

ELECTRICAL ENGINEERING DEPARTMENT

PROGRAMMABLE
DATA COLLECTION PLATFORM
STUDY

Final Report

May 1976

UNIVERSITY OF TENNESSEE

**KNOXVILLE
TN 37916**

The University of Tennessee
Department of Electrical Engineering
Knoxville, Tennessee 37916

PROGRAMMABLE
DATA COLLECTION PLATFORM
STUDY

Final Report

May 1976

Contract No. NAS5-22495

Prepared for

National Aeronautics and Space Administration
Goddard Space Flight Center
Greenbelt, Maryland 20771

**Page
Intentionally
Left Blank**

1955
W. H. R. ...
...

PREFACE

This report represents the findings of the work completed under a nine-month study of programmable data collection platforms. The system has been implemented with a microcomputer, and the results show that programmable data collection platforms can carry out all the functions of hardwired data collection platforms with capacity left to perform a number of desirable computational functions.

CONTENTS

<u>Section</u>		<u>Page</u>
1	INTRODUCTION.	1-1
	1.1 BACKGROUND AND PURPOSE	1-1
	1.2 SUMMARY.	1-2
2	MICROCOMPUTER HARDWARE AND SOFTWARE	2-1
	2.1 HARDWARE ARCHITECTURE OF THE MICROPROCESSOR.	2-4
	2.1.1 Arithmetic/Logic Unit (ALU)	2-4
	2.1.2 Registers	2-8
	2.1.2.1 Program Counter Register	2-9
	2.1.2.2 Stack and Stack Pointer.	2-10
	2.1.2.3 Accumulator and General Purpose Registers.	2-11
	2.1.2.4 Instruction Register	2-11
	2.1.2.5 Address Registers.	2-12
	2.1.3 Control Logic	2-12
	2.2 MICROCOMPUTER SOFTWARE ASPECTS	2-19
	2.2.1 Microcomputer Instruction Sets.	2-22
	2.2.2 Microcomputer Programming	2-25
	2.2.2.1 Programming Languages.	2-27
	2.2.2.2 Utility Programs	2-29
	2.2.2.3 Microprogramming	2-31
3	PDCP SYSTEM HARDWARE.	3-1
	3.1 SENSOR CLASSIFICATION AND INTERFACING.	3-1
	3.1.1 Sensor Classification	3-1
	3.1.2 Sensor Interfacing.	3-9
	3.1.2.1 Sensor Interface	3-11
	3.1.2.2 Microprocessor Interface	3-11
	3.2 DIGITAL CONTROL LOGIC AND MEMORY	3-12
	3.3 TRANSMITTER, RECEIVER, AND ANTENNA	3-15
	3.4 SUMMARY.	3-16

CONTENTS (Continued)

<u>Section</u>	<u>Page</u>
4 PDCP SYSTEM SOFTWARE	4-1
4.1 PDCP SOFTWARE ORGANIZATION	4-1
4.2 PDCP SYSTEM TIMING	4-3
4.3 PDCP SOFTWARE DEVELOPMENT	4-5
4.3.1 Applications-Oriented Software Development. .	4-5
4.3.2 Assembly Language and Microprogrammed Subroutines	4-7
4.4 PDCP PROGRAM EXAMPLES	4-8
4.4.1 Data Input and Sensor Control	4-8
4.4.1.1 Frequency Measurement.	4-9
4.4.1.2 Autoranging.	4-13
4.4.1.3 Software Controlled Analog-to-Digital Conversion	4-17
4.4.2 Data Compaction and Preprocessing	4-19
4.4.2.1 General-Purpose Binary Math Package.	4-22
4.4.2.2 Determination of Minimum and Maximum Data Values.	4-27
4.4.2.3 Zero-Order Floating Aperture Predictor.	4-27
4.4.2.4 Data Average	4-40
4.4.2.5 Mean, Variance, and Standard Deviation	4-40
4.4.3 Data Formatting and Transmitter Control	4-50
4.4.3.1 TWERLE Format Transmitter Subroutine	4-57
4.4.3.2 GOES Format Transmitter Routine. . . .	4-68
4.4.3.3 TIROS-N Format Transmitter Routine.	4-68
4.4.3.4 Landsat Format Transmitter Subroutine	4-84
4.4.4 Special Computations.	4-84
4.4.4.1 The FFT Program.	4-95
4.4.4.2 The Bayes Classifier	4-103

CONTENTS (Continued)

<u>Section</u>		<u>Page</u>
5	PROGRAMMABLE DATA COLLECTION PLATFORM DEMONSTRATION SYSTEM.	5-1
5.1	UT PDCP DEVELOPMENT SYSTEM HARDWARE.	5-1
5.1.1	Basic Pro-Log System.	5-2
5.1.1.1	Central Processing Unit Card	5-5
5.1.1.2	Random-Access Memory Cards	5-5
5.1.1.3	Read-Only Memory Cards	5-6
5.1.1.4	Latched Parallel Output Card	5-6
5.1.1.5	Parallel Input Card.	5-7
5.1.2	Special Purpose Interface Cards	5-8
5.1.2.1	Dual Serial I/O Card	5-9
5.1.2.2	Analog Interface Card.	5-10
5.1.2.3	Cassette Analog Interface and Baud Rate Clock Card.	5-10
5.1.2.4	General-Purpose Utility Card	5-11
5.1.3	Miscellaneous Hardware Components of the UT PDCP Development System	5-11
5.2	UT PDCP DEVELOPMENT SYSTEM SOFTWARE PACKAGE.	5-13
6	FUTURE PDCP SYSTEMS	6-1
6.1	EVALUATION OF AVAILABLE MICROPROCESSORS.	6-1
6.2	TECHNOLOGY FORECAST.	6-14
APPENDICES		
A	BINARY MATH PACKAGE	A-1
B	A NEW PRINCIPLE FOR FAST FOURIER TRANSFORMATION	B-1
C	8080A FAST FOURIER TRANSFORM PROGRAM.	C-1

ILLUSTRATIONS

<u>Figure</u>		<u>Page</u>
2.1(1)	General Block Diagram of CPU.	2-5
2.1.1(1)	Typical Arithmetic/Logic Unit	2-6
2.2(1)	Coding Levels	2-20
2.2.1(1)	Data Movement in the Microcomputer.	2-23
2.2.2(1)	Typical Flow Chart Symbols.	2-26
3(1)	DPC Hardware Subsystems	3-2
3.2(1)	Microprocessor Based PDCP Control Subsystem Organization.	3-13
4.1(1)	PDCP Software System Organization	4-2
4.4.1.1(1)	Flow Chart for Frequency Sensor Input Subroutine.	4-12
4.4.1.2(1)	Microprocessor-Controlled Variable-Gain Amplifier	4-14
4.4.1.2(2)	Analog Data Acquisition System for a PDCP	4-17
4.4.1.2(3)	Flow Chart for Variable Gain Amplifier Control Program	4-18
4.4.1.3(1)	Flow Chart for Successive Approximation ADC Subroutine.	4-21
4.4.1.3(2)	Simplified Schematic of the ADC Hardware.	4-21a
4.4.2.1(1)	Flow Chart for Square Root Subroutine	4-26
4.4.2.3(1)	Flow Chart for the Double Precision Zero-Order Floating-Aperture Predictor Subroutine.	4-35
4.4.2.3(2)	Flow Chart for the Multiprecision Zero-Order Floating Aperture Predictor Subroutine.	4-39
4.4.2.4(1)	Flow Chart for the Data Accumulation Subroutine	4-42
4.4.2.4(2)	Flow Chart for the Data Averaging Subroutine.	4-44
4.4.2.5(1)	Flow Chart for Data Preparation for Mean, Variance, and Standard Deviation Subroutine	4-51
4.4.2.5(2)	Flow Chart for Mean, Variance, and Standard Deviation Subroutine.	4-52

ILLUSTRATIONS (Continued)

<u>Figure</u>		<u>Page</u>
4.4.3(1)	Bit Assignments for Transmitter Control Port XCNTR	4-54
4.4.3.1(1)	Simplified Flow Chart for Subroutine TWXMIT	4-64
4.4.3.1(2)	Detailed Flow Chart for the Data Transmission Block of Subroutine TWXMIT	4-65
4.4.3.2(1)	Flow Chart for GOES Transmitter Subroutine	4-76
4.4.3.2(2)	Flow Chart for Subroutine BIPHS	4-77
4.4.3.3(1)	Flow Chart for TIROS-N Transmitter Subroutine	4-83
4.4.3.4(1)	Flow Chart for the PDCP Convolutional Encoder Subroutine	4-90
4.4.4(1)	Seismic Record Preprocessing and Classification	4-96
4.4.4(2)	FFT Data Flow Chart	4-97
5.1(1)	Major Components of the UT PDCP Development System	5-3
5.1(2)	UT PDCP Microcomputer Development System Block Diagram	5-4

PROGRAMS

<u>Program</u>	<u>Page</u>
4.4.1.1(1) Frequency Sensor Input	4-10
4.4.1.2(1) Variable Gain Amplifier Control	4-15
4.4.1.3(1) Analog-to-Digital Conversion	4-20
4.4.2.2(1) Minimum-Maximum Value Determination	4-29
4.4.2.3(1) Zero-Order Floating-Aperture Predictor	4-32
4.4.2.3(2) Multiprecision Zero-Order Floating-Aperture Predictor	4-36
4.4.2.4(1) Data Accumulation Subroutine	4-41
4.4.2.4(2) Data Averaging Subroutine	4-43
4.4.2.5(1) Data Preparation Subroutine for Mean, Variance and Standard Deviation	4-45
4.4.2.5(2) Mean, Variance and Standard Deviation	4-47
4.4.3.1(1) TWERLE Format Transmitter Subroutine	4-59
4.4.3.2(1) GOES Format Transmitter Subroutine	4-70
4.4.3.3(1) TIROS-N Format Transmitter Subroutine	4-78
4.4.3.4(1) PDCP Convolutional Encoder Subroutine	4-86
4.4.3.4(2) Landsat Format Transmitter Subroutine	4-91
A(1) Short Multiprecision Add	A-1
A(2) Short Multiprecision Subtract	A-2
A(3) Double-Precision Add With Memory	A-3
A(4) Double-Precision Subtract with Memory	A-4
A(5) Double-Precision Compare	A-5
A(6) Double-Precision Multiply	A-6
A(7) Double Precision Divide	A-8
A(8) Square Root	A-10

A(9)	Multibyte Binary Addition and Subtraction . . .	A-13
A(10)	Multibyte Compare	A-15
A(11)	Multibyte Memory-to-Memory Move	A-17
A(12)	General Purpose Exit Subroutine	A-18
C(1)	8080A Fast Fourier Transform	C-1

TABLES

<u>Table</u>		<u>Page</u>
3.1.1(1)	Satellite Data Collection Disciplines	3-3
3.1.1(2)	Sensor Measurement Parameters and Associated Disciplines	3-4
3.1.1(3)	Typical Satellite Data Collection Sensors . .	3-10
4.4.2.1(1)	General Purpose PDCP Binary Math Package . .	4-23
4.4.2.1(2)	Binary Math Subroutine Execution Times . . .	4-28
4.4.3(1)	Transmitter Control Signals	4-56
4.4.3(2)	Transmission Rates for the Landsat, GOES, TWERLE, and TIROS-N Data Collection Systems .	4-57
4.4.3.1(1)	TWERLE Data Transmission Sequence Specifications	4-58
4.4.3.1(2)	Format of the TWERLE RAM Data Block	4-66
4.4.3.2(1)	GOES Data Transmission Sequence Specifications	4-69
4.4.3.3(1)	TIROS-N Data Transmission Sequence Specifications	4-82
4.4.3.4(1)	Landsat Data Transmission Sequence Specifications	4-85
6.1(1)	System Constraints	6-4
6.1(2)	Application Constraints	6-5
6.1(3)	Software Constraints	6-6

1. INTRODUCTION

1.1 BACKGROUND AND PURPOSE

Data collection by satellite is a rather young branch of science which was first demonstrated in 1967 using the ATS-1 satellite. This first NASA demonstration was the Omega Position Location Equipment System (OPLE) which primarily determined that an accurate position fix could be obtained from platforms in remote locations. This system was followed by the Interrogation, Recording and Location System (IRLS) flown on the Nimbus-3 satellite in 1969. The IRLS was closely followed by the French Eole satellite which was solely devoted to the tasks of data collection and position location for remote platforms distributed around the globe.

One fact that stood out from these early experiments was that the user costs for platform and data reduction must be kept low if the satellite data collection concepts are to be utilized on a large-scale basis. A way to reduce costs was incorporated by the Landsat data collection system flown in 1972. The platform transmissions were random rather than ordered. This simplified the platforms by eliminating on-board receivers and reduced the costs substantially. In 1974 the GOES satellite carried a data collection system which again utilized an ordered system, but costs were kept low because of improvements in semiconductor technology.

In 1975 the Nimbus-6 satellite carried the Tropical Wind Energy-conversion and Reference Level Experiment (TWERLE) into orbit. The TWERLE utilized the low cost of the random transmission system combined with new low-cost integrated circuits to maintain a still lower platform cost.

In 1977 the TIROS-N satellite will be launched carrying a data collection system designed by France. The system will use the random transmission feature which will help to lower user costs and aid in making the system internationally acceptable.

A primary purpose of this study is to determine if the recent advances in semiconductor technology can be incorporated in the design of data collection platforms to further reduce their cost and, consequently, make their application more desirable to the user.

At the time of the instigation of this study, all data collection platforms used hardwired logic circuitry to implement the identification and formatting of the data collected by a platform. Hardwired circuitry was also used to control the sensors and the transmitter. In the last two years, a new semiconductor device which has the potential of drastically reducing hardware costs while increasing the capability of the data collection platform has appeared on the commercial market. The device is the microprocessor. The microprocessor is a computer central-processing unit (CPU) on a single integrated circuit chip. The equivalent of thousands of discrete electronic components are fabricated on a single microprocessor chip with unit costs projected to be in the ten-dollar range.

The goals of this study are to determine what present data collection functions can be accomplished by substituting a microprocessor for most of the hardwired logic, to uncover new tasks which would enhance the utility of data collection platforms by virtue of having a microprocessor available, and to determine if the programmable feature of the microprocessor will allow future platforms to be compatible with more than one satellite data collection system. To prove the concepts developed in achieving these goals, a model programmable data collection platform (PDCP) development system has been implemented. The details of the University of Tennessee (UT) PDCP development system are described in Section 5 of this report.

1.2 SUMMARY

This report describes the results of a study of the feasibility of incorporating microprocessors in data collection platforms (DCP's). An introduction to microcomputer hardware and software concepts is provided in Section 2. Thus, readers who are not familiar with the microprocessor

field are furnished necessary background information on the basic organization of a microcomputer and software development techniques.

Programmable data collection platform (PDCP) hardware design goals include minimizing power consumption, maintenance, weight, and cost while providing accurate and reliable operation. Section 3 discusses the influence of microprocessor technology on the design of PDCP hardware. A standard, modular PDCP design capable of meeting the design goals listed above is proposed. The microprocessor contributes to this design by minimizing PDCP control logic and simplifying sensor interfaces. Also, standard PDCP sensor and transmitter interfaces will promote further cost reductions.

Although the standard, modular PDCP design proposed in Section 3 is economically desirable, the PDCP must be sufficiently flexible to operate efficiently with a number of different sensors, data compaction techniques, and data transmission formats. The PDCP software described in Section 4 is the key to attaining these goals. Numerous examples of potential PDCP programs are presented to demonstrate that a microprocessor is capable of performing all tasks of current DCP random logic control units. Furthermore, the PDCP software system will contribute to reduced hardware costs by replacing random logic and complex sensor interfaces with inexpensive program memory. In addition, a PDCP can economically perform data compaction operations that are impractical for random logic implementation. This will allow users to obtain more information from each PDCP without exceeding transmission channel bandwidth limitations.

Section 4 also discusses the process of developing PDCP programs. A specific PDCP software organization designed to minimize software development costs is described. Traditional program development techniques are inadequate for PDCP software development by applications oriented users. A PDCP software development system which would allow applications oriented users to define the software structure of their individual PDCP's in familiar terms is recommended for a future study. The proposed editor/translator software development system retains the

efficiency of assembly language programming because users are allowed to create software systems from a library of subroutines written by experienced assembly language programmers.

An integral part of this study was the design and construction of the UT PDCP development system described in Section 5. This system is useful in the development, evaluation, and demonstration of potential PDCP programs. The UT PDCP provides all the capabilities of a PDCP control unit. Programs developed for use on the UT PDCP include sensor data input subroutines, data compaction subroutines, and data transmission subroutines. The UT PDCP is intended for use primarily as a program development and demonstration system. Therefore, the design of the system is not intended to represent the design of an actual PDCP. The system will serve as a machine through which NASA personnel can acquaint themselves with the details of PDCP software and PDCP software development.

A PDCP design should be based on a knowledge of currently available microprocessors and projected future microprocessors. Section 6 describes a weighting matrix technique for evaluating microprocessors and provides a microprocessor technology forecast covering the next five years. The weighting matrix provides a systematic procedure for generating PDCP application performance measures for the various microprocessors.

2. MICROCOMPUTER HARDWARE AND SOFTWARE

The control logic found in most current data collection platform designs is an example of the custom random logic approach to process control system design. Custom random logic combines individual logic elements (such as flip-flops, gates, counters, etc.) to perform a particular system control task. The primary tasks of the random logic of a DCP are to control the sensors, input sensor data, format the data, and control the DCP transmitter. CMOS logic is prevalent in current DCP designs due to the extremely low power requirements for CMOS circuits.

A major disadvantage of current hardwired DCP's is that they are based on customized designs which are intended to perform only specific tasks. In practice, applications for DCP's vary considerably due to the diversified requirements of the various users. As a result, the introduction of new DCP applications or a new data collection satellite can require the development of a new DCP design. This increases DCP costs due to both the increase in direct development cost and the increase in manufacturing cost resulting from the production of smaller quantities of each specific design.

What are the alternatives for a state-of-the-art, cost-effective DCP design? A close look at modern process control systems suggests an answer. Until the introduction of the microprocessor, state-of-the-art process control systems depended on custom built random logic designs or the dedicated minicomputer.

Custom built random logic can be cost effective if a large number of identical systems are required where the initial high design costs can be effectively shared by a large number of users. This approach certainly has merit in the DCP field, but the question still remains as to whether a more versatile and perhaps more cost effective approach exists with modern technology. Also, a DCP user requiring only a small number of platforms must pay an extremely high price for a new design unless a present design can satisfy his requirements.

The dedicated minicomputer approach would provide versatility in a DCP design as modification of the DCP tasks could be implemented in software. Unfortunately, the minicomputer is physically too large and heavy to be practical in a DCP design. Also, power requirements for the minicomputer cannot be met by the typical DCP power supply.

The microprocessor is causing a revolution in the process control field [1-4].* A microprocessorized DCP would be programmable like the minicomputer, yet cost, size, weight, and power requirements would be greatly reduced. Produced in quantity, a microprocessorized process control system provides reduced hardware costs over an equivalent custom random-logic process-control system. This is due mainly to a reduction in the number of components required [1-3, 5-7].

A microprocessor-based DCP could not only provide reduced hardware costs, but later modification of the DCP task could be achieved at a significantly lower cost compared to a custom random-logic design. The cost of redesigning a custom random logic system could well exceed the initial design cost. The microprocessor-based DCP would probably require only a software revision. Hardware cost would be limited to replacement or reprogramming of one or more read-only memories (ROM's). Software costs should in general be low compared to high redesign and hardware costs for a custom random logic-system.

Besides providing great flexibility [3, 4] in altering the task performed by the DCP, a microprocessor-based design offers the added advantage of providing data processing capability at a small additional cost. Providing data processing capability would require adding the necessary software to perform the additional task. Again, hardware cost would be minimal requiring only the addition of one or more ROM's. Software cost could be spread out among the large number of DCP users. Field programmable ROM's would be used in prototype models, and cost effective mask-programmed ROM's would be used in production DCP's to minimize hardware costs. Further hardware aspects of the proposed programmable DCP (PDCP) are discussed later in this report (Section 3.2).

*References are located at the end of each chapter.

To fully appreciate the potential capabilities and advantages of a microprocessor based DCP, one must be familiar with the basic concepts of a general computer system and the specific hardware and software characteristics of a microprocessor. Therefore, the remainder of this section is devoted to a discussion of the basic characteristics of a microcomputer.

The definition of a microcomputer differs from author to author; however, a suitable definition for the purpose of this study is that a microcomputer is a computer whose major component, the central processing unit, is a single microprocessor chip or microprocessor chip set. To complete the definition, the term computer must be defined.

Briefly, a computer is a device or machine which performs a programmed sequence of operations on data. A computer is comprised of three major components:

- Central Processing Unit (CPU)
- Memory
- Input/Output (I/O)

Instructions and data are placed in the computer memory. The central processing unit (CPU) fetches instructions from the memory and executes the instructions. The instructions are programmed to cause the CPU to process data. The set of instructions executed by the CPU is called a program. Data may be initially in memory, processed by the CPU, and results returned to memory. Input/output ports of the computer provide a means of entering and retrieving the data from the CPU and/or the memory. Note that data can be control words or signals as well as numerical quantities. Thus, the computer is capable of performing the following tasks:

1. Input data and control signals
2. Process and format data
3. Output control signals and data

In particular, a computer could input DCP sensor data; process the data (convolutional encoding, Manchester encoding, data compaction, etc.); provide and recognize DCP control signals; and format the platform data to emulate any of the typical DCP data formats.

Sections 2.1 and 2.2 discuss the hardware architecture and software aspects of the typical microprocessor. These sections reveal the processing and control capabilities of the microcomputer and provide insight into the question as to whether the microcomputer is indeed capable of performing the task of a typical DCP.

2.1 HARDWARE ARCHITECTURE OF THE MICROPROCESSOR

There are three basic components of the typical central processing unit (CPU) as depicted in Figure 2.1(1).

- Arithmetic/Logic Unit (ALU)
- Registers (Temporary Storage)
- Control Logic

The arithmetic/logic unit (ALU), as the name implies, performs the arithmetic and logical operations on the data processed by the CPU. Registers provide temporary internal storage for operands and results as well as address pointers for input/output and memory. The control logic provides the various signals for initiating processor functions and controlling external circuits. Recognition of external control signals is also a function of the control logic. The next three subsections explain in detail the functions of the ALU, registers, and control logic.

2.1.1 Arithmetic/Logic Unit (ALU)

All microprocessors have an arithmetic/logic unit (ALU) which performs the arithmetic and logical operations. As shown in Figure 2.1.1(1), the ALU generally has two multibit inputs.

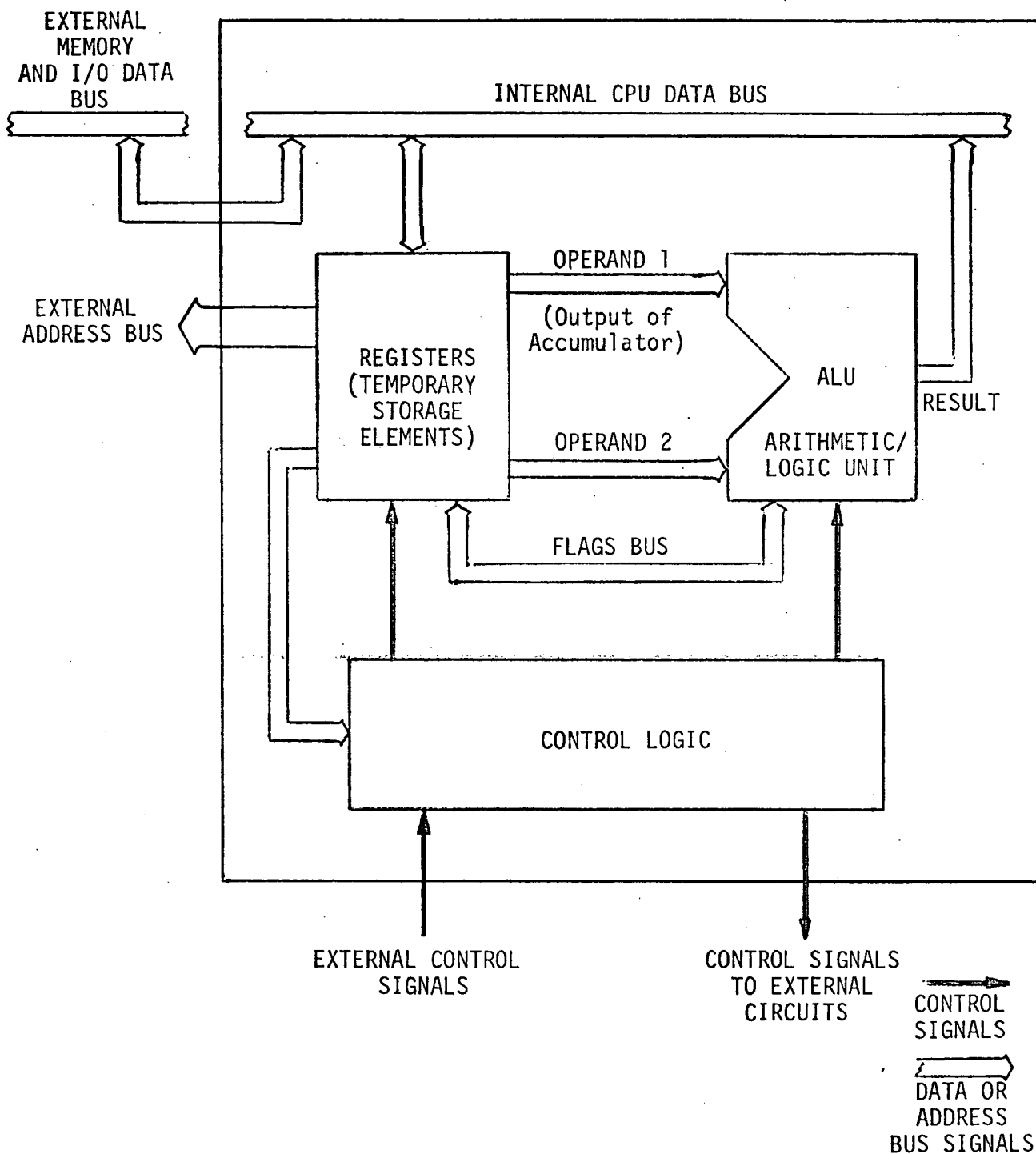


Figure 2.1(1) General Block Diagram of CPU.

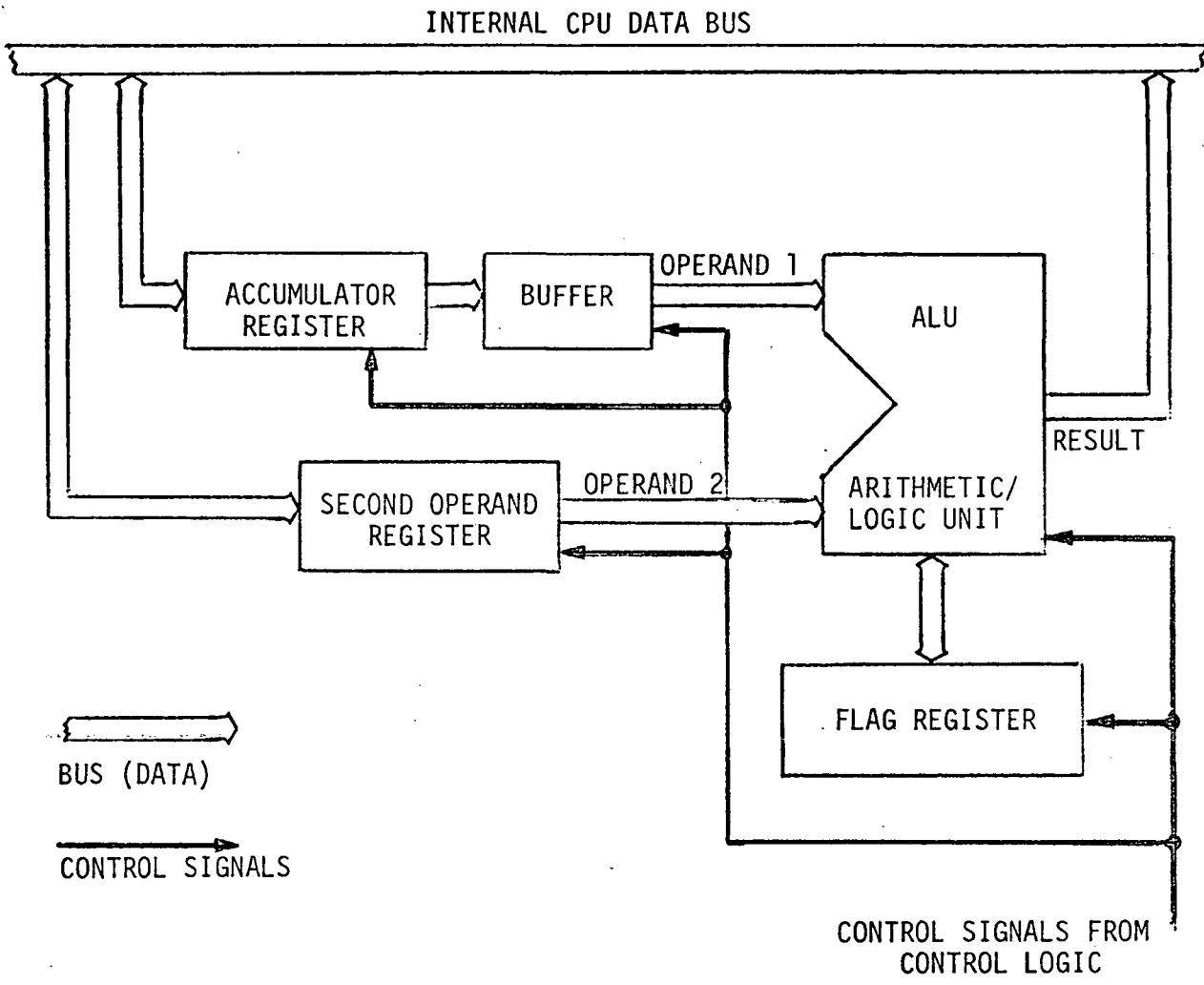


Figure 2.1.1(1) Typical Arithmetic/Logic Unit.

One input (Operand No. 1) to the ALU is usually the output of an accumulator buffer. The buffer stores the accumulator data throughout the ALU operation. Typically, all arithmetic operations are performed between the accumulator register and a temporary register, which in some CPU's may be a second accumulator. Input to the accumulator or second operand register may be memory data, an input port, or some other hardware register. The result of the operation is placed on the internal CPU data bus. Often the result is returned to the accumulator. In this case, the accumulator requires a buffer so that the original data in the accumulator can be stored and applied to the ALU while the arithmetic or logical operation is being performed. Thus, the result can be returned to the accumulator even though the result is a function of the accumulator's original contents.

All ALU's perform the basic arithmetic function of binary addition. Additional arithmetic/logical operations performed by most CPU's include binary subtraction, logical AND, logical OR, logical exclusive OR, complement and register bit shift. Although not yet common functions, some of the more recent microprocessors such as Texas Instruments' TMS 9900 also include binary multiplication and division.

A very simple ALU may only provide binary addition. In this case, binary subtraction, multiplication, and division may be performed only after the programmer has written software routines to perform these functions. Routines can be written to perform binary subtraction, multiplication, and division with only the basic binary addition instruction in conjunction with the ALU flags. Algorithms for performing various binary operations in terms of simpler binary operations can be written as needed for a given processing task. For example, multiplication by two is equivalent to a single bit shift left, and division by two is equivalent to a single bit shift right.

Although even the simplest ALU provides binary addition, the number of additional binary operations provided varies widely from microprocessor to microprocessor. One of the fundamental problems in selecting a microprocessor for a particular task is deciding how much power in performing arithmetic/logical operations is sufficient.

Referring again to Figure 2.1.1(1), notice that the data path between the flag register is bidirectional. The operations performed by the ALU produce an output which may be zero, have a carry or borrow, have even or odd parity, or have positive or negative sign. These auxiliary outputs are called flags and are stored in the flag register for subsequent ALU operations. The flag register is thus a source and destination register.

Besides providing additional data to the ALU, the flag register often provides information for the control logic. Program branching may be conditional on the result of an ALU operation (value of a flag). For example, the programmer may wish to call an overflow subroutine if an ALU operation produces a carry. By placing a call on carry instruction after the ALU operation instruction, the overflow subroutine would be called only if the result of the operation produced a carry. References 10-14 discuss the use of flags and conditional branching in more detail.

