the results of the error bound analysis. It was noted that the greatest computational error occurred when the vehicle traveled a direct path with constant inputs. This was due to

Type of Path	Nav. Mode	Type Input	Computed X-Loc. FORTRAN	Computed X-Loc. MCS-4	Computed Y-Loc. FORTRAN	Computed Y-Loc. MCS-4	Maximum Error Due to NAV Program	Maximum Error Due to Input
Point to Point	Integrated	Exact	-0.9814	-0.9811	1.0427	1.0425	.03%	
	Inertial		-1.1041	-1.1038	0.9813	0.9811	.03%	
	Doppler		-0.8588	-0.8586	1.1041	1.1038	.03%	
	Air-Mass		-1.0912	-1.0909	0.9812	0.9808	.04%	
Point to Point and Back	Integrated	Exact	-0.0001	0.0	0.0	0.0	0.0	
	Inertial		0.0	0.0	0.0	0.0	0.0	
	Doppler		-0.0001	0.0	-0.0001	0.0	0.0	
	Air-Mass		0.0	0.0	-0.0001	-0.0001	.01%	
Point to Point	Integrated	Max Error	-1.0037	-1.0034	1.0428	1.0423	.05%	2.23%
	Inertial		-1.1228	-1.1227	0.9625	0.9622	.03%	1.88%
	Doppler		-0.8845	-0.8841	1.1230	1.1225	.05%	2.57%
	Air-Mass		-1.1088	-1.1081	0.9636	0.9639	.07%	1.76%

TABLE IX. RESULTS OF FORTRAN SIMULATION

the linear addition of the truncation error when the inputs remain constant. It was also noted that the computational error was zero when the vehicle returned to the departing position indicating that the truncation error cancelled in the opposite direction.

VII. CONCLUSIONS

The results of this design study indicated that a microcomputer is both fast enough and powerful enough to handle the complex task of navigation. The total computational time of the navigational cycle, 80 milliseconds, is well below the 200 millisecond cycle time used in current navigation computers. Table look-up routines for such complex tasks as multiply, Cosine, and Sine proved an effective means of trading inexpensive memory for computational speed. The total hardware cost of the ten LSI cips, excluding a board to hold the DIP packages, was $95.00.

The software aids investigated in this study were very effective in decreasing programming and debugging time. Graph Theory, in the form of process graphs, was an excellent means of visualizing the actual flow of the data throughout the computations. The initial problem was successively broken into smaller and smaller discrete parts until each operation could be programmed easily in the assembly language of the MCS-4. The operational process graph was then used to combine the programmed discrete operations into the required navigation computations. The actual writing of the program was greatly simplified by this method.

The error bound analysis of the microprogram was greatly simplified by the use of process graphs. The maximum error in each of the discrete operations was initially computed.

The summing of these errors as they propagated throughout the program was easily identified with the aid of the operational process graph. The testing of actual erros in the computation, however, proved very difficult. Sample problems were hand calculated with the aid of the process graphs and the results were compared with the results computed by the actual program. When these results did not agree, both the hand calculated procedure and the computer program had to be debugged. This procedure was very time consuming with only a limited number of tests being made. The FORTRAN simulation program was written to aid in testing and debugging the navigation program.

The FORTRAN simulation program proved to be an effective aid in testing the navigation program. The set up of the simulation program allowed the results of the navigation program to be compared at different points in the computations with a FORTRAN solution of the same required equations. With the aid of the IBM 360/67 computer, many tests were run on the FORTRAN simulation program. The results of these runs identified those areas in the navigation program that needed rewriting. Before expanding the navigation program to include negative inputs from different true heading quadrants, the FORTRAN simulation program was changed to incorporate the required changes. After all the changes had been tested and optimized on the simulation program, the MCS-4 navigation program was rewritten to incorporate those changes.

The simulation program proved very effective in optimizing, expanding, debugging, and testing the navigation program.

It is concluded from this design study that many of the dedicated computational tasks being done by large general purpose computers on board naval ships and aircraft can be done by microcomputers. It is recommended that research and development of a multi-microcomputer system be begun to replace one or more of the costly general purpose computer systems currently being used. The development of the MCS-8080 microcomputer with its increased computational speed and power over the MCS-4 will make this task easier.

It is also recommended that before programming a microcomputer, the desired program should be written in a higher level language simulation program before investing in the software costs of a microprogram. The development of PL/M, derived from PL/I, a high level language for the MCS-8 and MCS-8080 greatly increases the ease of programming and testing programs for the microcomputer.