2.1.2 Registers

The hardware registers internal to the CPU provide temporary storage locations for data, instructions, and address pointers. The number of bits per register is usually an integer multiple of the characteristic word length of the CPU. All computers have a characteristic word length. The characteristic word length is generally determined by the size of the internal storage elements (registers) and the interconnecting buses. As an example, a CPU that has mostly eight-bit registers and eight-bit buses that transfer information between the registers is said to have an eight-bit word length. Some confusion results when the address bus is other than the characteristic word length.

Perhaps a better indication of the word length of the CPU is the maximum number of data bits stored at a single memory address. Most hardware registers of the CPU are usually equal in size to the memory word length. An example is the 8080A used in the UT PDCP development system. The word length of memory is eight bits. The internal general purpose registers in the 8080A CPU are eight bits. Certain address registers such as the program counter and stack pointer are

comprised of 2 eight-bit registers. Registers, then, are simply temporary storage elements in the CPU. There are many types of special purpose registers which are summarized below.

2.1.2.1 Program Counter Register - All CPU's contain a program counter register. This register stores the memory address from which instructions are fetched from memory. Each time an instruction is fetched, the program counter is normally incremented by one to point to the memory location where the next instruction is located. Multibyte instructions may require multiple incrementing of the program counter in order to read the entire instruction (usually accomplished automatically). Some instructions contain immediate data stored sequentially following the instruction. In this case, the program counter is incremented one or more times so that the entire instruction is read in. After the last byte of the instruction is read, the program counter register is incremented again to point to the first byte of the next instruction.

Most CPU's provide at least two means to alter an otherwise sequential fetching of instructions. The jump-type instruction alters the program counter register contents according to a particular addressing mode specified by the jump instruction (see References 10-14). The subroutine call instruction allows program flow to another area of memory, and upon completion of the "called subroutine" the program returns to the instruction following the call. Execution of a subroutine call requires a stack to store the address of the instruction following the call instruction. When a call instruction is encountered, the address of the instruction following the call instruction is stored in the stack. Then the program counter is loaded with the address of the subroutine and the subroutine is executed. Upon completion of the subroutine, the address previously stored in the stack is returned from the stack and placed into the program counter. This action is initiated by placing a return from subroutine instruction as the last executed instruction at the end of the subroutine. This causes program execution to return to the calling program.

2.1.2.2 Stack and Stack Pointer - A CPU that provides subroutine calls must have a stack. A stack is a last in, first out (LIFO) buffer storage element; that is, the last value stored is the first to be read out.

Two types of stacks are found in microprocessors. The hardware stack (e.g., the Intel 8008) is a LIFO buffer storage element internal to the microprocessor integrated circuit. The software stack (e.g., Intel 8080A or Motorola 6800) is a LIFO buffer storage element implemented in the computer random-access memory or some memory external to the CPU integrated circuit.

An advantage of the software stack is that stack size is limited only by the amount of external memory the programmer dedicates to the stack function. In the case of a software stack, an internal (to the CPU) address register called a stack pointer register is used to address the memory area which is software programmable to serve as the CPU stack. The stack pointer register is generally the same size as the program counter register.

Besides storing addresses for subroutine calls, the stack may also serve as a data storage element. For example, in the 8080A, there are "push" and "pop" instructions which provide for storage of the contents of the various hardware registers. An instruction set possessing this capability has a significant advantage over the ones that do not. Suppose an external interrupt request occurs while the CPU is executing a main program. Further, suppose the interrupt service routine may change CPU register contents depending on which device issues the interrupt. Without stack storage of processor status (contents of CPU registers), the interrupt may have to be delayed until a point is reached in the main program where loss of CPU status is not critical. With stack storage of CPU status, the interrupt can be processed almost immediately with processor status "pushed" onto the stack at the beginning of the interrupt service subroutine and CPU status restored by "popping" the stack at the end of the interrupt subroutine.

2.1.2.3 Accumulator and General Purpose Registers - One of the operands used in ALU operations is generally the current contents of the accumulator. The number of bits in the accumulator register is a good indication of the characteristic word length of the CPU.

During an ALU operation, the accumulator register's contents and some other register's contents are applied to the inputs of the ALU [see Figure 2.1.1(1)]. The result of the operation is usually returned to the accumulator via the internal data bus of the CPU.

In contrast to other operand registers which may be input to the ALU, the accumulator contents are, in general, altered following the completion of an ALU operation. The accumulator is a source for operands and, in most cases, also a destination for results. The buffer register following the accumulator provides the temporary storage of the accumulator register data while the ALU operation takes place. This allows the result of the ALU operation to be returned to the accumulator.

Other hardware registers which serve only as a source or destination register (but not both simultaneously) are called general purpose registers. Some microprocessor architectures do not provide general purpose registers, thus requiring all ALU or other CPU operations to be performed between accumulators, memory, and I/O only. Microprocessors lacking general purpose registers often provide more flexible memory addressing modes to allow single or double byte instructions to execute ALU or other operations with different memory locations. In other words, external memory is used for general purpose registers.

2.1.2.4 Instruction Register - The instruction register is used for temporary storage of the instruction fetched from memory. At the beginning of the instruction cycle (see Section 2.1.3), the instruction is fetched from memory and loaded into the instruction register. The word length of the instruction register is thus equal to the word length of memory.

Corresponding to each particular operation the CPU must perform is a unique code called the instruction or operation code. For an n-bit

machine, there are 2^n unique instruction codes that could be used. Thus, an eight-bit machine allows for up to 2^8 (256) unique instruction codes. The codes are stored sequentially in memory and are fetched one by one into the instruction register where they are stored during decoding and execution of the instruction.

2.1.2.5 Address Registers - The stack pointer and program counter are examples of special function address registers. Some CPU's have other address registers which provide absolute memory pointers, relative memory pointers, input port address pointers, and output port address pointers.

In some CPU's, a general purpose address register may be used for many functions. For example, the 8080A has two general purpose registers (H and L) which are absolute memory address pointers; yet, they may also be used for data storage, stack pointer storage, and even double precision shift left and add.

Memory reference instructions require an address register to hold the address of the memory location to be referenced by the instruction. Sometimes this address register is loaded with an address fetched by memory as part of a multibyte instruction. Some CPU's such as the Motorola 6800 provide an address index register which is used as an indexed memory pointer. Data can be stored or retrieved from the address specified by the index register plus or minus a fixed amount (relative addressing).

2.1.3 Control Logic

The primary and most complex component of the CPU is the control logic. The control logic provides the sequential signals that perform the various processing tasks.

A necessary input signal to the control logic of all CPU's is a master clock. Some microprocessors such as the MOS Technology MCS 6502 have an internal clock while others require an external clock. The popular Intel 8080A used in the UT PDCP requires an external two-phase clock.

Other inputs to the control logic include decoded instructions from the instruction register, flags generated by previous ALU operations, and external control signals such as interrupt, wait, or DMA (direct memory access) requests. The function of the control logic is to take all these input signals and output the necessary signals in the proper sequence so as to execute the appropriate processing task.

The CPU operates in a cycle. That is, the processor fetches an instruction, decodes the instruction, executes the instruction, fetches the next instruction, etc. The master clock provides a reference timing signal to synchronize the events in a processor cycle.

An instruction fetch, instruction decode, and instruction execution is often called an instruction cycle. The instruction cycle generally requires one or more subcycles called machine cycles. Each distinct function performed during a machine cycle is called a state. A state generally requires one or more periods of the master clock. The first state of every instruction cycle is an instruction fetch. During the instruction fetch state, the program counter provides the address of the next instruction, and a memory read subcycle places the instruction fetched from memory on the internal data bus. From the data bus, the instruction is loaded into the instruction register. The instruction remains in the instruction register throughout the instruction cycle.

Following the decoding of the instruction, cycle status information is provided by the control logic. Cycle status signals indicate the type of machine cycle that is to be performed. For example, the instruction cycle may be input/output (I/O), memory referencing (memory read or write), internal CPU processing, or a combination of these.

Input/output cycles may not be provided by some microprocessors. This class of microprocessors relies on a technique called memory-mapped I/O. That is, input and output ports are treated like any other memory location. Data sent to or from a memory-mapped I/O port appears to the CPU to have been written into or read from memory. All microprocessors have the capability of memory-mapped I/O. Those microprocessors providing I/O cycles offer a second method to input and output data to and from

the CPU. The I/O cycle is flagged by I/O status bits following the decoding of an I/O instruction. The I/O flag bits combined with the I/O port address provide the signals to select the external I/O ports and whether the operation is to be an input or output. Memory mapped I/O uses memory read or write flag bits combined with reserved memory addresses to operate the external I/O ports.

Memory reference subcycles are either memory read (data is returned from memory) or memory write (data is stored in memory). Several machine states are usually required to execute a memory reference subcycle. For example, in a memory read subcycle status information is first provided by the control logic, indicating a memory read cycle is to be executed. Following status information, the memory address is sent to the external address bus. At this point in the subcycle, depending on the capabilities of the microprocessor, the memory read signal is issued or delayed if necessary to allow for minimum memory access time. Finally, data is returned from memory on the external data bus and routed to the appropriate destination register within the CPU.

For a memory write subcycle, the sequential operation is reversed slightly compared to the memory read subcycle. In this case, the first and second steps are similar; that is, status information is sent out by the control logic indicating a memory write subcycle is to be performed. Then, the memory address is placed on the external memory address bus. The third step differs in that data (instead of a control signal) is sent out by the CPU to the external memory data bus. Finally, the memory write signal is sent out by the control logic. Again, some microprocessors provide a means to delay the memory write signal to allow for minimum memory access time.

Microprocessors providing delay for slow memory or I/O have what is called a ready control signal input and a special machine state called a wait state. As long as the ready control signal input is true, CPU operation takes place at full machine speed set by the CPU master clock. Making the ready input false temporarily freezes CPU operation while retaining all CPU status. Usually, the ready input is checked just prior

to the memory or I/O, read or write control signal. Thus, address and data are held constant while in the wait state. If memory or I/O is too slow when the CPU operates at full speed, a wait-cycle generator can be used to place the CPU in the wait state for a minimum period of time required for memory or I/O access. Memory or I/O that is fast enough to operate with the CPU at full machine speed is said to be capable of operating synchronously with the CPU. Memory or I/O which is not fast enough and requires the CPU to synchronize with the external memory or I/O cycle is called asynchronous memory or I/O. Assuming the microprocessor has a ready signal input, slow memory requires an external wait-cycle generator to delay the appropriate read or write control signal.

The class of microprocessors that do not provide a wait state require additional hardware to operate with asynchronous memory or I/O. First, a circuit is needed to recognize the memory or I/O cycle. The problem arises at this point as the only alternative to suspend CPU operation, yet retain data on the address and data buses, is to provide external address and data storage and place the CPU in DMA if indeed DMA is provided. A possible second alternative is to stop the CPU master clock. Stopping the clock, however, is usually not a solution since many processors are dynamic and internal status requires clock refresh cycles. In practice, the clock pulse is stretched just long enough to allow the memory or I/O access. The hardware to provide this operation is considerably more involved than a wait-cycle generator circuit.

Using slow memory or I/O with a microprocessor may be possible without any external wait-cycle generation circuitry. If the application does not require the microprocessor to be run faster than the memory or I/O access times, the CPU clock frequency can be reduced to meet memory or I/O access times. This solution is applicable only if the minimum CPU clock frequency requirement is met. For example, the Pro-Log MPS system used in the UT PDCP (see Section 5.1) uses a $1.66\overline{66}$ μsec state time which allows the use of slow ROM without need for a wait-cycle generator.

Internal processing subcycles generally require the least amount of time (fewest number of states) to execute. Typical internal processing cycles are register-to-register move, arithmetic operations between registers, and increment a register's contents. Since the internal operations involve only internal storage elements, no time is required for access of external I/O or memory. The minimum cycle time specified for a particular machine is usually the minimum time in which some simple internal CPU operation can be performed. Minimum cycle time for a CPU is not necessarily a good criteria for evaluation of a particular microprocessor. Although the minimum cycle time of two microprocessors may be the same, one may perform a larger function than the other during the minimum cycle time.

As presented earlier, some instruction cycles may be a combination of basic I/O, memory reference, or internal CPU operation subcycles. For example, one instruction may involve reading data from memory and adding the data to the internal accumulator register. Usually two machine subcycles are required to perform an add memory to accumulator type instruction. The first machine subcycle would fetch the add memory instruction, decode the instruction, send out the memory address of the data to be added, and fetch the data to an internal temporary storage register. Several machine states would be required to perform this part of the instruction. The second machine subcycle would be an internal CPU operation subcycle which would be to input the accumulator and the temporary storage register contents to the ALU; direct the ALU to add the operands; and finally, place the result in a destination register (usually the accumulator). Many combinations of machine subcycles can be generated to perform various instruction cycles. The programming of machine subcycles comes under the topic of microprogramming which is discussed in Section 2.2.2.3.

Interrupt and hold control signal inputs are provided by most microprocessors. An interrupt, as the name implies, allows temporary interruption of main program processing to allow for execution of some subroutine. The principle advantage of interrupt capability is evident when using a peripheral much slower than the CPU. Suppose output to a

slow peripheral is required. Without interrupt provision, the processor must wait for the slow output device. With interrupt capability, the processor can be continuously processing, and when the slow output device is ready to take data, an interrupt request is sent to the CPU. The output routine to service the interrupt is executed, and the processor returns to the main processing task. The CPU need never wait for a slow peripheral.

The hold signal input provides just the opposite speed difference capability. Suppose an external peripheral is capable of outputting or inputting data faster than the CPU can process or generate data. If the processor provides a hold cycle, data may be passed between memory and the external device at a rate equal to the access time of the memory plus the time required for the CPU to recognize a hold request. When the control logic recognizes a hold request, the address and data buses are placed in the tri-state mode (floating), and a flag is generated by the control logic to indicate to the external circuitry that the address and data buses are now available for use by the external device that requested the CPU hold cycle. The external device may then address memory directly and perform a memory read or write cycle independent of the CPU. This operation is called direct memory access or simply DMA.

In PDCP applications, the data rate between CPU and sensors, or CPU and the platform transmitter, is generally slow compared to the processing speed of the microprocessor. Therefore, in PDCP applications, the DMA or hold capability will probably not be required.

The only remaining topic of the hardware architecture of the microprocessor that must be considered is the problem of system start-up. All microprocessors have the capability to load the program counter with some particular address upon the application of an external start-up signal. This is equivalent to initiation of a program since the program counter is loaded with the starting address of the program. Perhaps the best way to discuss hardware start-up is through examples.

The 8080A microprocessor provides three techniques for initiating a program. The most fundamental of these is application of an external reset signal to the reset control signal input of the 8080A chip. The control logic recognizes a reset request and places zero in the program counter. All other CPU status is unchanged. Thus, a vectored start-up to absolute address zero is performed. The problem with this method is that either location zero must be the starting point of the start-up program or the location must be preloaded with a jump instruction to the desired location. If location zero is ROM, then on power-up, the system may be started immediately with only a reset pushbutton required. If location zero is volatile RAM, location zero must be preloaded. A further complication results if all memory is volatile; in this case, a short boot-strap loader program must be toggled in by hand, and system start-up would include jumping to the boot-strap loader program which loads the system program(s). Usually, the simple boot-strap loader is used to load a more complicated loader with some sort of error checking. The more complicated loader then loads the system program(s) and passes control to the system program(s) at the conclusion of the loading routine. One microprocessor, the RCA CPD 1802 provides a hardware boot-strap. The CPD 1802 provides a special DMA cycle which permits sequential memory loading starting at location zero.

The second and third methods of starting the 8080A series microprocessors involve the interrupt capabilities of the device. There are eight single-bit restart instructions which cause an unconditional call to eight particular locations in memory (locations 0, 10, 20, 30, 40, 50, 60, and 70 octal). That is, the present value of the program counter is saved in the stack, and the program counter is loaded with one of the eight appropriate restart locations. All other CPU status is unaffected. The restart \emptyset (restart to location \emptyset) is similar to a reset, only additional hardware is required to jam the interrupt restart instruction onto the data bus. Also, the stack is affected by the restart instruction. The original 8080 provides only the reset and restart interrupt methods of system start-up. The newer 8080A series of microprocessors permit a three-byte instruction interrupt. That is, a three-byte instruction may be jammed onto the data bus during the interrupt. This permits unlimited

vectoring of the program counter since a call or jump instruction can specify a branch to any of the 2^{16} (65,536) possible memory locations. Of course, additional hardware is required to interrupt with three bytes since three bytes must be presented sequentially to the CPU data bus instead of only one.

Most microprocessors provide some sort of reset input that vectors the program counter to a particular location although the location is not always zero. The Motorola 6800 provides an interesting start-up technique which differs from the three methods just described. The desired program starting address is preloaded in the top two locations of memory. The reset signal initiates a special machine cycle which loads the program counter with the address stored in the top two memory locations. The resulting operation is equivalent to an unconditional jump to the memory location specified by the contents of the top two memory locations. Again, the top memory locations must be either non-volatile or pre-programmed.

2.2 MICROCOMPUTER SOFTWARE ASPECTS

A microcomputer must be programmed in order to perform a process control task. In the case of a microprocessorized DCP, the DCP system algorithms must be converted into a set of instructions (the microcomputer program) that can be directly loaded into the system memory. The process of converting system algorithms into machine executable instructions is often called coding.

As depicted in Figure 2.2(1), there are many levels of microcomputer coding. At the top of the scale are compilers and interpreters such as FORTRAN, PL/1, and BASIC. These high-level languages are machine independent in that a program written in any high-level language can be executed on any machine which supports that particular high-level language. For example, one could write a FORTRAN program to run on a DEC PDP-11/45 and also execute the program on an Intel 8080A microcomputer with little or no change in the original FORTRAN source program. Of course the original program would require recompiling using an 8080A FORTRAN compiler.

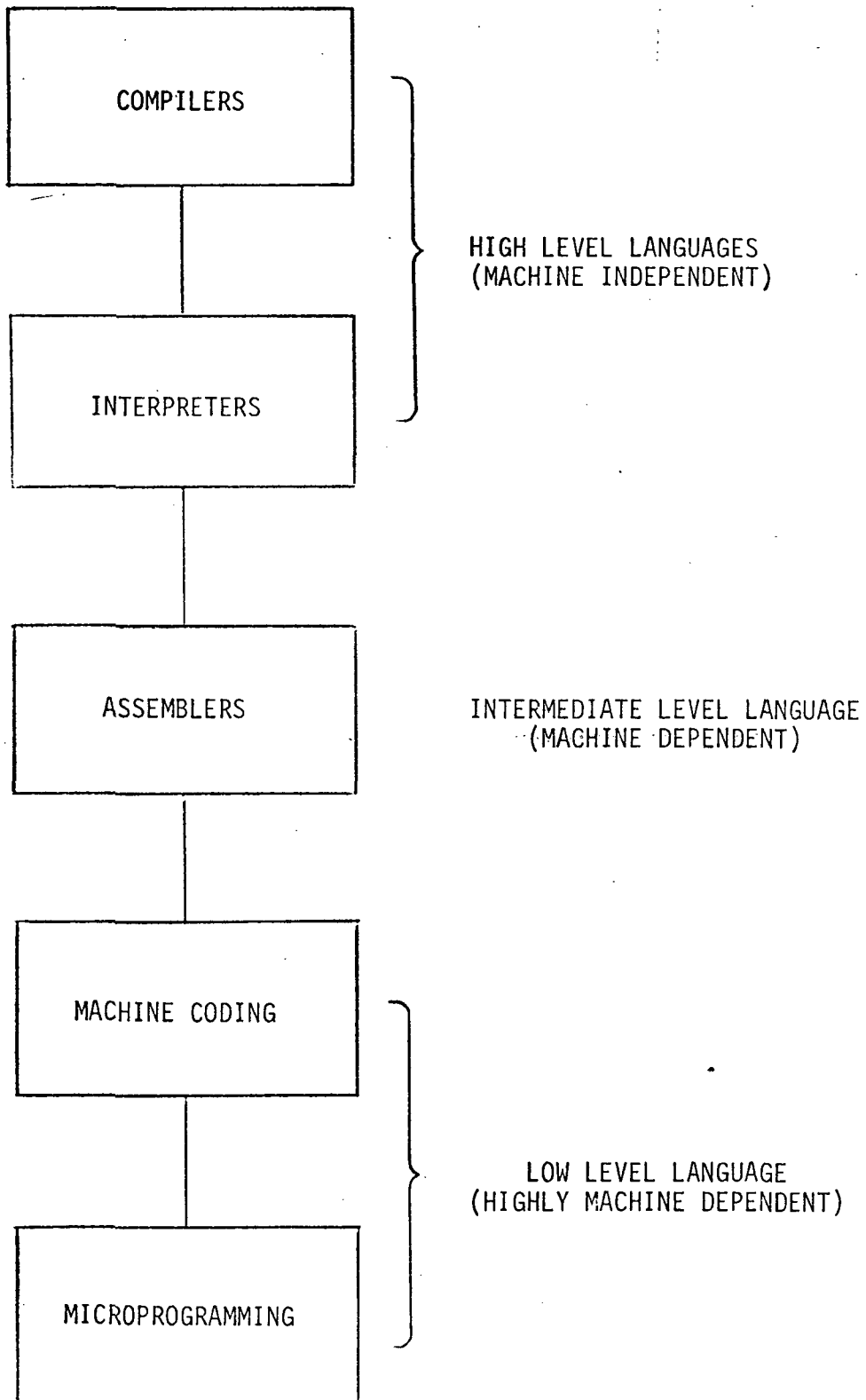


Figure 2.2(1) Coding Levels.

Intermediate-level programming is a more complex process than high-level programming. An assembly language programmer must have a thorough knowledge of the particular microprocessor's instruction set and a complete understanding of the particular microcomputer's hardware architecture. Assembly language programming is called machine dependent programming since instruction sets vary greatly from one microprocessor to another.

At the bottom of the coding scale is low-level programming. Machine language programming is extremely time consuming and should generally be avoided. Machine language programming is the process of hand loading the machine binary digits representing the various instructions the programmer wishes the CPU to perform. The binary digits are literally "toggled in" from a switch register or typed in as numbers from a keyboard. This process requires considerable time and effort from the programmer. Machine coding should not be required for PDCP applications since assembly language programs are available for most current microprocessors. The UT PDCP system described in Section 4 provides machine level coding capabilities as well as simplified symbolic assembly language programming.

The most basic level of microcomputer coding is a highly machine dependent technique called microprogramming. Microprogramming is sometimes referred to as the bridge between hardware and software [8]. This is because a microprogram consists of the control words used to decode machine instructions (software) into machine operations (hardware). The microprogram resides in the control storage element of the CPU. The control storage device is either a read only memory (ROM) or programmable logic array (PLA). Some microprocessor manufacturers allow the user to specify the microprogram. Since the microprogram defines the instruction set of the microprocessor, this allows the user to develop the optimal software structure for a particular application.

In addition to programming languages, the microcomputer programmer requires several utility programs to complete the programming task. A text editor program is useful in preparing the source code of a program.

Source code is the name given to the human readable program. Source code is compiled or assembled into equivalent machine executable object code. Another required utility program is a loader which is used to load the object code output of a compiler, interpreter or assembler program directly into the microcomputer memory. Finally, debugging and simulator programs aid in testing the object program. The remainder of this section provides a more detailed description of the software aspects of the microcomputer.

2.2.1 Microcomputer Instruction Sets

A microcomputer instruction provides the binary information required by the control logic of the CPU to perform a particular processing task. According to Weiss [9], microprocessor instructions can be conveniently grouped into four basic categories:

1. Data Movement
2. Data Manipulation
3. Decision and Control
4. Input/Output

Data movement instructions control the movement of data from register-to-register, register-to-memory, memory-to-register, and memory-to-memory. Figure 2.2.1(1) graphically illustrates possible sources and destinations for data movement. Note that data flow to and from input/output ports is also depicted in Figure 2.2.1(1). Input/output data movement is a special case of the general class of data movement instructions.

Data manipulation instructions provide arithmetic and logical operations on data. Arithmetic instructions include add, subtract, multiply, divide, and increment/decrement. Typical logical operations are logical AND, OR, exclusive OR, complement, compare, and rotate/shift. All microprocessor instruction sets provide the basic binary add instruction. If other data manipulation instructions are required but are not included in the instruction set, the needed instructions can usually be implemented with a subroutine.

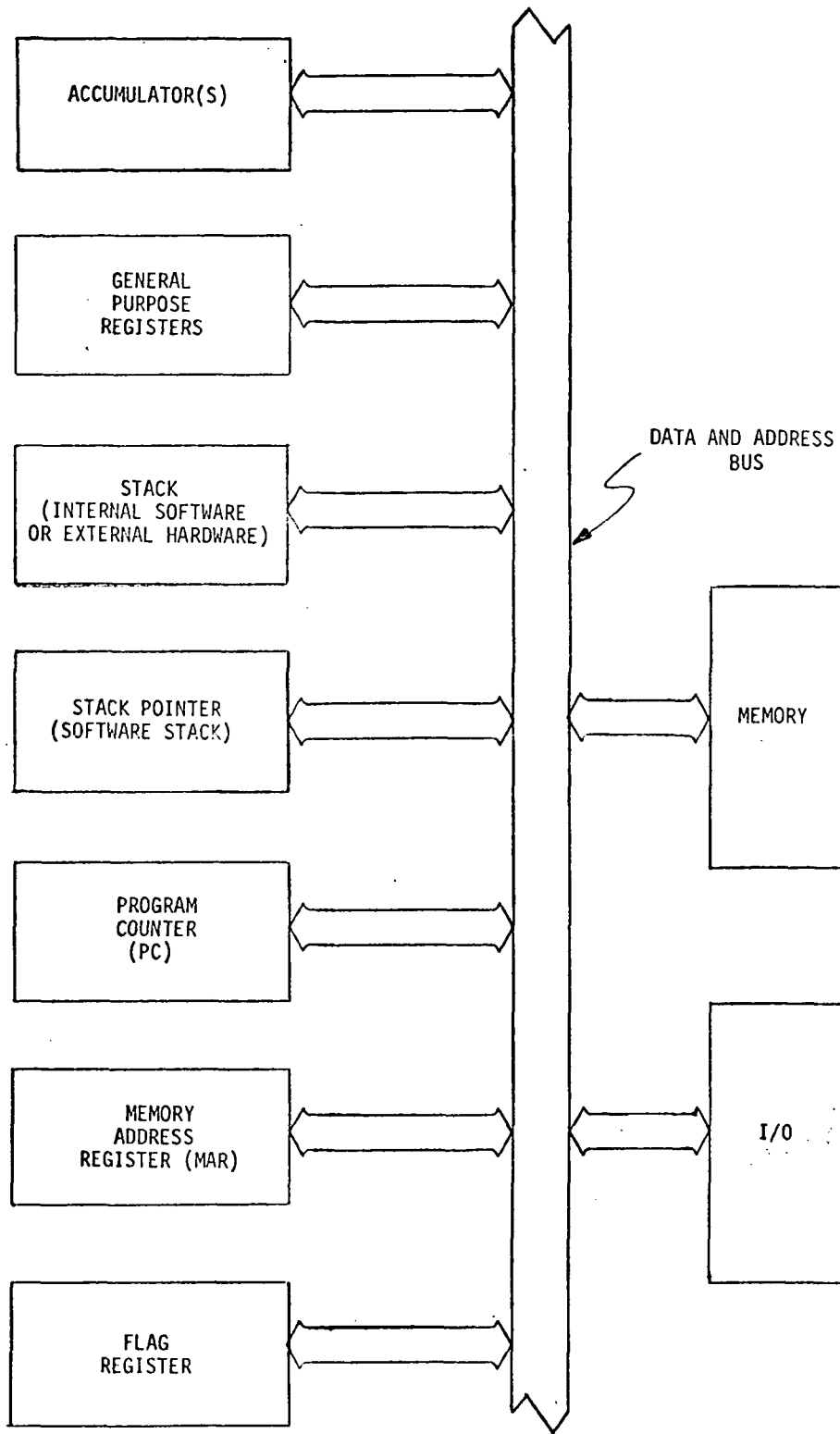


Figure 2.2.1(1) Data Movement in the Microcomputer.

Microprocessors execute instructions in a sequential manner unless otherwise directed by a decision and control instruction. Non-sequential alteration of the program counter is a result of the execution of a conditional or unconditional jump, jump to subroutine, return from subroutine, or skip instruction. Unconditional jump or call instructions are sometimes called branch instructions. Also, some authors prefer the use of the simpler term "call" to indicate a jump to subroutine type instruction.

The choice of input/output device interfacing depends largely on the nature of the input/output instructions provided by the particular microprocessor. Many microprocessors provide no formal input/output instructions. Instead, memory referencing instructions must be used with a technique called memory-mapped I/O (see Section 2.1.3). Input/output instructions provide data transfer between the microcomputer and external I/O devices.

An explanation of the instruction set of a particular microprocessor can generally be found in the user's manual published by that microprocessor's manufacturer. A complete description of the Intel 8080A instruction set used in the UT PDCP development system is contained in the Intel 8080A User's Manual [10]. A copy of the Intel 8080A User's Manual is included with the UT PDCP System Operation Manual.

Microcomputer instructions are similar to typical minicomputer instructions. The major difference is that the execution speed of instructions is generally much faster for a minicomputer. Recent technological advances, however, are narrowing this speed gap. Since the form and resulting operations of microcomputer instructions are similar to minicomputer instructions, texts and papers discussing general minicomputer instruction sets are applicable to microcomputer instructions. References 9, 11, and 12 provide excellent discussions of typical mini- and microcomputer instruction sets.

Microcomputer instructions sets vary considerably from manufacturer to manufacturer. The ability of a particular microprocessor to efficiently implement a PDCP is primarily a function of that particular

microprocessor's instruction set. Evaluation of the microprocessor's instruction set must be a primary consideration in the selection of a particular microprocessor for a PDCP design. As a result of the software development incorporated in this study (see Section 4), certain types of instructions appear to be highly desirable for PDCP applications. Choosing a microprocessor that contains as many of the desirable instruction types as possible will result in reduced memory costs due to more efficient coding. These desired instruction types are given higher weighting factors in the microprocessor evaluation presented in Section 6 of this report.

2.2.2 Microcomputer Programming

As indicated at the beginning of Section 2.2, the procedure for converting process control system algorithms to machine executable instructions is often called coding. Efficient coding requires a well structured algorithm. Flowcharting is a useful technique for reducing the algorithm to a form which can easily be converted to a computer program. The flow chart is a graphical representation of the distinct operations required to implement the computer program. Numerous examples of program flowcharting appear throughout Section 4 of this report. Typical flow chart symbols are presented in Figure 2.2.2(1). Once the flow chart for a program is developed, the programmer proceeds to implement each block of the flow chart using an appropriate programming language.

The basic programming tools available to the microcomputer programmer are:

1. Programming languages
2. Utility programs
3. Microprogramming

The first two tools are programs that are actually run on the particular microcomputer (resident program) or on some other computer (cross programming). Programming languages and utility programs are discussed in

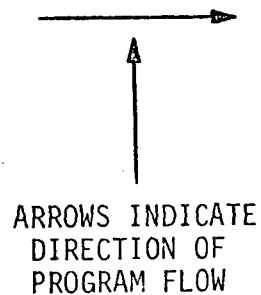
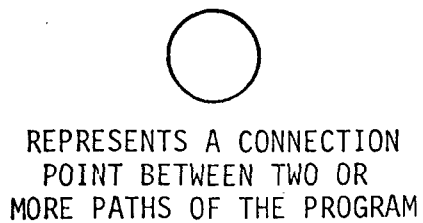
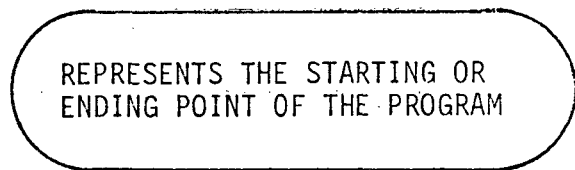
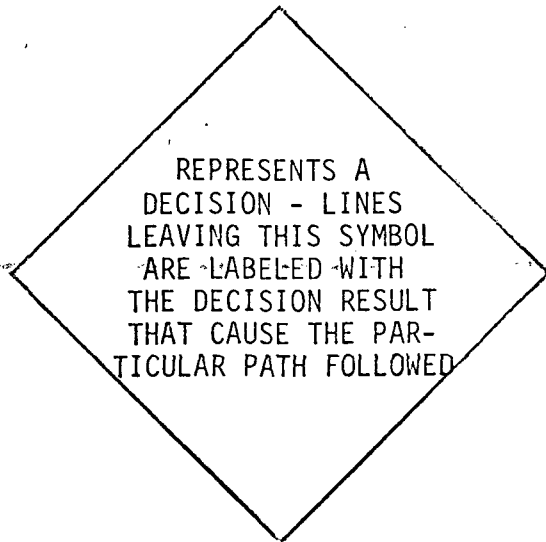
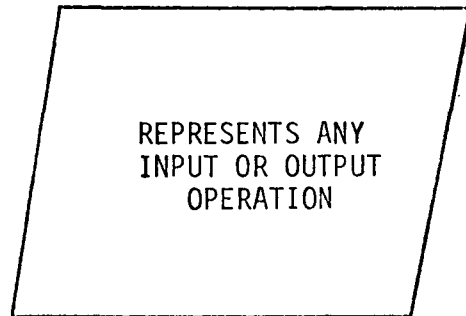
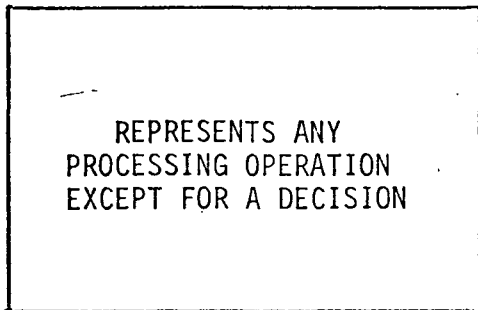


Figure 2.2.2(1) Typical Flow Chart Symbols.

Sections 2.2.2.1 and 2.2.2.2, respectively. The third tool, microprogramming, is discussed in Section 2.2.2.3 of this report.

2.2.2.1 Programming Languages - Programming languages are conveniently grouped into three classes [see Figure 2.2(1)]:

1. High-level languages (compilers and interpreters)
2. Intermediate level languages (assemblers)
3. Low-level languages (machine language and microprogramming)

High-level languages include compilers and interpreters such as FORTRAN, PL/M, COBOL, FOCAL, or BASIC. All five of these languages are currently available for one or more microprocessors. For example, the Intersil 6100 will support all software available for the DEC PDP-8E minicomputer including FOCAL, BASIC, COBOL, and FORTRAN. The Intel 8080A used in the UT PDCP development system is supported by FORTRAN, PL/M, BASIC, FOCAL, and COBOL. The significant advantages of high-level programming are machine independence and a reduction in programming time. High-level languages can reduce programming time by 50 to 80 percent. Also, existing high-level language programs can be recompiled to run on a microprocessor which supports the particular high-level language used. The high-level programming languages were each developed for particular programming tasks. FORTRAN is designed primarily to solve mathematical problems and thus may be useful for PDCP data processing. COBOL, however, is a business oriented language and would have little application in the PDCP field. PL/M, a microprocessor compiler originated by Intel, is a subset of IBM's powerful PL/1 compiler. PL/M provides instructions oriented toward commercial control and scientific problem solving. BASIC is an interpretive language in that each line of source code is compiled as the program executes. BASIC requires a relatively large amount of memory to execute even short programs. This is because the BASIC source program and the BASIC interpreter program must both be resident in memory during program execution. A compiler differs from an interpreter in that the final compiled program is machine executable object code. The compiler is not needed once the object code is

generated. Finally, FOCAL is a language which was developed by Digital Equipment Corporation to fill the gap between BASIC and FORTRAN. Like BASIC, FOCAL is an interpretive language that requires considerable memory overhead to run even short programs. Neither BASIC nor FOCAL are recommended for potential PDCP application due to their high memory requirements.

The principle disadvantage of using any high-level language for PDCP programming is the inefficiency of the compiled object code. Another significant disadvantage is inadequate input/output flexibility. Also, program execution times are generally unknown and uncontrollable. PDCP software timing routines require precise knowledge of the number of machine states required to execute various subroutines. Even interrupt timing techniques (see Section 4) require knowledge of execution times. In this case, worst case execution times must be known to guarantee that the subroutine will be completed within the allotted time span.

Assembly languages are intermediate level languages. An assembler translates symbolic machine instructions or mnemonics directly into machine executable object code. A mnemonic is a short form symbolic name given to each instruction in the instruction set of a particular microprocessor. Mnemonics used to represent instructions vary greatly from manufacturer to manufacturer. This is due to the wide variation in microprocessor instruction sets. Since microprocessor instruction sets vary from machine to machine, assembly language programming is machine dependent. A primary disadvantage of assembly language programming is a direct result of the variation of instruction sets. The microcomputer programmer must be thoroughly familiar with the instruction set of the machine to be programmed. Furthermore, efficient assembly language programming requires complete understanding of the particular microcomputer's hardware architecture.

Many types of microcomputer assemblers are available. Cross assemblers are run on a machine other than the microcomputer for which the program is written, whereas resident assemblers are run on the machine for which the program is written. All assemblers translate symbolic

instructions to their equivalent object code. Several additional features are often found on current microcomputer assemblers. The most useful feature is a provision for symbolic addresses and constants. These symbolic names and labels can be used to represent address and data constants. This feature reduces programming errors by freeing the programmer from the burden of keeping track of absolute addresses and constants. Some assemblers even provide algebraic manipulation of address and data expressions. Recognition of pseudo assembly directives is provided by most assemblers. For example, the pseudo directive "END" informs a typical 8080A assembler that the location of the END directive marks the end of the source code to be assembled. Macroassemblers usually provide for all the above features plus the assembly of macroinstructions. A macroinstruction is a single line instruction used to represent a multi-instruction sequence. This feature supplies some of the programming simplicity of a high-level language since a single line of source code can replace many machine instructions. Note, however, that assembly language efficiency is retained. Other features provided by some microcomputer assemblers include conditional assembly directives, relocation and linkage of multiple program segments, optional assembler listing formats, and loading of object code output directly into memory or to some external storage device for later loading into memory. Microcomputer assemblers are quite similar to typical minicomputer assemblers. References 13 and 14 provide general discussions of typical mini- and microcomputer assemblers.

2.2.2.2 Utility Programs - Once a program has been written and translated to object code, the object code must be loaded into the system memory and the program must be checked for correct operation. A loader program is used to load the object code directly into memory. A loader program is not required when using a programming language that outputs object code directly to memory. This feature is provided by some microprocessor assemblers. Certain high-level languages do not have object code output (such as BASIC). In this case, a loader is required to enter the source code. Several types of program loaders are available for current microcomputers. The simplest is a boot-strap loader. The boot-strap loader program is purposely very short and

provides none of the error checking or other advanced features typical of the more complex general purpose loaders. A simple boot-strap loader could be used to load any program; however, the boot-strap loader is typically used to read a more sophisticated loader. Control is then transferred to the second loader which usually provides error-checking and automatic program start-up following the loading process. Unless a machine has non-volatile memory pre-programmed with a loader, the programmer must manually load a boot-strap loader program to initiate subsequent program loading. The UT PDCP development system includes an error-checking cassette loader routine as an integral part of the system monitor program (see the System Operation Manual). The loader routine is stored in PROM (programmable read only memory) and is available to the user at all times.

Once a user program is loaded into memory, the program can be executed and checked for correct operation. If the program runs correctly, the software task is complete. Typically, however, programs contain "bugs" which prevent correct operation of the program. The bugs must be removed by a technique called program debugging. Microcomputer programs which allow examination and alteration of memory contents to assist the programmer in debugging and correcting the program are available. Often these programs include a routine which will print the CPU status (register contents). Breakpoints may be set in memory to temporarily suspend program execution when a specified point in the program is encountered. Thus, breakpoints permit the programmer to examine CPU status and memory at any point of program execution. This enables the programmer to debug the program in small sections. As bugs are found, they are corrected, if possible, by altering the instruction sequence in memory. Often the programmer will initially insert "no operation" instructions throughout an untested program. This provides memory space between the original program instructions so that additional instructions can be inserted if they are required. The UT PDCP system monitor contains extensive program debugging aids.

A simulator program provides a very powerful technique for testing and debugging a microcomputer program. A simulator program, as the name

implies, simulates the operation of the microcomputer. Program development and testing is simplified by using a large computer system supported by peripherals such as a video display terminal, a high-speed line printer, and a disk drive. The full potential of a simulator can only be realized by using such a developmental system.

The simulator provides the basic function of program instruction tracing. That is, the result of executing each instruction can be displayed and verified by the programmer. Any discrepancy between what the program actually does and what the programmer feels should be happening is immediately evident. The programmer can then patch the program to correct errors as they are found. The process continues until the program runs without error. Some simulators provide additional features such as counting machine states required to execute a program segment, optional tracing and listing modes, and simulated memory examination and alteration. Final program verification must be performed on the actual PDCP system since the simulator cannot verify correct operation of I/O and control interfacing. For example, external A/D conversion may be dependent on software timing which must be verified on the actual PDCP hardware system. Full simulator potential plus final checkout could be achieved simultaneously by using a microcomputer development system that includes the same hardware interfaces that are used on the PDCP system.

2.2.2.3 Microprogramming - A microprogram is an integral part of the control logic of most microprocessors. In a microprogrammed control unit, the numerous individual operations required to execute each instruction cycle are defined by microinstructions fetched from a read-only-memory or a programmable logic array. Therefore, each macroinstruction written by a user is actually executed by a microprogram, and the instruction set of the microprocessor is defined by the set of microprograms stored in the CPU control unit read-only-memory.

Some microprocessor organizations are designed to permit users to specify the microprogram. These microprogrammable microprocessors offer users the potential advantage of defining an optimal instruction set for their particular application. Microprogramming essentially extends the advantages of programmed logic one step further down the quantum ladder.

The constraint of a predefined instruction set is removed leaving only the basic constraints imposed by the hardware architecture of a particular microprocessor.

The PDCP software development portion of this study (Section 4) provides some insight into the problem of defining an optimal microcomputer instruction set for a PDCP. In Section 4.3, the potential contributions of microprogramming to the development of block operator subroutines for the library of an applications-oriented software development system are outlined. General microprogramming concepts are discussed in References 8, 15, 16, 17, 18, and 19.

REFERENCES

1. Lewis, D. R., "Microprocessor or Random Logic," Electronics Design 18, September 1, 1973.
2. Electronics Staff, "Diverse Industry Users Clamber Aboard the Microprocessor Band Wagon," Electronics, July 11, 1974, pp. 81-108.
3. Altman, L., "Single-Chip Microprocessors Open Up New World of Applications," Electronics, April 18, 1974, pp. 81-87.
4. Smith, H., "Impact of Microcomputers on the Designer," 1973 WESCON Technical Papers, Session 2.
5. Morris, J. H., Patel, H., and Schwartz, M., "Scamp Microprocessor Aims to Replace Mechanical Logic," Electronics, September 18, 1975, pp. 81-87.
6. Chung, D., "Four-Chip Microprocessor Family Reduces System Parts Counts," Electronics, March 6, 1975, pp. 87-92.
7. Weissberger, A. J., "Microprocessors Simplify Industrial Control," Electronic Design 22, October 25, 1975, pp. 96-99.
8. Galey, J. M., "Microprogramming: The Bridge Between Hardware and Software," Computer, August 1975, Vol. 8, No. 8, p. 23.
9. Weiss, C. D., "Software for MOS/LSI Microprocessors," Electronic Design 7, April 1, 1974, pp. 50-57.
10. Intel 8080 Microcomputer User's Manual, July 1975, pp. 4-1 to 4-15.
11. Mano, M. M., Computer Logic Design, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, pp. 326-377.
12. Korn, G. A., Minicomputers for Engineers and Scientists, McGraw-Hill, New York, New York, pp. 44-63.
13. Korn, G. A., Minicomputers for Engineers and Scientists, McGraw-Hill, New York, New York, pp. 206-213.
14. Weiss, C. D., "MOS/LSI Microcomputer Coding," Electronic Design 8, April 12, 1974, pp. 66-71
15. Jones, L. H., "A Survey of Current Work in Microprogramming," Computer, August 1975, Vol. 8, No. 8, pp. 33-37.
16. Broadbent, J. K., "Microprogramming and System Architecture," The Computer Journal, Vol. 17, 1974, pp. 2-8.

17. Cox, G. W. and Schneider, V. B., "On Improving Operating System Efficiency Through Use of a Microprogrammed, Low-Level Environment," Preprints, Micro 7, pp. 297-298.
18. Montangero, C., "An Approach to the Optimal Specification of Read-Only Memories in Microprogrammed Digital Computers," IEEE - Trans. Computers, Vol. C-23, April 1974, pp. 375-387.
19. Vandling, G. C. and Walddecker, D. E., "The Microprogram Control Technique for Digital Logic Design," Computer Design, August 1969, pp. 44-51.

3. PDCP SYSTEM HARDWARE

PDCP system hardware must be designed to satisfy the constraints imposed by the remote data collection application. Typically, design goals include minimizing power consumption, weight, maintenance, and cost while providing accurate and reliable operation. As illustrated in Figure 3(1), DCP hardware can be partitioned into three systems consisting of the sensors, the digital control logic, and the transmitter. This section discusses the influence of microprocessor technology on the design of each system of a PDCP. In addition, standards which could promote lower PDCP costs are recommended, and the development of a standard, modular PDCP design is proposed.

3.1 SENSOR CLASSIFICATION AND INTERFACING

The most variable elements in data collection systems are the sensors. With measurements being made in such diverse fields as agriculture, ecology, hydrology, and search and rescue, many types of sensors must be accommodated by the data collection platform. A signal conditioner is generally placed between the sensor output and the DCP to convert the sensor output to a standard electrical signal which is compatible with the DCP. In previous systems, this signal has been a voltage, a frequency, or a digital logic level depending upon the particular design of a data collection platform telemetry system. For programmable data collection platforms, the ideal interface occurs at digital logic levels. For analog sensors, an analog-to-digital converter is required as a signal conditioner. The sensor interface is then made at the input to the analog-to-digital converter.

3.1.1 Sensor Classification

User requirements have been documented in recent studies [1]. Table 3.1.1(1) gives a list of the disciplines which have a need for data collection by satellite. Table 3.1.1(2) provides a list of parameters for which sensors are required and the discipline in which the

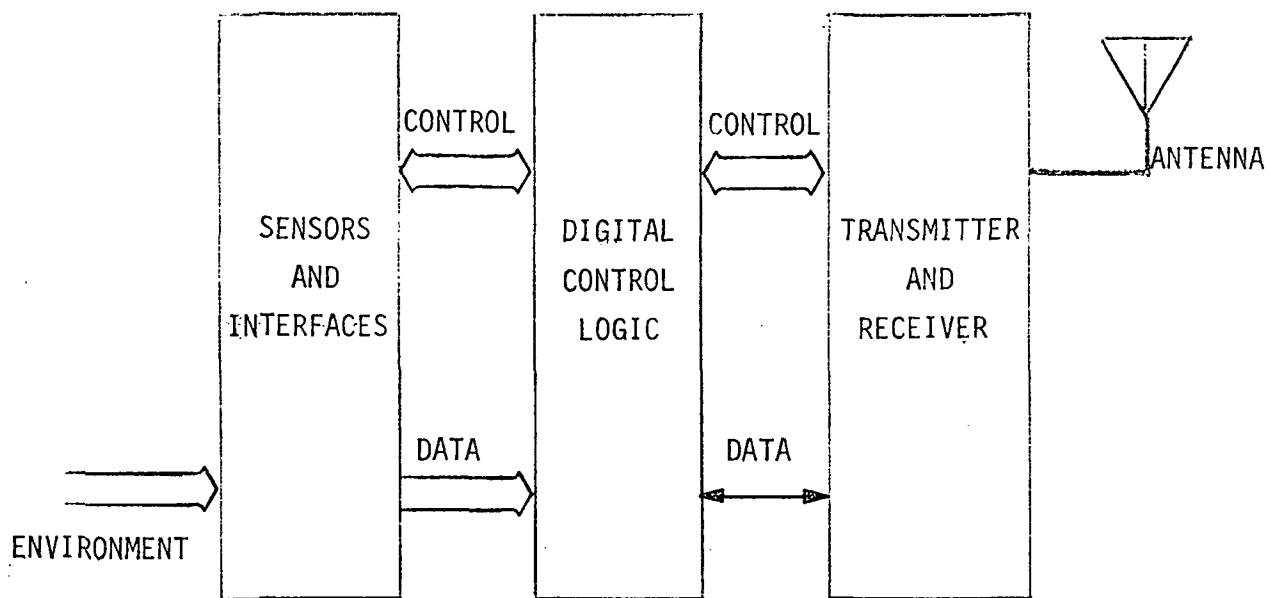


Figure 3(1) DCP Hardware Subsystems.

TABLE 3.1.1(1)
SATELLITE DATA COLLECTION DISCIPLINES

Disciplines	
Agriculture	Meteorology
Biological Behavior	Navigation
Ecology	Oceanography
Geology	Search and Rescue
Hydrology	Transportation

TABLE 3.1.1(2)

SENSOR MEASUREMENT PARAMETERS AND
ASSOCIATED DISCIPLINES

Parameter	Discipline
Acceleration	Biological Behavior
Acoustic Noise	Oceanography
Acoustic Output	Biological Behavior
Activity (Time Up and Down)	Biological Behavior
Aerosols	Meteorology
Air Particles	Biological Behavior Ecology
Air Pollution	Agriculture
Albedo	Geology
Bioluminescence	Oceanography
Blood Pressure	Biological Behavior
Body Position	Biological Behavior
Cargo Security	Transportation
Chlorine	Oceanography
Cloud Cover	Agriculture
Conductivity	Ecology Geology
Creepmeter	Geology
Crop Condition	Agriculture
Dew Point/Front Point	Hydrology
Dissolved Oxygen (Water)	Ecology Hydrology
Diving Depth	Biological Behavior
EKG	Biological Behavior Ecology
Evaporation	Hydrology
Geographical Fix	Agriculture Biological Behavior Ecology Geology Oceanography Search and Rescue Transportation

TABLE 3.1.1(2) (continued)

Parameter	Discipline
Heading (Compass)	Biological Behavior
Heart Rate	Biological Behavior
Humidity (Relative)	Agriculture Ecology Geology Meteorology Oceanography
Ice Presence	Hydrology
Ice Thickness	Hydrology
Light at Surface	Ecology
Light Penetration	Ecology
Magnetic Field	Geology
Oxidents	Meteorology
Oxygen Concentration	Agriculture
Particle Counts (Airborne)	Ecology
pH (Soil)	Ecology
pH (Stomach)	Biological Behavior
pH (Water)	Ecology Hydrology
Phytoplankton Counting	Ecology Oceanography
Pipeline Current	Geology
Pipeline Voltage	Geology
Pitch Angle	Biological Behavior
Pore Pressure	Geology
Precipitation	Agriculture Ecology Hydrology Meteorology
Pressure, Atmospheric	Meteorology Oceanography
River Stage	Ecology
Salinity	Agriculture Ecology Oceanography

TABLE 3.1.1(2) (continued)

Parameter	Discipline
Sediment	Ecology
Seismometer	Geology
Snow Cover	Agriculture Meteorology
Snow Depth	Agriculture Hydrology
Snow Moisture Equivalent	Hydrology
Soil Moisture	Agriculture Ecology Hydrology
Solar Radiation (Surface and Depth)	Ecology
Solar Radiation (Net Allware)	Agriculture Biological Behavior Ecology Hydrology Meteorology
Specific Conductance	Hydrology
Storm Surge	Meteorology
Strain, Biaxial	Geology
Strain Gage	Oceanography
Speed, Swimming	Biological Behavior
Tail Beat (EKL)	Biological Behavior
Telluric Current (On Pipe)	Geology
Telluric Current Frequency	Geology
Temperature, Air	Agriculture Biological Behavior Ecology Geology Meteorology Oceanography
Temperature, Body Surface	Biological Behavior
Temperature, Cargo	Transportation
Temperature, Deep Body	Biological Behavior

TABLE 3.1.1(2) (continued)

Parameter	Discipline
Temperature, Sea Surface	Meteorology
Temperature, Soil	Ecology Meteorology
Temperature, Stomach	Biological Behavior
Temperature, Surface	Biological Behavior
Temperature, Water	Biological Behavior Ecology Hydrology Oceanography
Tide Height	Ecology
Tilt, Biaxial	Geology
Time	Biological Behavior
Time In and Out of Water	Biological Behavior
Total Organic Carbon	Ecology
Tsunami	Geology
Turbidity	Ecology Hydrology Oceanography
Vapor Pressure	Agriculture
Water Current Profile	Oceanography
Water Depth	Biological Behavior
Water Direction	Oceanography
Water Level	Geology Hydrology
Water Table Depth	Hydrology
Water Velocity	Biological Behavior Ecology Hydrology Oceanography
Wave Height	Transportation
Wave Spectra	Oceanography
Wildland Fire	Ecology
Wind Profile	Agriculture

TABLE 3.1.1(2) (continued)

Parameter	Discipline
Wind Speed	Biological Behavior Ecology Geology Hydrology Meteorology Oceanography
Wind Vector (Direction)	Agriculture Hydrology Meteorology Oceanography

sensor could be utilized. Among this list are sensors which have already seen service in satellite data collection systems. A typical group composed of hydrology sensors is listed in Table 3.1.1(3) along with their signal conditioner outputs. While the direct output of each sensor may be a measurement in volts, amperes, or ohms with nonlinear and suppressed zero scales, the signal conditioners have conveniently translated the output ranges to lie between zero and plus five volts.

3.1.2 Sensor Interfacing

A number of sensors already exist which could interface directly with a programmable data collection platform including those in Table 3.1.1(3) with the addition of an analog-to-digital converter. Some sensors include their own signal conditioning equipment which converts the basic sensor output to a digital logic level. For example, the temperature sensor in the TWERLE experiment converts the measured temperature to a frequency output. A counter measures the number of cycles of this frequency over a fixed period; the contents of the counter registers are a digital representation of the temperature value.

Two basic classes of signal conditioners will probably be required in a PDCP system. One class converts the non-standard basic sensor output to a standard output. This will normally be the responsibility of the sensor designer. A second class of signal conditioners converts the standard output to digital logic levels which are compatible with the microcomputer. In many cases, the second class of signal conditioners will be time-shared between a number of sensors and would be provided as a standard module by the PDCP designer. Of course, a direct conversion of the basic sensor output to a logic level is very desirable since this eliminates the need for the second class of signal conditioners; however, the second class of signal conditioners will be required to utilize existing sensors. Fortunately, these signal conditioners convert from a standard input to a standard output; therefore, their cost is low.

TABLE 3.1.1(3)
TYPICAL SATELLITE DATA COLLECTION SENSORS

Sensor	Signal Conditioning Output Specification
Dissolved Oxygen	0.0 to +5.0 Volts dc
pH	0.0 to +5.0 Volts dc
Specific Conductance	0.0 to +5.0 Volts dc
Temperature	0.0 to +5.0 Volts dc

3.1.2.1. Sensor Interface - Little can be said about the basic sensor output since this can be almost any variable. The sensor signal conditioner takes this output and converts it to a standard output. Traditionally, for analog sensors this output has been zero to plus five volts dc. Another standard range for low-voltage or low-current devices has been zero to 50 millivolts dc. If at this point in time a recommendation were to be made for a standard output range for analog sensors, it would have to be zero to plus five volts. This, however, may present a problem if some of the newer technologies are used in fabricating the microprocessor. The integrated injection logic (I^2L) technology (see Section 6.1) can operate at voltages lower than one volt, and operating the signal conditioner from the same power supply voltage would be advantageous. This would require a low-voltage output from the signal conditioner.

As previously mentioned, some sensors produce a digital output directly. The analog-to-digital conversion process may be built into the sensor. The output of such a sensor should have an interface compatible with CMOS logic. In contrast to the analog output of zero to plus five volts dc, the digital sensor output should be as follows: a zero would be represented by a level between zero and 0.8 volts, and a logical one would be represented by a level of 3.5 to 5 volts dc. Until more experience is gained with I^2L devices, the above specification for the output of the digital sensor would be a reasonable recommendation.

3.1.2.2. Microprocessor Interface - The second level of signal conditioning converts the standard interface signals to signals which match the logic characteristics of the microprocessor. The most common signal conditioner of this type is the analog-to-digital converter (ADC). This device takes a zero to plus five-volt dc analog signal and converts it to an equivalent digital word. For the PDCP application, the ADC output should be coded in natural binary at CMOS compatible logic levels. A parallel output word will generally be required for direct interface to the microprocessor data bus. However, in some applications where the ADC and sensor must be located in an area remote from the PDCP a clocked serial output is desirable to minimize the number of wires required for the data link. Both the parallel and serial outputs should be three-state outputs to permit the signal lines to be shared with other devices.

In a three-state device two states are the usual high or low output (input) states. A control signal converts this output (input) to a high impedance state effectively removing the device from the bus. Suitable low-power CMOS-logic ADC's which would interface directly to the bus of a CMOS microprocessor-based PDCP are commercially available.

Similar to the analog-to-digital converter is the frequency-to-digital converter which is needed by sensors having an output frequency proportional to the measured variable. This conversion of frequency to a digital signal can be performed by the microprocessor with no external hardware. Using the Intel 8080A-1 microprocessor, sensor frequencies up to approximately 20 kilohertz can be digitized. The frequency-to-digital conversion capability of the microprocessor provides a low cost interface to a variety of sensors since most sensor outputs can conveniently be produced as a frequency.

A big advantage in interfacing the microprocessor with a set of sensors is that sensors with different resolution requirements can easily be accommodated. For example, one sensor may require a resolution of eight bits and another require a resolution of sixteen bits from a time-shared ADC. For the eight-bit resolution sensor only the eight highest-order bits need to be multiplexed from the ADC while the whole sixteen bits of the high resolution sensor would be transferred to the microprocessor. This is easily accomplished through program control.

3.2 DIGITAL CONTROL LOGIC AND MEMORY

In current DCP designs, the digital control system is generally a hardwired logic circuit manufactured with standard CMOS integrated circuits. Because of the cost and complexity of the random logic design, control functions are limited to elementary sensor control, data formatting, and transmitter control. A microprocessor-based PDCP will be capable of performing additional tasks such as data compaction and data preprocessing without excessive cost increases (see Section 4).

A block diagram of a microprocessor-based PDCP control system is illustrated in Figure 3.2(1). The number of integrated circuits required to implement this system will depend upon input/output require-

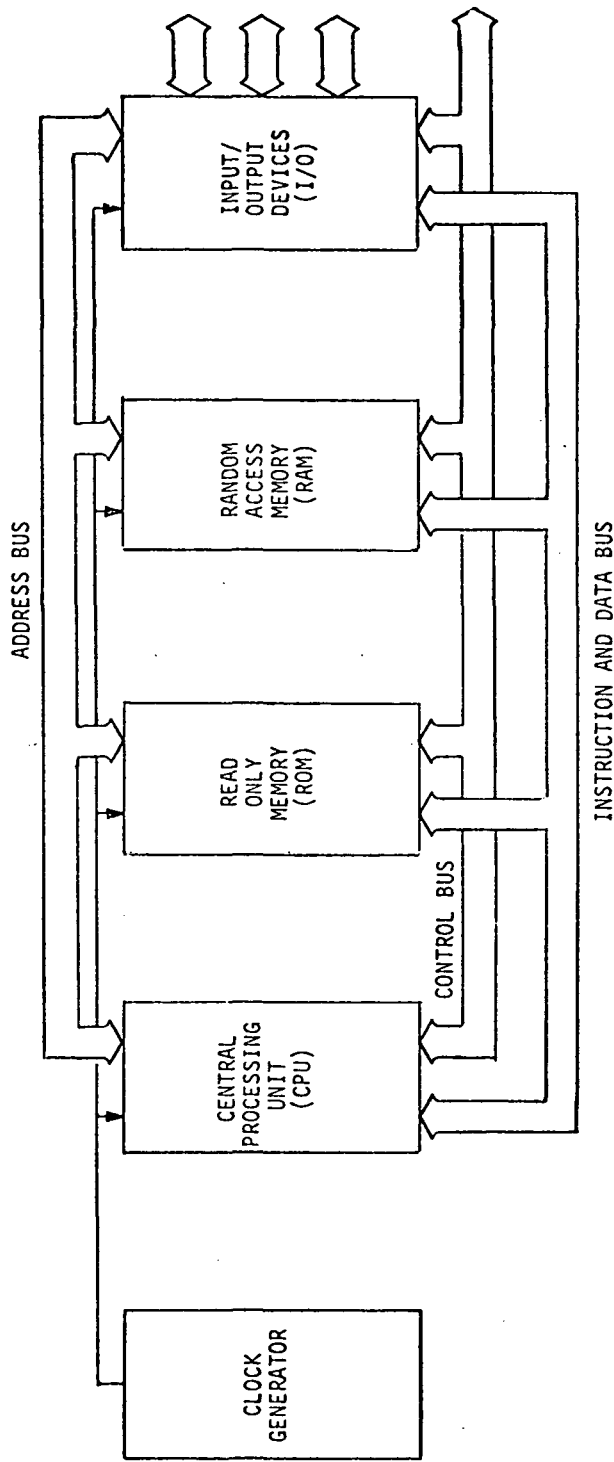


Figure 3.2(1) Microprocessor Based PDCP Control Subsystem Organization.

ments, memory requirements, and the microprocessor architecture. Already, two-chip microprocessor sets containing a clock circuit, 64 bytes of RAM, 1024 bytes of ROM, and several input/output ports are available. Thus, a minimal PDCP control system may require as few as two integrated circuits.

Future DCP user requirements are expected to range from basic platforms capable of performing the tasks of current DCP's through advanced platforms capable of extensive data preprocessing and data compaction. From the standpoint of economics, a standard, universal DCP design is desirable. This goal is impractical for a random logic design but is practical for a microprocessor-based PDCP design. As illustrated in Figure 3.2(1), the microprocessor-based PDCP control system organization is based upon address, control, and data buses. This organization produces a system which is modularly expandable. A basic PDCP capable of replacing current DCP designs will utilize all of the functions shown in Figure 3.2(1). An advanced PDCP with extensive data compaction capabilities will utilize the identical functions. The distinction between these systems will be the amount of ROM, RAM, and I/O provided and their software organizations. Section 4 demonstrates that a unique PDCP software organization can be defined for each user by plugging a different ROM into the system. Thus, a PDCP design which will function efficiently as a basic DCP but which can easily be expanded to form an advanced processing PDCP can be developed.

Physically, the standard PDCP design could be implemented as a modular set of printed circuit (PC) boards. A basic PDCP system would probably require one or two small PC boards. The system could be expanded by adding extra integrated circuits in spaces provided on the basic PC card set. Large scale expansion would involve adding standard memory or I/O cards and, possibly, bus driver circuits. This implementation of the standard PDCP design would provide most potential users with the capability they require at a minimum cost since the design and development expenses could be prorated. If a potential user did require a unique PDCP construction, the electrical design would not necessarily have to be changed. Additional PC board development cost would be incurred, but additional circuit design cost would not be incurred.

3.3 TRANSMITTER, RECEIVER, AND ANTENNA

Section 4.4.3 demonstrates that a microprocessor-based PDCP can easily control transmitter modules which have been developed for current DCP designs. Transmitter control inputs should be digital signals compatible with the technology chosen for the PDCP control system. Otherwise, no constraints are imposed upon the electrical design of transmitters and antennas for PDCP's. Traditional design techniques are satisfactory.

Since the sensor interface and digital control systems of a PDCP will be developed as modular PC board sets, the transmitter system could be similarly designed. One approach is to provide individual transmitter modules designed specifically for each data transmission technique. The PDCP user could then customize the modular system by selecting a transmitter module (Landsat, GOES, TWERLE, or TIROS-N), a basic or expanded control module, and various sensor interface modules. For data collection platforms installed near land lines, a telephone modem module could be chosen instead of a specific transmitter module.

A better approach would be to utilize the flexible nature of the microprocessor to control a single modulator which can emulate the modulation of all of the data collection systems. A standard PDCP system will be capable of operating with all current data transmission formats under software control by using the techniques demonstrated in Section 4.4.3. As a result, a universal transmitter design that can switch between the various data transmission modes under software control could be designed. The required carrier frequencies might be developed from a standard frequency reference by a phase-locked loop containing a programmable frequency divider. This system would eliminate the requirement for multiple transmitter modules and would be more adaptable to any future changes in data transmission techniques. A further study will be required to determine the potential cost effectiveness and performance of the two transmitter construction techniques outlined above.