The rapid improvements being made in microcomputers, and the current availability of inexpensive microcomputers, make it imperative that the Navy begin now in the development of systems utilizing this technology.

```
      ************************************************
      THIS IS THE MCS-4 MICROCOMPUTER NAVIGATION PROGRAM
      USED TO COMPUTE THE PRESENT POSITION OF AN AIRCRAFT
      FROM DISTANCE, VELOCITY, AND AIRSPEED INPUTS FROM
      INERTIAL, DOPPLER, AND/OR AIR MASS SYSTEMS;
      ************************************************


      ************************************************
      THIS SECTION CONSISTS OF THE COMMON SUBROUTINES
      USED BY THE NAVIGATION PROGRAM AND THE MULTIPLY
      AND COSINE SUBROUTINES;
      ************************************************


      ************************************************
      THIS SUBROUTINE ADDS THE CONTENTS OF THE RAM MEMORY
      ADDRESSED BY INDEX REGISTERS 0 AND 1 TO THE CONTENTS
      OF INDEX REGISTERS "C" THRU "F" AND STORES THE RESULT
      INTO THE INDEX REGISTERS "C" THRU "F";
      ************************************************

ADDRAMIR:
        CLC;
        SRC R0;
        RDM;
        ADD RC;
        XCH RC;
        INC R1;
        SRC R0;
        RDM;
        ADD RD;
        XCH RD;
        INC R1;
        SRC R0;
        RDM;
        ADD RE;
        XCH RE;
        INC R1;
        SRC R0;
        RDM;
        ADD RF;
        XCH RF;
        CLC;
        BBL 0;


      ************************************************
      THIS SUBROUTINE TAKES A FOUR CHAR HEX NUMBER ADDRESSED
      BY THE CONTENTS OF INDEX REGISTERS 0 AND 1, AND ADDS
      IT TO A FOUR CHAR HEX NUMBER ADDRESSED BY INDEX REGS
      2 AND 3, AND LOADS THE RESULT INTO RAM MEMORY ADDRESS
      BY INDEX REGISTERS 2 AND 3;
      ************************************************

ADDRAM:
        CLC;
        LDM 12;
        XCH R5;
     AD:
        SRC R0;
        RDM;
        SRC R2;
        ADM;
        WRM;
        INC R1;
        INC R3;
        ISZ R5 AD;
        BBL 0;
```

```
      ************************************************
      THIS SUBROUTINE TAKES A FOUR CHARACTER HEX NUMBER
      LOCATED IN INDEX REGISTERS "C" THRU "F", DIVIDES
      THIS NUMBER BY TWO AND STORES THE RESULT IN RAM;
      ************************************************
DIV2IR:
        CLC;
        SRC R0;
        LD RF;
        RAR;
        WRM;
        LDM 14;
        XCH R1;
        SRC R0;
        LD RE;
        RAR;
        WRM;
        LDM 13;
        XCH R1;
        SRC R0;
        LD RD;
        RAR;
        WRM;
        LDM 12;
        XCH R1;
        SRC R0;
        LD RC;
        RAR;
        WRM;
        BBL 0;

      ************************************************
      THIS SUBROUTINE DIVIDES THE CONTENTS OF RAM MEMORY
      ADDRESSED BY INDEX REGISTERS 2 AND 3 BY TWO;
      ************************************************

DIV2:
        CLC;
        SRC R2;
        RDM;
        RAL;
        JCN ZCY PDIV;
        CLC;
        RAR;
        STC;
        JUN NDIV;
     PDIV:
        RAR;
     NDIV:
        RAR;
        WRM;
        LDM 14;
        XCH R3;
        SRC R2;
        RDM;
        RAR;
        WRM;
        LDM 13;
        XCH R3;
        SRC R2;
        RDM;
        RAR;
        WRM;
        LDM 12;
        XCH R3;
        SRC R2;
        RDM;
        RAR;
        WRM;
        BBL 0;
```

```
      ************************************************
      THIS SUBROUTINE LOADS THE DESIRED TIME CYCLE FOR THE
      NAVIGATION ROUTINE INTO INDEX REGISTERS "8" THRU "B";
      ************************************************

TIME:
       LDM 0;
       XCH R8;
       LDM 0;
       XCH R9;
       LDM 0;
       XCH RA;
       LDM 4;
       XCH RB;
       BBL 0;



      ************************************************
      THIS SUBROUTINE LOADS A FOUR CHAR HEX NUMBER FROM
      INDEX REGISTERS "C" THRU "F" INTO RAM MEMORY ADDRESS
      BY INDEX REGISTERS 2 AND 3;
      ************************************************

IRRAMC:
       SRC R2;
       LD RC;
       WRM;
       INC R3;
       SRC R2;
       LD RD;
       WRM;
       INC R3;
       SRC R2;
       LD RE;
       WRM;
       INC R3;
       SRC R2;
       LD RF;
       WRM;
       BBL 0;



      ************************************************
      THIS SUBROUTINES LOADS A FOUR CHAR HEX NUMBER INTO
      INDEX REGISTERS "C" THRU "F" FROM RAM MEMORY LOCATION
      ADDRESSED BY THE CONTENTS OF INDEX REGISTERS 2 AND 3;
      ************************************************
RAMIRC:
       SRC R2;
       RDM;
       XCH RC;
       INC R3;
       SRC R2;
       RDM;
       XCH RD;
       INC R3;
       SRC R2;
       RDM;
       XCH RE;
       INC R3;
       SRC R2;
       RDM;
       XCH RF;
       BBL 0;
```

```
      ************************************************
      THIS SUBROUTINE LOADS A FOUR CHAR HEX NUMBER INTO
      INDEX REGISTERS 8 THRU "B" FROM A RAM MEMORY LOCATION
      ADDRESSED BY INDEX REGISTERS 0 AND 1;
      ************************************************

RAMIR8:
        SRC R0;
        RDM;
        XCH R8;
        INC R1;
        SRC R0;
        RDM;
        XCH R9;
        INC R1;
        SRC R0;
        RDM;
        XCH RA;
        INC R1;
        SRC R0;
        RDM;
        XCH RB;
        BBL 0;
```

```
      ************************************************
      THIS SUBROUTINE SUBTRACTS THE CONTENTS OF INDEX
      REGISTERS "C" THRU "F" FROM THE CONTENTS OF INDEX
      REGISTERS 4 THRU 7 AND STORES THE RESULTS IN INDEX
      REGISTERS "C" THRU "F";
      ************************************************

     SUBIR:
        LD R4;
        SUB RC;
        XCH RC;
        CMC;
        LD R5;
        SUB RD;
        XCH RD;
        CMC;
        LD R6;
        SUB RE;
        XCH RE;
        CMC;
        LD R7;
        SUB RF;
        XCH RF;
        CMC;
        BBL 0;
```

```
      ************************************************
      THIS SUBROUTINE SUBTRACTS THE CONTENTS OF INDEX REGS
      "C" THRU "F" FROM THE CONTENTS OF RAM MEMORY ADDRESSED
      BY INDEX REGS 2 AND 3 WITH THE RESULT LOADED INTO RAM
      MEMORY ADDRESSED BY INDEX REGISTERS 0 AND 1;
      ************************************************

SUBRAMIR:
       CLC;
       SRC R2;
       RDM;
       SUB RC;
       SRC R0;
       WRM;
       CMC;
       INC R3;
       INC R1;
       SRC R2;
       RDM;
       SUB RD;
       SRC R0;
       WRM;
       CMC;
       INC R3;
       INC R1;
       SRC R2;
       RDM;
       SUB RE;
       SRC R0;
       WRM;
       CMC;
       INC R3;
       INC R1;
       SRC R2;
       RDM;
       SUB RF;
       SRC R0;
       WRM;
       CMC;
       BBL 0;
```

```
      ************************************************
      THIS SUBROUTINE TRANSFERS THE CONTENTS OF THE RAM
      MEMORY ADDRESSED BY INDEX REGISTERS 2 AND 3 INTO
      RAM MEMORY ADDRESSED BY INDEX REGISTERS 0 AND 1;
      ************************************************

TRANRAM:
       SRC R2;
       RDM;
       SRC R0;
       WRM;
       INC R1;
       ISZ R3 TRANRAM;
       BBL 0;
```

```
      ************************************************
      THIS SUBROUTINE TAKES A FOUR CHARACTER HEX NUMBER THAT
      IS LOADED IN RAM MEMORY, TAKES THE COMPLEMENT OF THAT
      NUMBER AND RETURNS IT BACK TO RAM MEMORY;
      ************************************************

COMPLEMENT:
        CLB;
        SRC R2;
        RDM;
        CMA;
        IAC;
        WRM;
        INC R3;
        SRC R2;
        LDM 0;
        XCH R0;
        RDM;
        CMA;
        ADD R0;
        WRM;
        INC R3;
        SRC R2;
        RDM;
        CMA;
        ADD R0;
        WRM;
        INC R3;
        SRC R2;
        RDM;
        CMA;
        ADD R0;
        WRM;
        CLB;
        BBL 0;



      ************************************************
      THIS SUBROUTINE FINDS THE COMPLEMENT OF AN ANGLE
      LOADED IN INDEX REGISTERS "C" THRU "F";
      ************************************************

COMANGLE:
        CLC;
        LDM 2;
        SUB RC;
        XCH RC;
        CMC;
        LDM 2;
        SUB RD;
        XCH RD;
        CMC;
        LDM 9;
        SUB RE;
        XCH RE;
        CMC;
        LDM 1;
        SUB RF;
        XCH RF;
        CLB;
        BBL 0;
     ENDSUB: NOP;
```

```
*****************************************************
THIS SECTION IS THE EXECUTIVE PART OF THE
NAVIGATION PROGRAM;
*****************************************************

PROG:                           REM BEGIN NAV CYCLE;
      CLC;                      REM CLEAR CARRY;
      LDM 2;                    REM LOAD INDEX REGISTERS
                                    90 DEG AND TEST THE
                                    TH QUADRANT;

      XCH RC;
      LDM 2;
      XCH RD;
      LDM 9;
      XCH RE;
      LDM 1;
      XCH RF;
      FIM R2 92;                REM LOCATION OF TRUE HEAD;
      FIM R0 72;                REM LOCATION OF TH WORK SPC;
      JMS TRANRAM;              REM TRANSFER TH TO WORK SPC;
      FIM R0 56;                REM LOCATION OF SECOND
                                    WORK SPACE;
      FIM R2 72;                REM TH INPUT;
      JMS SUBRAMIR;             REM TH-90DEG LOADED IN RAM;
      JCN NZCY Q0;              REM JUMP IF TH IN QUAD 0;
      FIM R0 72;                REM TEST IF TH IN QUAD 1;
      FIM R2 56;
      JMS SUBRAMIR;             REM TH-90DEG;
      JCN NZCY Q1;              REM JUMP IF QUAD 1;
      FIM R0 56;                REM NOW TEST FOR QUAD 2;
      FIM R2 72;
      JMS SUBRAMIR;             REM SUB ANOTHER 90DEG;
      JCN NZCY Q2;              REM JUMP IF QUAD 2;
Q3:                             REM TH IN QUAD 3;
      FIM R2 56;                REM TH-270DEG;
      JMS RAMIRC;               REM LOAD IR'S WITH TH-270;
      JMS COSINE;               REM COS(TH-270);
      FIM R0 79;                REM MEMORY SPACE FOR SINE;
      JMS DIV2IR;               REM SHIFT FOR SIGN BIT;
      FIM R2 56;
      JMS RAMIRC;               REM LOAD IR WITH TH-270;
      JMS COMANGLE;             REM COMPLEMENT TH;
      JMS COSINE;               REM COS OF TH IN QUAD 3;
      FIM R0 31;                REM MEMORY LOCATION FOR COS;
      JMS DIV2IR;               REM SHIFT FOR SIGN BIT;
      JUN THSET;                REM QUAD 3 TH SET
                                    JUMP TO MAIN PROGRAM;
Q2:                             REM FIND COS AND SINE FOR TH
                                    IN QUAD 2;

      CLC;
      FIM R2 56;
      JMS COMPLEMENT;           REM SHIFT TH INTO 0-90DEG;
      FIM R2 56;
      JMS RAMIRC;
      JMS COSINE;               REM FIND SIN(TH);
      FIM R0 79;
      JMS DIV2IR;               REM SHIFT FOR SIGN BIT;
      FIM R2 72;
      JMS RAMIRC;
      JMS COSINE;               REM COS(TH);
      FIM R0 31;
      JMS DIV2IR;               REM SHIFT FOR SIGN BIT;
      FIM R2 28;
      JMS COMPLEMENT;           REM COS NEGATIVE IN QUAD 2;
      JUN THSET;                REM JUMP TO MAIN PROGRAM;
Q1:                             REM COS AND SIN IN QUAD 1;
      CLC;
      FIM R2 72;
      JMS COMPLEMENT;           REM SHIFT TH INTO 0-90DEG;
      FIM R2 72;
      JMS RAMIRC;
```

```
      JMS COSINE;              REM FIND COS(TH);
      FIM R0 31;
      JMS DIV2IR;              REM SHIFT FOR SIGN BIT;
      FIM R2 28;
      JMS COMPLEMENT;          REM COS(TH) NEG IN QUAD 1;
      FIM R2 56;
      JMS RAMIRC;
      JMS COSINE;              REM FIND SIN(TH);
      FIM R0 79;