Some DCP's have been developed with receivers for remote command applications. A PDCP with this capability can be produced by adding a receiver module to the system. The receiver should be interfaced to the PDCP buses at standard logic levels. PDCP software can be developed to poll the receiver and decode the command signals as they are received. Alternately, the receiver can provide a signal to the interrupt line of the microprocessor to indicate that a command signal is being received. The microprocessor would then suspend data processing operations temporarily to input and decode the control message.

3.4 SUMMARY

The problem of standardization of sensor output and microprocessor input is a difficult one to solve; however, attempts should be made towards standardization to reduce the types of signal conditioners required.

It is recommended that new designs for analog sensors have outputs which are 0.0 to +5.0 volts dc. The recommendation for digital interface for CMOS logic is zero to 0.8 volts for a logical zero and 3.5 to 5 volts dc for a logical one.

REFERENCES

1. Wolfe, E. A., Cote, C. E., and Painter, J. E., Satellite Data Collection User Requirements Workshop, NASA - Goddard Space Flight Center, Greenbelt, Maryland, May 21, 1975.

4. PDCP SYSTEM SOFTWARE

As demonstrated in Section 2, significant advantages can often be obtained by replacing a hardwired circuit with a software-controlled microprocessor system. The potential advantages of a programmed logic system arise from substitution of relatively inexpensive software storage for complex hardwired circuits. Therefore, the flexibility and cost effectiveness of a PDCP will be highly dependent upon the system software and the extent to which PDCP functions can be software controlled. Unfortunately, system constraints can preclude the use of programmed logic, and, as shown in Section 2, the cost of developing special purpose software for low quantity applications can be excessive. This section will demonstrate that microprocessors are capable of performing PDCP tasks and describe a potential PDCP software package organization designed to minimize software development cost. Included also are discussions of the effects of various PDCP system constraints on the software package and descriptions of subroutines which have been developed for typical PDCP applications.

4.1 PDCP SOFTWARE ORGANIZATION

Standardization is the key to minimum PDCP system cost. However, a PDCP design must be sufficiently flexible to operate efficiently with a number of different sensors, data compaction techniques, and data transmission formats. In addition, a software system organization which provides a simple technique by which individual experimenters and organizations can create special purpose PDCP programs is desirable. The PDCP software system organization outlined in Figure 4.1(1) is designed to meet these goals.

A PDCP software package based on the organization proposed in Figure 4.1(1) would contain an executive program and a number of called subroutines. The executive program controls the operation of the PDCP by executing a set of conditional and unconditional calls to various subroutines. A different executive routine will generally be required for

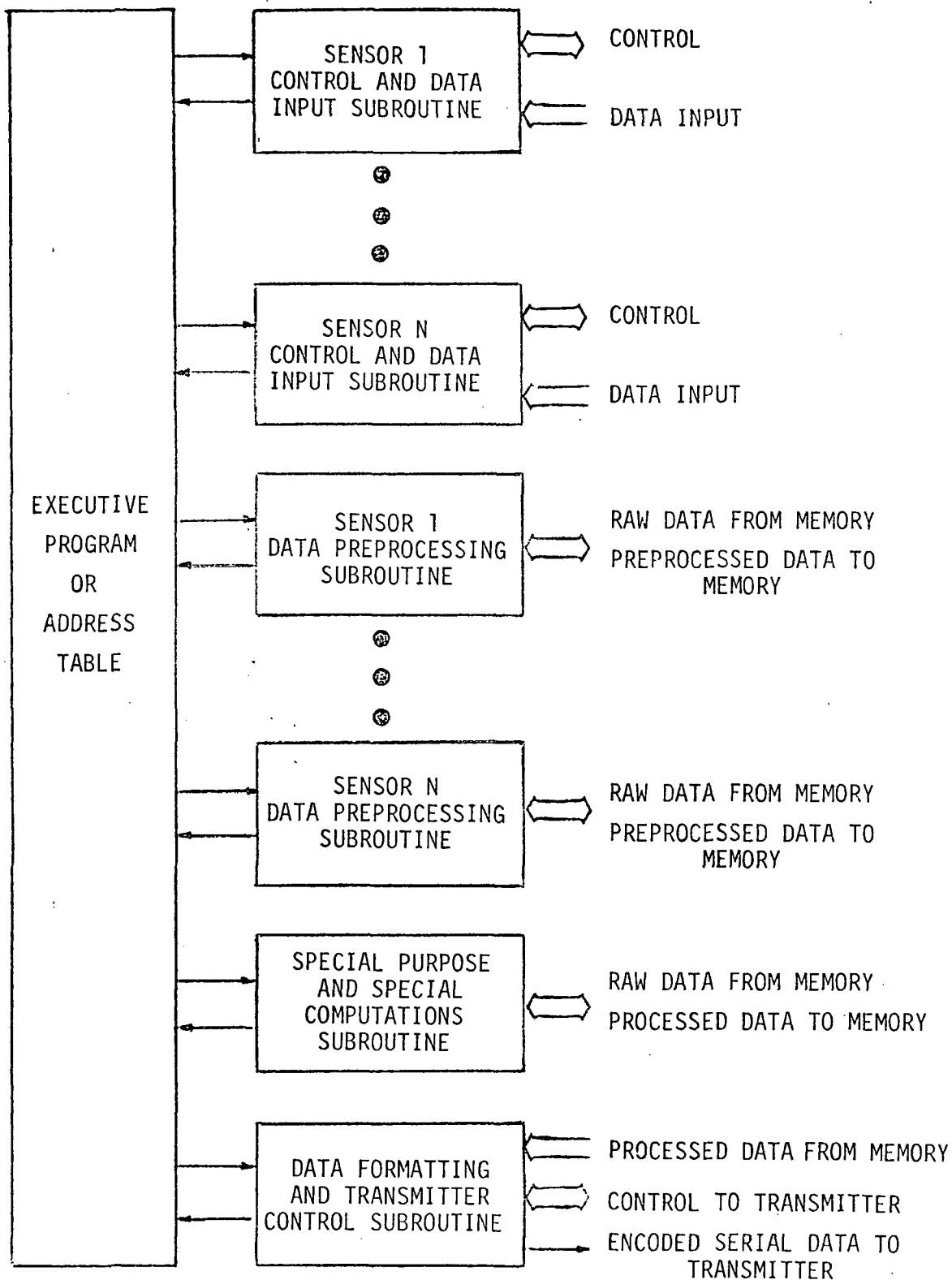


Figure 4.1(1) PDCP Software System Organization.

each different application. Major subroutines called by the executive program control the sampling of a raw sensor data, preprocessing of raw data, special computations, and data transmission to the satellite. The executive program and major subroutines share multipurpose minor subroutines such as software delay loops. Standard sensor data input, general purpose data preprocessing, and transmitter control routines can be provided to users as firmware ROM options for the PDCP. Using this approach, most PDCP software development costs can be prorated over a large number of PDCP's so that the effective software cost will be minimal. The additional program development required for new PDCP software systems could generally be restricted to an executive routine and, when necessary, special purpose subroutines. Since complex control and data processing tasks would generally be controlled by subroutines, the cost of developing executive programs should be low. Special subroutine development costs can be minimized by maintaining a user program library.

4.2 PDCP SYSTEM TIMING

The program organization proposed above is independent of the various system constraints imposed upon PDCP software and the actual techniques used for program development. The major PDCP system constraints which will affect software development is timing. In many cases, the usual goal of minimum execution time will apply to data preprocessing and special purpose subroutines. In general, however, sensor control and transmitter subroutines will require precise timing between specified events, and the total elapsed time between data transmissions should be precisely controlled. These constraints must be considered during PDCP software system development.

Software and interrupt timing are the two basic techniques which can be employed to insure correct PDCP transmission intervals. The microprocessor and associated support logic must operate continuously if software timing is employed. For low power CMOS or I²L devices, this is not necessarily a disadvantage since maximum throughput can be obtained with a continuously operating system. In addition, however, the precise execution time for each PDCP software routine must be known, and provision must be made for holding the total program execution time between

transmissions constant. Most programs contain various mutually exclusive branches which may require different execution times. As a result, delay must be inserted within certain program branches to equalize execution times. Alternately, program execution time can be allowed to vary if each time the subroutine is called a flag is set to indicate the actual execution time. The executive program may then use any excess time to call optional subroutines (for example, system maintenance or low priority data input routines) and a variable delay subroutine. Although average throughput can be increased using this technique, worst-case throughput is not necessarily improved. The major advantage of software timing is that neither an interval timer nor interrupt logic is required.

Interrupt timing can be relatively independent of system software but requires the use of an external interval timer and interrupt logic. The microprocessor can be powered-down between transmissions or operated continuously. Worst-case execution time for each subroutine must be known in order to guarantee that the microprocessor will either complete data input or processing before a new transmission is required or will be executing an interruptable routine. In the event that the transmission request interrupt is allowed to occur before data processing has been completed, the programmer will have the choice of discarding unprocessed data or completing the processing at the conclusion of the transmission sequence.

Assembly language programming can be used to develop subroutines utilizing either interrupt or software timing. Programs written in assembly language generally provide maximum performance while requiring minimal storage area. Efficient assembly language programming, however, requires a thorough knowledge of the available instruction set and entails the highest development cost per program. Subroutines which require precise internal timing, and general purpose subroutines which will be included in firmware ROM packages provided to PDCP users should be written in assembly language. This will insure precise timing and minimal program storage area. Unfortunately, assembly language programming cost for small-quantity, special-purpose systems could be excessive and should be avoided when possible.

The development of many types of numeric data processing subroutines could be simplified by using a high-level language such as PL-1, FORTRAN, COBAL, or BASIC. Each of these languages is represented by a compiler written for at least one microprocessor. The major disadvantages of these languages are inadequate input/output structures, inefficient memory utilization, and poorly defined timing. Subroutines written using these high-level languages cannot be used in a PDCP software system which relies upon software timing unless the resulting machine code is investigated to determine all possible execution times. Even for an interrupt-timed system, the maximum execution time of the subroutines must be determined to insure that processing can be completed between transmissions.

4.3 PDCP SOFTWARE DEVELOPMENT

As presented above, neither assembly languages nor standard high-level languages are ideally suited for the development of PDCP software systems. This problem is not unique to the data collection field but is of general concern in the areas of mini- and microcomputer programming. Therefore, a new program development technique which retains the advantages of both high-level and assembly languages should be developed.

4.3.1 Applications-Oriented Software Development

A simplified technique for microcomputer programming by applications-oriented nonprogrammers has recently been proposed by Korn [1]. This technique is based on a software system organization nearly identical to the one illustrated in Figure 4.1(1). Korn [1] assumes that a set of well-defined assembly language, block-operator subroutines are available to perform each of the subfunctions which may be required by an applications program. The applications program is specified by means of a block-diagram employing the standard functions. An interactive editor/translator program running on a small minicomputer is used to translate the block-diagram specifications into an address table. The address table serves the same function as the executive program proposed in Figure 4.1(1). During program execution, the address table specifies successive jumps to the standard subroutines required to correctly execute

the applications program. A choice between the address table and executive program techniques will depend upon the extent to which indirect addressing and subroutine calls are supported by a particular microprocessor's instruction set.

Korn [1] states that the editor/translator programming system should satisfy the following requirements:

1. The language used to communicate with the editor/translator program should be easy to learn and understand.
2. Simple elementary operations well known to engineers and scientists should be used to build up complex operations.
3. The language should be entirely independent of the specific microcomputer used.
4. The editor/translator system should generate microprocessor code as efficient in the use of time and memory as that developed by a good assembly language programmer.

These goals are applicable to the proposed PDCP program development system but are not sufficient. A PDCP program development system should also satisfy the following requirements:

1. In addition to the elementary operations, as many major PDCP system programs as possible should be available to the user in block form. For example, this would include transmitter programs (see Section 3.3) and the zero-order floating aperture predictor subroutine (see Section 3.2).
2. Memory requirements should be indicated to the user in a form that is representative of memory cost. This implies that memory required for storage of the executive program (usually, user defined ROM or PROM) should be listed separately from data storage or scratch pad RAM memory. Also, the number of firmware ROM modules required for storage of the standard subroutines should be indicated.

3. Program execution time should be indicated to the user, and an error should be noted if the program execution time exceeds a specified transmission interval.
4. Alternate subroutine structures should be available for software and interrupt-timed systems. This is necessary to obtain maximum system throughput with minimum program storage area.

A PDCP software development system with the capabilities described above could significantly enhance usage of PDCP's by applications-oriented nonprogrammers.

4.3.2 Assembly Language and Microprogrammed Subroutines

To insure efficiency and accurate timing, the functional block subroutines employed in the programming system described above should be developed by experienced programmers using assembly language and/or microprogramming techniques. Within the constraints of a given instruction set, assembly language programming can produce the optimum realization of a particular software task. Many microprocessors have general purpose instruction sets which are fixed by hardwired control logic integrated within the microprocessor chip itself. These instruction sets must be individually evaluated to determine their relative merits in particular applications such as PDCP programming (see Section 6). In contrast, the instruction set of a microprogrammable microprocessor can be optimized for a particular application. A microprogrammed microprocessor utilizes a programmed logic control unit rather than a hardwired control unit. This permits the user to define and sequence microprocessor operations at the most elementary level. As a result, the power of a microprogrammed instruction set is limited only by the amount of control storage available and the capabilities of the microprocessor's arithmetic/logic unit. In fact, when sufficient control storage is available, microprogramming need not be limited to the development of a simple instruction set. Those operations which are used most frequently, or require minimum execution time, can be implemented as microprogrammed subroutines. Regardless of whether or not the microprocessor chosen for the PDCP has a microprogrammable instruction set, PDCP operations which are not included in the

machine's basic instruction set can be implemented using assembly language programming. Examples of assembly language subroutines which could be included in the library of the editor/translator program are presented in the following section.

4.4 PDCP PROGRAM EXAMPLES

The hardwired logic control units used in most current DCP designs perform four basic tasks: data input, elementary sensor control, data formatting, and transmitter control. In this section, example programs are presented to demonstrate the ability of a microprocessor based control unit to perform each of these tasks. These example programs show that, even when a microprocessor based control unit is used to perform all basic DCP tasks, excess computational capability exists. This capability can be used for data preprocessing or special computations which would reduce the load on the satellite data link. Fourier analysis of seismic data is presented as a potential special computation. Examples of data preprocessing subroutines are also shown.

Unless otherwise indicated, the programs listed in the following sections have been written to be executed on the 8080A-based PDCP demonstration system (see Section 5). Standard Intel Corporation 8080A mnemonics [2] have been used except within macroinstructions. Instructions such as NOP and MOV A,A are sometimes used to produce a required delay without otherwise affecting program execution. Macro delay instructions have been defined so that this usage of an instruction will be clear within the context of the program. The mnemonic DLY XX is used for these macroinstruction groups. The decimal number which replaces XX indicates the length of the delay in states. Since the assembly language program listings were produced using a PDP 11/40 minicomputer, constants and addresses are represented as octal numbers except as noted by the use of a decimal point.

4.4.1 Data Input and Sensor Control

Data input and elementary sensor control are two of the four basic tasks performed by the hardwired control logic currently used in most

DCP designs. A microprocessor based control unit is ideally suited to performing these tasks. Each PDCP sensor can be serviced by a simple subroutine which provides the functions of data input and sensor control. Because of the versatility of the microprocessor, the cost and complexity of the data acquisition channels can potentially be reduced without a significant increase in control logic. Subroutines which demonstrate microprocessor control of the data input and sensor control functions are listed below.

4.4.1.1 Frequency Measurement - Many sensors currently in use on DCP's produce an output signal whose frequency is proportional to the parameter of interest (temperature or air pressure for example). Program 4.4.1.1(1) has been developed to demonstrate that a PDCP can digitize the output of these sensors directly. A flow chart for Program 4.4.1.1(1) is illustrated in Figure 4.4.1.1(1). This frequency counter program is presented in the form of a subroutine which is suitable for use as an input block operation in the programming system described in Section 4.3.1. The sensor output signal is input to the microprocessor on one bit of input port zero. Signal frequency is determined by counting 0 to 1 logic level transitions for a preset interval of time. Two parameters, the number of samples to be taken and the input bit assigned to the sensor, are passed to the subroutine from the calling program. The input bit assignment for the sensor is specified by a mask byte in register C. This byte contains a one in the bit position corresponding to the bit position of the input signal and zero's in all other bits. Thus, a single frequency counter subroutine can service up to eight sensors without multiplexing. The number of samples to be taken is specified by the contents of register pair DE. Since the number of samples is proportional to the measurement interval, the resolution and scale factor of the digitization process are determined by the number of samples taken. With a measurement interval of one second, the sensor frequency is measured directly in units of Hertz (Hz). The binary output is returned to the calling program in register pair DE.

The sampling interval establishes an upper bound on the frequency which can be measured using this subroutine. A minimum sampling interval

```

; THIS SUBROUTINE IS USED TO CONVERT A FREQUENCY SENSOR OUTPUT TO
; PARALLEL DATA. THE NUMBER OF SAMPLES TO BE TAKEN IS PRELOADED
; BY THE EXECUTIVE IN THE (DE) REGISTER PAIR. SINCE A FIXED
; AMOUNT OF TIME IS REQUIRED TO TAKE A SAMPLE, THE NUMBER OF
; SAMPLES TAKEN IS PROPORTIONAL TO THE SAMPLING PERIOD. 77
; MACHINE STATES ARE REQUIRED TO MAKE ONE SAMPLE. THE VALUE
; OF THE COUNT IS RETURNED IN THE (DE) REGISTER PAIR. UP TO
; EIGHT DIFFERENT INPUT CHANNELS CAN BE SAMPLED, ONE AT A
; TIME. THE EXECUTIVE MUST PLACE THE BIT MASK IN THE A REG-
; PRIORITY CALLING THE COUNT ROUTINE. THE MASK DETERMINES
; WHICH BIT OF INPUT PORT 0 IS TO BE SAMPLED. CPU STATUS OF
; B,C,H,L REGISTERS PRESERVED.
;
; 36 * 77 * #SAMPLES TAKEN MACHINE STATES, 36 BYTES ROM AND 6 BYTES
; STACK.

```

```

000000          COUNT:      PUSH  B      ; SAVE CPU STATUS OF BC AND HL.
000000          305        PUSH  H      ; REGISTER PAIR
000000          345        LXI  H,0      ; ZERO COUNT REGISTERS
000000          041        MVI  C,1      ; 1 ----> C
000000          016        MOV  B,A      ; PUT BIT MASK PASSED FROM EXECUTIVE INTO B
000000          107        IN   0        ; INPUT SAMPLE OF SIGNAL
000000          333        ANA  B        ; PICK OFF APPROPRIATE BIT BY MASKING WITH B
000000          240        JZ   C2       ; NOT POSITIVE EDGE, DON'T INCREMENT COUNT
000000          312        CMP  C        ; COMPARE TO LAST SAMPLE
000000          271        JZ   C3       ; NOT A TRANSITION SO DON'T INCREMENT COUNT
000000          312        INX  H        ; IT'S A TRANSITION AND A POS. EDGE, SO INCREMENT COUNT
000000          043        DLY5        ; DELAY FOR EQUAL TIME BRANCHES
000000          177

```

Program 4.4.1.1(1) Frequency Sensor Input Subroutine.

```

000024      C1:
000024      MOV     C,A      ; STORE SAMPLE IN C
000024      DCX     D          ; DECREMENT TIME COUNT
000025      MOV     A,E      ; CHECK FOR ZERO IN DE
000026      DRA     D
000027      JNZ     CO      ; NOT ZERO TIME OUT SO SAMPLE AGAIN
000028      XCHG
000029      POP     H          ; PUT COUNT FROM HL INTO DE
000030      POP     B          ; RESTORE CPU STATUS
000031      RET
000032      DLY14      ; ADD DELAY SO THREE PATHS ARE
000033      JMP     C1      ; EQUAL TIME WISE
000034      END
000035
000036      C2:
000037      DLY14      ; ADD DELAY SO THREE PATHS ARE
000038      JMP     C1      ; EQUAL TIME WISE
000039      END
000040
000041      C3:
000042      DLY14      ; ADD DELAY SO THREE PATHS ARE
000043      JMP     C1      ; EQUAL TIME WISE
000044      END
000045

```

Program 4.4.1.1(1) (continued)

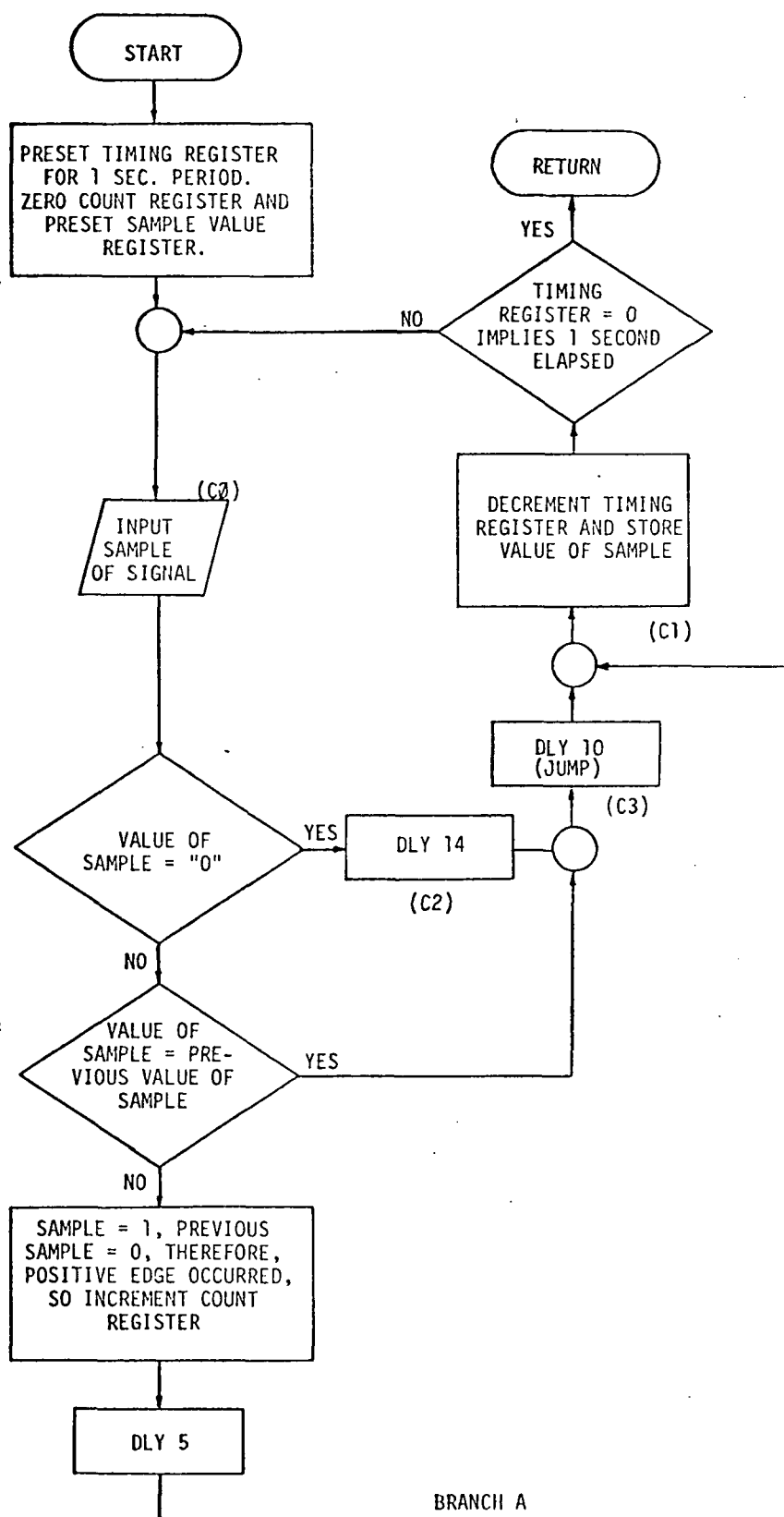


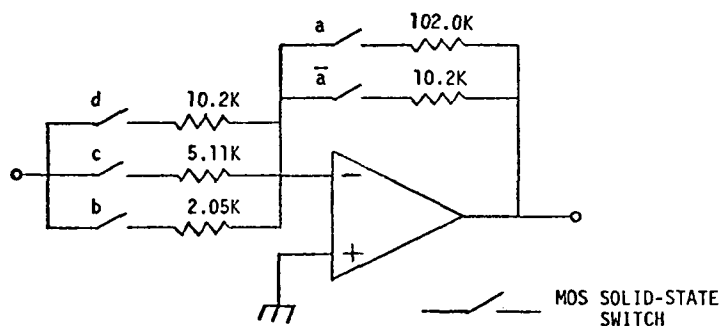
Figure 4.4.1.1(1) Flow Chart for Frequency Sensor Input Subroutine.

is desirable, but the interval must be constant. As shown in Figure 4.4.1.1(1), the program executes one of three branches depending upon the value of the current and the preceding sample. The longest natural program branch (Branch A) occurs when a positive signal transition is detected. Delay must be added to the other two program loops so that all three loops will execute in the same number of states. Unfortunately, arbitrary delays are not possible because the microprocessor is synchronized to a constant frequency clock. As a result, a minimum delay of five states must be added to Branch A so that the execution time of all three branches can be equalized.

Program 4.4.1.1(1) requires 77 states between samples of the input signal. Each state is a minimum of 500 nsec for an 8080A microprocessor or 325 nsec for a high speed 8080A-1 microprocessor. Thus, the maximum sampling frequencies are 25,974 Hz and 39,960 Hz, respectively. From the sampling theorem [3], these sampling frequencies establish upper bounds on the input signal frequencies of 12,987 Hz and 19,980 Hz, respectively. Frequency prescaling will be required for sensors which produce higher output frequencies.

4.4.1.2 Autorangeing - In some PDCP applications, the parameter of interest is a relatively slowly changing function of time which can vary over a wide range. The design of an accurate data acquisition channel for parameters of this type is difficult because linearity must be maintained over a wide dynamic range. If the data acquisition channel gain is set to produce a detectable signal level for the minimum sensor output, the later stages of the data acquisition channel may saturate as the sensor output increases. A solution to this problem is to vary the gain of the data acquisition channel so that the signal is always within the most linear and accurate system dynamic range. This can be accomplished by including a logarithmic amplifier in the front end of the data channel. The logarithmic amplifier, however, introduces a log conformity error which is unsatisfactory for systems requiring high resolution throughout the voltage range. In addition, the logarithmic scaling of the digitized data is inconvenient for most PDCP data preprocessing subroutines. These problems can be avoided by using a variable gain amplifier in place of the logarithmic amplifier.

The concept of a microprocessor-controlled variable gain amplifier is illustrated in Figure 4.4.1.2(1). A block-diagram of a typical data acquisition system using the variable gain amplifier is shown in Figure 4.4.1.2(2). Channel 1 is designed to acquire data from a sensor which produces a slowly varying, wide-range (200:1) output voltage. This signal is digitized by the analog-to-digital converter and input to the PDCP's microprocessor. Program 4.4.1.2(1) controls data acquisition on Channel 1. A flow chart for Program 4.4.1.2(1) is shown in Figure 4.4.1.2(3). Each time the executive PDCP program calls the data acquisition subroutine for Channel 1, a sample is obtained. If the sample is between 10 percent and 90 percent of full scale, the value of the sample and a code representing the current channel gain are stored in memory for later transmission. Whenever the sample is not within the ideal range of 10 percent - 90 percent of full scale, the gain of the amplifier is adjusted and another sample is obtained. Thus, a maximum system accuracy and an eight-bit resolution are maintained over a 200:1 dynamic range.



NOMINAL GAIN	CONTROL SIGNALS abcd
-1	0001
-2	0010
-3	0011
-6	0100
-7	0110
-10	1001
-20	1010
-30	1011
-50	1100
-70	1110

Figure 4.4.1.2(1) Microprocessor-Controlled Variable-Gain Amplifier.

```

; AMPLIFIER VARIABLE GAIN CONTROL ROUTINE
;
; THIS ROUTINE IS USED TO VARY THE GAIN OF A VARIABLE
; GAIN AMPLIFIER. THE AMPLIFIER GAIN IS VARIED BETWEEN
; -1.0 AND +70.0 TO KEEP THE ADC INPUT VOLTAGE BETWEEN
; 10% AND 90% OF FULL SCALE (5.0 VOLTS)
;
VOC:      ; START OF VAR GAIN CONTROL ROUTINE
          PUSH   PSM
          PUSH   B
          PUSH   D
          PUSH   H
          LHL   2000
          ; PLACE ADDRESS OF CONTROL WORD
          ; INTO REGISTERS H AND L
          MOV   A,M
          OUT   3
          ; LATCH CONTROL WORD INTO OUTPUT PORT 3
          STA  2002
          ; STORE CODE FOR GAIN SETTING IN MEMORY
          LXI  B,15346
          ; LOAD BYTE REPRESENTING
          ; 0.51 V INTO REGISTER B AND BYTE
          ; REPRESENTING 4.49 V INTO REGISTER C
          IN   3
          CMP   C
          ; CHECK FOR OVERRANGE
          JNC  DEC
          ; TRY TO DECREASE GAIN IF OVERRANGE
          CMP   B
          ; CHECK FOR UNDERRANGE
          JC   INC
          ; TRY TO INCREASE GAIN IF UNDERRANGE
          STA  2003
          ; STORE CORRECT ADC DATA IN MEMORY
          ; FOR LATER PROCESSING BY MAIN PROGRAM
          ; OR TO AWAIT TRANSMISSION
          SHLD 2000
          ; STORE LAST CONTROL WORD ADDRESS
          POP   H
          ; RESTORE WORKING REGISTERS AND
          POP   D
          ; RETURN TO MAIN PROGRAM
          POP   B
          POP   PSM
          RET

```

000044							
000045			D, 31				
000046	031		D				; CHECK FOR GAIN=-1
000047	272		STORE				; STORE PREVIOUSLY SAMPLED DATA IF TRUE
000050	312	000	L				; IF NOT, DECREASE GAIN AND SAMPLE
000053	055		SAMPLE				; AGAIN
000054	303	000	D, 351				
000057	026		D				; CHECK FOR GAIN=-70
000061	272		STORE				; STORE PREVIOUSLY SAMPLED DATA IF TRUE
000062	312	000	L				; IF NOT, INCREASE GAIN AND SAMPLE
000065	054		SAMPLE				; AGAIN
000066	303	007					
000066		000					

Program 4.4.1.2(1) (continued)

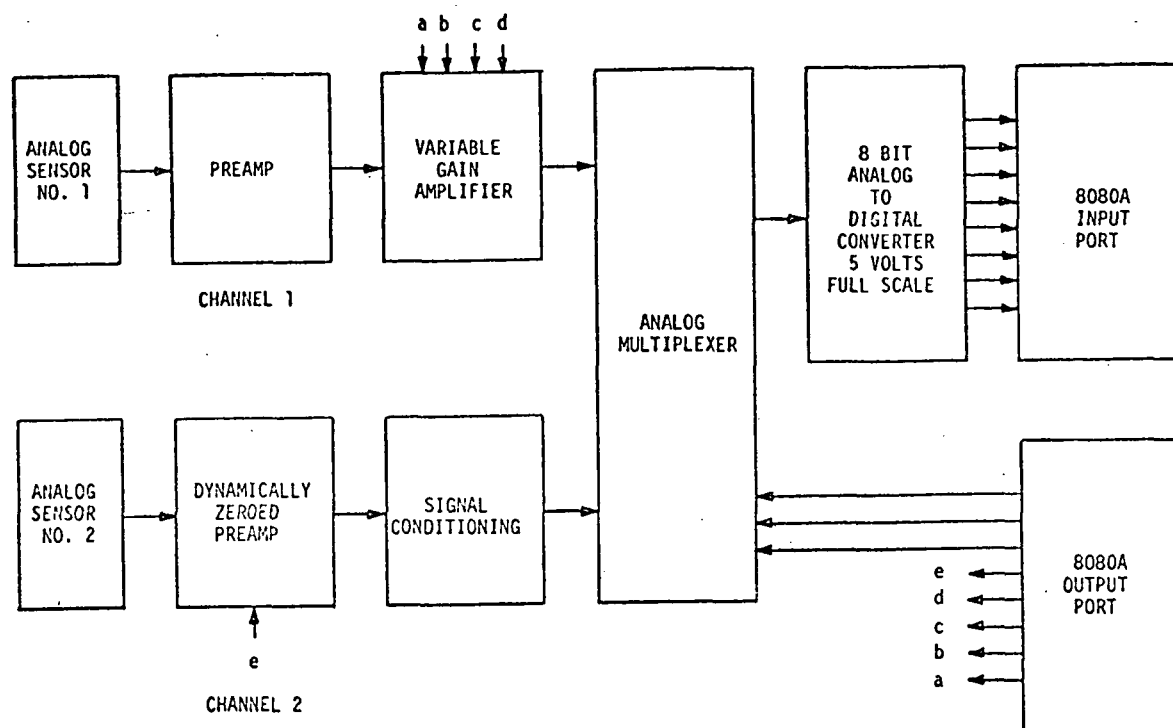


Figure 4.4.1.2(2) Analog Data Acquisition System for a PDCP.

4.4.1.3 Software Controlled Analog-to-Digital Conversion - The traditional analog-to-digital converter (ADC) consists of a digital-to-analog converter, a comparator, and some control logic. A digital word loaded into the digital-to-analog converter (DAC) produces an analog output signal (voltage or current) which is applied to one input of the comparator. The second comparator input is the analog signal which is to be digitized. The comparator output signal indicates whether the analog input is greater than or less than the reference signal produced by the DAC. This signal is fed back to the control logic which uses a fixed algorithm to update the DAC input. The above process continues until the DAC output equals the analog signal input. At that point, the digital word in the DAC is the digital representation of the analog input voltage.

A number of different algorithms can be employed in the analog-to-digital conversion process but one of the most widely used is successive approximation. In a successive approximation ADC, the unknown analog

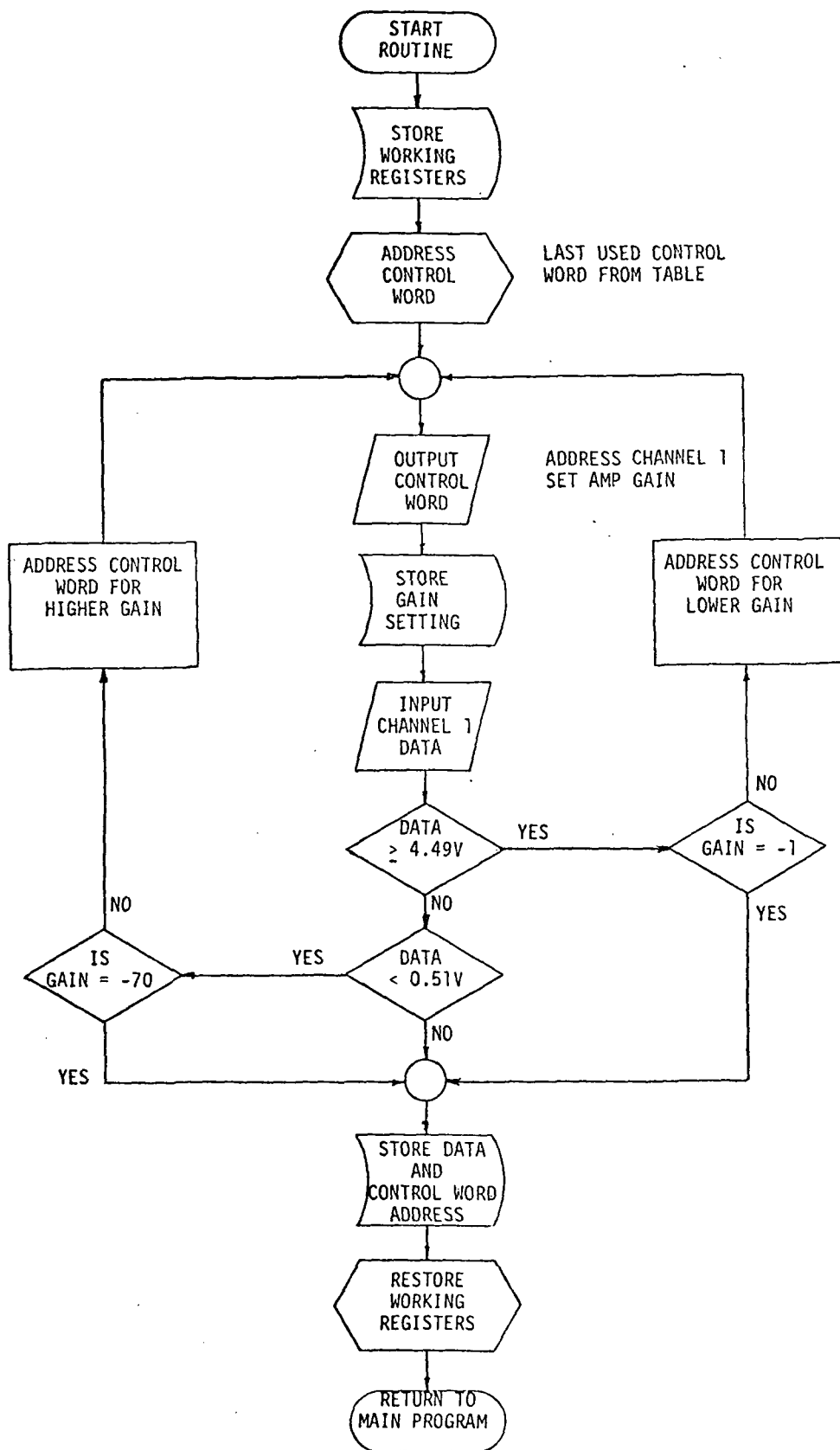


Figure 4.4.1.2(3) Flow Chart for Variable Gain Amplifier Control Program.

input is compared to the DAC output with only the most significant bit (MSB) true. This DAC output corresponds to one-half of full scale. If the input is greater than the MSB, the MSB is left in the logical "1" or true state and the next bit (which corresponds to one-quarter of full scale) is set to a logical "1". If the second bit does not cause the DAC output to exceed the unknown analog input, it is left in the logical "1" state and the next bit is set to "1". If the second bit causes the DAC output to exceed the unknown analog signal, it is set back to a logical "0" as the next bit is set to a logical "1" for another comparison. This process continues in order of descending bit weight until the proper state of the last bit has been determined. At that time the digital input to the DAC represents the correct digitization of the previously unknown analog voltage or current.

All of the digital control logic normally found in the successive approximation ADC can be replaced by a software subroutine using the microprocessor. Program 4.4.1.3(1) is a successive approximation analog-to-digital conversion subroutine written for the UT PDCP development system. A flow chart for this program is illustrated in Figure 4.4.1.3(2) and a simplified schematic of the ADC hardware used in the development system is shown in Figure 4.4.1.3(3).

The successive approximation subroutine uses only 34 bytes of ROM and six bytes of RAM. As currently implemented, the analog-to-digital conversion requires a fixed time of 748 states. For a standard 8080A operating with a 2-MHz clock, this corresponds to 2673 conversions per second, a rate which should be more than satisfactory for the PDCP application.

4.4.2 Data Compaction and Preprocessing

The amount of data which must be handled by data collection satellites is constantly increasing. The major factors contributing to this increase are the use of high sampling-rate sensors and a general growth in the number of DCP's. PDCP's can be expected to contribute substantially to the increase in remote data collection activities by opening new areas of application. In addition to taxing the bandwidth limits of

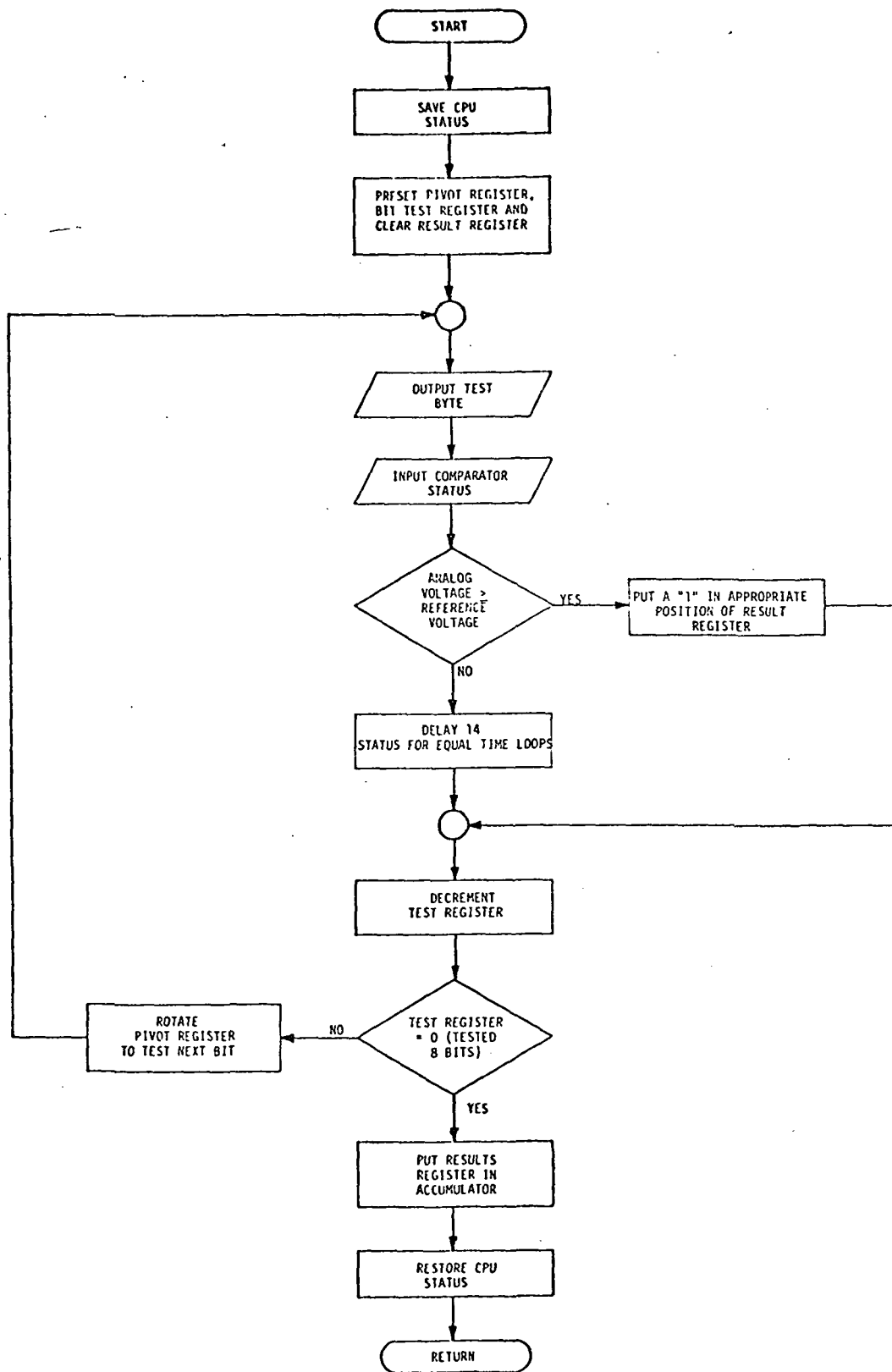


Figure 4.4.1.3(1) Flow Chart for Successive Approximation ADC Subroutine.

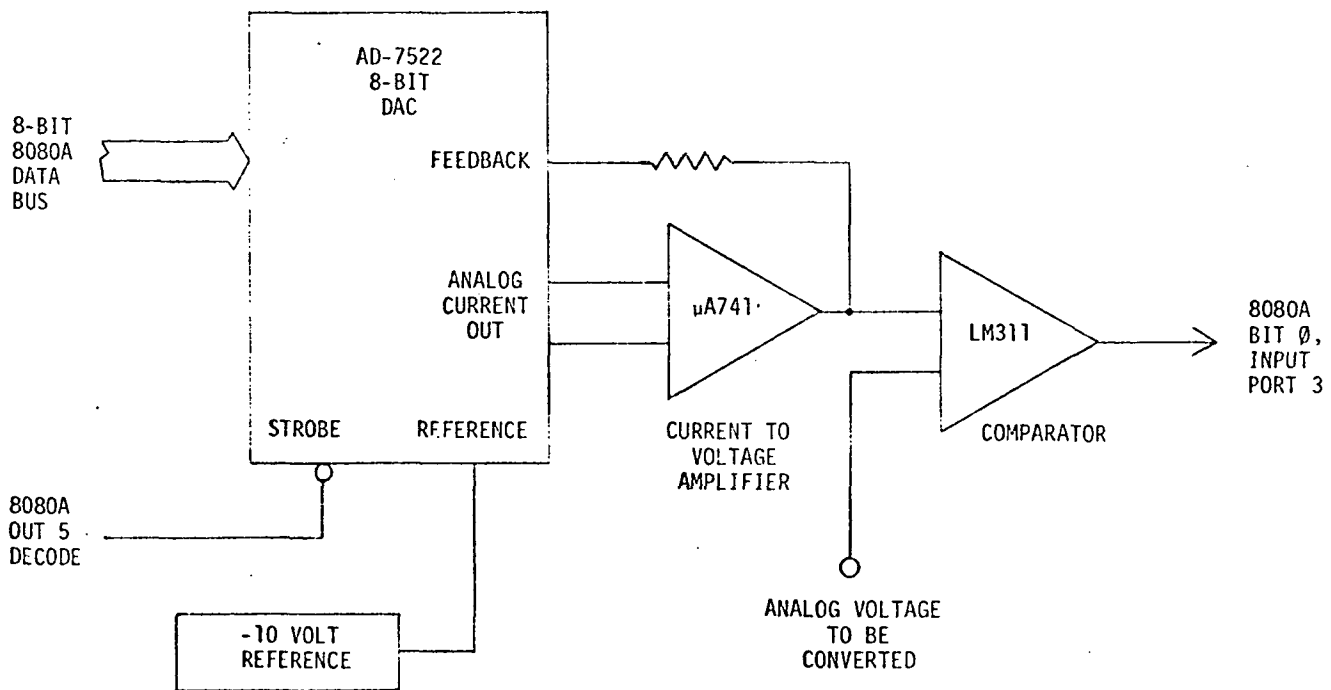


Figure 4.4.1.3(2) Simplified Schematic of the ADC Hardware.

present data collection satellites, significant increases in data flow would require corresponding increases in the capacity of present ground-to-ground communication lines, data storage areas, and data processing facilities. Therefore, techniques should be developed for reducing the volume of data flow without significantly reducing the amount of information acquired.

One of the most useful ways to reduce data flow to manageable proportions is to apply compression or compaction before the data is transmitted from the PDCP. There are two basic types of data compression techniques: entropy-reducing and information-preserving [4]. Entropy-reducing algorithms perform an irreversible transformation on the input data and therefore cause a pre-transmission loss of some information. Information-preserving algorithms perform a reversible transformation on the input data. As a result, the information carried by the original data can be recovered within a specified allowable tolerance or peak error. Bit compaction rates of two-to-one can normally be achieved by information-preserving algorithms. Much higher degrees of data compaction can be obtained by using entropy-reducing algorithms.

Data compaction techniques have not been employed in previous DCP designs because of the extra hardware and expense involved in hardwired implementations of the algorithms. Many data compression algorithms, however, can be economically used with PDCP's. In most cases, the only additional hardware required to provide data compression capabilities will be the memory used to store subroutine implementations of the various data compression algorithms. Data compression subroutines are excellent candidates for the subroutine library within the applications-oriented PDCP programming system (see Section 4.3.1). Several examples of data compression programs are presented in the following sections.

4.4.2.1 General-Purpose Binary Math Package - Most data compression algorithms are based upon mathematical operations. Therefore, a general-purpose binary math package has been developed for use in data compaction operations. The math package consists of subroutines which perform the unsigned binary operations specified in Table 4.4.2.1(1). Unsigned binary format is assumed since PDCP data will usually be represented in this

TABLE 4.4.2.1(1)

GENERAL PURPOSE PDCP BINARY MATH PACKAGE

Basic Operation	Form (Allowable Number Size In Bytes)	Memory Requirements In Bytes		Execution Time In States (N=Number of Bytes)	Subroutine Title
		Program Storage (ROM)	Stack (RAM)		
ADDITION	Double Precision (2+2→2)	11	2	83	DADDM
	Short Multiprecision (2+4→4)	20	4	141	SMADD
	Multiprecision ([1-256]+[1-256]→[1-256])	17	10	56N+95	MADD
SUBTRACTION	Double Precision (2+2→2)	11	2	83	DSUBM
	Short Multiprecision (2+4→4)	20	4	141	SMSUB
	Multiprecision ([1-256]+[1-256]→[1-256])	17	10	56N+95	MSUB
MULTIPLICATION	Double Precision (2x2→4)	73	14	4683-4843	DPMULT
DIVISION	Double Precision (4÷2→2)	64	8	3645-8456	DPDIV
SQUARE ROOT	Double Precision ($\sqrt{2}$ → 1)	84	14	4336-4746	SQRT

TABLE 4.4.2.1(1) (Continued)

Basic Operation	(Allowable Number Size In Bytes)	Memory Requirements In Bytes		Execution Time In States (N=Number of Bytes)	Subroutine Title
		Program Storage (ROM)	Stack (RAM)		
COMPARISON	Double Precision (2-2)	12	4	62-73	DCMPM
	Multiprecision ([1-256]-[1-256])	54	10	110N+108	MCMP
MEMORY-TO-MEMORY MOVE	Multiprecision ([1-256]MOVE TO[1-256])	15	10	49N+91	MOVE
SUBROUTINE EXIT RESTORING ALL STATUS	POPS ALL STATUS	5	-	60	EXIT

form. Listings of the math package subroutines are provided as Programs A(1) through A(12) in Appendix A.

The most important PDCP math subroutines (addition and subtraction) are available in three formats: double precision (numbers up to 16 bits long), short multiprecision (mixed 16-bit and 32-bit number operations), and multiprecision (numbers up to 2048 bits long). This allows the PDCP programmer to consider memory requirements, execution time, and operand resolution in selecting the best subroutine for a particular task. Most PDCP data words will be eight to ten bits long and can most appropriately be manipulated using the double precision subroutines. The short multiprecision subroutines supplement the double precision subroutines and are intended for use in operations such as summing a large number of data points where the result could overflow 16 bits (i.e., a result greater than 65,535). Short multiprecision and double precision subroutines should be sufficient for PDCP data compaction operations. Multiprecision subroutines capable of operating on numbers up to 256 bytes long are provided primarily to illustrate that numbers of any magnitude can be processed by a microprocessor based PDCP if sufficient calculation time is available.

The comparison, multiplication, division, and square-root subroutines permit significant data compaction operations such as data averaging to be performed by the PDCP. Since a special square-root algorithm developed by Kostopoulos [5] was implemented, a flow chart for the square-root subroutine is shown in Figure 4.4.2.1(1). The MOVE subroutine is provided for transferring multiprecision data blocks between temporary storage and working memory areas when using multiprecision arithmetic subroutines. EXIT is a general purpose block of code which is used to restore all CPU status when returning from selected subroutines. The complete binary math package is 403 bytes long and would occupy only 40 percent of a 8K ROM.

Normally, a PDCP will not require all of the subroutines listed in Table 4.4.2.1(1). A basic math package consisting of double and short multiprecision addition and subtraction, double precision compare, and EXIT would be sufficient for elementary data compaction operations. This

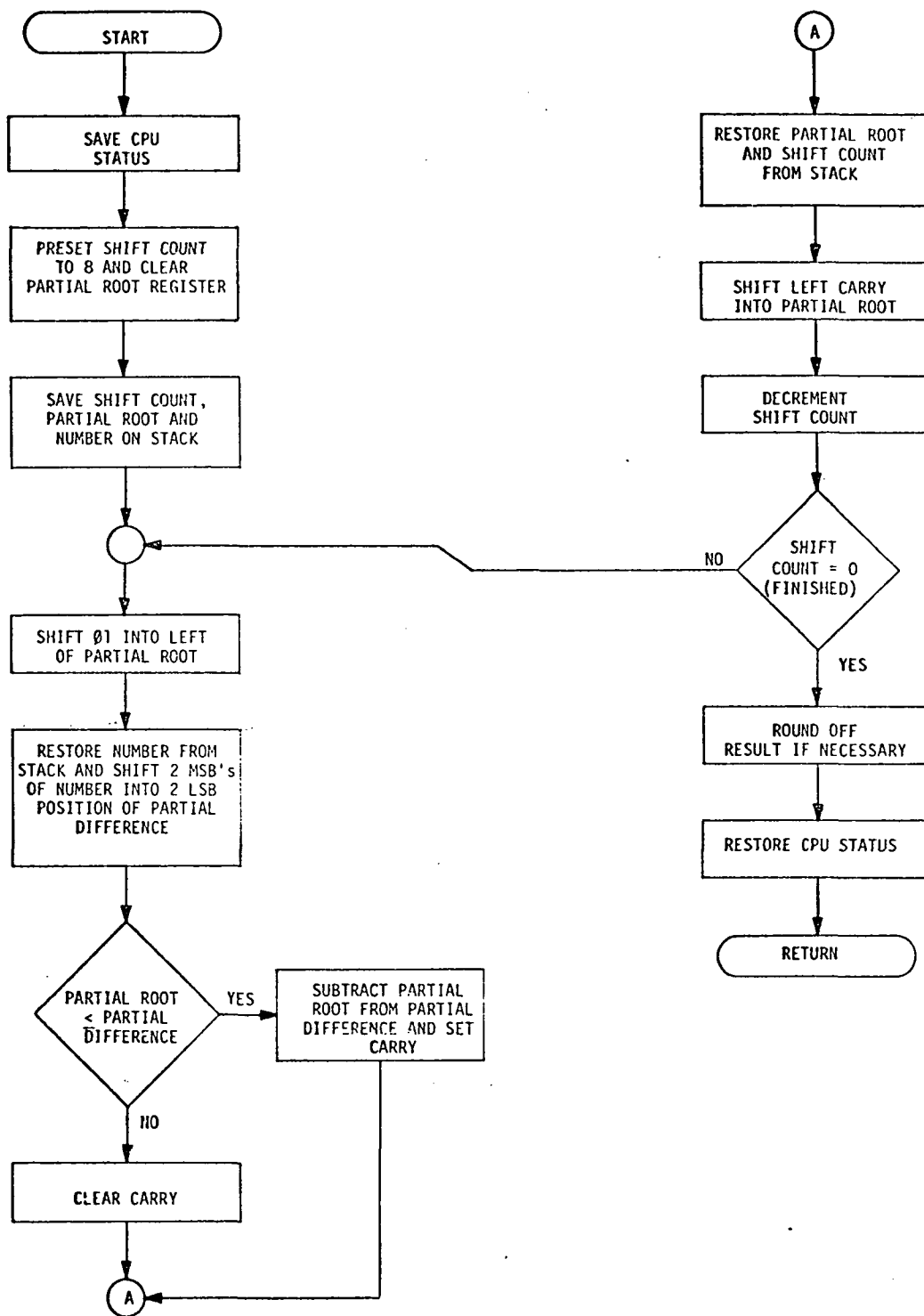


Figure 4.4.2.1(1) Flow Chart for Square Root Subroutine.

basic math package can be stored in 79 bytes of ROM. Multiplication and division can be added to the basic math package to support more advanced data compaction operations such as data averaging. A total of 216 bytes are required to store this form of the math package. Standard deviation calculations require the addition of a square-root subroutine which increases the memory storage area used by the extended version of the basic math package to 300 bytes. Execution times of the PDCP binary math subroutines are shown in Table 4.4.2.1(2).

4.4.2.2 Determination of Minimum and Maximum Data Values - Program 4.4.2.2(1) is an entropy-reducing data compaction subroutine which determines the minimum and maximum data samples obtained from a sensor over a particular time interval. The DMINMAX program uses the double precision compare subroutine from the binary math package and operates on numbers up to 16 bits long. Each time the DMINMAX routine is called by the PDCP executive program, the value of a new data point is compared to previously established upper and lower bounds. The carry and zero flags indicate the result of this comparison. If the new data point is outside one of the boundaries, the data point is stored as a new boundary. Therefore, just prior to transmission, the upper and lower bounds will represent the minimum and maximum data samples obtained since the last transmission. Program 4.4.2.2(1) is 29 bytes long and requires eight bytes of RAM for stack operations. Worst case execution time is 264 states which would permit a data rate of over 7,500 samples per second.

A multiprecision MINMAX subroutine is listed as Program 4.4.2.2(2). The MINMAX program uses two subroutines from the multiprecision binary math package and operates on numbers up to 256 bytes long. The value of a memory location named TMCODE indicates the result of the MINMAX program execution. Program 4.4.2.2(2) is 120 bytes long and requires 20 bytes of RAM for stack operations. These figures include the required multi-byte COMPARE and MEMORY-TO-MEMORY MOVE subroutines from the math package.

4.4.2.3 Zero-Order Floating Aperture Predictor - A potentially more useful data compression algorithm is implemented in Program 4.4.2.3(1). The zero-order floating aperture predictor algorithm is an information-preserving polynomial data compression technique [4]. The algorithm is

TABLE 4.4.2.1(2)

BINARY MATH SUBROUTINE EXECUTION TIMES

Operation	Number Size (8-bit Bytes)	States	Execution Time	
			Microseconds Using 500-nsec 8080A	Microseconds Using 325-nsec 8080A-1
ADD	2	83	41.5	26.98
	2,4	141	70.5	45.83
	N	95+56N	47.5+28N	30.88+18.2N
SUBTRACT	2	83	41.5	26.98
	2,4	141	70.5	45.83
	N	95+56N	47.5+28N	30.88+18.2N
COMPARE	2	62-73	31.0-36.5	20.15-23.73
	N	108+110N	54.0+55N	35.1+35.75N
MULTIPLY	2	4683-4843	2341.5-2421.5	1521.98-1573.98
DIVIDE	2,4	3645-8456	1822.5-4228	1184.63-2748.20
SQUARE ROOT	2	4336-4746	2168-2373	1409.20-1542.45
MOVE	N	91+49N	45.5+24.5N	29.58+15.93N
EXIT	-	60	30	19.5


```

MULTI-PRECISION
MINIMUM-MAXIMUM VALUE DETERMINATION SUBROUTINE
;
; THIS SUBROUTINE COMPARES THE VALUE OF A NEW DATA POINT (ND)
; TO UPPER (UB) AND LOWER (LB) BOUNDS. IF ND<LB, THE VAL...
; UE OF ND BECOMES THE NEW LOWER BOUND. IF ND>UB, THE
; VALUE OF ND BECOMES THE NEW UPPER BOUND. A FLAG
; IS SET TO INDICATE THE RESULT OF THE COMPARISONS
; AND THE EXECUTION TIME OF THE SUBROUTINE. ALL
; NUMBERS ARE ASSUMED TO BE IN UNSIGNED BINARY
; FORMAT.
;
; (A) = # BYTES / NUMBER. SPECIFY TO GUARANTEE NO
; OVERFLOW ON ADDITION.
; (BC) = ADDRESS OF NEW DATA POINT'S LEAST SIG BYTE
; (HL) = ADDRESS OF UPPER BOUND LEAST SIGNIFICANT BYTE.
; THE LEAST SIG BYTE OF THE LOWER BOUND MUST BE
; STORED IMMEDIATELY ABOVE THE MOST SIGNIFICANT
; BYTE OF THE UPPER BOUND.
;
; ALL DATA WORDS IN MEMORY ARE ASSUMED TO BE STORED IN ASCEND...
; MEMORY LOCATIONS BEGINNING AT THE ADDRESSES SPECIFIED ABOVE.
;
; THIS SUBROUTINE SAVES AND LATER RETURNS THE MACHINE STATUS.
; TMCODE IS USED AS A FLAG BYTE. ON RETURN, TMCODE=0 IF THE
; NEW DATA IS BETWEEN THE BOUNDS.
;
; TMCODE = 1 IF NEW DATA IS GREATER THAN THE UPPER BOUND
; TMCODE = 2 IF NEW DATA IS LESS THAN THE LOWER BOUND
;
; THIS SUBROUTINE DOES NOT ALTER THE VALUE OR LOCATION OF THE
; NEW DATA POINT. EXECUTION TIME IS A FUNCTION OF TMCODE
; AND THE NUMBER OF BYTES PER NUMBER AS DEFINED BELOW:
;
; IF TMCODE = 0 THEN T = 262*A+414
; IF TMCODE = 1 THEN T = 159*A+380
; IF TMCODE = 2 THEN T = 262*A+510
; WHERE T = EXECUTION TIME IN STATES
;
; TMCODE = 17776 ; TIME CODE STORAGE BYTE LOCATION

```

017776

```

000000
000000
000000 065
000001
000001 005
000002
000002 025
000003
000003 045

```

```

MINMAX:  PUSH  PSW          ; SAVE CPU STATUS
          PUSH  B
          PUSH  D
          PUSH  H

```

Program 4.4.2.2(2) Multi-Precision Minimum-Maximum Value Determination.

```

000004 MOV     E,A      ; SAVE NUMBER OF BYTES PER NUMBER IN E
000004 CALL   MCOMP   ; COMPARE NEW DATA (ND) POINT TO UPPER BOUND (UB)
000005 DCR   D      ; DECREMENT D TWICE TO SET CONDITIONS FOR FLAG REGISTER D
000010 DCR   D
000011 JZ    DR      ; JUMP IF NDDUB
000012 MVI   D,0    ; 0 ----> D
000015 DAB   D      ; OTHERWISE (DE) + (HL) ----> (HL), NOW CHL - LB
000017 CALL  MCOMP ; COMPARE ND TO LOWER BOUND (LB)
000020 DCR   D      ; DECREMENT TO TEST FLAG REGISTER D
000023 JZ    UR      ; JUMP IF NDCLB, OTHERWISE ND IS WITHIN BOUNDS
000024 MVI   A,0    ; 0 ----> A
000027 STA  TMCODE ; ZERO THE EXECUTION TIME REGISTER
000031 JMP   EXIT    ; AND EXIT ROUTINE
000034 DR:
000037 MVI   A,1    ; 1 ----> A SINCE NDDUB
000037 JMP   UR+2    ; CONTINUE AT UR+2
000041 MVI   A,2    ; 2 ----> A SINCE NDCLB
000044 STA  TMCODE ; STORE A AS TIME CODE INDICATING EXECUTION
000046 MOV   A,E    ; TIME AND RESULT
000051 CALL  MOVE    ; #BYTES/NUMBER ----> A
000052 JMP   EXIT    ; MOVE ND TO LOCATION RESERVED FOR UB
000055 MVI   A,0    ; GO TO EXIT ROUTINE
000055

```

Program 4.4.2.2(2) (continued)

```

; DOUBLE PRECISION ZERO ORDER FLOATING APERTURE PREDICTOR
;
; (BC) = NEW DATA POINT (16 BITS)
; (HL) = ADDRESS OF LEAST SIGNIFICANT BYTE OF LOWER BOUND
; THE LEAST SIGNIFICANT BYTE OF THE UPPER BOUND SHOULD
; BE STORED AT THE MEMORY LOCATION ADDRESSED BY (HL)+2
; (DE) = ADDRESS OF TOLERANCE LEAST SIGNIFICANT BYTE
;
; ON RETURN:
;
; Z = 1 IF NEW DATA IS WITHIN TOLERANCE
; CY = 0 IF NEW DATA IS LESS THAN LOWER BOUND
; CY = 1 IF NEW DATA IS GREATER THAN UPPER BOUND
; FLAGS AND ACCUMULATOR ARE DESTROYED
;
; 56 BYTES ROM + 34 BYTES FROM MATH SUBROUTINES, 4 BYTES STACK
; USAGE AND 79-545 MACHINE STATES.
;
DZOFAB:
000447 CALL DCMPM ; COMPARE ND TO LB
000447 RZ ; ND=LB, DATA WITHIN TOLERANCE
000452 JC CKOR ; JUMP IF ND<LB
000452 PUSH PSW ; ND<LB, FLAG FOR TRANSMISSION, SAVE STATUS
000453 JMP NREF ; SET UP NEW REFERENCE POINT AS ND
000454 INX H ; POINT (HL) TO LSB OF UB BY INCREMENTING TWICE
000462 INX H ;
000463 CALL DCMPM ; COMPARE ND TO UB
000464 DCX H ; RESTORE (HL)
000467 DCX H ;
000470 JNC SZRET ; JUMP IF ND<= UB, DATA WITHIN TOLERANCE
000471 PUSH PSW ; ND>UB, FLAG FOR TRANSMISSION SAVING STATUS
000474 XCHG ; ESTABLISH ND AS NEW REFERENCE POINT
000475 CALL DSUBM ; SUBTRACT TOLERANCE FROM ND AND STORE AS NE LB
000476 JNC ENUB ; JUMP IF NO UNDERFLOW OCCURRED
000477 XRA A ; ELSE CLEAR ACC
000501 CALL LMEM ; UNDERFLOW ..... SO SET LB=0
000501
000504
000504
000505
000505

```

Program 4.4.2.3(1) Double Precision Zero-Order Floating Aperture Predictor Subroutine.

based on the prediction that later data samples will be within a specified tolerance of the first sample. If this prediction is true, the data sample is considered redundant and should not be transmitted. The first sample which is not within the specified tolerance is flagged for transmission and becomes the new reference point for subsequent predictions. The result of the comparison is indicated by the states of the zero and carry flags after the program has been executed. The zero flag must be tested first to avoid incorrect results. A flow chart for the zero-order floating aperture predictor subroutine is illustrated in Figure 4.4.2.3(1). This program is 56 bytes long and uses three subroutines from the binary math package requiring an additional 34 bytes. Six bytes of RAM are required for stack operations. At present, 100-piece CMOS ROM prices, the incremental component cost of providing the zero-order floating aperture predictor subroutine and the required math package subroutines as part of a standard PDCP ROM, is approximately \$7.21.* As shown in Section 4.4.3.2, the longest data formatting and transmitter control subroutine (GOES format) is 236 bytes long. Therefore, the double precision zero-order floating-aperture predictor algorithm and a transmitter subroutine would occupy only 32 percent of the capacity of an 8K bit ROM. The remaining 698 bytes of ROM should be sufficient to contain an executive program and several sensor data input and control subroutines. For example, the frequency sensor input subroutine [Program 4.4.1.1(1)] which is capable of serving eight different sensors requires only 36 additional bytes of ROM.