      JMS DIV2IR;              REM SHIFT FOR SIGN BIT;
      FIM R2 76;
      JMS COMPLEMENT;          REM SIN(TH) NEG IN QUAD 1;
      JUN THSET;               REM JUMP TO MAIN PROGRAM;
   Q0:                         REM COS AND SIN IN QUAD 0;
      CLC;
      FIM R2 56;
      JMS COMPLEMENT;          REM SHIFT TH INTO 0-90DEG;
      FIM R2 56;
      JMS RAMIRC;
      JMS COSINE;              REM SIN(TH);
      FIM R0 79;
      JMS DIV2IR;              REM SHIFT FOR SIGN;
      FIM R2 76;
      JMS COMPLEMENT;          REM SIN(TH) NEG IN QUAD 0;
      FIM R2 72;
      JMS RAMIRC;
      JMS COSINE;              REM COS(TH);
      FIM R0 31;
      JMS DIV2IR;

THSET:                         REM START MAIN PROGRAM;
      CLB;                     REM CLEAR AC AND CY;
      FIM R0 32;               REM MEM LOC OF INPUT TAS;
      FIM R2 12;               REM MEM LOC OF SMOOTH TAS;
      JMS ADDRAM;              REM INPUT TAS ADDED TO
                                   SMOOTH TAS;
      FIM R2 15;               REM MEM LOC OF SUM;
      JMS DIV2;                REM DIV SUM BY 2;
      FIM R0 12;               REM MEM LOC NEW SMOOTH TAS;
      JMS RAMIR8;              REM LOAD TAS INTO IR8-IRB;
      FIM R2 28;               REM MEM LOC OF COS(TH);
      JMS RAMIRC;              REM LOAD COS(TH);
      JMS MULT;                REM MULT TAS*COS(TH)=VAY;
   DYINS:                      REM TEST IF INS GOOD;
      FIM R2 0;                REM MEM LOC INS FLAG;
      SRC R2;
      RDM;                     REM LOAD FLAG INTO AC;
      JCN ZAC DNIY;            REM JUMP IF INS DOWN;
   INSUP:                      REM INS UP COMPUTE WIND;
      FIM R2 60;               REM MEM LOC VAY;
      JMS IRRAMC;              REM LOAD VAY INTO RAM MEM;
      FIM R2 60;               REM MEM LOC OF VAY;
      JMS COMPLEMENT;          REM NEG OF VAY;
      FIM R0 68;               REM MEM LOC OF VGYI;
      FIM R2 60;               REM SET WIND SMOOTHING ROUT;
      JMS ADDRAM;              REM (VGYI-VAY)=VWYR;
      FIM R2 60;               REM MEM LOC VWYR;
      FIM R0 52;               REM MEM LOC VWY;
      JMS ADDRAM;              REM VWYR+VWY;
      FIM R2 60;               REM MEM LOC (VWYR+VWY);
      FIM R0 52;               REM MEM LOC VWY;
      JMS ADDRAM;              REM (VWYR+VWY)+VWY;
      FIM R2 60;               REM MEM (VWYR+VWY+VWY);
      FIM R0 52;               REM MEM VWY;
      JMS ADDRAM;              REM (VWYR+VWY+VWY)+VWY;
      FIM R2 63;               REM MEM (VWYR+3VWY)
      JMS DIV2;                REM (VWYR+3VWY)/2;
      FIM R2 63;
      JMS DIV2;                REM (VWYR+3VWY)/4;
      FIM R0 52;               REM MEM LOC VWY;
      FIM R2 60;               REM MEM LOC (VWYR+3VWY)/4;
      JMS TRANRAM;             REM VWY=(VWYR+3VWY)/4;
```

```
DYI:
   FIM R2 68;                    REM COMPUTE DYI;
   JMS RAMIRC;                   REM MEM LOC VGYI;
   JMS TIME;                     REM VGYI LOADED IRC-IRF;
   JMS MULT;                     REM TIME INC IR8-IRB;
   FIM R2 24;                    REM (VGYI)*(TIME)=DYI;
   JMS IRRAMC;                   REM MEM LOC DY;
   JUN UPIY;                     REM DY=DYI;
                                 REM JUMP TO COMPUTE
                                     VWX AND DXI;
DNIY:                            REM COMPUTE DYW;
   FIM R0 52;                    REM MEM LOC VWY;
   JMS ADDRAMIR;                 REM (VAY+VWY)=VGYW;
   JMS TIME;                     REM LOAD TIME INCR;
   JMS MULT;                     REM (VGYW)*(TIME)=DYW;
   FIM R2 24;                    REM MEM LOC DY;
   JMS IRRAMC;                   REM DY=DYW;
UPIY:                            REM COMPUTE VWX AND DXI;
   FIM R2 76;                    REM MEM LOC SIN(TH);
   JMS RAMIRC;                   REM LOAD SIN(TH);
   FIM R0 12;                    REM MEM LOC TAS;
   JMS RAMIR8;                   REM LOAD TAS;
   JMS MULT;                     REM (TAS)*(SIN(TH))=VAX;
DXINS:                           REM TEST IF INERTIAL GOOD;
   FIM R2 0;                     REM INERTIAL FLAG;
   SRC R2;
   RDM;                          REM READ FLAG INTO AC;
   JCN ZAC DNIX;                 REM JUMP IF INERTIAL DOWN;
   FIM R2 44;                    REM MEM LOC VAX;
   JMS IRRAMC;                   REM STORE VAX;
   FIM R2 44;                    REM MEM LOC VAX;
   JMS COMPLEMENT;               REM NEGATE VAX;
   FIM R2 44;                    REM (-VAX);
   FIM R0 64;                    REM (VGXI);
   JMS ADDRAM;                   REM (VGXI)+(-VAX)=VWXR;
   FIM R2 44;                    REM (VWX);
   FIM R0 48;                    REM (VWX);
   JMS ADDRAM;                   REM (VWXR)+(VWX);
   FIM R2 44;                    REM (VWXR+VWX);
   FIM R0 48;                    REM (VWX);
   JMS ADDRAM;                   REM (VWXR+VWX)+(VWX);
   FIM R2 44;                    REM (VWXR+2VWX);
   FIM R0 48;                    REM (VWX);
   JMS ADDRAM;                   REM (VWXR+2VWX)+(VWX);
   FIM R2 47;                    REM (VWXR+3VWX);
   JMS DIV2;                     REM (VWXR+3VWX)/2;
   FIM R2 47;                    REM ((VWXR+3VWX)/2);
   JMS DIV2;                     REM ((VWXR+3VWX)/2)/2;
   FIM R0 48;                    REM (VWX);
   FIM R2 44;                    REM ((VWXR+3VWX)/4);
   JMS TRANRAM;                  REM VWX=((VWXR+3VWX)/4);
DXI:                             REM COMPUTE DXI;
   FIM R2 64;                    REM (VGXI);
   JMS RAMIRC;                   REM LOAD VGXI;
   JMS TIME;                     REM LOAD TIME INCR;
   JMS MULT;                     REM (VGXI)*(TIME)=DXI;
   FIM R2 8;                     REM DX;
   JMS IRRAMC;                   REM DX=DXI;
   JUN UPIX;                     REM COMPUTE DYD AND DXD;
DNIX:                            REM COMPUTE DXW IF INS DOWN;
   FIM R0 48;                    REM (VWX);
   JMS ADDRAMIR;                 REM (VWX)+(VAX)=VGXW;
   JMS TIME;                     REM LOAD TIME INCR;
   JMS MULT;                     REM (VGXW)*(TIME)=DXW;
   FIM R2 8;                     REM (DX);
   JMS IRRAMC;                   REM DX=DXW;
UPIX:                            REM COMPUTE DYD AND DXD;
   FIM R2 1;                     REM (DOPPLER FLAG);
   SRC R2;
   RDM;                          REM LOAD DOPP FLAG INTO AC;
   JCN ZAC DOWND;                REM JUMP IF DOPP DOWN;
   FIM R2 76;                    REM (SIN(TH));
   JMS RAMIRC;                   REM LOAD SIN(TH);
```

```
   FIM R0 84;              REM (SDD);
   JMS RAMIR8;             REM LOAD SDD;
   JMS MULT;               REM (SDD)*(SIN(TH));
   FIM R2 96;              REM (SDD*SIN(TH));
   JMS IRRAMC;             REM STORE SDD*SIN(TH);
   FIM R2 96;              REM (SDD*SIN(TH));
   JMS COMPLEMENT;         REM NEG SDD*SIN(TH);
   FIM R2 28;              REM (COS(TH));
   JMS RAMIRC;             REM LOAD COS(TH);
   FIM R0 80;              REM (SHD);
   JMS RAMIR8;             REM LOAD SHD;
   JMS MULT;               REM (SHD)*(COS(TH));
   FIM R0 96;              REM -(SDD*SIN(TH));
   JMS ADDRAMIR;           REM SHD*COS(TH)-SDD*SIN(TH)
                               = DYD;
   FIM R2 140;             REM (DYD);
   JMS IRRAMC;             REM STORE DYD;
   FIM R2 28;              REM (COS(TH));
   JMS RAMIRC;             REM LOAD COS(TH);
   FIM R0 84;              REM (SDD);
   JMS RAMIR8;             REM LOAD SDD;
   JMS MULT;               REM (SDD)*COS(TH);
   FIM R2 100;             REM WORK AREA;
   JMS IRRAMC;             REM STORE SDD*COS(TH);
   FIM R2 76;              REM (SIN(TH));
   JMS RAMIRC;             REM LOAD SIN(TH);
   FIM R0 80;              REM (SHD);
   JMS RAMIR8;             REM LOAD SHD;
   JMS MULT;               REM (SHD)*(SIN(TH));
   FIM R0 100;             REM (SDD*COS(TH));
   JMS ADDRAMIR;           REM SHD*SIN(TH)+SDD*COS(TH)
                               = DXD;
   FIM R2 0;               REM TEST INERTIAL FLAG;
   SRC R2;
   RDM;                    REM LOAD INS FLAG INTO AC;
   JCN ZAC DOWNI;          REM JUMP IF INS DOWN;
   FIM R0 8;               REM (DX);
   JMS ADDRAMIR;           REM (DX)+(DXD);
   FIM R0 127;             REM (DX WORK AREA);
   JMS DIV2IR;             REM LOAD (DX+DXD)/2;
   FIM R2 124;             REM (DX+DXD)/2;
   FIM R0 8;               REM (DX);
   JMS TRANRAM;            REM DX=(DX+DXD)/2;
   FIM R0 24;              REM (DY);
   FIM R2 140;             REM (DYD);
   JMS ADDRAM;             REM (DY)+(DYD);
   JMS DIV2;               REM (DY+DYD)/2;
   FIM R0 24;              REM (DY);
   JMS TRANRAM;            REM DY=(DY+DYD)/2;
   JUN DOWND;              REM JUMP TO END OF CYCLE;
DOWNI:                     REM IF INS DOWN USE DXD
                               AND DYD;
   FIM R2 8;               REM (DX);
   JMS IRRAMC;             REM DX=DXD;
   FIM R2 140;             REM (DYD);
   FIM R0 24;              REM (DY);
   JMS TRANRAM;            REM DY=DYD;

DOWND:                     REM END OF NAV CYCLE;
   JUN PROG;               REM RETURN AND START
                               NEXT NAV CYCLE;
```

```
*************************************************
THIS SECTION CONTAINS THE NAVIGATION SUBROUTINES
*************************************************


*************************************************
THIS IS THE TABLE OF VALUES USED BY THE COSINE ROUTINE
THE FIRST COLUMN CONSISTS OF THE VALUE OF THE COSINE
FROM 0 TO 90 DEGREES IN INCREMENTS OF .08 HEX RADIANS;
THE SECOND COLUMN CONSISTS OF VALUES USED TO
INTERPOLATE BETWEEN VALUES IN THE FIRST COLUMN;
*************************************************

CON  "FFFF";          CON  "0040";
CON  "0EFF";          CON  "FFB0";
CON  "08FF";          CON  "BF31";
CON  "0EEF";          CON  "2F81";
CON  "10EF";          CON  "1E32";
CON  "2ECF";          CON  "8CB2";
CON  "38BF";          CON  "4A33";
CON  "6E9F";          CON  "37B3";
CON  "B08F";          CON  "3334";
CON  "1F5F";          CON  "3EA4";
CON  "A93F";          CON  "F725";
CON  "601F";          CON  "70A5";
CON  "63EE";          CON  "9716";
CON  "A2BE";          CON  "2D86";
CON  "3E7E";          CON  "1107";
CON  "364E";          CON  "4377";
CON  "9A0E";          CON  "93E7";
CON  "78CD";          CON  "F158";
CON  "F88D";          CON  "3EB8";
CON  "F24D";          CON  "4829";
CON  "B9FC";          CON  "1099";
CON  "3DAC";          CON  "85F9";
CON  "9D5C";          CON  "685A";
CON  "CA0C";          CON  "C8BA";
CON  "05BB";          CON  "661B";
CON  "5C5B";          CON  "417B";
CON  "C00B";          CON  "49CB";
CON  "82AA";          CON  "5E1C";
CON  "814A";          CON  "607C";
CON  "0ED9";          CON  "5FBC";
CON  "1879";          CON  "1B0D";
CON  "BF09";          CON  "835D";
CON  "15A8";          CON  "B89D";
CON  "5838";          CON  "7ADD";
CON  "89C7";          CON  "C81E";
CON  "B857";          CON  "835E";
CON  "26E6";          CON  "BA8E";
CON  "C176";          CON  "4EBE";
CON  "DBF5";          CON  "2EEE";
CON  "6485";          CON  "4A1F";
CON  "9805";          CON  "924F";
CON  "8194";          CON  "276F";
CON  "4614";          CON  "D78F";
CON  "0A93";          CON  "A4AF";
CON  "CC13";          CON  "9D8F";
CON  "0F92";          CON  "82DF";
CON  "6022";          CON  "83EF";
CON  "41A1";          CON  "90FF";
CON  "C121";          CON  "A9FF";
CON  "F1A0";          CON  "BEFF";
```

```
      ************************************************
      THIS SUBROUTINE COMPUTES THE COSINE OR SINE OF A VALUE
      INPUTTED IN HEX RADIANS USING A SECOND ORDER NEWTON
      DIVIDED-DIFFERENCE INTERPOLATING ROUTINE WITH TABLE