Program 4.4.2.3(2) is a multiprecision implementation of the zero-order floating-aperture predictor algorithm. A flow chart for this program is illustrated in Figure 4.4.2.3(2). Program execution times and

* For example, RCA's CDP1831CD 512 x 8 ROM with a 400 nsec access time and 0.5 mW typical quiescent power dissipation costs \$41.00 each for the first 100 devices, including the mask charge. Therefore,

$$\Delta \text{ Cost} = \frac{41.00}{512 \text{ bytes}} \times 90 \text{ bytes} = \$7.21.$$

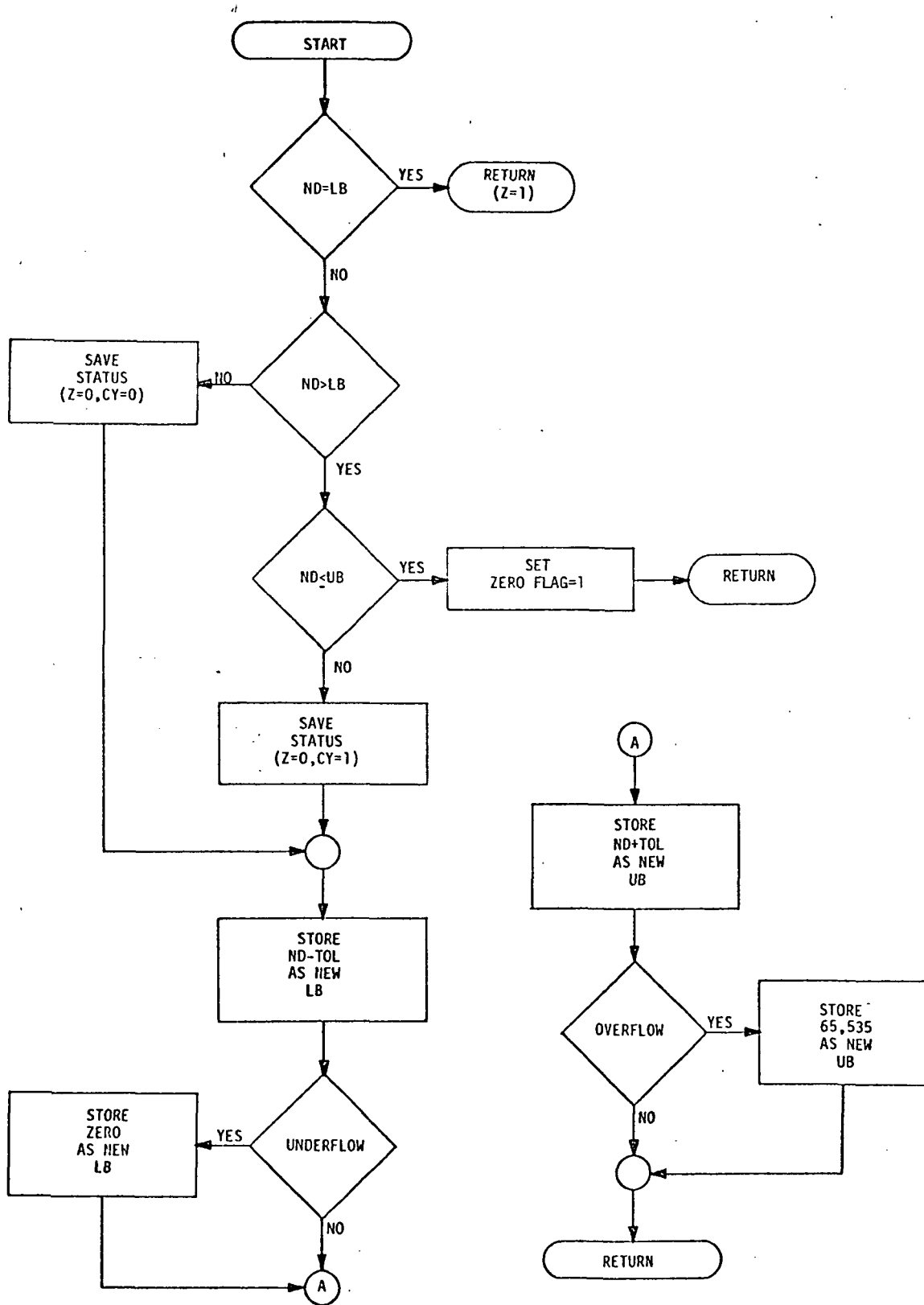


Figure 4.4.2.3(1) Flow Chart for the Double Precision Zero-Order Floating-Aperture Predictor Subroutine.


```

ZERO-ORDER FLOATING-APERTURE PREDICTOR SUBROUTINE
THIS SUBROUTINE IMPLEMENTS THE ZERO-ORDER FLOATING-APERTURE
PREDICTOR ALGORITHM USING AN 8080A MICROPROCESSOR. THIS
IS AN INFORMATION PRESERVING POLYNOMIAL DATA COMPRESSION
TECHNIQUE BASED ON THE PREDICTION THAT LATER SAMPLES
WILL BE WITHIN A SPECIFIED TOLERANCE OF THE FIRST
SAMPLE. IF THIS PREDICTION IS TRUE, THE DATA
SAMPLE IS CONSIDERED REDUNDANT AND IS NOT
TRANSMITTED. THE FIRST SAMPLE WHICH IS
NOT WITHIN THE SPECIFIED TOLERANCE IS
TRANSMITTED AND BECOMES THE NEW REFERENCE
POINT FOR SUBSEQUENT PREDICTIONS. ALL NUMBERS ARE ASSUMED TO
BE IN N-BYTE, UNSIGNED BINARY
FORMAT:
(A) = # BYTES / NUMBER. SPECIFY TO GUARANTEE NO
OVERFLOW ON ADDITION.
(BC) = ADDRESS OF NEW DATA POINT'S LEAST SIG BYTE
(DE) = ADDRESS OF TOLERANCE LEAST SIGNIFICANT BYTE
(HL) = ADDRESS OF UPPER BOUND LEAST SIGNIFICANT BYTE.
THE LEAST SIG BYTE OF THE LOWER BOUND MUST BE
STORED IMMEDIATELY ABOVE THE MOST SIGNIFICANT
BYTE OF THE UPPER BOUND.
ALL DATA WORDS IN MEMORY ARE ASSUMED TO BE STORED IN ASCENDING
MEMORY LOCATIONS BEGINNING AT THE ADDRESSES SPECIFIED ABOVE.
THIS SUBROUTINE SAVES AND LATER RETURNS THE MACHINE STATUS.
TMCODE IS USED AS A FLAG BYTE. ON RETURN, TMCODE=0 IF THE
NEW DATA POINT WAS REDUNDANT. TMCODE>0 IF THE NEW DATA POINT
IS NOT REDUNDANT AND THEREFORE HAS BEEN ESTABLISHED AS THE
NEW REFERENCE POINT. SPECIFICALLY:
TMCODE = 1 IF NEW DATA IS GREATER THAN THE UPPER BOUND
OF THE REFERENCE PLUS TOLERANCE
TMCODE = 2 IF NEW DATA IS LESS THAN THE LOWER BOUND
OF THE REFERENCE MINUS TOLERANCE
THIS SUBROUTINE DOES NOT ALTER THE VALUE OR LOCATION OF THE
NEW DATA POINT. EXECUTION TIME IS A FUNCTION OF TMCODE
AND THE NUMBER OF BYTES PER NUMBER AS DEFINED BELOW:
IF TMCODE = 0 THEN T = 220*A+414
IF TMCODE = 1 THEN T = 320*A+771
IF TMCODE = 2 THEN T = 430*A+201
WHERE T = EXECUTION TIME IN STATES
A = NUMBER OF BYTES PER NUMBER

```

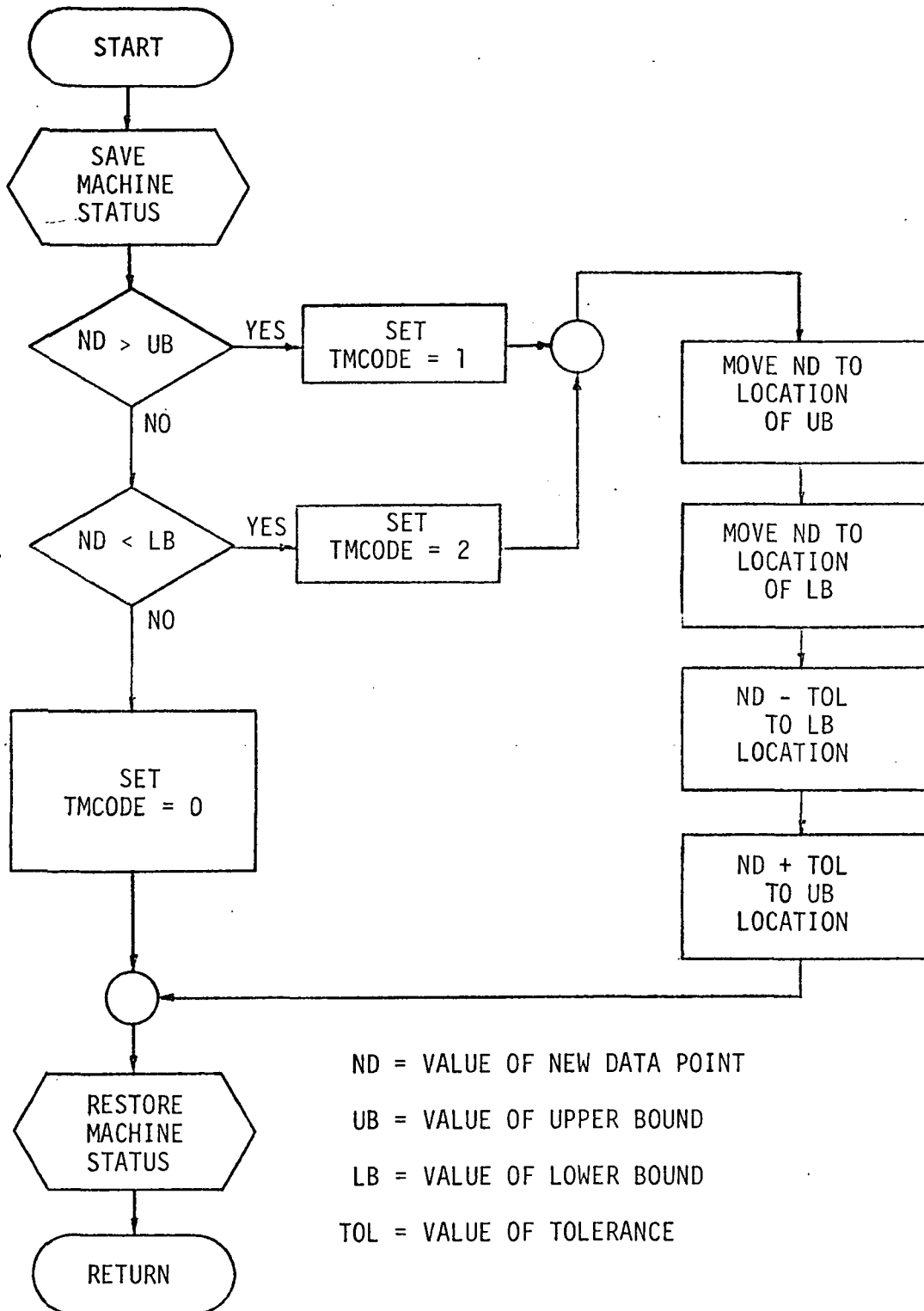



Figure 4.4.2.3(2) Flow Chart for the Multiprecision Zero-Order Floating Aperture Predictor Subroutine.

the results of the multiprecision zero-order floating-aperture predictor subroutine are indicated by the value returned in the memory location called TMCODE. This program is 71 bytes long and uses four subroutines from the multiprecision binary math package for a minimum storage requirement of 178 bytes. Twenty bytes of RAM are required for stack operations.

4.4.2.4 Data Average - Data averaging is one useful form of data preprocessing which can easily be performed by a PDCP. Normally two distinct subroutines will be used to obtain the average value of a set of N data points. A data accumulation subroutine such as Program 4.4.2.4(1) will be called each time a data sample is obtained. When the average value of the data is desired, a final data averaging subroutine will be called. As illustrated by the flow chart in Figure 4.4.2.4(1), the data accumulation subroutine will simply increment the sample count (N) and add the new sample to the previous sum of samples. Program 4.4.2.4(1) can process up to 65,535 samples of 16-bit sensor data while using only 18 bytes of ROM and 10 bytes of stack.

Program 4.4.2.4(2) is an example of a data averaging subroutine. A flow chart for this program is shown in Figure 4.4.2.4(2). The data averaging subroutine calls the short multiprecision divide routine to divide the accumulated sum of samples by the number of samples, N. The resultant average is returned in register pair DE. Only 24 bytes of ROM and 18 bytes of stack are required for this subroutine. Therefore, data averaging can be accomplished by the PDCP at a cost of only 42 bytes of ROM and 18 bytes of stack in addition to the general purpose binary math subroutines.

4.4.2.5 Mean, Variance, and Standard Deviation - Much more information can be obtained from the mean or average of a number of data samples if the variance and standard deviation are also known. Programs 4.4.2.5(1) and 4.4.2.5(2) can be used in conjunction with the data accumulation subroutine [Program 4.4.2.4(1)] to calculate this information aboard the PDCP. Program 4.4.2.5(1) is called each time a data sample is obtained. This program prepares the data for the mean, variance, and standard deviation subroutine by calculating the number of samples, the

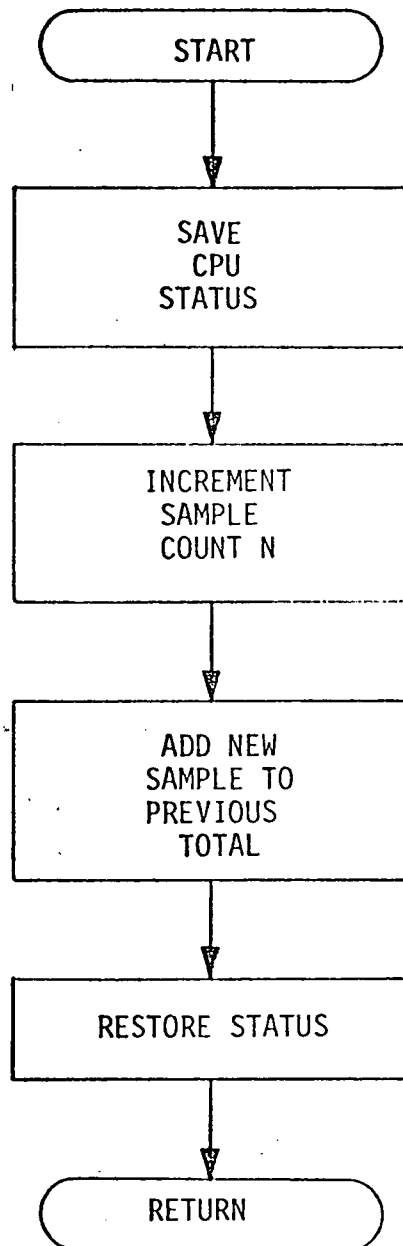


Figure 4.4.2.4(1) Flow Chart for the Data Accumulation Subroutine.

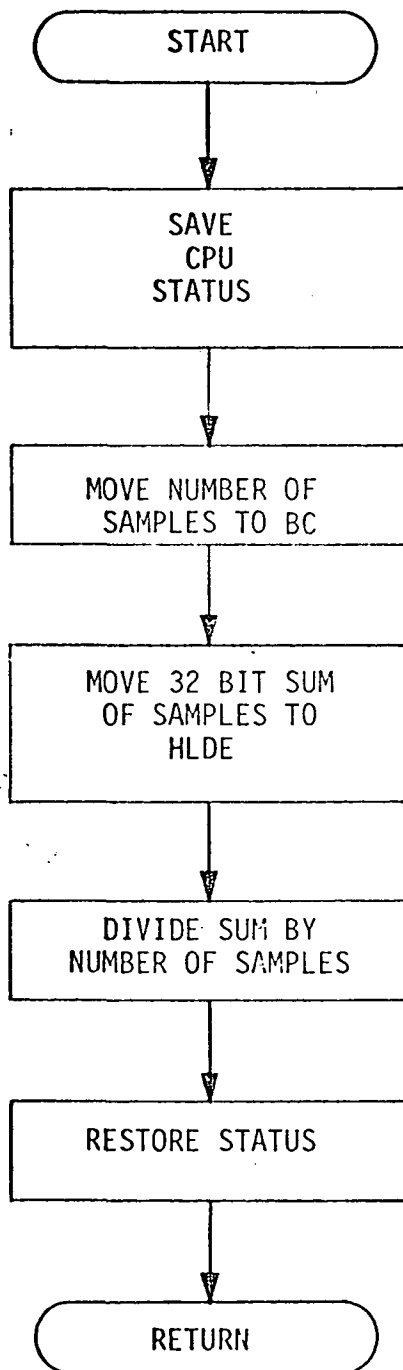


Figure 4.4.2.4(2) Flow Chart for the Data Averaging Subroutine.

000740	MOV	B,M	POINT HL TO SCRATCH AREA BY INCREMENTING 3 TIMES
000740	INX	H	
000741	INX	H	
000742	INX	H	
000743	INX	H	
000744	CALL	DFMULT	MULTIPLY DE*BC TO SQUARE SUM OF SAMPLES
000744	CALL	MVSDI	
000747	CALL	DFDIV	(HLDE)/(BC) -----> DE
000752	POP	H	POINT HL TO N
000755	PUSH	H	SAVE HL AGAIN
000756	LXI	B,10	10 -----> BC
000757	DAD	B	ADD 10. TO HL TO POINT TO SUM OF SQUARED SAMPLES
000762	MOV	B,D	QUOTIENT TO BC
000763	MOV	C,E	
000764	CALL	SMSUB	SUBTRACT (BC) FROM (MC*HL) TO FORM (N-1)*VARIANCE
000765	CALL	MVSDI	MOVE RESULT TO HLDE AND # OF SAMPLES TO BC
000770	DCX	B	N-1 -----> BC
000770	CALL	DFDIV	DIVIDE DIFFERENCE BY N-1 TO GET VARIANCE
000773	POP	H	GET ADDRESS OF N
000774	PUSH	H	SAVE HL AGAIN
000777	INX	H	INCREMENT HL TWICE
000777	INX	H	
001000	MOV	M,E	STORE VARIANCE ABOVE #SAMPLES IN MEMORY
001000	INX	H	
001001	MOV	M,D	
001002	XCHG		
001003	CALL	SORT	SORT. OF VARIANCE TO ACCUMULATOR
001004	INX	D	
001005	STAX	D	STORE STANDARD DEVIATION IN MEMORY
001006	INX	D	
001007			
001012			
001012			
001013			
001013			
001014			

Program 4.4.2.5(2) (continued)

```

001014 XFC A ;
001015 STAX D ;
001016 JMP EXIT ; RESTORE STATUS AND RETURN TO CALLING PROGRAM
001017
001022
001022 MOV E,M ; SUBROUTINE TAKES 4 BYTES FROM MEMORY BEGINNING AT
001022 INX H ; CHL AND PUTS THEM IN HL. ALSO LOADS BC WITH
001023 MOV D,M ; (MCLAST VALUE ON STACK)
001024 INX H ; MOVE LS TWO BYTES OF RESULT TO DE
001025
001026 PUSH D ; SAVE LS TWO BYTES
001027 MOV E,M ; MOVE MS TWO BYTES TO DE
001028 INX H ;
001029 MOV D,M ;
001031 INX SP ; POINT SP TO STACK AREA WHERE THE ADDRESS
001032 INX SP ; OF THE NUMBER OF SAMPLES IS STORED BY INCREMENTING
001033 INX SP ; 4 TIMES
001034
001035 INX SP ;
001036 POP H ; GET ADDRESS OF NUMBER OF SAMPLES
001037 PUSH H ; RESAVE IT
001040 DCX SP ; RESTORE SP BY DECREMENTING 4 TIMES
001041 DCX SP ;
001042 DCX SP ;
001043 DCX SP ;
001044 MOV C,M ; MOVE # SAMPLES TO BC
001045 INX H ;
001046 MOV B,M ;
001047 POP H ; PUT 2 MS BYTES OF RESULT INTO HL
001050 XCHG ;
001051 RET ; RETURN

```

Program 4.4.2.5(2) (continued)

sum of the samples, and the sum of the squared samples. A flow chart for Program 4.4.2.5(1) is provided in Figure 4.4.2.5(1).

Program 4.4.2.5(2) calculates the mean, variance, and standard deviation of the data using the formulas

$$\text{MEAN} = \frac{1}{n} \sum_{i=1}^n x_i ,$$

$$\text{Variance } (S^2) = \frac{1}{n-1} \left[\sum_{i=1}^n x_i^2 - \frac{1}{n} \left(\sum_{i=1}^n x_i \right)^2 \right], \text{ and}$$

$$\text{Standard Deviation}(S) = \sqrt{S^2} .$$

A PDCP with this capability would typically calculate and transmit the mean and variance of sensor data acquired between transmissions rather than transmitting a large number of data points. This will significantly reduce the load on the satellite data collection system by reducing bandwidth and data storage requirements. In addition, this data preprocessing operation provides the user with immediate knowledge of the average size of the sample values and the dispersion of the samples about the mean.

A flow chart for Program 4.4.2.5(2) is shown in Figure 4.4.2.5(2). The data accumulation; data preparation for mean, variance, and standard deviation; and mean, variance, and standard deviation subroutines require a total of 150 bytes of program storage and 28 bytes of stack.

4.4.3 Data Formatting and Transmitter Control

Data formatting and transmitter control are two of the four basic tasks performed by the hardwired control logic currently used in most DCP designs. Special emphasis has been placed on developing data formatting and transmitter control subroutines for each of the major data collection satellites because these are essential PDCP tasks which require

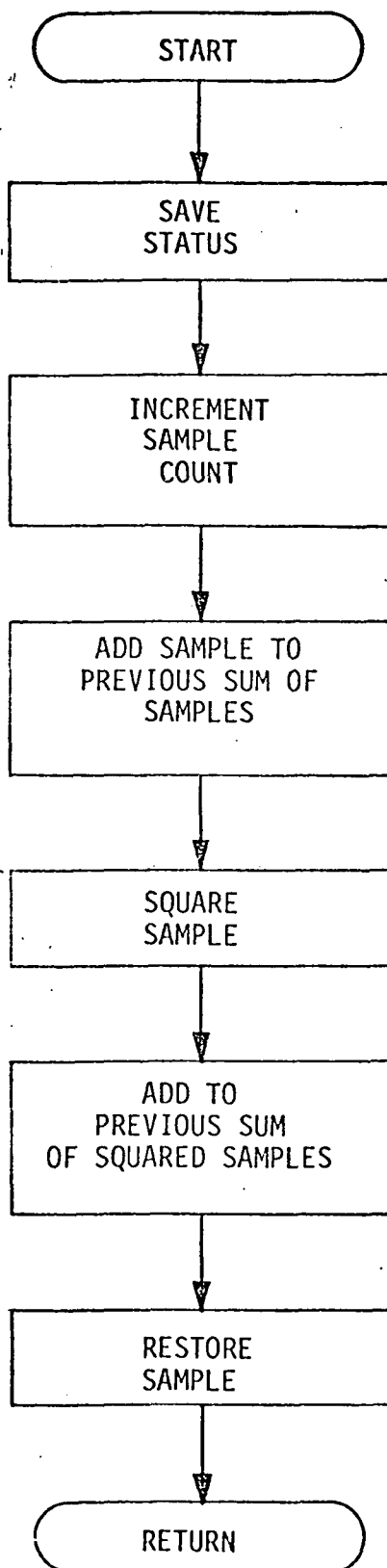


Figure 4.4.2.5(1) Flow Chart for Data Preparation for Mean, Variance, and Standard Deviation Subroutine.

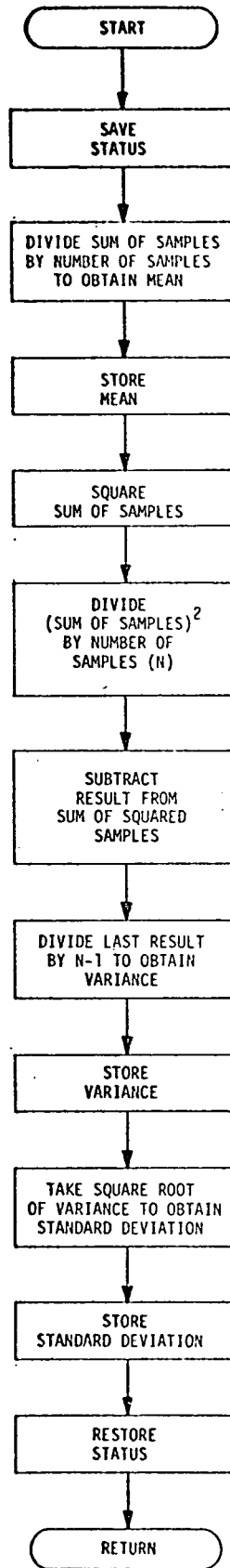


Figure 4.4.2.5(2) Flow Chart for Mean, Variance, and Standard Deviation Subroutine.

precise timing. Normally, both data formatting and transmitter control will be provided by a single subroutine in order to reduce the amount of RAM required for data storage. For example, in the GOES data transmission format an eight-bit data byte is encoded as two 11-bit ASCII words. Each ASCII word is then Manchester encoded before transmission. Thus, as many as 44 bites of RAM would be required to store only eight bits of data. However, if the microprocessor is sufficiently fast, the RAM allocation for data storage can be reduced 82 percent by performing both the ASCII and Manchester encoding in real time during transmission.

The transmitter control programs illustrated in this section assume transmitter modules which operate in basically the same manner as the Landsat-GOES convertible transmitter designed by Ball Brothers Corporation [6]. This transmitter module was chosen because control signal specifications were readily available. Note that the programs presented in this section can easily be modified to function with any reasonable transmitter design.

Ball Brothers' transmitter control signals INTEGRATE, TRANSMITTER ENABLE, and $\pm 15V$ POWER ON are designated as INT, XMITE, and XPWR, respectively. To demonstrate the versatility of a PDCP, additional control signals which could select Landsat (LS), GOES (GO), TIROS-N (TN), or TWERLE (TW) transmitter modes are provided. Normally, a particular PDCP would be required to transmit data to only one type of satellite, and thus a simple modular transmitter designed specifically for that satellite could be used. In this case, control signals LS, GO, TN, and TW would not be required. However, if an electrically convertible transmitter is available and a PDCP is required to transmit data to more than one type of satellite, the LS, GO, TN, and TW signals would enable the correct transmitter mode.

The 8080A programs listed in this section assign transmitter control to output port XCNTR (Port 1). Biphase data to the transmitter appears on the least significant bit of output port XMIT (Port 0, see Section 5.1.1.4). Bit assignments for the individual control signals on port XCNTR are illustrated in Figure 4.4.3(1). The function of each

OUTPUT PORT XCNTR		BIT NUMBER
7	TN	0
6	XPWR	1
5	XMITE	2
4	GO	3
3	LS	4
2	$\overline{\text{INT}}$	5
1	TW	6
0	(UNUSED)	7

Figure 4.4.3(1) Bit Assignments for Transmitter Control Port XCNTR.

control signal is specified in Table 4.4.3(1). As an example of microprocessor control of the transmitter, consider the 8080A assembly language instructions

```
MVI A,124
```

```
OUT XCNTR .
```

The first instruction moves the octal byte 124 into the accumulator, and the second instruction causes the contents of the accumulator to be latched into output port XCNTR. As a result, power is applied to the transmitter, GOES mode is selected, and the integrator is disabled as specified by the binary code $\emptyset 1 \emptyset 1 \emptyset 1 \emptyset \emptyset$.

Timing is an important consideration in transmitter subroutines. Correct reception of data at the satellite depends upon the self-clocking characteristic of the Manchester code. The transmission rate must be held constant during each transmission, and the $\emptyset \rightarrow 1$ or $1 \rightarrow \emptyset$ transitions representing data must occur precisely at the middle of each bit time. In addition, the transmission rate must be held within a reasonably tight tolerance over the operational life of the PDCP, which can be greater than one year. Because some variation in the frequency of the PDCP master clock must be expected, the transmitter subroutines should not be allowed to contribute to the total error budget. The entire error budget can then be applied to the system oscillator so that the cost of the clock will be minimized.

Since microprocessor controlled operations can only occur at discrete time intervals defined by the system clock, the clock frequency must be chosen to provide an integer number of states during each phase of a data bit. This technique has been employed in the design of the model PDCP (see Section 5.1.1.1). A low frequency clock was chosen so that the microprocessor could be interfaced with inexpensive, low speed memory without the need for a wait-cycle generator. Further, the specific 600-KHz CPU clock was chosen so that an integer number of states will occur within each phase of the Manchester encoded data for all typical

TABLE 4.4.3(1)
TRANSMITTER CONTROL SIGNALS

Signal	Bit Number	Function
(UNUSED)	0	UNUSED BIT provides don't care condition to minimize the number of instructions required to initialize the transmitter data port (Port XMIT)
TW	1	Selects TWERLE transmitter mode
$\overline{\text{INT}}$	2	$\overline{\text{INTEGRATE}}$ inhibits carrier modulation and thus control clear carrier transmission
LS	3	Selects Landsat transmitter mode
GO	4	Selects GOES transmitter mode
XMITE	5	Enables the transmitter RF signal
XPWR	6	Controls power supply to transmitter
TN	7	Selects TIROS-N transmitter mode

transmission rates. This restriction on the microprocessor clock does not significantly reduce the potential speed of the microprocessor. For example, only a 0.225 percent increase in the theoretical minimum cycle time of the higher speed 8080A-1 microprocessor is required to satisfy the restriction. Table 4.4.3(2) lists the transmission rates for each major data collection satellite system and the corresponding number of states within each phase of the Manchester-encoded data stream for an 8080A with a 600-KHz clock and for an 8080A-1 with a 3.07-MHz clock. Note that the microprocessor must output to port XMIT at a rate equivalent to twice the data rate.

TABLE 4.4.3(2)

TRANSMISSION RATES FOR THE LANDSAT, GOES, TWERLE,
AND TIROS-N DATA COLLECTION SYSTEMS

System	Transmission Rate (Bits Per Second)	Number of States Between Phases of the Manchester-Encoded Data Stream	
		8080A μ P 600-KHz Clock	8080A-1 μ P 3070-KHz Clock
TWERLE	100	3000	15,350
GOES	100	3000	15,350
TIROS-N	400	750	7,675
Landsat	5000	60	307

4.4.3.1 TWERLE Format Transmitter Subroutine - Specifications for the TWERLE data transmission sequence are shown in Table 4.4.3.1(1). Program 4.4.3.1(1) is a listing of the TWERLE format transmitter routine written for the UT PDCP. This subroutine provides complete control of the transmitter and generates a Manchester-encoded data stream at a rate of precisely 100 bits per second. A simplified flow chart for Program 4.4.3.1(1) is illustrated in Figure 4.4.3.1(1), and a detailed flow chart for the data transmission block is presented in Figure 4.4.3.1(2).