      LOOKUP;
      ************************************************


COSINE:CLB;
       FIM R0 2;
       SRC R0;
       LDM 1;
       WRM;
       LDM 0;
       XCH RF;
       RAR;
       XCH RE;
       RAR;
       XCH R0;
       JCN NZCY CR;
       LDM 0;
       XCH R1;
    NCR:
       LD RD;
       RAL;
       JCN NZCY ACR;
       JUN OUT;
    CR:
       LDM 8;
       XCH R1;
       JUN NCR;
    ACR:
       CLC;
       RAR;
       XCH RD;
       LDM 4;
       ADD R1;
       XCH R1;
    OUT:
       FIN R4;
       INC R1;
       FIN R6;
       INC R1;
       FIN R8;
       INC R1;
       FIN RA;
       LD RD;
       XCH RE;
       LDM 0;
       XCH RC;
       XCH RD;
       JMS MULT;
       JMS SUBIR;
       FIM R0 2;
       SRC R0;
       LDM 0;
       WRM;
       BBL 0;
```

```
;*****************************************************
;THIS ROM CONTAINS A 16X16 HEX MULTIPLICATION TABLE.
;EACH VALUE REPRESENTS A MULTIPLICATION OF TWO SINGLE
;HEX-DIGIT NUMBERS.THE ZERO ROW IS USED FOR TABLE
;INSTRUCTIONS;
;*****************************************************

TABLE:  FIN R0;     BBL 0;     NOP;       NOP;
        NOP;        NOP;       NOP;       NOP;
        NOP;        NOP;       NOP;       NOP;
        NOP;        NOP;       NOP;       NOP;
  ROW1: CON "00"; CON "01"; CON "02"; CON "03";
        CON "04"; CON "05"; CON "06"; CON "07";
        CON "08"; CON "09"; CON "0A"; CON "0B";
        CON "0C"; CON "0D"; CON "0E"; CON "0F";
  ROW2: CON "00"; CON "02"; CON "04"; CON "06";
        CON "08"; CON "0A"; CON "0C"; CON "0E";
        CON "10"; CON "12"; CON "14"; CON "16";
        CON "18"; CON "1A"; CON "1C"; CON "1E";
  ROW3: CON "00"; CON "03"; CON "06"; CON "09";
        CON "0C"; CON "0F"; CON "12"; CON "15";
        CON "18"; CON "18"; CON "1E"; CON "21";
        CON "24"; CON "27"; CON "2A"; CON "2D";
  ROW4: CON "00"; CON "04"; CON "08"; CON "0C";
        CON "10"; CON "14"; CON "18"; CON "1C";
        CON "20"; CON "24"; CON "28"; CON "2C";
        CON "30"; CON "34"; CON "38"; CON "3C";
  ROW5: CON "00"; CON "05"; CON "0A"; CON "0F";
        CON "14"; CON "19"; CON "1E"; CON "23";
        CON "28"; CON "2D"; CON "32"; CON "37";
        CON "3C"; CON "41"; CON "46"; CON "4B";
  ROW6: CON "00"; CON "06"; CON "0C"; CON "12";
        CON "18"; CON "1E"; CON "24"; CON "2A";
        CON "30"; CON "36"; CON "3C"; CON "42";
        CON "48"; CON "4E"; CON "54"; CON "5A";
  ROW7: CON "00"; CON "07"; CON "0E"; CON "15";
        CON "1C"; CON "23"; CON "2A"; CON "31";
        CON "38"; CON "3F"; CON "46"; CON "4D";
        CON "54"; CON "5B"; CON "62"; CON "69";
  ROW8: CON "00"; CON "08"; CON "10"; CON "18";
        CON "20"; CON "28"; CON "30"; CON "38";
        CON "40"; CON "48"; CON "50"; CON "58";
        CON "60"; CON "68"; CON "70"; CON "78";
  ROW9: CON "00"; CON "09"; CON "12"; CON "1B";
        CON "24"; CON "2D"; CON "36"; CON "3F";
        CON "48"; CON "51"; CON "5A"; CON "63";
        CON "6C"; CON "75"; CON "7E"; CON "87";
  ROWA: CON "00"; CON "0A"; CON "14"; CON "1E";
        CON "28"; CON "32"; CON "3C"; CON "46";
        CON "50"; CON "5A"; CON "64"; CON "6E";
        CON "78"; CON "82"; CON "8C"; CON "96";
  ROWB: CON "00"; CON "0B"; CON "16"; CON "21";
        CON "2C"; CON "37"; CON "42"; CON "4D";
        CON "58"; CON "63"; CON "6E"; CON "79";
        CON "84"; CON "8F"; CON "9A"; CON "A5";
  ROWC: CON "00"; CON "0C"; CON "18"; CON "24";
        CON "30"; CON "3C"; CON "48"; CON "54";
        CON "60"; CON "6C"; CON "78"; CON "84";
        CON "90"; CON "9C"; CON "A8"; CON "B4";
  ROWD: CON "00"; CON "00"; CON "1A"; CON "27";
        CON "34"; CON "41"; CON "4E"; CON "5B";
        CON "68"; CON "75"; CON "82"; CON "8F";
        CON "9C"; CON "A9"; CON "B6"; CON "C3";
  ROWE: CON "00"; CON "0E"; CON "1C"; CON "2A";
        CON "38"; CON "46"; CON "54"; CON "62";
        CON "70"; CON "7E"; CON "8C"; CON "9A";
        CON "A8"; CON "B6"; CON "C4"; CON "D2";
  ROWF: CON "00"; CON "0F"; CON "1E"; CON "2D";
        CON "3C"; CON "4B"; CON "5A"; CON "69";
        CON "78"; CON "87"; CON "96"; CON "A5";
        CON "B4"; CON "C3"; CON "D2"; CON "E1";
```

```
      ************************************************
      THIS SUBROUTINE MULTIPLIES A FOUR HEX-DIGIT NUMBER
      BY A HEX-DIGIT NUMBER RESULTING IN A FOUR HEX-DIGIT
      CHOPPED NUMBER;
      ************************************************

MULT: CLC;
      FIM R2 160;          REM ZERO SIGN COUNTER;
      SRC R2;
      LDM 0;
      WRM;
      LDM 4;               REM STORE CONTENTS OF INDEX
                               REGISTERS INTO MEMORY;
XCH R3; SRC R2; LD R4; WRM;
INC R3; SRC R2; LD R5; WRM;
INC R3; SRC R2; LD R6; WRM;
INC R3; SRC R2; LD R7; WRM;
INC R3; SRC R2; LD R8; WRM;
INC R3; SRC R2; LD R9; WRM;
INC R3; SRC R2; LD RA; WRM;
INC R3; SRC R2; LD RB; WRM;
INC R3; SRC R2; LD RC; WRM;
INC R3; SRC R2; LD RD; WRM;
INC R3; SRC R2; LD RE; WRM;
INC R3; SRC R2; LD RF; WRM;

REM   THIS SECTION OF THE MULTIPLY SUBROUTINE
      TESTS THE TYPE AND SIGN OF THE INPUT VALUES,
      FIM R0 2;            REM LOAD COS FLAG;
      SRC R0;
      RDM;
      JCN NZAC COS;        REM JUMP IF COS;
      LDM 15;              REM TEST SIGN OF FIRST INPUT;
      XCH R3;
      SRC R2;
      RDM;
      RAL;
      JCN ZCY PM;          REM JUMP IF POSITIVE;
      LDM 12;              REM COMPLEMENT NEGATIVE;
      XCH R3;
      JMS COMPLEMENT;
      LDM 0;               REM SET SIGN COUNTER;
      XCH R3;
      SRC R2;
      LDM 8;
      WRM;
   PM:                     REM TEST SIGN OF SECOND INPUT;
      LDM 11;
      XCH R3;
      SRC R2;
      RDM;
      RAL;
      JCN ZCY NM;          REM JUMP IF POSITIVE;
      LDM 8;               REM COMPLEMENT NEG INPUT;
      XCH R3;
      JMS COMPLEMENT;
      LDM 0;               REM SET SIGN COUNTER;
      XCH R3;
      SRC R2;
      LDM 8;
      ADM;
      WRM;
   NM:                     REM NORMALIZE INPUTS;
      LDM 8;
      XCH R3;
      CLC;
    MULT2:
      SRC R2;
      RDM;
      RAL;
      WRM;
      ISZ R3 MULT2;
```

```
REM   THIS BEGINS THE MAIN PORTION OF MULTIPLY SUBROUTINE.
      INDEX REGISTERS 0 AND 1 ARE USED TO FETCH VALUES FROM
      MULTIPLICATION TABLE.THE TABLE VALUES ARE ADDED AS IN
      LONG-HAND MULTIPLICATION;


     COS:
       LDM 15; XCH R3; SRC R2; RDM;
       JCN ZAC POS1;
       XCH R0;
       LDM  8; XCH R3; SRC R2; RDM;
       XCH R1;
       JMS TABLE;
       XCH R0;
       XCH R4;
       JUN LOC1;
POS1:  XCH R4;
LOC1:  LDM 14; XCH R3; SRC R2; RDM;
       JCN ZAC POS2;
       XCH R0;
       LDM  9; XCH R3; SRC R2; RDM;
       XCH R1;
       JMS TABLE;
       XCH R0;
       XCH R5;
       JUN LOC2;
POS2:  XCH R5;
LOC2:  LD R4;
       ADD R5;
       XCH R4;
       TCC;
       XCH R7;
       LDM 15; XCH R3; SRC R2; RDM;
       JCN ZAC POS3;
       XCH R0;
       LDM  9; XCH R3; SRC R2; RDM;
       XCH R1;
       JMS TABLE;
       XCH R1;
       XCH R5;
       XCH R0;
       XCH R6;
       JUN LOC3;
POS3:  XCH R5;
       LDM 0;
       XCH R6;
LOC3:  LD R4;
       ADD R5;
       XCH R4;
       TCC;
       XCH R5;
       LDM 14; XCH R3; SRC R2; RDM;
       JCN ZAC POS4;
       XCH R0;
       LDM 10; XCH R3; SRC R2; RDM;
       XCH R1;
       JMS TABLE;
       XCH R1;
       XCH R8;
       XCH R0;
       XCH R9;
       JUN LOC4;
POS4:  XCH R8;
       LDM 0;
       XCH R9;
LOC4:  LD R6;
       ADD R7;
       ADD R9;
       XCH R6;
       TCC;
       XCH R7;
```

```
        LDM 15; XCH R3; SRC R2; RDM;
        JCN ZAC POS5;
        XCH R0;
        LDM 10; XCH R3; SRC R2; RDM;
        XCH R1;
        JMS TABLE;
        XCH R1;
        XCH R9;
        XCH R0;
        XCH RA;
        JUN LOC5;
POS5:   XCH R9;
        LDM 0;
        XCH RA;
LOC5:   LD R5;
        ADD R6;
        XCH R5;
        TCC;
        XCH R6;
        LD R5;
        ADD R9;
        XCH R5;
        TCC;
        ADD R6;
        XCH R6;
        LDM 14; XCH R3; SRC R2; RDM;
        JCN ZAC POS6;
        XCH R0;
        LDM 11; XCH R3; SRC R2; RDM;
        XCH R1;
        JMS TABLE;
        XCH R1;
        XCH RB;
        XCH R0;
        XCH R9;
        JUN LOC6;
POS6:   XCH RB;
        LDM 0;
        XCH R9;
LOC6:   LD R7;
        ADD R9;
        ADD RA;
        XCH R7;
        TCC;
        XCH R9;
        LDM 15; XCH R3; SRC R2; RDM;
        JCN ZAC POS7;
        XCH R0;
        LDM 11; XCH R3; SRC R2; RDM;
        XCH R1;
        JMS TABLE;
        XCH R1;
        XCH RA;
        XCH R0;
        XCH RC;
        JUN LOC7;
POS7:   XCH RA;
        LDM 0;
        XCH RC;
LOC7:   LD R9;
        ADD RC;
        XCH R9;
        LD R6;
        ADD R7;
        XCH R6;
        TCC;
        XCH R7;
        LD R6;
        ADD RA;
        XCH R6;
        TCC;
        ADD R7;
```

```
        ADD R9;
        XCH RF;
        LDM 13; XCH R3; SRC R2; RDM;
        JCN ZAC POS8;
        XCH R0;
        LDM 10; XCH R3; SRC R2; RDM;
        XCH R1;
        JMS TABLE;
        XCH R0;
        XCH R7;
        JUN LOC8;
POS8:   XCH R7;
LOC8:   LD R4;
        ADD R7;
        XCH R4;
        TCC;
        XCH R7;
        LD R4;
        ADD R8;
        XCH R4;
        TCC;
        XCH R8;
        LDM 13; XCH R3; SRC R2; RDM;
        JCN ZAC POS9;
        XCH R0;
        LDM 11; XCH R3; SRC R2; RDM;
        XCH R1;
        JMS TABLE;
        XCH R1;
        XCH R9;
        XCH R0;
        XCH RA;
        JUN LOC9;
POS9:   XCH R9;
        LDM 0;
        XCH RA;
LOC9:   LD RA;
        ADD R7;
        ADD R5;
        XCH R5;
        TCC;
        ADD R6;
        XCH R6;
        TCC;
        XCH RA;
        LDM 12; XCH R3; SRC R2; RDM;
        JCN ZAC POSA;
        XCH R0;
        LDM 11; XCH R3; SRC R2; RDM;
        XCH R1;
        JMS TABLE;
        XCH R0;
        XCH R7;
        JUN LOCA;
POSA:   XCH R7;
LOCA:   LD R7;
        ADD R9;
        XCH R7;
        TCC;
        ADD R5;
        XCH R5;
        TCC;
        XCH RE;
        LD R5;
        ADD R8;
        XCH R5;
        TCC;
        ADD RE;
        ADD R6;
        XCH RE;
        TCC;
        ADD RF;
```

```
        XCH RF;
        LD R4;
        ADD R7;
        XCH RC;
        TCC;
        ADD R8;
        ADD R5;
        XCH RD;
        TCC;
        ADD RE;
        XCH RE;
        TCC;
        ADD RA;
        ADD RF;
        XCH RF;

                                REM END TABLE LOOKUP MULT,PRODUCT
                                    LOCATED IN IR'S "C" THRU "F";

        CLB;
        FIM R0 2;               REM TEST COS FLAG;
        SRC R0;
        RDM;
        JCN NZAC CSIN;          REM JUMP IF COS;
        FIM R0 111;             REM SHIFT FOR SIGN BIT;
        JMS DIV2IR;
        FIM R2 108;
        FIM R0 168;
        JMS TRANRAM;
        CLB;
        FIM R2 160;             REM TEST SIGN OF PRODUCT;
        SRC R2;
        RDM;
        RAL;
        JCN ZCY PPROD;          REM JUMP IF POSITIVE;
        FIM R2 168;             REM COMPLEMENT PRODUCT;
        JMS COMPLEMENT;

    PPROD:                      REM LOAD PRODUCT INTO IRC TO IRF;
        FIM R2 168;
        JMS RAMIRC;
        JUN ENDMULT;

    CSIN:                       REM LOAD INDEX REGISTERS 4 THRU 7
                                    WITH INITIAL VALUE BEFORE
                                    MULT ROUTINE CALLED;
        LDM 4 ; XCH R3; SRC R2; RDM;
        XCH R4;
        INC R3;
        SRC R2;
        RDM;
        XCH R5;
        INC R3;
        SRC R2;
        RDM;
        XCH R6;
        INC R3;
        SRC R2;
        RDM;
        XCH R7;
    ENDMULT: BBL 0;
END
```

APPENDIX D

```
C*****************************************************
C THIS PROGRAM SIMULATES THE MICROCOMPUTER
C IN THE NAVIGATIONAL PROGRAM TO COMPUTE
C THE PRESENT POSITION  OF THE AIRCRAFT
C*****************************************************


      IMPLICIT INTEGER   (A-B,D-V)
      DIMENSION R(16),RAM(16,16),ROM(16,32),Y(53),Z(52),T(4)
      DATA XTAS/0.400/,XVGYI/-.2774/,XVGXI/0.3236/,XVWY/00.0
     1/,IF/0/,DF/0/,XTASR/0.400/,XTH/0.7854/,XSDD/.01810/,XS
     2HD/.1418/,XT/0.347/
      DO 1 I=1,16
      R(I)=0
      DO 1 J=1,16
    1 RAM(I,J)=0
      I=2
      Y(1)=.99999
      X=.03125
   30 Y(I)=COS(X)
      X=X+.03125
      I=I+1
      IF(X.GT.1.6325)  GO TO 40
      GO TO 30
   40 DO 60 J=1,52
   60 Z(J)=(Y(J)-Y(J+1))/.03125
      L=1
      DO 63 J=1,13
      DO 63 K=1,7,2
      FIX1=Y(L)*65536.0*16.0
      FIX2=Z(L)*65536.0*16.0
      REM=FIX1-(FIX1/16)*16
      FIX1=FIX1/16+REM/8
      REM=FIX2-(FIX2/16)*16
      FIX2=FIX2/16+REM/8
      DO 64 I=1,4
      ROM(J,4*(K-1)+I)=FIX1-(FIX1/16 )*16
      FIX1=FIX1/16
      ROM(J,4*K+I)=FIX2-(FIX2/16)*16
   64 FIX2=FIX2/16
      L=L+1
   63 CONTINUE
      DO 1001 IF=1,2
      DO 1002 DF=1,2
      XTH= 3.1415/4.0
      XTH= XTH+ 0.0087266
      XVGXI=-XVGXI
      XVGYI=-XVGYI
      XLOCF = 0.0
      YLOCF = 0.0
      XLOCM = 0.0
      YLOCM = 0.0
      DO 100 M=1,10
      CALL NAV(R,RAM,ROM, T,TE,XTAS,XVGYI,XVGXI,XVWY,XVWX,IF
     1 XTASR,XTH,XSDD,XSHD,XT,XLOCF,YLOCF,XLOCM,YLOCM)
  100 CONTINUE
      XTH=3.1415 + XTH
      XVGYI = -XVGYI
      XVGXI = -XVGXI
      DO 110 M=1,10
      CALL NAV(R,RAM,ROM, T,TE,XTAS,XVGYI,XVGXI,XVWY,XVWX,IF
     1 XTASR,XTH,XSDD,XSHD,XT,XLOCF,YLOCF,XLOCM,YLOCM)
  110 CONTINUE
 1002 CONTINUE
 1001 CONTINUE
      RETURN
      END
```

```
      ************************************************
      THIS SUBROUTINE SIMULATES THE FUNCTION
      OF THE MICROCOMPUTER EXECUTIVE ROUTINE
      ************************************************


      SUBROUTINE  NAV(R,RAM,ROM,T,TE,XTAS,XVGYI,
     1XVGXI,XVWY,XVWX,IF,DF,XTASR,
     2XTH,XSDD,XSHD,XT,XLOCF,YLOCF,XLOCM,YLOCM)
      IMPLICIT INTEGER (A-B,D-V)
      DIMENSION R(16), RAM(16,16), ROM(16,32), T(4), TE(4)
      CALL CONVRT(1,13,XTASR,RAM,R,0)
      CALL CONVRT(1,5,XT,RAM,R,0)
      CALL CONVRT(3,1,XTAS,RAM,R,0)
      CALL CONVRT(3,5,XTH,RAM,R,1)
      CALL CONVRT(4,1,XVWX,RAM,R,0)
      CALL CONVD(4,1,RAM,R,XVA,0)
      WRITE (6,103) XVA
  103 FORMAT(10X,'VWX=',F15.6)
      CALL CONVRT(4,5,XVWY,RAM,R,0)
      CALL CONVRT(5,1,XVGXI,RAM,R,0)
      CALL CONVRT(5,5,XVGYI,RAM,R,0)
      CALL CONVRT(6,1,XSHD,RAM,R,0)
      CALL CONVRT(6,5,XSDD,RAM,R,0)
      XSIN=-COS(3.1415/2.0-XTH)
      XCOS=COS(XTH)
      WRITE(6,200) XSIN,XCOS
  200 FORMAT(10X,'XSIN=',F10.6,2X,'XCOS=',F10.6)
      CALL COSPL(R,ROM,RAM)

C         SIN  AND  COS  HAVE  JUST  BEEN  CALCULATED

      CALL CONVD(5,13,RAM,R,XVAL1,0)
      CALL CONVD(2,13,RAM,R,XVAL2,0)
      WRITE(6,200) XVAL1,XVAL2
      XTAS=(XTASR+XTAS)/2.0
      XVAY=XTAS*XCOS
      XVAX=XTAS*XSIN
      WRITE(6,201) XTAS,XVAY,XVAX
  201 FORMAT(10X,'XTAS=',F10.3,2X,'XVAY=',F10.3,'XVAX=',F10.
     13)
C     DO  THE  ABOVE  CALCULATIONS  WITH  MCS-4
      R(3)=3
      R(4)=1
      CALL DIV2(R,RAM)
      R(3)=1
      R(4)=13
      CALL DIV2(R,RAM)
      R(1)=3
      R(2)=1
      R(3)=1
      R(4)=13
      CALL ADDRAM(R,RAM)
      R(3)=1
      R(4)=13
      CALL RAMIRC(R,RAM)
      R(1)=1
      R(2)=13
      R(3)=2
      R(4)=13
      CALL MULTPN(R,RAM)
      R(3)=4
      R(4)=9
      CALL IRRAMC(R,RAM)
      R(3)=5
      R(4)=13
      CALL MULTPN(R,RAM)
      R(3)=5
      R(4)=9
      CALL IRRAMC(R,RAM)
      CALL CONVD(4,9,RAM,R,XVAL1,0)
      CALL CONVD(5,9,RAM,R,XVAL2,0)
```

```
      WRITE (6,201) XTAS, XVAL1,XVAL2
C     WE HAVE JUST COMPARED OUTPUTS VAY AND VAX
      IF(DF.NE.0) GO TO 600
      XDXD=XSDD*XCOS+XSHD*XSIN
      XDYD=XSHD*XCOS-XSDD*XSIN
      WRITE (6,202) XDXD,XDYD
  202 FORMAT( 10X,'XDXD=',F10.4,2X,'XDYD=',F10.4)
C     DO THE  ABOVE STATEMENTS ON THE MICROCOMPUTER
      R(1)=6
      R(2)=5
      R(3)=2
      R(4)=13
      CALL MULTPN(R,RAM)
      R(3)=7
      R(4)=5
      CALL IRRAMC(R,RAM)
      R(1)=6
      R(2)=1
      R(3)=2
      R(4)=13
      CALL MULTPN(R,RAM)
      R(3)=7
      R(4)=1
      CALL IRRAMC(R,RAM)
      R(3)=5
      R(4)=13
      CALL MULTPN(R,RAM)
      R(1)=7
      R(2)=5
      CALL ADRMIR(R,RAM)
      R(3)=6
      R(4)=9
      CALL IRRAMC(R,RAM)
      R(1)=6
      R(2)=5
      R(3)=5
      R(4)=13
      CALL MULTPN(R,RAM)
      R(1)=7
      R(2)=1
      CALL COMPLC(R,RAM)
      CALL ADRMIR(R,RAM)
      R(3)=3
      R(4)=9
      CALL IRRAMC(R,RAM)
      CALL CONVD(6,9,RAM,R,XVAL1,0)
      CALL CONVD(3,9,RAM,R,XVAL2,0)
      WRITE(6,203) XVAL1,XVAL2
  203 FORMAT(10X,'DXD=',F10.4,2X,'DYD=',F10.4)
C     WE HAVE  JUST COMPLETED  THE  CALCULATIONS  WHICH
C     MAKE  USE  OF  THE  DOPPLER  RADAR
  600 IF(IF.NE.0) GO TO 300
      XVWY=(3.0*XVWY+(XVAY-XVGYI))/4.0
      XVWX=(3.0*XVWX+(XVAX-XVGXI))/4.0
      WRITE(6,601) XVWY,XVWX
  601 FORMAT(10X,'XVWY=',F10.4,2X,'XVWX=',F10.4)
C     WE ARE  GOING  TO  PROCEED  WITH MCS 4
      R(3)=5
      R(4)=5
      CALL RAMIRC(R,RAM)
      CALL COMPLC(R,RAM)
      R(1)=4
      R(2)=9
      CALL ADRMIR(R,RAM)
      R(1)=4
```

```
      R(2)=5
      CALL ADRMIR(R,RAM)
      CALL ADRMIR(R,RAM)
      CALL ADRMIR(R,RAM)
      R(3)=4
      R(4)=5
      CALL IRRAMC(R,RAM)
      CALL DIV2(R,RAM)
      CALL DIV2(R,RAM)
      R(3)=5
      R(4)=1
      CALL RAMIRC(R,RAM)
      CALL COMPLC(R,RAM)
      R(1)=5
      R(2)=9
      CALL ADRMIR(R,RAM)
      R(1)=4
      R(2)=1
      CALL ADRMIR(R,RAM)
      CALL ADRMIR(R,RAM)
      CALL ADRMIR(R,RAM)
      R(3)=4
      R(4)=1
      CALL IRRAMC(R,RAM)
      CALL DIV2(R,RAM)
      CALL DIV2(R,RAM)
      CALL CONVD(4,5,RAM,R,XVAL1,0)
      CALL CONVD(4,1,RAM,R,XVAL2,0)
  303 FORMAT(10X,'VWY=',F10.4,2X,'VWX=',F10.4)

C     WE  HAVE  COMPLETED  CALCULATING  VWY, VWX

      XDY=XT*XVGYI
      XDX=XT*XVGXI
      WRITE(6,301) XDX,XDY
  301 FORMAT(10X,'XDX=',F10.5,2X,'XDY=',F10.5)

C     REPEAT  THE  ABOVE  CALCULATIONS  ON  MCS-4

      R(1)=1
      R(2)=5
      R(3)=5
      R(4)=5
      CALL MULTPN(R,RAM)
      R(1)=2
      R(2)=9
      CALL DIV2IR(R,RAM)
      R(1)=1
      R(2)=5
      R(3)=5
      R(4)=1
      CALL MULTPN(R,RAM)
      R(1)=1
      R(2)=9
      CALL DIV2IR(R,RAM)
      CALL CONVD(1,9,RAM,R,XVAL1,0)
      CALL CONVD(2,9,RAM,R,XVAL2,0)
       WRITE(6,304) XVAL1,XVAL2
  304 FORMAT(10X,'DX=',F10.4,2X,'DY=',F10.4)

C     WE HAVE  COMPLETED  CALCULATING  DY, DX

  300 IF(.NOT.((IF.EQ.0).AND.(DF.EQ.0))) GO TO 400
      XDY=(XDY+XDYD)/2.0
      XDX=(XDX+XDXD)/2.0
      WRITE(6,301) XDX,XDY

C     REPEAT  THE  ABOVE  SEQUENCE  WITH  MCS-4

      R(1)=3
      R(2)=9
      R(3)=2
```

```
      R(4)=9
      CALL ADDRAM(R,RAM)
      CALL DIV2(R,RAM)
      R(1)=6
      R(2)=9
      R(3)=1
      R(4)=9
      CALL ADDRAM(R,RAM)
      CALL DIV2(R,RAM)
      CALL CONVD(2,9,RAM,R,XVAL2,0)
      CALL CONVD(1,9,RAM,R,XVAL1,0)
       WRITE(6,304) XVAL1,XVAL2
C     DY  AND  DX  HAVE  JUST  BEEN  CALCULATED
      GO TO 540
  400 IF(.NOT.( (IF.EQ.1). AND.(DF.EQ.0))) GO TO 500
      XDY=XDYD
      XDX=XDXD
       WRITE(6,301) XDX,XDY
C     REPEAT THE ABOVE SEQUENCE  WITH  MCS-4
      R(3)=3
      R(4)=9
      R(1)=2
      R(2)=9
      CALL TRANRM(R,RAM)
      R(3)=6
      R(4)=9
      R(1)=1
      R(2)=9
      CALL TRANRM(R,RAM)
      CALL CONVD(2,9,RAM,R,XVAL2,0)
      CALL CONVD(1,9,RAM,R,XVAL1,0)
       WRITE(6,304) XVAL1,XVAL2
C     DY  AND  DX  HAVE  JUST  BEEN  CALCULATED
      GO TO 540
  500 IF((IF.NE.1).OR.(DF.NE.1)) GO TO 540
      XDY=XT*(XVWY+XVAY)
      XDX=XT*(XVWX+XVAX)
       WRITE(6,301) XDX,XDY
C     REPEAT  THE ABOVE CALCULATION WITH MCS-4