TABLE 4.4.3.1(1)
 TWERLE DATA TRANSMISSION SEQUENCE
 SPECIFICATIONS

Transmission Interval: 1 second nominal

Transmission Rate: 100 bits per second

Coding: Manchester encoded with a 0 → 1 transition representing a 1

Transmission Sequence:

1. Transmitter power-up followed by a 1 second warm-up delay
 2. Clear carrier transmission for 0.32 to 0.36 seconds
 3. Data transmission
 - a. Bit synchronization code (10101010) (8 bits)
 - b. Frame synchronization code (110101100000) (12 bits)
 - c. Address code (assigned to user) (10 bits)
 - d. Mode bits (2 MSB's of radio altimeter data, LSB first) (2 bits)
 - e. Data bits (32 bits, LSB first)
 - 1) Radio altimeter data (8 bits)
 - 2) Air temperature data (8 bits)
 - 3) Air pressure data (8 bits)
 - 4) Pressure temperature data (8 bits)
 4. Transmitter power-down
-

```

; TWERLE FORMAT TRANSMITTER SUBROUTINE
;
; THIS PROGRAM CONTROLS THE TRANSMISSION OF 34
; BITS OF MANCHESTER ENCODED DATA IN TWERLE
; FORMAT USING AN 8080A MICROPROCESSOR
; WITH A 1.6666 MICROSECOND STATE
; TIME. ALL TIMING FUNCTIONS
; ARE ACCOMPLISHED IN
; SOFTWARE
;

```

DATA CONSTANT DEFINITIONS

```

020000 TWD = 20000 ; TWERLE DATA BLOCK AT 4K POINT
000000 XMIT = 0 ; OUTPUT PORT ASSIGNMENTS
000001 XCNTR = 1 ;
000252 BSYNC = 252 ; BIT SYNC PATTERN
000153 FSYNC = 153 ; FRAME SYNC PATTERN
000106 TWFO = 106 ; TWERLE POWER ON CONSTANT
000146 TWCC = 146 ; TWERLE CLEAR CARRIER CONSTANT
000142 TWDT = 142 ; TWERLE DATA TRANSMISSION CONSTANT
000004 TO = 4 ; TRANSMITTER OFF CONSTANT (POWER OFF)
000000 ADRS = 0 ; TWERLE PDCP ADDRESS INPUT PORT ASSIGNMENT
000001 ADRS1 = 1 ; TWERLE PDCP ADDRESS - 2MSB INPUT PORT

```

```

000000 ORG 3244 ; START PROGRAM AT 1700.

003244 TWXMIT: MVI A, TWFO ; STARTING POINT OF TWERLE TRANSMITTER ROUTINE
003244 ; APPLY TRANSMITTER POWER IN TWERLE MODE
003246 OUT XCNTR ; WITH RF TRANSMISSION INHIBITED
003246 LXI H, 24966H ; PROVIDE 1 SECOND WARM-UP DELAY
003250 CALL DELAY1
003253 MVI A, TWCC ; ENABLE TRANSMISSION WITHOUT MODULATION
003256 OUT XCNTR ; DO IT!
003260
003260

```


003322	175	STAX	B	; BITS AND THE FIRST FOUR PLATFORM ADDRESS BITS	
003323	002	INX	B	; IN <TWD-2>	
003323	002			; POINT BC TO TWD-1	
003324	003	MOV	A,H	; H <---->A AND STORE THE 6 MSB'S OF THE PLATFORM	
003325	174	STAX	B	; ADDRESS IN <TWD-1>	
003326	002	XCHG		; RETURN VALUE OF PLATFORM ADDRESS TO HL	
003327	353	INX	H	; INCREMENT HL SO HL . NE. 0	
003330	043	CALL	DELAY	; ADD DELAY OF 40*(HL) MICROSECONDS. THE TOTAL	
003331	315			DELAY	044 007
003334	001	LXI	B,3410	; CLEAR CARRIER TRANSMISSION IS 0.31952 TO	
003334	001			; 0.36048 SECONDS DEPENDING ON THE PLATFORM ADDRESS	010 007
003337	041	LXI	H,TWD-4	; ESTABLISH HL AS AN ADDRESS REGISTER POINTING TO	
003337	041			THE FIRST BYTE TO BE TRANSMITTED FROM RAM	374 037
003342	076	MVI	A,TWDT	; PREPARE TO TRANSMIT THE FIRST BIT OF BIT	
003342	076			XCNT	142
003344	323	OUT	XMIT	; SYNC BY PLACING A '1' IN THE TRANSMITTER	
003344	323			XCNT	000
003346	323	OUT	XCNT	; DATA BIT LATCH AND THEN ENABLE TRANSMITTER	
003346	323			MODULATION	001
003350	176	MOV	A,M	; BSYNC BYTE <----> ACCUMULATOR	
003350	176			LXI	305 000
003351	021	LXI	D,197	; SET UP DELAY TO ESTABLISH	
003351	021			DEL	305 000
003354	303	JMP	TW2-1	; 100 BPS TRANSMISSION RATE	
003354	303				365 006

```

; THIS SUBROUTINE CONTROLS TRANSMISSION OF THE REMAINING 7 BITS ;
; OF BIT SYNC, THE 12 BITS OF FRAME SYNC, THE 10 BIT PLATFORM ;
; ADDRESS, AND THE 34 DATA BITS FROM RAM BEGINNING AT RAM ;
; ADDRESS TWD-4

```

003357	TW1:	INX	H	; POINT HL AT THE NEXT DATA BYTE
003357		MVI	B,10	; BIT COUNT=8 ----> B
003360		MOV	A,M	; GET THE DATA BYTE INTO THE ACCUMULATOR
003362		MVI	E,194	; DELAY TO PRODUCE 100 BPS DATA RATE
003363		DLY4		
003365		DCR	E	
003366		JNZ	TW2	
003367		CMA		; COMPLEMENT ACCUMULATOR
003372		OUT	XMIT	; OUTPUT FIRST PHASE OF CURRENT BIT
003373		CMA		; COMPLEMENT ACCUMULATOR
003375		MVI	E,198	; DELAY TO PRODUCE 100 BPS DATA RATE
003376		DCR	E	
003400		JNZ	TW3	
003401		DLY9		
003404		OUT	XMIT	; TRANSMIT THE SECOND PHASE OF THE CURRENT BIT
003405		RRC		; ROTATE THE DB RIGHT ONCE TO GET NEXT BIT
003406		DCR	B	; IF ALL BITS OF THE CURRENT BYTE HAVE NOT
003410		DLY5		; BEEN TRANSMITTED, PREPARE TO JUMP TO TW2
003411		MVI	E,197	; SET UP DELAY
003412		JNZ	TW2	; JUMP IF THE LAST BIT OF THIS BYTE HAS NOT
003413		DCR	C	; BEEN TRANSMITTED
003415		JNZ	TW1	; GO GET THE NEXT BYTE
003415		DLY5		; OTHERWISE DELAY UNTIL THE END OF THE SECOND
003420				
003420				
003421				
003421				
003424				
003424				

Program 4.4.3.1(1) (continued)

```

003425 MVI E,195 ; PHASE OF THE LAST BIT
003425 DCR E
003427 JNZ TW4
003430 MVI A,TO ; PREPARE TO SHUT DOWN TRANSMITTER RF
003433 OUT XCNTR ; DO IT!
003435 RET ; RETURN TO EXECUTIVE
003437
003437
; SUBROUTINES FOR DELAY ;
DELAY1: DLY21
003440
003440
003442
003442
003444
003444
003445
003445
003446
003446
003447
003447
003452
003452
END

003425 MVI E,195 ; PHASE OF THE LAST BIT
003425 DCR E
003427 JNZ TW4
003430 MVI A,TO ; PREPARE TO SHUT DOWN TRANSMITTER RF
003433 OUT XCNTR ; DO IT!
003435 RET ; RETURN TO EXECUTIVE
003437
003437
; SUBROUTINES FOR DELAY ;
DELAY1: DLY21
003440
003440
003442
003442
003444
003444
003445
003445
003446
003446
003447
003447
003452
003452
END

```

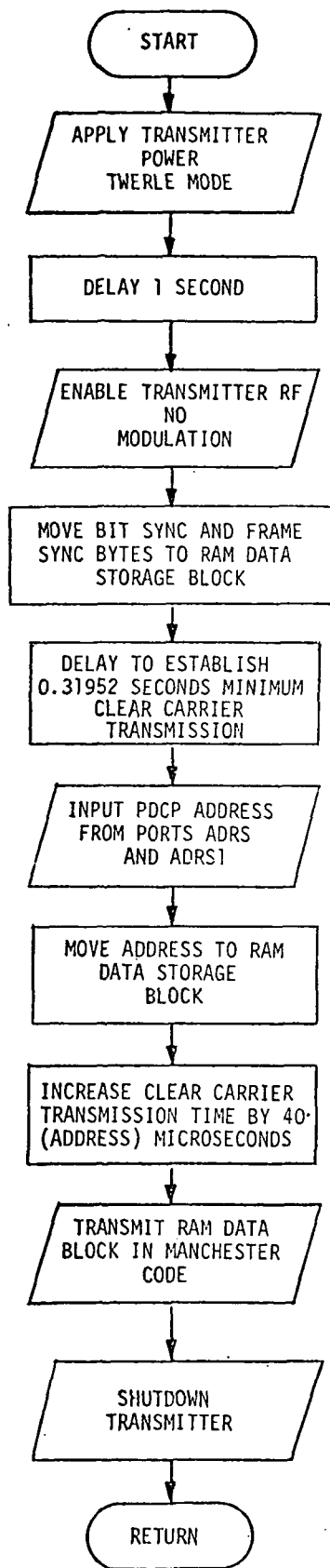


Figure 4.4.3.1(1) Simplified Flow Chart for Subroutine TWXMIT.

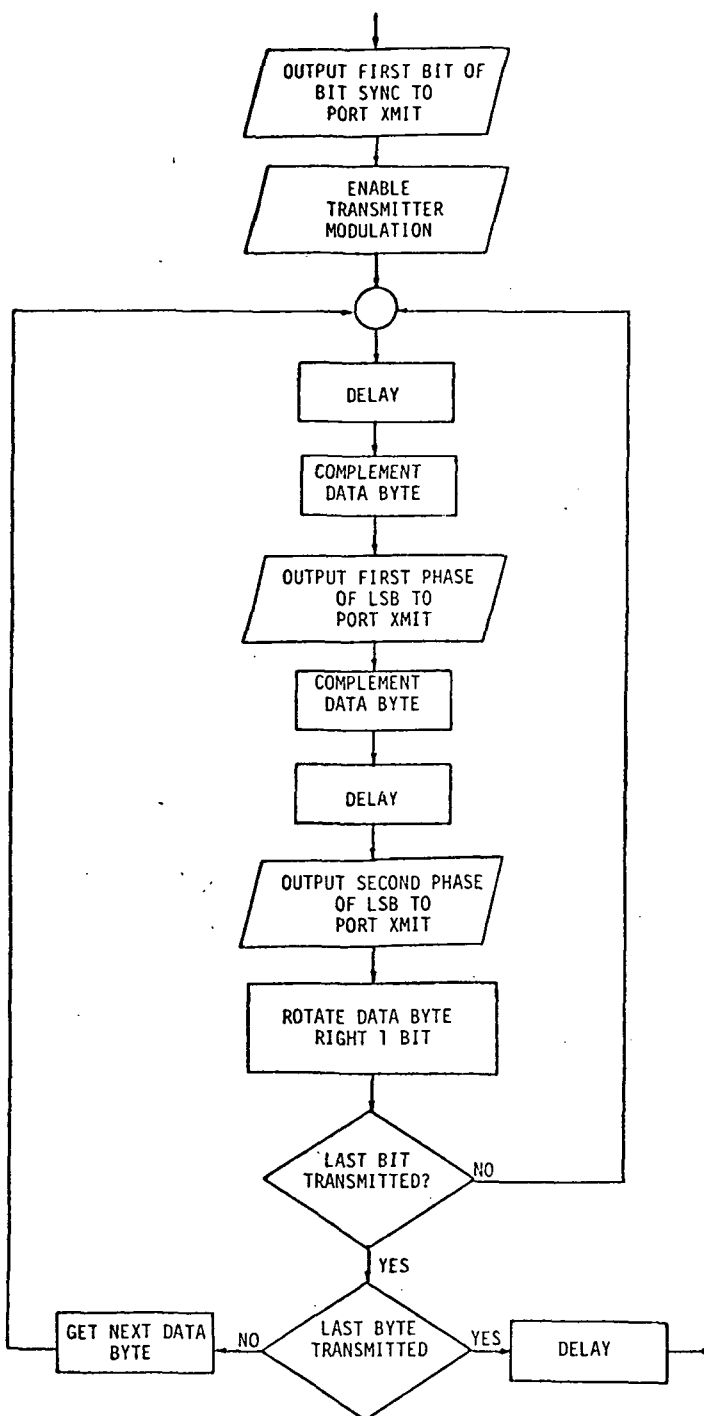


Figure 4.4.3.1(2) Detailed Flow Chart for the Data Transmission Block of Subroutine TWXMIT.

Each TWERLE DCP is assigned a unique ten-bit address code which is normally plug-wired [7]. The TWXMIT subroutine assumes a similar arrangement in which the PDCP address is plug-wired on two bits of input port ADRS1 and eight bits of input port ADRS. These input ports are sampled during the clear carrier portion of each transmission, and the plug-wired address is temporarily stored in the RAM data block. The bit and frame synchronization codes are also moved to the RAM data storage block temporarily so that the data can be transmitted from sequential memory locations. Table 4.4.3.1(2) illustrates the format of the RAM data block.

TABLE 4.4.3.1(2)
FORMAT OF THE TWERLE RAM
DATA BLOCK

Memory Address	Data Byte							
	MSB							LSB
TWD-4	1	0	1	0	1	0	1	0
TWD-3	0	1	1	0	1	0	1	1
TWD-2	X ₃	X ₂	X ₁	X ₀	0	0	0	0
TWD-1	A ₉	A ₈	X ₉	X ₈	X ₇	X ₆	X ₅	X ₄
TWD	A ₇	A ₆	A ₅	A ₄	A ₃	A ₂	A ₁	A ₀
TWD+1	T ₇	T ₆	T ₅	T ₄	T ₃	T ₂	T ₁	T ₀
TWD+2	P ₇	P ₆	P ₅	P ₄	P ₃	P ₂	P ₁	P ₀
TWD+3	Pt ₇	Pt ₆	Pt ₅	Pt ₄	Pt ₃	Pt ₂	Pt ₁	Pt ₀

KEY: X - TWERLE PDCP Address Plug-Wired to Ports ADRS and ADRS+1

A - Altimeter Data

T - Air Temperature Data

P - Pressure Data

Pt - Pressure Temperature Data

TWERLE DCP's transmit to the RAM's system aboard the Nimbus-F satellite. This system relies on time and frequency division multiplexing of the data from the various DCP's to prevent transmission interference [8]. In present TWERLE DCP designs, time-division multiplexing depends upon establishing a random variation in the length of the clear carrier transmission from each DCP. An average clear carrier transmission time of 0.34 seconds with nominal range of 0.32 to 0.36 seconds is desired. Currently, an imprecise one-shot circuit which must be manually adjusted for proper timing is used to control the clear carrier transmission time [7] and prevent two systems from interfering with each other for an excessive period. The TWXMIT subroutine listed as Program 4.4.3.1(1) makes a substantial improvement over this system. A software delay routine establishes a minimum clear carrier transmission time of 0.31952 seconds which may be increased to a maximum of 0.36048 seconds according to the formula

$$T = 319.52 + 40A,$$

where T = clear carrier transmission time in microseconds, and A = decimal equivalent to PDCP TWERLE address. Since each TWERLE PDCP is assigned a unique address, this technique guarantees that two systems will not remain in time sequence and interfere with each other. In addition, the transmission interval for each TWERLE PDCP can be computed from the address assigned to that system. Any variation from the assigned transmission interval would provide an early warning of possible system malfunction. Also, the need for manual adjustment of the transmission interval is eliminated.

The TWERLE format transmitter routine uses a general-purpose, long-delay routine which is 11 bytes long. Only 123 additional bytes of ROM are required to store the program. At present 100-piece CMOS 512 byte ROM prices, the incremental component cost of providing the TWXMIT subroutine as part of a standard PDCP ROM is approximately \$9.85. In addition, this subroutine utilizes 12 bytes of RAM for data and stack storage, 10 input bits to specify the platform address, and 8 output bits to provide data and control to the transmitter.

4.4.3.2 GOES Format Transmitter Routine - Specifications for the GOES data transmission sequence [6] are presented in Table 4.4.3.2(1). Program 4.4.3.2(1) is an assembly language listing of the GOES format transmitter routine written for the UT PDCP. This subroutine provides all necessary data formatting and encoding as well as complete transmitter control. In contrast to the TWERLE format transmitter subroutine, Program 4.4.3.2(1) assumes that the unique 31-bit address code assigned to each GOES PDCP is either stored in PROM or plug-wired to a memory-oriented input port. Both ASCII and Manchester data encoding are accomplished in real time during transmission to minimize data storage requirements.

A flow chart for the basic GOES format transmitter subroutine is illustrated in Figure 4.4.3.2(1), and a flow chart for the BIPHS subroutine called by the GOES subroutine is shown in Figure 4.4.3.2(2). Since the amount of GOES data transmitted is variable, the first byte of the RAM data storage block is used to indicate the current number of data bytes. Twelve additional bytes of RAM are required for stack operations. The GOES transmitter program, including subroutine BIPHS and the table area required to store preamble data, occupies 225 bytes of ROM. This program also requires the same 11-byte, general-purpose long-delay subroutine used by the TWERLE format transmitter routine. Primarily because of the ASCII encoding requirement, the GOES format transmitter subroutine is significantly longer than either the TWERLE, TIROS-N, or Landsat transmitter programs. Nevertheless, the GOES transmitter routine would occupy only one-fourth of an 8K-bit ROM. The incremental component cost of providing this subroutine in a standard PDCP ROM is approximately \$18.02.

4.4.3.3 TIROS-N Format Transmitter Routine - Program 4.4.3.3(1) is an assembly language listing of the TIROS-N format transmitter subroutine written for the UT PDCP. This program is based on the preliminary TIROS-N DCP specifications [9] presented in Table 4.4.3.3(1). A flow chart for Program 4.4.3.3(1) is illustrated in Figure 4.4.3.3(1). The TIROS-N transmitter subroutine provides real-time Manchester encoding of the preamble and the 32 data bits. Data is transmitted at a rate of 400 bits per second using the standard transmitter control signals

TABLE 4.4.3.2(1)
GOES DATA TRANSMISSION SEQUENCE
SPECIFICATIONS

Transmission Interval: Selectable up to 24.75 hours in 0.25 hour increments

Transmission Rate: 100 bits per second

Coding: Preamble - Manchester encoded with a 0 → 1 transition representing a 1
Data and EOT characters - ASCII encoded then Manchester encoded as above

Transmission Sequence:

1. Transmitter power-up followed by a 1-second warm-up delay
 2. Clear carrier transmission for a minimum of 5 seconds
 3. Preamble transmission
 - a. Bit synchronization clock (250 bits minimum)
 - b. Frame synchronization code (100010011010111) (15 bits)
 - c. Address code (assigned to user) (31 bits)
 4. Data transmission (up to 2000 bits)
 5. EOT code transmission - three ASCII end-of-transmission characters (33 bits)
 6. Transmitter power-down
-

001047	CMA						
001047	OUT	XMIT					
001050	DAD	H		DELAY 10 STATES			
001052	DAD	H		DELAY 10 STATES			
001053	DAD	H		DELAY 10 STATES			
001054	LDA	GDATA		FIND THE NUMBER OF DATA BYTES TO BE			
001055	MOV	C, A		TRANSMITTED AND STORE THE NUMBER			
001060	LXI	H, PAD-1		ESTABLISH HL AS AN ADDRESS REGISTER POINTING			
001061	MVI	D, 45					
001064	DLYS						
001066	INX	H		INCREMENT HL TO POINT TO NEXT DB MEMORY LOCATION			
001067	MVI	B, 10		B SPECIFIES THE NUMBER OF BITS TO BE TRANSMITTED			
001070	MOV	A, M		FROM THE CURRENT BYTE			
001072	PUSH	H		MOVE THE NEXT DB ---> ACCUMULATOR			
001073	MOV	H, M		DELAY 7 STATES			
001074	MVI	H, 197					
001075	DLY4						
001077	CALL	BIPH5		CALL BIPHASE ROUTINE WHICH CONTROLS			
001100	POP	H		TRANSMISSION OF THE CURRENT DATA WORD			
001103	MVI	A, 7		IN MANCHESTER CODE AT 100 BPS			
001104	DCR	D		IF THE FINAL PDCP GOES ADDRESS BYTE			
001106	JZ	GX3		HAS BEEN TRANSMITTED, GOTO GOES DATA			
001107	CMP	D		TRANSMISSION ROUTINE			
001112	JC	GX1		IF THE LAST FULL CLOCK BYTE HAS BEEN			
001113	JMP	GX2		TRANSMITTED, JUMP TO GX1 AND INCREMENT HL			
001116				OTHERWISE GOTO GX2 AND CONTINUE TRANSMITTING			
001116							

```

001121      MOV      A,M      ; CLOCK BITS
001121      LXI      H,DATA    ; DELAY 7 STATES AND THEN ESTABLISH HL AS
001122      LXI      H,DATA    ; AN ADDRESS REGISTER POINTING TO THE GOES
001122      O41      000      040
001125      INX      H        ; RAM DATA STORAGE AREA
001125      MOV      A,M      ; INCREMENT HL TO POINT TO THE NEXT DB MEMORY LOCATION
001126      RLC          ; MOVE THE NEXT DB --> ACCUMULATOR
001127      RLC          ; ROTATE THE DB LEFT ONE BIT
001130      MVI      D,2      ; D IS A COUNTER WHICH FLAGS WHEN BOTH NIBBLES OF
001130      MVI      B,10     ; THE CURRENT DB HAVE BEEN TRANSMITTED
001132      MVI      E,7      ; B SPECIFIES THE NUMBER OF BITS TO BE TRANSMITTED
001132      MVI      E,7      ; FROM THE CURRENT BYTE
001134      ANI      36       ; PREPARE TO CONVERT THE CURRENT NIBBLE TO ASCII. E IS
001134      ORI      140      ; THE PARITY FLAG
001136      JPO      GX10     ; CHECK PARITY AND DECREMENT E ONLY IF PARITY IS EVEN
001136      DCR      E        ; EVEN PARITY IS SENT--TO CHANGE TO ODD, CHANGE JPO
001140      DCR      E        ; INSTRUCTION TO JPE
001142      DLY5         ; INSTRUCTION TO JPE
001144      DLY5         ; INSTRUCTION TO JPE
001146      PUSH     H        ; SAVE HL ON STACK
001147      MVI      H,183   ; ESTABLISH LENGTH OF INITIAL BIPHASE
001150      DLY13        ; DELAY
001152      CALL    BIPHS   ; CALL BIPHASE SUBROUTINE TO TRANSMIT THE FIRST 8 BITS
001152      MVI      B,3     ; OF THE CURRENT NIBBLES ASCII CODE
001153      MOV      A,E     ; FROM THE CURRENT BYTE
001154      MVI      H,191   ; MOVE LAST 3 BITS OF CURRENT ASCII WORD TO ACCUMULATOR
001155      CALL    BIPHS   ; CALL BIPHASE TO TRANSMIT THE REMAINING 3 BITS
001160      POP      H        ; OF THE CURRENT NIBBLE'S ASCII CODE
001160      DCR      D        ; RESTORE HL FROM THE STACK
001162      DCR      D        ; IF BOTH NIBBLES OF THE CURRENT BYTE HAVE BEEN
001163      DCR      D        ; TRANSMITTED, RESTORE HL FROM THE STACK
001165      DCR      D        ; IF BOTH NIBBLES OF THE CURRENT BYTE HAVE BEEN
001165      DCR      D        ; TRANSMITTED, RESTORE HL FROM THE STACK
001170      DCR      D        ; IF BOTH NIBBLES OF THE CURRENT BYTE HAVE BEEN
001170      DCR      D        ; TRANSMITTED, RESTORE HL FROM THE STACK
001171      DCR      D        ; IF BOTH NIBBLES OF THE CURRENT BYTE HAVE BEEN
001171      DCR      D        ; TRANSMITTED, RESTORE HL FROM THE STACK

```

001172	JZ	210	002	GX7	; CONVERTED TO ASCII, JUMP TO GX7
001172	MOV	210	002	A,M	; IF NOT, MOVE THE ORIGINAL BYTE INTO ACCUMULATOR
001175	RRC	176			; AND ROTATE RIGHT THREE TIMES
001176	RRC	017			
001177	RRC	017			
001200	RRC	017			
001201	DLY16				
001201		000			
001203		000			
001205	JMP	132	002	GX5	; RETURN TO GX5 TO TRANSMIT ASCII CODE OF SECOND NIBBLE
001210	DLY5				
001210		177			
001211	DCR	015		C	; DECREMENT THE DATA BYTE COUNTER
001211		015			
001212	JNZ	302	002	GX4	; JUMP TO GX4 TO TRANSMIT THE NEXT DATA BYTE
001212		302	002		
001215	MVI	016	003	C,3	; IF C IS NOT ZERO
001215		016	003		; ELSE RESET C TO 3
001217	MVI	076	010	A,10	; 3 ----> ACCUMULATOR
001217	MOV	107		B,A	; 3 ----> B
001221	LXI	041	000	H,48128	; DELAY 3 STATES AND ESTABLISH LENGTH OF INITIAL
001222		041	000		
001225	CALL	315	316	BIPH3	; BIPHASE DELAY BY LOADING 188. TO H
001225		315	316		; CALL BIPHASE TO TRANSMIT THE FIRST 3 BITS OF
001230	MVI	076	006	A,6	; AN END OF TRANSMISSION (EOT) CHARACTER
001230		076	006		
001232	MVI	006	003	B,3	
001232		006	003		
001234	MVI	046	277	H,191	; ESTABLISH LENGTH OF INITIAL BIPHASE DELAY
001234	DLY3				
001236		000	000		
001236		000	000		
001240	CALL	315	316	BIPH3	; CALL BIPHASE TO TRANSMIT THE REMAINING 3
001240		315	316		
001243	MOV	146		H,M	; BITS OF AN 'EOT' CHARACTER
001243		146			; DELAY 7 STATES
001244	DAD	051		H	; DELAY 10 STATES
001244		051			
001245	DAD	051		H	; DELAY 10 STATES
001245		051			
001246	DAD	051		H	; DELAY 10 STATES
001246		051			
001247	DCR	015		C	; IF THE LAST (3RD) 'EOT' CHARACTER HAS NOT BEEN
001247		015			
001250	JNZ	302	217	GX8	; TRANSMITTED, JUMP TO GX8 AND TRANSMIT THE NEXT CHAR.
001250		302	217		

Program 4.4.3.2(1) (continued)


```

; THIS SUBROUTINE CONTROLS TRANSMISSION OF THE BYTE CURRENTLY ;
; STORED IN THE ACCUMULATOR. THE DATA IS MANCHESTER ENCODED ;
; AS IT IS TRANSMITTED ;
;
001313          DLY4
001313          000
001314          MVI          H,194.
001314          046          302
001316          BIFHS:
001316          343          DLY36
001320          B11:      DCR          H          ; DELAY TO CREATE 100 BPS TRANSMISSION RATE
001320          045
001321          302          002          JNZ          B11
001324          303          002          JMP          B1OUT
001327          B1OUT:   CMA          ; COMPLEMENT THE DATA BYTE
001327          057
001330          323          000          OUT          XMIT          ; TRANSMIT THE 1ST PHASE OF THE CURRENT LSB
001332          057          CMA          ; COMPLEMENT THE DATA BYTE AGAIN
001333          041          171          LXI          H,121.
001336          365          PUSH          PSW
001337          315          002          CALL         DELAY          ; DELAY TO CREATE 100 BPS TRANSMISSION RATE
001342          361          POP          PSW
001343          323          000          OUT          XMIT          ; TRANSMIT THE 2ND PHASE OF THE CURRENT LSB
001345          017          RRC          ; ROTATE THE DB RIGHT 1 BIT POSITION
001346          005          DCR          B          ; IF BIT '0' OF THE CURRENT DB HAS NOT BEEN
001347          302          313          002          JNZ          B12          ; TRANSMITTED, JUMP TO B12 AND INSERT DELAY
001352          311          RET          ; OTHERWISE RETURN TO THE MAIN GXMIT PROGRAM
001353          END

```

Program 4.4.3.2(1) (continued)

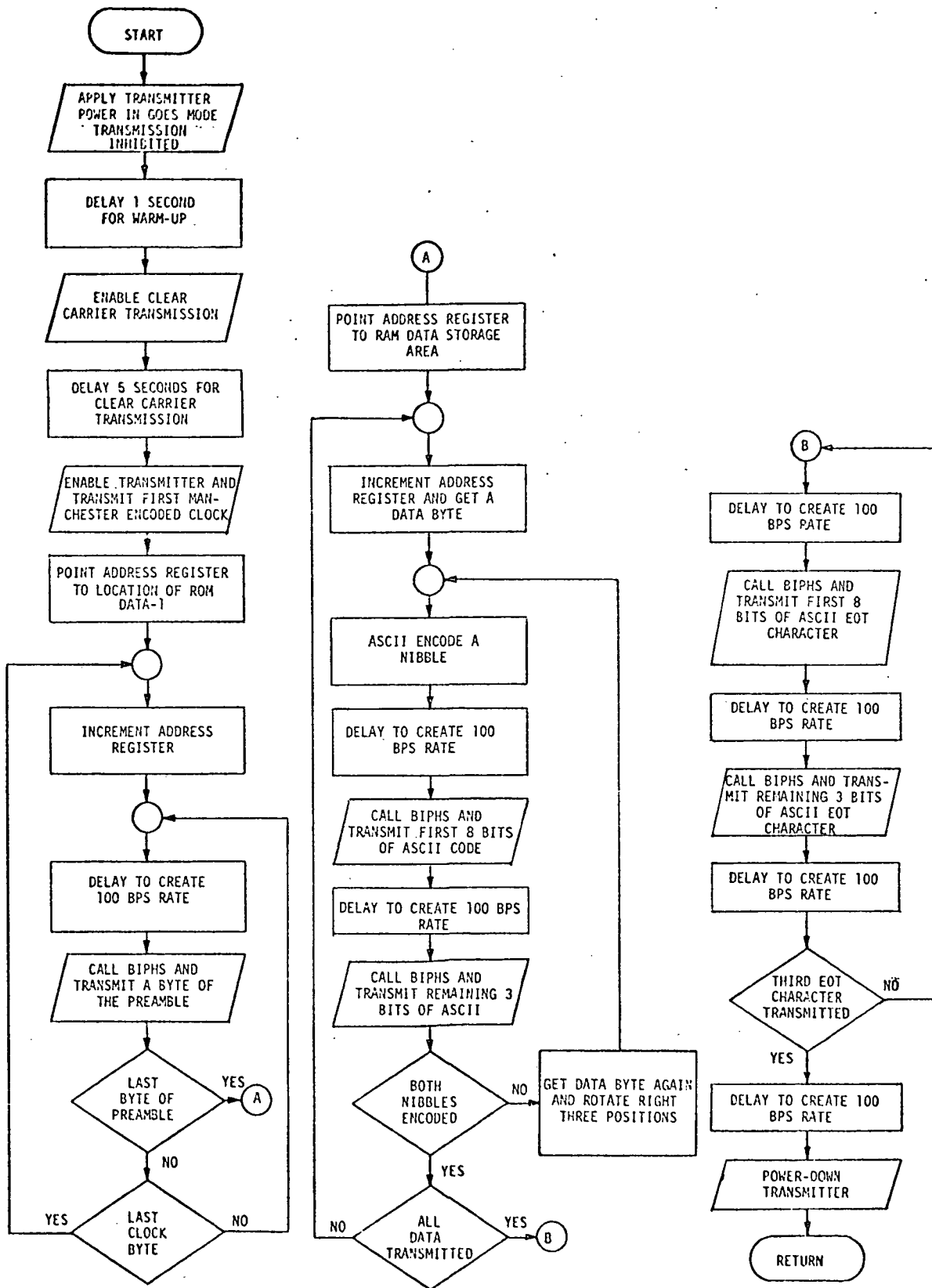


Figure 4.4.3.2(1) Flow Chart for GOES Transmitter Subroutine.