      R(1)=4
      R(2)=5
      R(3)=4
      R(4)=9
      CALL ADDRAM(R,RAM)
      R(1)=1
      R(2)=5
      CALL MULTPN(R,RAM)
      R(3)=2
      R(4)=9
      CALL IRRAMC(R,RAM)
      R(1)=4
      R(2)=1
      R(3)=5
      R(4)=9
      CALL ADDRAM(R,RAM)
      R(1)=1
      R(2)=5
      CALL MULTPN(R,RAM)
      R(3)=1
      R(4)=9
      CALL IRRAMC(R,RAM)
      CALL CONVD(2,9,RAM,R,XVAL2,0)
      CALL CONVD(1,9,RAM,R,XVAL1,0)
       WRITE(6,304) XVAL1,XVAL2
```

```
C      CALCULATIONS ARE NOW COMPLETE  FOR  DX AND DY

 540   XLOCF = XLOCF + XDX
       XLOCM = XLOCM + XVAL1
       YLOCF = YLOCF + XDY
       YLOCM = YLOCM + XVAL2
       WRITE(6,550) XLOCF,YLOCF
       WRITE(6,560) XLOCM,YLOCM
  550 FORMAT (8X,'XLOCF = ',F10.5,4X,'YLOCF = ',F10.5)
  560 FORMAT (8X,'XLOCM = ',F10.5,4X,'YLOCM = ',F10.5,//)
       RETURN
       END
```

```
      ************************************************************
      THIS SUBROUTINE SIMULATES THE FUNCTIONS
      OF THE MCS-4 SUBROUTINE COSINE
      ************************************************************

      SUBROUTINE COSINE(IR,RAM)
      IMPLICIT INTEGER  (A-Z)
      DIMENSION IR(16),RAM(16,32),T(4),TE(4)
      I=IR(16)*8+IR(15)/2
      J=(IR(15)-(IR(15)/2)*2)*16+IR(14)/8*8
      DO 1 K=5,12
    1 IR(K)=RAM(I+1,J+K-4)
      IR(15) = IR(14) - IR(14)/8*8
      IR(14) =IR(13)
      IR(13)=0
      IR(16)=0
      DO 2 K=1,4
      T(K)=IR(12+K)
    2 TE(K)=IR(8+K)
      CALL MULT(T,TE,IR)
      ACC=(IR(8)-IR(16))*16**3+(IR(7)-IR(15))*16**2+(IR(6)-I
     1R(14))*16+(IR(5)-IR(13))
      ACC=ACC/2
      DO 3 K=1,4
      IR(12+K)=ACC-(ACC/16)*16
    3 ACC=ACC/16
      RETURN
      END
```

```
      ************************************************************
      THIS SUBROUTINE SIMULATES THE FUNCTIONS
      OF THE MCS-4 SUBROUTINE ADDRAM
      ************************************************************

      SUBROUTINE ADDRAM(IR,RAM)
      IMPLICIT INTEGER   (A-V)
      DIMENSION IR(16),RAM(16,16)
      SUM=0
      DO 1 J=1,4
    1 SUM=SUM+(RAM(IR(1),IR(2)+J-1)+RAM(IR(3),IR(4)+J-1))*16
     1**(J-1)
      DO 2  J=1,4
      RAM(IR(3),IR(4)+J-1)=SUM-(SUM/16)*16
    2 SUM=SUM/16
      RETURN
      END
```

```
C*****************************************************
C THIS SUBROUTINE SIMULATES THE FUNCTIONS
C OF THE MCS-4 SUBROUTINE SUBIR
C*****************************************************

      SUBROUTINE SUBRIR(IR,RAM)
      IMPLICIT INTEGER   (A-V)
      DIMENSION IR(16),RAM(16,16)
      SUM=0
      M=IR(3)
      N=IR(4)
      DO 1 J=1,4
      TERM=(RAM(M,N+J-1)-IR(12+J))*16**(J-1)
    1 SUM=SUM+TERM
      DO 2 J=1,4
       RAM(IR(1),IR(2)+J-1)=SUM-(SUM/16)*16
    2 SUM=SUM/16
      RETURN
      END

C*****************************************************
C THIS SUBROUTINE SIMULATES THE FUNCTIONS
C OF THE MCS-4 SUBROUTINE DIV2IR
C*****************************************************

      SUBROUTINE DIV2IR(IR,RAM)
      IMPLICIT INTEGER   (A-V)
      DIMENSION IR(16),RAM(16,16)
      SUM=0
      F1=0
      IF(IR(16).LT.8) GO TO 3
      F1=1
      CALL COMPLC(IR,RAM)
    3 DO 1 J=1,4
      TERM=(IR(12+J))*16**(J-1)
    1 SUM=SUM+TERM
      SUM=SUM/2
      DO 2 J=1,4
      IR(J+12)=SUM-(SUM/16)*16
    2 SUM=SUM/16
      IF(F1.EQ.0) GO TO 5
      CALL COMPLC(IR,RAM)
    5 DO 6 J=1,4
    6 RAM(IR(1),IR(2)+J-1)=IR(J+12)
      RETURN
      END

C*****************************************************
C THIS SUBROUTINE SIMULATES THE FUNCTIONS
C OF THE MCS-4 SUBROUTINE ADDRAMIR
C*****************************************************

      SUBROUTINE ADRMIR(IR,RAM)
      IMPLICIT INTEGER   (A-V)
      DIMENSION IR(16),RAM(16,16)
      SUM=0
      K=IR(1)
      L=IR(2)
      DO 1 J=1,4
      TERM=(RAM(K,L+J-1)+IR(12+J))*16**(J-1)
    1 SUM=SUM+TERM
      DO 2 J=1,4
      IR(12+J)=SUM-(SUM/16)*16
    2 SUM=SUM/16
      RETURN
      END
```

```
*****************************************************
THIS SUBROUTINE SIMULATES THE FUNCTIONS
OF THE MCS-4 SUBROUTINE RAMIR8
*****************************************************

      SUBROUTINE RAMIR8(IR,RAM)
      IMPLICIT INTEGER   (A-V)
      DIMENSION IR(16),RAM(16,16)
      DO 1 J=1,4
    1 IR(8+J)=RAM(IR(1),IR(2)+J-1)
      RETURN
      END



*****************************************************
THIS SUBROUTINE SIMULATES THE FUNCTIONS
OF THE MCS-4 SUBROUTINE IRRAMC
*****************************************************


      SUBROUTINE IRRAMC(IR,RAM)
      IMPLICIT INTEGER   (A-V)
      DIMENSION IR(16),RAM(16,16)
      DO 1 J=1,4
    1 RAM(IR(3),IR(4)+J-1)=IR(12+J)
      RETURN
      END



*****************************************************
THIS SUBROUTINE SIMULATES THE FUNCTIONS
OF THE MCS-4 SUBROUTINE RAMIRC
*****************************************************

      SUBROUTINE RAMIRC(IR,RAM)
      IMPLICIT INTEGER   (A-V)
      DIMENSION IR(16),RAM(16,16)
      DO 1 J=1,4
    1 IR(12+J)=RAM(IR(3),IR(4)+J-1)
      RETURN
      END



*****************************************************
THIS SUBROUTINE SIMULATES THE FUNCTIONS
OF THE MCS-4 SUBROUTINE TRANRAM
*****************************************************


      SUBROUTINE TRANRM(IR,RAM)
      IMPLICIT INTEGER   (A-V)
      DIMENSION IR(16),RAM(16,16)
      DO 1 J=1,4
    1 RAM(IR(1),IR(2)+J-1)=RAM(IR(3),IR(4)+J-1)
      RETURN
      END
```

```
      ************************************************************
      THIS SUBROUTINE SIMULATES THE FUNCTIONS
      OF THE MCS-4 SUBROUTINE DIV2
      ************************************************************


      SUBROUTINE DIV2(IR,RAM)
      IMPLICIT INTEGER   (A-Z)
      DIMENSION IR(16),RAM(16,16)
      SUM=0
      K=IR(3)
      L=IR(4)
      F1=0
      DO 1 J=1,4
1     IR(J+12)=RAM(K,L+J-1)
      IF(IR(16).LT.8) GO TO 3
      F1=1
      CALL COMPLC(IR,RAM)
3     DO 4 J=1,4
4     SUM=SUM+IR(J+12)*16**(J-1)
      SUM=SUM/2
      DO 2 J=1,4
      IR(J+12)=SUM-(SUM/16)*16
2     SUM=SUM/16
      IF(F1.EQ.0) GO TO 5
      CALL COMPLC(IR,RAM)
5     DO 6 J=1,4
6     RAM(K,L+J-1)=IR(J+12)
      RETURN
      END


      ************************************************************
      THIS SUBROUTINE SIMULATES THE FUNCTIONS
      OF THE MCS-4 MULTIPLICATION TABLE
      ************************************************************


      SUBROUTINE TABLE(A,B,C,D)
      IMPLICIT INTEGER   (A-Z)
      D=(A*B)/16
      C=A*B-D*16
      RETURN
      END


      ************************************************************
      THIS SUBROUTINE SIMULATES THE FUNCTIONS
      OF THE MCS-4 SUBROUTINE ADDRAM
      ************************************************************


      SUBROUTINE ADD(A,B,C,D)
      IMPLICIT INTEGER   (A-Z)
      D=(A+B)/16
      C=(A+B)-D*16
      RETURN
      END
```

```
      ************************************************
      THIS SUBROUTINE SIMULATES THE FUNCTIONS
      OF THE MCS-4 SUBROUTINE MULT
      ************************************************


      SUBROUTINE MULT(X,Y,R)
      IMPLICIT INTEGER   (A-Z)
      DIMENSION X(4),Y(4),R(16),TEMP(16)
      DO 1 I=1,16
    1 TEMP(I)=R(I)
      CALL TABLE (X(1),Y(4),BL,R(5))
      CALL TABLE(X(2),Y(3),BL,R(6))
      CALL ADD(R(5),R(6),R(5),R(8))
      CALL TABLE(X(2),Y(4),R(6),R(7))
      CALL ADD(R(6),R(5),R(5),R(6))
      CALL TABLE(X(3),Y(3),R(9),R(10))
      CALL ADD(R(7),R(8),R(8),BL)
      CALL ADD(R(8),R(10),R(7),R(8))
      CALL TABLE(X(3),Y(4),R(10),R(11))
      CALL ADD(R(6),R(7),R(6),R(7))
      CALL ADD(R(10),R(6),R(6),CY)
   12 CALL ADD(CY,R(7),R(7),BL)
      CALL TABLE(X(4),Y(3),R(12),R(10))
      CALL ADD(R(8),R(10),R(8),BL)
      CALL ADD(R(8),BL,R(8),BL)
  145 CALL ADD(R(8),R(11),R(8),R(10))
      CALL TABLE( X(4),Y(4),R(11),R(13))
      CALL ADD(R(13),R(10),R(10),BL)
      CALL ADD(R(7),R(8),R(7),R(8))
      CALL ADD(R(11),R(7),R(7),CY)
      CALL ADD(CY,R(8),ACC,BL)
   21 CALL ADD(ACC,R(10),R(16),BL)
      CALL TABLE(X(3),Y(2),BL,R(8))
      CALL ADD(R(8),R(5),R(5),R(8))
      CALL ADD(R(9),R(5),R(5),R(9))
      CALL TABLE(X(4),Y(2),R(10),R(11))
      CALL ADD(R(11),R(8),ACC,BL)
      CALL ADD(ACC,R(6),R(6),CY)
      CALL ADD(CY,R(7),R(7),R(11))
   28 CALL TABLE(X(4),Y(1),BL,R(8))
      CALL ADD(R(10),R(8),R(8),CY)
      CALL ADD(CY,R(6),R(6),R(15))
      CALL ADD(R(6),R(12),R(6),CY)
      CALL ADD(CY,R(15),ACC,BL)
      CALL ADD(ACC,R(7),R(15),CY)
      CALL ADD(CY,R(16),R(16),BL)
      CALL ADD(R(8),R(5),R(13),CY)
      CALL ADD(CY,R(9),R(9),BL)
      CALL ADD(R(6),R(9),R(14),CY)
   38 CALL ADD(R(15),CY,R(15),CY)
      CALL ADD(CY,R(11),ACC,CY)
      CALL ADD(ACC+CY,R(16),R(16),BL)
      DO 2 I=1,12
    2 R(I)=TEMP(I)
      RETURN
      END
```

```
      ************************************************************
      THIS SUBROUTINE SIMULATES THE FUNCTIONS
      OF THE MCS-4 ASSEMBLER DUMP AND TRACE
      ************************************************************



      SUBROUTINE PRINT(N,IR,RAM,ROM,K,L)
      IMPLICIT INTEGER   (A-Z)
      DIMENSION IR(16),RAM(16,16),ROM(16,32)
      IF(N.NE.1) GO TO 5
      WRITE(6,101) IR
101   FORMAT(5X,'IR',2X,' 1 2 3 4 5 6 7 8 9 10 11 12 13 14 1
     19X,9I2,7I3,/)
      RETURN
    5 IF(N.NE.2) GO TO 10
      WRITE(6,102) K,L,(RAM(K,L+I-1),I=1,4)
102   FORMAT(5X,'RAM LOC',I2,I3,4I3,/)
      RETURN
   10 IF(N.NE.3) RETURN
      WRITE(6,103) K,L,(ROM(K,L+I-1),I=1,8)
103   FORMAT(5X,'ROM LOC',I2,I3,8I3,/)
      RETURN
      END
```

```
      ************************************************************
      THIS SUBROUTINE SIMULATES THE FUNCTIONS
      OF THE MCS-4 SUBROUTINE COMPLEMENT
      ************************************************************



      SUBROUTINE COMPLC(R,RAM)
      IMPLICIT INTEGER (A-Z)
      DIMENSION R(16),RAM(16,16)
      IF((R(16).LT.0).OR.(R(15).LT.0).OR.(R(14).LT.0).OR.(R(
     1GO TO 5
      L=1
      DO 1 I=1,4
      R(12+I)=15-R(12+I)+L
      IF(R(12+I).NE.16) GO TO 3
      R(12+I)=0
      L=1
      GO TO 1
3     L=0
1     CONTINUE
      RETURN
5     DO 2 I=1,4
2     R(12+I)=-R(12+I)
      RETURN
      END
```

```
      ************************************************
      THIS SUBROUTINE SIMULATES THE FUNCTIONS
      OF THE MCS-4 EXPANDED MULTIPLY
      ************************************************


      SUBROUTINE MULTPN(R,RAM)
      IMPLICIT INTEGER(A-Z)
      DIMENSION R(16),RAM(16,16),T(4),TE(4)
      F1=0
      F2=0
      DO 1 I=1,4
    1 R(12+I)=RAM(R(1),R(2)+I-1)
      IF(R(16).LT.8) GO TO 2
      F1=1
      CALL COMPLC(R,RAM)
    2 SUM=R(16)
      DO 3 I=1,3
    3 SUM=SUM*16+R(16-I)
      SUM=SUM*2
      DO 4 I=1,4
      T(I)=SUM-(SUM/16)*16
    4 SUM=SUM/16
      DO 11 I=1,4
   11 R(12+I)=RAM(R(3),R(4)+I-1)
      IF(R(16).LT.8) GO TO 12
      F2=1
      CALL COMPLC(R,RAM)
   12 SUM=R(16)
      DO 13 I=1,3
   13 SUM=SUM*16+R(16-I)
      SUM=SUM*2
      DO 14 I=1,4
      TE(I)=SUM-(SUM/16)*16
   14 SUM=SUM/16
      CALL MULT(T,TE,R)
      SUM=R(16)
      DO 23 I=1,3
   23 SUM=SUM*16+R(16-I)
      SUM=SUM/2
      SUM=SUM/2
      DO 24 I=1,4
      R(I+12)=SUM-(SUM/16)*16
   24 SUM=SUM/16
      IF((F1+F2).NE.1) GO TO 30
      CALL COMPLC(R,RAM)
   30 RETURN
      END


      ************************************************
      THIS SUBROUTINE SIMULATES THE FUNCTIONS
      OF THE TH QUADRANT TEST IN THE
      MCS-4 EXECUTIVE ROUTINE
      ************************************************


      SUBROUTINE COSPL(R,ROM,RAM)
      IMPLICIT INTEGER (A-Z)
      DIMENSION R(16),ROM(16,32),RAM(16,16)
C     PLACE PI/2 INTO REGISTERS
      R(13)=2
      R(14)=2
      R(15)=9
      R(16)=1
C     LOCATION OF TH AND INTERMEDIATE STORAGE
      R(3)=3
      R(4)=5
      R(1)=10
      R(2)=1
      CALL SUBIR(R,RAM)
      IF((RAM(10,1).LT.0).OR.(RAM(10,2).LT.0).OR.(RAM(10,3).
```

```
     1(RAM(10,4).LT.0)) GO TO 1
      R(3)=10
      R(4)=1
      R(2)=5
      CALL SUBRIR(R,RAM)
      IF((RAM(10,5).LT.0).OR.(RAM(10,6).LT.0).OR.(RAM(10,7).
     1(RAM(10,8).LT.0)) GO TO 2
      R(4)=5
      R(2)=1
      CALL SUBRIR(R,RAM)
      IF((RAM(10,1).LT.0).OR.(RAM(10,2).LT.0).OR.(RAM(10,3).
     1(RAM(10,4).LT.0))  GO TO 3
C     TI-PI/2  IS NOW IN RAM(10,1)
      R(4)=5
C     TRANSFER    PI/2  TO  RAM(10,5)
      CALL IRRAMC(R,RAM)
      R(4)=1
      CALL RAMIRC(R,RAM)
      CALL COSINE(R,ROM)
      R(3)=5
      R(4)=13
      CALL IRRAMC(R,RAM)
C     TRANSFER   TH- PRIME TO R(13)
      R(3)=10
      R(4)=1
      CALL RAMIRC(R,RAM)
      R(4)=5
      R(1)=10
      R(2)=1
      CALL SUBRIR(R,RAM)
C     COMPLEMENTARY  ANGLE  IS NOW  IN  RAM(1,10)
      R(4)=1
      CALL RAMIRC(R,RAM)
      CALL COSINE (R,ROM)
      R(3)=2
      R(4)=13
      CALL IRRAMC(R,RAM)
C     COS(TH) HAS NOW BEEN STORED
      RETURN
   3  R(3)=10
      R(4)=1
C     COMPLEMENT   RAM(10,1)
      CALL RAMIRC(R,RAM)
      CALL COMPLC(R,RAM)
      CALL COSINE(R,ROM)
      R(3)=5
      R(4)=13
      CALL IRRAMC(R,RAM)
C        SIN(TH)   HAS BEEN FOUND
      R(3)=10
      R(4)=5
      CALL RAMIRC(R,RAM)
      CALL COSINE(R,ROM)
      CALL COMPLC(R,RAM)
      R(3)=2
      R(4)=13
       CALL IRRAMC(R,RAM)
C     COS(TH)  HAS  BEEN  FOUND
      RETURN
   2  R(3)=10
      R(4)=5
      CALL RAMIRC(R,RAM)
      CALL COMPLC(R,RAM)
      CALL COSINE(R,ROM)
      CALL COMPLC(R,RAM)
      R(3)=2
      R(4)=13
       CALL IRRAMC(R,RAM)
C     COSINE  HAS  BEEN  FOUND
      R(3)=10
      R(4)=1
      CALL RAMIRC(R,RAM)
```

```
      CALL COSINE(R,ROM)
      CALL COMPLC(R,RAM)
      R(3)=5
      R(4)=13
       CALL IRRAMC(R,RAM)
C       SIN(TH)  HAS  BEEN  FOUND
      RETURN
    1 R(3)=10
      R(4)=1
      CALL RAMIRC(R,RAM)
      CALL COMPLC(R,RAM)
      CALL COSINE(R,ROM)
      CALL COMPLC(R,RAM)
      R(3)=5
      R(4)=13
       CALL IRRAMC(R,RAM)
C            SIN(TH)  HAS  BEEN  FOUND
      R(3)=3
      R(4)=5
      CALL RAMIRC(R,RAM)
      CALL COSINE (R,ROM)
      R(3)=2
      R(4)=13
       CALL IRRAMC(R,RAM)
C        COS(TH)  HAS  BEEN  FOUND
      RETURN
      END

      ************************************************************
      THIS SUBROUTINE CONVERTS THE HEXIDECIMAL
      OUTPUT OF THE MCS-4 PROGRAM TO DECIMAL
      ************************************************************

      SUBROUTINE CONVD(N,M,RAM,R,XVAL,K)
      IMPLICIT INTEGER(A-V)
      DIMENSION RAM(16,16),R(16)
      FLAG=0
      R(3)=N
      R(4)=M
      CALL RAMIRC(R,RAM)
      IF(R(16).LT.8) GO TO 3
      FLAG=1
      CALL COMPLC(R,RAM)
    3 K1=K-1
      IF(K1) 8,9,10
    8 XVAL=0.0
      GO TO 4
    9 XVAL=R(16)
      GO TO 4
   10 XVAL=R(16)
      DO 1 I=1,K1
    1 XVAL=R(16-I)+XVAL*16.0
    4 K1=4-K
      IF(K1.LE.0) GO TO 6
      DO 2 I=1,K1
    2 XVAL=XVAL+R(13+K1-I)*16.0**(-I)
      IF(K.EQ.0) XVAL=XVAL*2.0
    6 IF(FLAG.EQ.0) RETURN
      XVAL=-XVAL
      RETURN
      END
```

```
      ************************************************************
      THIS SUBROUTINE CONVERTS THE DECIMAL INPUT
      INTO HEXADECIMAL FOR THE MCS-4 PROGRAM
      ************************************************************


      SUBROUTINE CONVRT(N,M,XVAR,RAM,R,K)
      IMPLICIT INTEGER(A-V)
      DIMENSION RAM(16,16),R(16)
      FLAG=0
      IF(XVAR.GE.0.0) GO TO 3
      XVAR=-XVAR
      FLAG=1
3     VAR=XVAR
      IF(K.LT.1) GO TO 5
      DO 1 I=1,K
      RAM(N,M+3-K+I)=VAR-VAR/16*16
1     VAR=VAR/16
5     K1=4-K
      IF(K1.LE.0) GO TO 10
      FR=XVAR
      XFR=XVAR
      DO 2 I=1,K1
      XFR=(XFR-FR)*16
      RAM(N,M+4-K-I)=XFR
2     FR=XFR
10    IF(FLAG.EQ.0) RETURN
      XVAR=-XVAR
      R(3)=N
      R(4)=M
      CALL RAMIRC(R,RAM)
      CALL COMPLC(R,RAM)
      CALL IRRAMC(R,RAM)
      RETURN
      END
```

BIBLIOGRAPHY

1. Adams, J.R. and others, The Preliminary Design of a Long Range Navigation System for Tactical Aircraft, Group Thesis Project 0910, United States Naval Postgraduate School, Monterey, 1970.

2. AGARD Conference Proceedings No. 43, Inertial Navigation-Systems and Components, May 1968.

3. AGARD-LS 52, Guidance and Control of Tactical Missiles, May 1972.

4. AGARD-AG-158, Computers in the Guidance and Control of Aerospace Vehicles, February 1972.

5. Altman, L., "Single-Chip Microprocessors Open Up A New World Of Applications," Electronics, V. 47, p. 81-87, 18 April 1974.

6. Beck, G.E., and others, Navigation Systems, Van Nostrand Reinhold, 1971.

7. Busacker, R.G., and Saaty, T.L., Finite Graphs and Networks: An Introduction with Applications, McGraw-Hill, 1965.

8. Carnahan, B., Luther, H.A., and Wilkes, J.O., Applied Numerical Methods, Wiley, 1969.

9. Cushman, R.H., "Microprocessors are Changing Your Future. Are You Prepared:," EDN, V. 18, p. 26-32, 5 November 1973.

10. Cushman, R.H., "Understanding the Microprocessor is No Trivial Task," EDN, V. 18, p. 42-49, 20 November 1973.

11. Cushman, R.H., "Understand the 8-bit P: You'll see a lot of it," EDN, V. 19, p. 48-54, 20 January 1974.

12. Cushman, R.H., "Don't overlook the 4-bit μP: They're here and they're cheap," EDN, V. 19, p. 44-50, 20 February 1974.

13. Cushman, R.H., "The Intel 8080: First of the Second-Generation Microprocessors," EDN, V. 19, p. 30-36, 5 May 1974.

14. Cushman, R.H., "Microprocessors are rapidly gaining on Minicomputers," EDN, V. 19, p. 16-20, 20 May 1974.

15. Forsyth, G.E., "Pitfalls In Computation, or Why a Math Book is not Enough," American Mathematics Monthly, p. 931-956, November 1970.

16. Holt, R.M., "Current Microcomputer Architecture," Computer Design, p. 65-73, February 1974.

17. Intel, MCS-4 Micro Computer Set, 1973.

18. Intel, MCS-8 Micro Computer Set, 1973.

19. Intel, PL/M Programming Manual, 1974.

20. Johnson, E.E. and Johnson, J.R., Graph Theory with Engineering Applications, Ronald, 1972.

21. Kayton, M., Fried, W.R., and others, Avionics Navigation Systems, Wiley, 1969

22. Keele, R.V., Microprocessor Trade-Off Study for Project 2175, Tentative and Unpublished NELC Technical Note, San Diego, California, 6 March 1973.

23. Klass, P.J., "New Gyro Nears Operational Use," Aviation Week, p. 50-52, 19 June 1972.

24. Klass, P.J., "New Guidance Technique Being Tested," Aviation Week, V. 100, p. 48-51, 25 February 1974.

25. Koenig, H.E., Tokad, Y., and Kesavan, H.K., Analysis of Discrete Physical Systems, McGraw-Hill, 1967.

26. Lockheed S-3A Avionics, p. 4-20, Aircraft Engineering, January 1974.

27. McCalla, T.R., Introduction to Numerical Methods and FORTRAN Programming, Wiley, 1964.

28. McCracken, D.D., and Dorn, W.S., Numerical Methods and FORTRAN Programming, Wiley, 1964.

29. NASA Technical Report R-329, A New Concept in Strapdown Inertial Navigation, by J.E. Bortz, Sr., March 1971.

30. Naval Air Development Center, Navigation FP-106, 1973.

31. Naval Air Systems Command 01-85 ADF-2-10.1, Integrated Weapons System Theory for A6E Aircraft, 1971.

32. Ogdin, J.L., "Survey of Microprocessors Reveals Limitless Variety," EDN, V. 19, p. 38-43, 20 April 1974.

33. Patrol Squadron Thirty-One, P3C Electronic Systems Operational Training Study Guide, 1970.

34. Ralston, A., A First Course in Numerical Analysis, McGraw-Hill, 1965.

35. Reyling, G., "Performance and Control of Multiple Microprocessor Systems," Computer Design, p. 81-86, March, 1974.

36. Ringo, R.L., "Cost-of-Ownership Design Philosophy for Inertial Navigators," Astronautics and Aeronautics, p. 59-62, June 1973.

37. Schultz, G.W. and Holt, R.M., "MOS LSI Minicomputer Comes of Age," Proceedings 1972 Fall Joint Computer Conference, p. 1069-1080, October 1972.

38. Schultz, G.W., Holt, R.M., and McFarland, H.L., "A Guide to Using LSI Microprocessors," Computer, p. 13-19, June 1973.

39. Sperry Rand Corporation, P-3C Operational Program, 1970.

40. The University of Michigan Engineering Summer Conferences, Modern Navigation Systems,

41. Wiener, H., "Computers that fit in Your Pocket," Computer Decisions, V. 5, p. 8-13, August 1973.

42. Weissberger, A.J., "MOS/LSI Microprocessor Selection," Electronic Design, p. 100-104, 7 June 1974.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Mr. W. McMahon P3C Software Naval Air Development Center Warminster, Pennsylvania	1
4.	Assoc. Professor U. R. Kodres, Code 53Kr Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
5.	Chairman, Department of Aeronautics Naval Postgraduate School Monterey, California 93940	1
6.	Asst. Professor G. A. Kildall, Code 53Kd Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
7.	Lt. W. L. McCracken, USN 136 Barber's Point Rd. NAS Alameda Alameda, California 94501	1