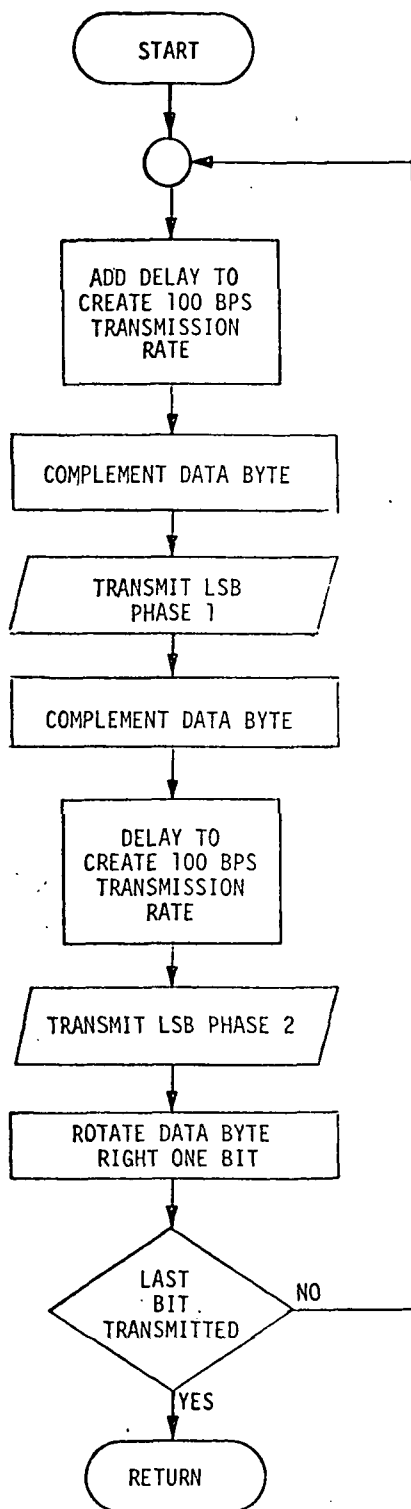


Figure 4.4.3.2(2) Flow Chart for Subroutine BIPHS.

TIROS-N FORMAT TRANSMITTER SUBROUTINE
 ; THIS PROGRAM CONTROLS THE TRANSMISSION OF 32 BITS OF MANCHESTER ;
 ; ENCODED DATA IN TIROS-N FORMAT USING AN 8080A MICROPROCESSOR ;
 ; WITH A 1.66666 MICROSECOND STATE TIME. ALL TIMING FUNCTIONS ;
 ; ARE ACCOMPLISHED IN SOFTWARE.

DATA CONSTANT DEFINITIONS:

```

020000 TND=20000 ; TIROS-N DATA BLOCK AT 4K POINT
000000 XMIT=0 ; OUTPUT PORT ASSIGNMENTS
000001 XCNR=1
000304 TTPO=304 ; TIROS TRANSMITTER POWER ON CONSTANT
000344 TCC=344 ; TIROS CLEAR CARRIER CONSTANT
000340 TDT=340 ; TIROS DATA TRANSMISSION CONSTANT
    
```

```

000000 ORG 3000 ; START PROGRAM AT 1.5K POINT
    
```

TNXMIT: ; STARTING POINT OF TIROS TRANSMITTER ROUTINE.

```

003000 MVI A,TTPO ; APPLY TRANSMITTER POWER IN TIROS-N MODE
003000 OUT XCNR ; WITH TRANSMISSION INHIBITED
003002 LXI H,24997H ; PROVIDE ONE-SECOND WARM-UP DELAY
003004 CALL DELAY
003007 DLY10
003012 MVI A,TCC ; ENABLE TIROS CLEAR CARRIER TRANSMISSION
003014 OUT XCNR ; DO IT
003016 LXI H,3297H ; PROVIDE 160 MILLISECOND DELAY
003020 CALL DELAY ; FOLLOWING THIS DELAY, THE FIRST BIT
003022 MVI A,TDT ; OF THE BIT SYNCHRONIZATION PATTERN
003024 OUT XMIT ; IN MANCHESTER CODE WITH A
003026 ; ONE REPRESENTED BY A 0-1 TRANSITION
003030 OUT XCNR ; ENABLE TIROS DATA TRANSMISSION MODE
003032
003032
    
```

003034	MVI	A, 60		TO TRANSMIT THE FIRST PHASE OF BIT 1
003034	DCR	A		INSERT DELAY TO CREATE 400 BPS
003036	JNZ	TNO		TRANSMISSION RATE
003037	DLY9		006	
003042	CMA			
003043	OUT	XMIT		TRANSMIT SECOND PHASE OF BIT 1
003044	LXI	B, 3412		7 → B ; 12 → C
003045	LXI	H, PADATA		
003047	MOV	A, M		
003052	MVI	E, 44		
003055	DLY8			
003056	JMP	TN2		
003060	DLY10			THIS SUBROUTINE CONTROLS
003062	INX	H		TRANSMISSION OF THE REMAINING 14-BITS
003065	MVI	B, 10		OF THE BIT SYNC PATTERN, THE 2-BIT FRAME
003070	MOV	A, M		SYNC PATTERN AND THE 24-BIT ADDRESS
003072	MOV	E, 40		CODE FROM THE 6 ROM LOCATIONS BEGINNING
003073	DLY8			AT ADDRESS PADATA. AFTERWARDS THE FOUR
003075	DLY4			8-BIT DATA BYTES ARE TRANSMITTED FROM RAM
003077	DCR	E		BEGINNING AT ADDRESS TNDATA.
003100	JNZ	TN2		DELAY
003101	CMA			COMPLEMENT ACCUMULATOR
003104	OUT	XMIT		TRANSMIT THE FIRST PHASE
003105	CMA			OF THE CURRENT LSB
003107	MVI	E, 60		COMPLEMENT ACCUMULATOR
003110				DELAY
003110				

Program 4.4.3.3(1) (continued)


```

003167
003167
003167
003170
003170
003171
003171
003172
003172
003175
003175
003177
003177

053
175
264
302
167
006
000
000
311

DCX H
MOV A,L
ORA H
JNZ DELAY
DLYS
RET
    
```

```

003200
003200
003200
003201
003201
003202
003202

003203
003203
003204
003204
003205
003205

003206

FADATA:
DB 377 ; THE FIRST THREE BYTES OF THIS TABLE
DB 177 ; SPECIFY THE LAST 14 BITS OF BIT SYNC
DB 364 ; AND 2 BITS OF FRAME SYNC

DB 001 ; THE REMAINING THREE BYTES SPECIFY
DB 000 ; THE 24 BIT FDCP TIRDS-N ADDRESS CODE
DB 000 ; (LSB TO MSB) -- THE ADDRESS SHOWN HERE IS 1

END
    
```

TABLE 4.4.3.3(1)

TIROS-N DATA TRANSMISSION
SEQUENCE SPECIFICATIONS

Transmission Interval: Variable 40, 60, or 80 seconds

Transmission Rate: 400 bits per second

Coding: Manchester encoded with a 0 → 1 transition representing a 1

Transmission Sequence:

1. Transmitter power-up followed by a 1 second warm-up delay
 2. Clear carrier transmission for 160 ± 2.5 milliseconds
 3. Preamble transmission
 - a. Bit synchronization clock (15 bits)
 - b. Frame synchronization code (000101111) (9 bits)
 - c. Address code (assigned to user) (24 bits)
 4. Data transmission - four bytes (32 bits)
 5. Transmitter power-down
-

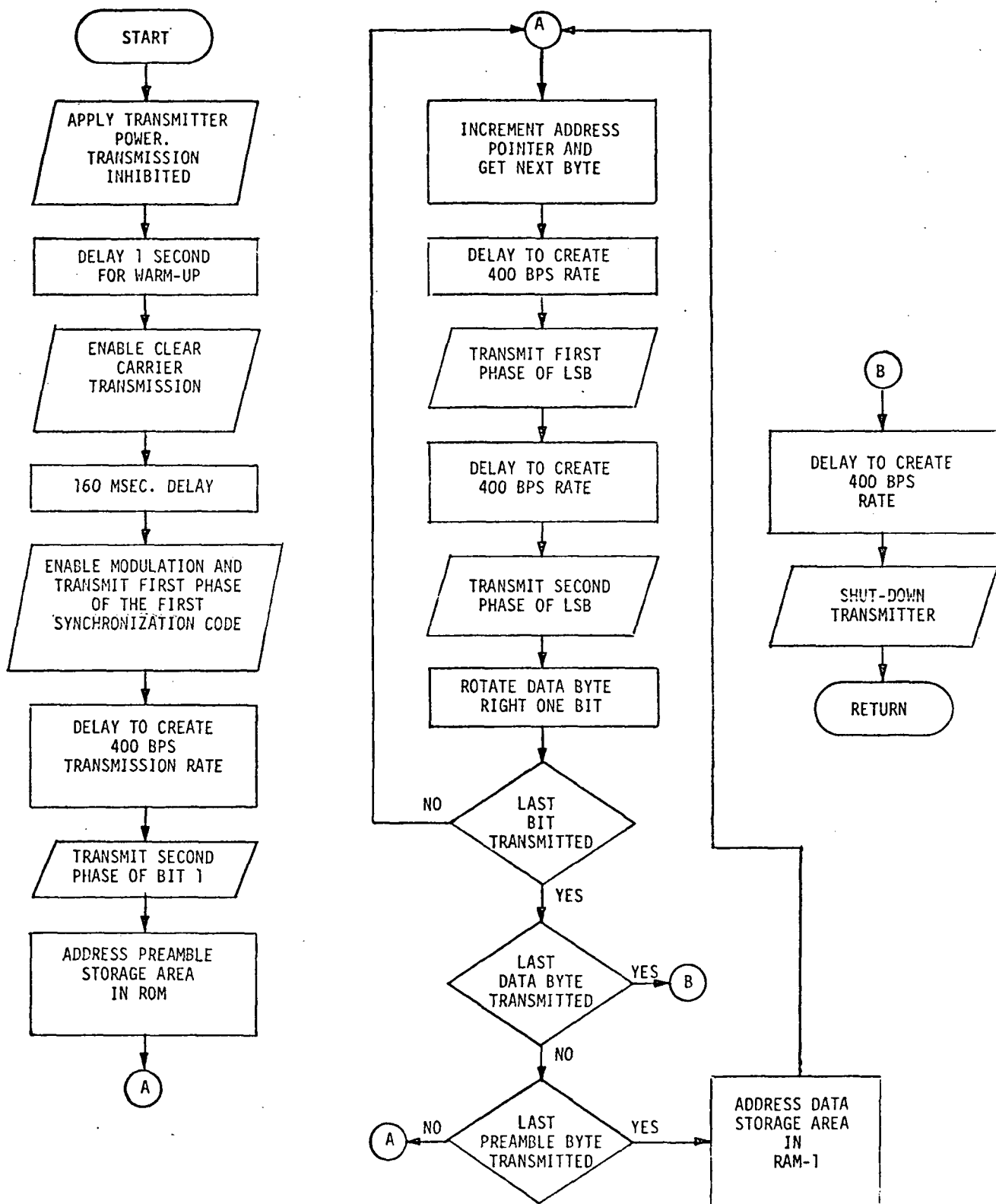


Figure 4.4.3.3(1) Flow Chart for TIROS-N Transmitter Subroutine.

defined in Section 4.4.3. The TIROS-N transmitter program utilizes eight bytes of RAM for data storage and stack operations and 124 bytes of ROM for program and preamble storage. In addition, this program also uses the standard long-delay subroutine.

4.4.3.4 Landsat Format Transmitter Subroutine - Specifications for the Landsat data transmission sequence [6] are presented in Table 4.4.3.4(1). The Landsat data transmission rate of 5000 bits per second is significantly higher than the data transmission rates used in the TWERLE, GOES, and TIROS-N data collection systems. Because of this high data transmission rate, convolutional encoding of Landsat data cannot be accomplished in real time during transmission from the UT PDCP system. Instead, a separate subroutine [Program 4.4.3.4(1)] is used to convolutionally encode the data prior to transmission. A flow chart for the PDCP convolutional encoder subroutine is illustrated in Figure 4.4.3.4(1). This subroutine is called by the executive program prior to each data transmission. The convolutional encoding process creates two code bits for each original bit of data but provides error detection and correction capabilities.

Manchester encoding, transmitter control, and actual Landsat data transmission are provided by Program 4.4.3.4(2). A flow chart for the Landsat format transmitter subroutine is shown in Figure 4.4.3.4(2). Together, Program 4.4.3.4(1) and Program 4.4.3.4(2) require 188 bytes of ROM for program storage and 31 bytes of RAM for data storage and stack operations.

4.4.4 Special Computations

Remote processing of data is a logical extension of the capabilities of the microprocessor-based PDCP. The question arises, however, as to what sort of numerical techniques are practical given the present time and memory restrictions of a PDCP system. In an effort to answer this question, the Fast Fourier Transform (FFT) was chosen for implementation. It is a sophisticated and powerful tool in numerical analysis, allowing the decomposition of time signals into their frequency components. It has been proposed, for example, that seismic events may be detected by

TABLE 4.4.3.4(1)
LANDSAT DATA TRANSMISSION SEQUENCE
SPECIFICATIONS

Transmission Interval: 90 or 180 seconds

Transmission Rate: 5000 bits per second

Coding: Convolutional encoding, rate 1/2,
constraint length 5

Manchester encoded, 1 → 0 transition
representing a 1

Transmission Sequence:

1. Transmitter and platform power-up for 50 milliseconds
 2. Turn on RF power simultaneous with first serial data bit
 3. Data transmission

a. Message preamble (0000000000000000)	(15 zeroes)
b. Platform address (assigned to user)	(12 bits)
c. Sensor data 1-8 eight-bit words	(8-64 bits)
d. Runout bits (0000)	<u>(4 zeroes)</u>
Total	39-95 bits
 4. Transmitter and platform power-down
-

NOTE: The 39-95 bits of Landsat message are before convolutional and Manchester encoding. The actual transmitted message is 4 x (39 to 95) bits.

```

; PDCP CONVOLUTIONAL ENCODER ROUTINE
;
; THIS PROGRAM PERFORMS CONVOLUTIONAL ENCODING OF CONSTRAINT
; LENGTH 5 ON A BLOCK OF DATA LOCATED AT THE DATA STARTING
; ADDRESS. THE ENCODING IS SUITABLE FOR ENCODING ERFS
; FORMAT TRANSMITTER DATA. FIRST BYTE OF DATA BLOCK
; IS ASSUMED TO CAONTAIN THE NUMBER OF BYTES TO
; BE CONVOLVED. STACK USEAGE = 6 BYTES.
;
; CONSTANT DEFINITIONS USED IN THIS PROGRAM ARE:

```

```

$1MSK = ^B00010111 ; EXCLUSIVE OR MASK CONSTANTS
$2MSK = ^B00011001

```

```

DATA = ^D4000 ; DATA BLOCK STARTS AT 2K POINT IN RAM

```

```

ORG 1000 ; START PROGRAM AT 1/2 K POINT
CONVOLV: ; STARTING POINT OF ROUTINE.

```

000000									
001000									
001000	021	000	010	LXI	D, DATA	;	POINT D AT DATA BLOCK ADDRESS		
001003	032			LDAX	D	;	#BYTES IN BLOCK ----> A		
001004	107			MOV	B, A	;	#BYTES IN BLOCK ----> B		
001005	257			XRA	A	;	CLEAR ACCUMULATOR		
001006	147			MOV	H, A	;	CLEAR H REGISTER		
001007				INX	D	;	LOAD NEXT THREE BYTES WITH ZERO		
001007	023			STAX	D	;			
001010	022			INX	D	;			
001011	023			STAX	D	;			
001012	022			INX	D	;			
001013	023			STAX	D	;			
001014	022			INX	D	;	POINT DE AT LOCATION FOR HIGH ADDRESS OF PLATFORM		
001015	023			LDA	HPADR	;	LOAD ACC FROM ROM LOCATION OF HIGH ADDRESS		
001016	072	142	002	STAX	D	;	STORE HIGH ADDRESS OF PLATFORM IN <DE>		
001021	022			INX	D	;	POINT DE TO LOACTION FOR LOW ADDRESS		
001022	023			INX	D	;			
001023	023			INX	D	;			

001024	LDA	LPADR	, GET HIGH PART OF PLATFORM ADDRESS
001024	STAX	D	, STORE HIGH PART IN DATA BLOCK
001027	LXI	D, DATA+4	, POINT DE AT HIGH ADDRESS OF PLATFORM IN DATA BLOCK
001030	INX	D	, POINT DE TO NEXT DB TO BE CONVOLVED
001033	LDAX	D	, PUT NEXT DB IN ACCUMULATOR
001034	MOV	L, A	, A ----> L
001035	CALL	C4BITS	, CONVOLVE 4 BITS OF A DB INTO 8 BITS AND
001036	INX	D	, WRITE INTO LOCATION ADDRESSED BY DE.
001041	INX	D	, POINT DATA BLOCK POINTER AT NEXT LOCATION
001042	CALL	C4BITS	, NOW CONVOLVE REMAINING 4 BITS AND WRITE
001042	CALL		, INTO NEXT LOCATION ADDRESSED BY DE.
001045	DCR	B	, FINISHED CONVOLVING ENTIRE BYTE SO
001045	JNZ	CON1	, DECREMENT BYTE COUNT.
001046	LHLD	DATA	, NOT DONE DO ANOTHER BYTE.
001051	DAD	H	, COMPUTE NUMBER OF BITS TO BE BIPHASED
001051	DAD	H	, FROM NUMBER OF BYTES IN DATA BLOCK
001054	DAD	H	
001055	DAD	H	
001055	DAD	H	
001056	DAD	H	
001057	DAD	H	
001057	DAD	H	, MULTIPLY #BYTES BY 16 BY DOUBLE ADDING HL 4 TIMES
001060	LXI	D, 14	, PUT 14 DECIMAL IN DE AND THEN
001060	DAD	D	, ADD TO HL
001063	SHLD	DATA	, NUMBER OF BITS TO BE ENCODED INTO FIRST BYTE OF DATA BLOCK
001064	RET		, ALL DONE, RETURN TO EXECUTIVE
001064			
001067			
001067			

Program 4.4.3.4(1) (continued)

001132	CALL	NEXTB			CONVOLVE ANOTHER BIT
001132					
001135	CALL	NEXTB			CONVOLVE FINAL BIT FOR THIS ROUTINE
001135					
001140	STAX	D			SAVE CONVOLVED BITS IN PRESENT DE LOCATION
001140					
001141	RET				
001141					
001142	HPADR:	DB	10		HIGH FOR BITS OF PLATFORM ADDRESS WOULD BE
001142					STORED IN ROM HERE
001143	LPADR:	DB	00		LOW 8 BITS OF PLATFORM ADDRESS STORED HERE
001143					
001144	END				

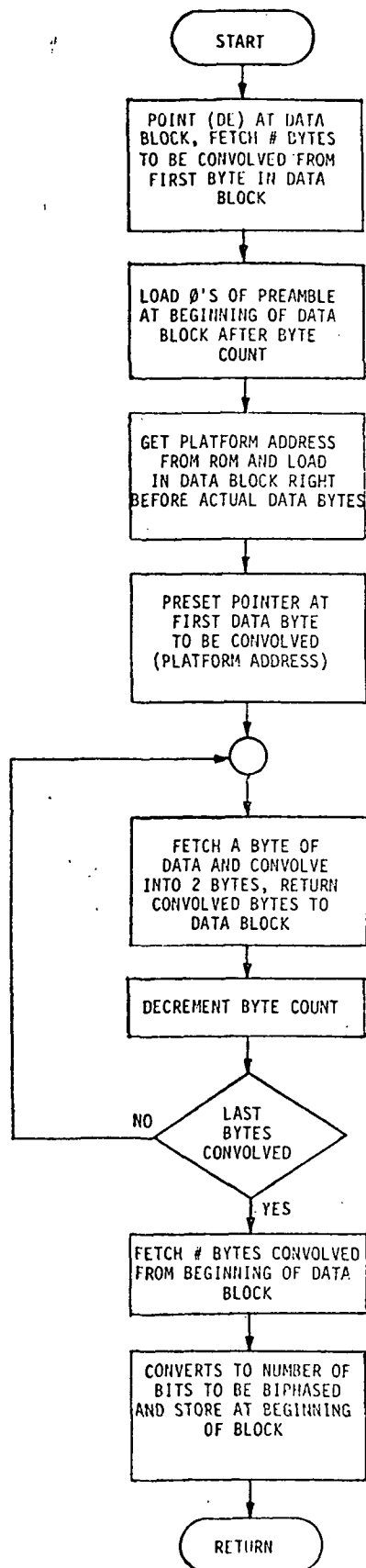


Figure 4.4.3.4(1) Flow Chart for the PDCP Convolutional Encoder Subroutine.


```

002066 AGI: CMA ; COMPLEMENT BIT TO SEND SECOND PHASE OF BIT
002066 OUT XMIT ; OUTPUT SECOND PHASE OF BIT
002067 CMA ; RESTORE BIT TO TRUE FORM
002071 RRC ; SHIFT BIT RIGHT ONE BIT POSITION
002072 DCR E ; DECREMENT BIT COUNT
002073 JZ NEXT ; AFTER 8 BITS GO GET NEXT DB
002074 DCR D ; DECREMENT TOTAL BIT COUNT (CHECK FOR NEXT
002077 DLY12 ; TO LAST BIT OUTPUTTED)
002100 ; MAKE LOOP = 60 MACHINE STATES
002100
002102
002103
002103
002106
002106
002110
002110
002112
002114
002114
002115
002115
002117
002117
002121
002121
002123
002125
002127
002130
002130
002132
002132
002133

000
004
004
000
060
000
177
343
000
040
177
301
000
001
311

AGI: CMA ; COMPLEMENT BIT TO SEND SECOND PHASE OF BIT
OUT XMIT ; OUTPUT SECOND PHASE OF BIT
CMA ; RESTORE BIT TO TRUE FORM
RRC ; SHIFT BIT RIGHT ONE BIT POSITION
DCR E ; DECREMENT BIT COUNT
JZ NEXT ; AFTER 8 BITS GO GET NEXT DB
DCR D ; DECREMENT TOTAL BIT COUNT (CHECK FOR NEXT
DLY12 ; TO LAST BIT OUTPUTTED)
; MAKE LOOP = 60 MACHINE STATES
JNZ AGAIN ; OUTPUT ANOTHER BIT IF NOT THROUGH
OUT XMIT ; OUTPUT LAST BIT, FIRST PHASE
DLY46 ; DELAY FOR BIT TIME OF 60 MACHINE STATES
CMA ; COMPLEMENT BIT TO SEND SECOND PHASE
OUT XMIT ; OUTPUT SECOND PHASE OF LAST BIT
SHUTDW: MVI A, PWRROFF ; LOAD ACCUMULATOR WITH SHUTDOWN CONTROL WORD.
DLY43 ; DELAY 30 BIT TIME IS 60 MACHINE STATES
OUT XCNTR ; SHUTDOWN PLATFORM POWER
RET ; ALL DONE, RETURN TO EXECUTIVE.
END

```

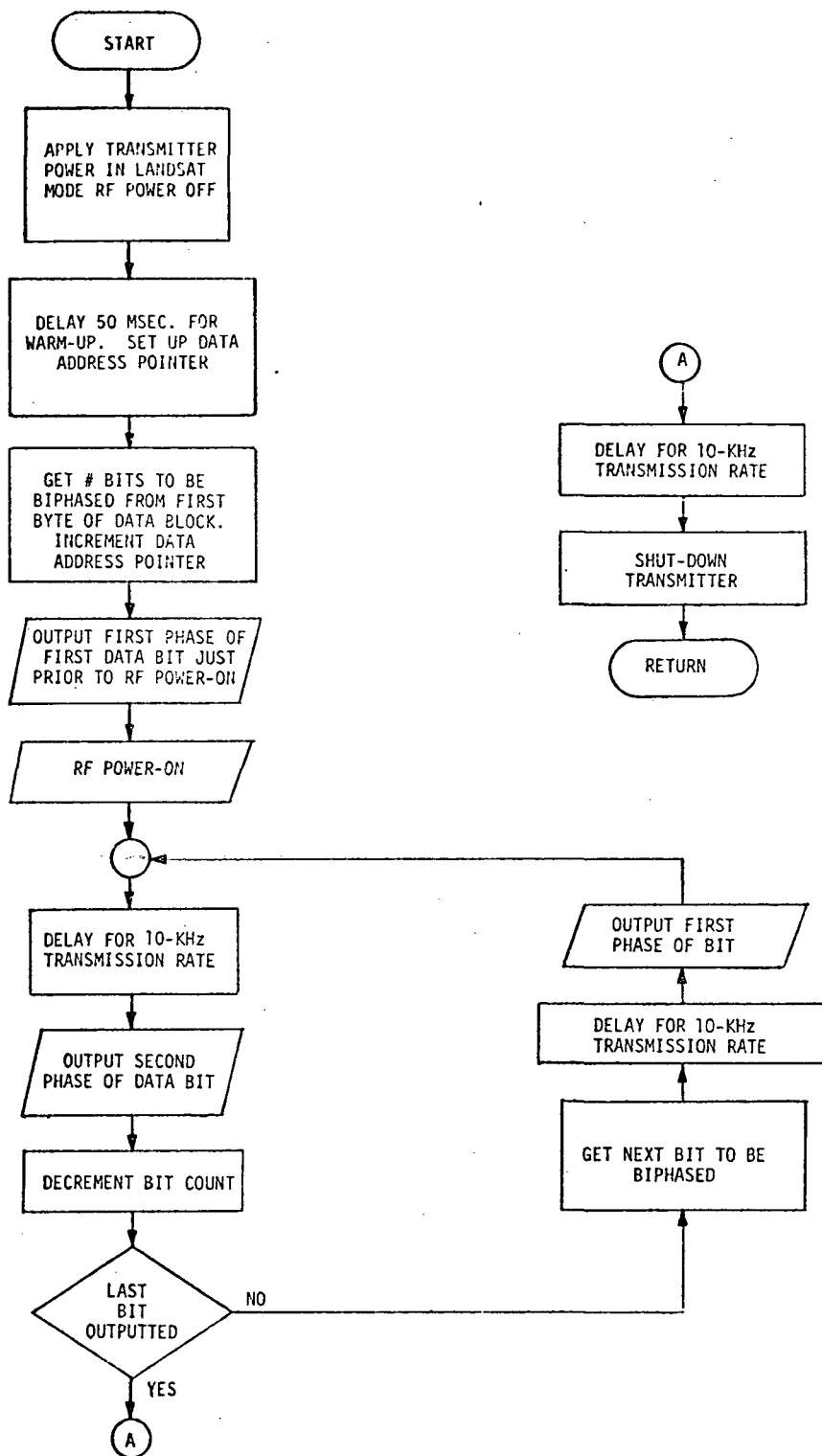


Figure 4.4.3.4(2) Flow Chart for the Landsat Transmitter Subroutine.

an analysis of their frequency components, and with this in mind a classifier based on the Bayes decision rule was investigated for its possible practical application to seismic event detection. Figure 4.4.4(1) shows how the FFT and Bayes classifier would work together in a detection scheme.

4.4.4.1 The FFT Program - The implementation of the fast Fourier transform (FFT) program was carefully studied. Because of limitations in execution time and memory size imposed by the PDCP, the commonly used forms of the algorithm were unacceptable. Most involve considerable manipulation of complex numbers, in particular, complex multiplication. Also, most existing FFT programs require the calculation of weighting factors, which are themselves complex and trigonometric in nature.

A method which avoids these time consuming problems was reported in a short paper by Dr. C. M. Rader and N. M. Brenner [9]. It outlines an approach whereby only pure imaginary weighting factors are required, thus decreasing the total number of multiplications necessary. In addition, the weighting factors fall in a convenient range of values, from $j0.5$ to $j5$ (for 64-point transforms or less).

The derivation of the Rader-Brenner form is outlined in Appendix B and need not be repeated here. However, it is helpful to see the flow of the process. Figure 4.4.4(2) outlines the sequence of events graphically for each data byte pair (complex data value) of a 16-point transform. First is the summation stage where c_n is calculated for the two-point, four-point, and eight-point cases. Recall from the derivation that $c_n = a_{2N+1} + a_{2N-1} + Q$, where $n = 0, 1, \dots, N/2-1$. Next, the data values are permuted so that the final output data will be in numerical and not bit-reverse order [10]. That is, cell one will contain the first Fourier coefficient, cell two will contain the second, and so on. Finally, the two-, four-, eight- and 16-point transforms are calculated.

As can be seen from the 8080A FFT program source listing, (Program C(1) in Appendix C) a FORTRAN program is used as the logical model for the assembly language program. There are several advantages to this approach. In the first place, FORTRAN offers a clear way to logically

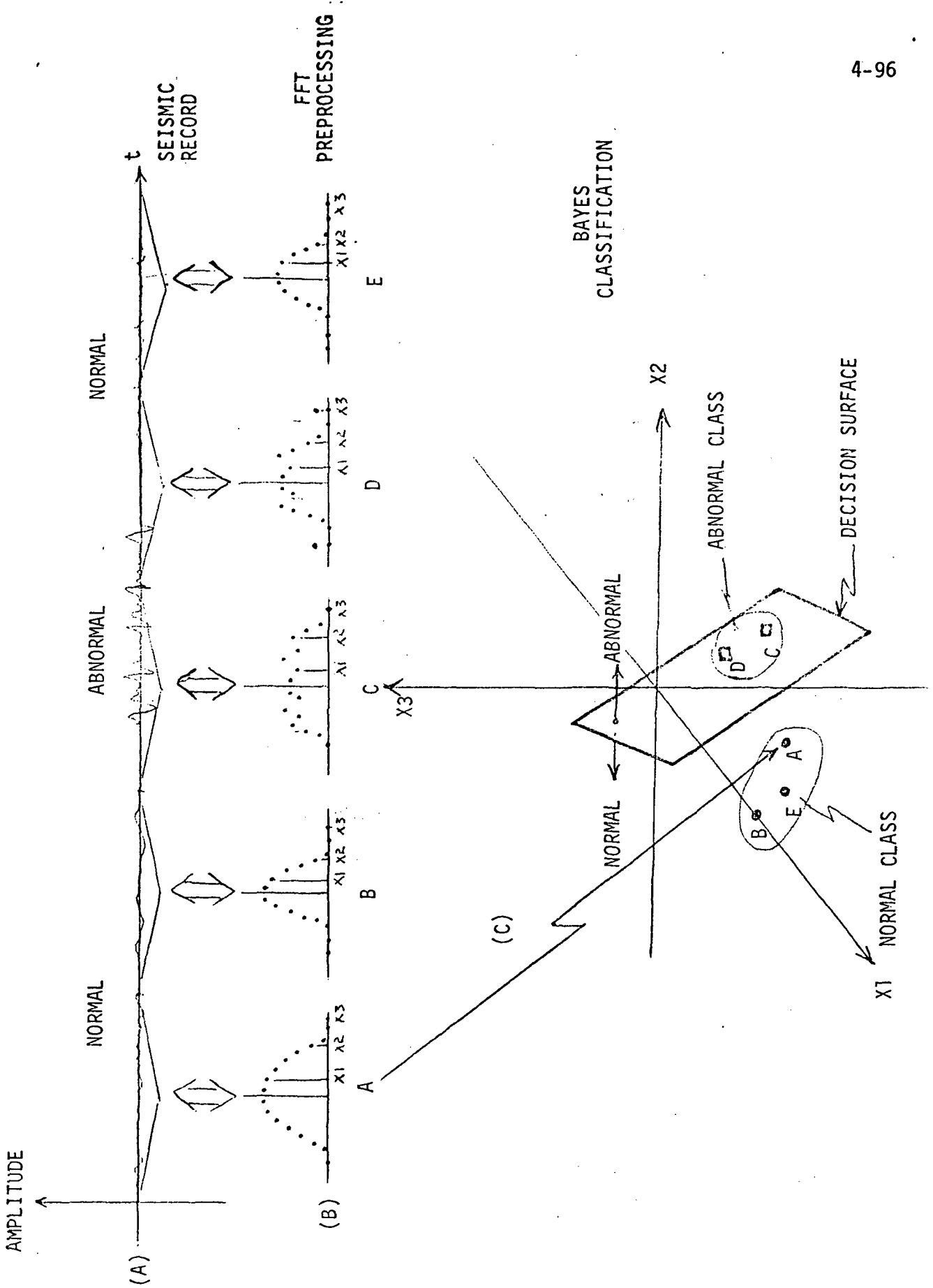


Figure 4.4.4(1) Seismic Record Preprocessing and Classification.

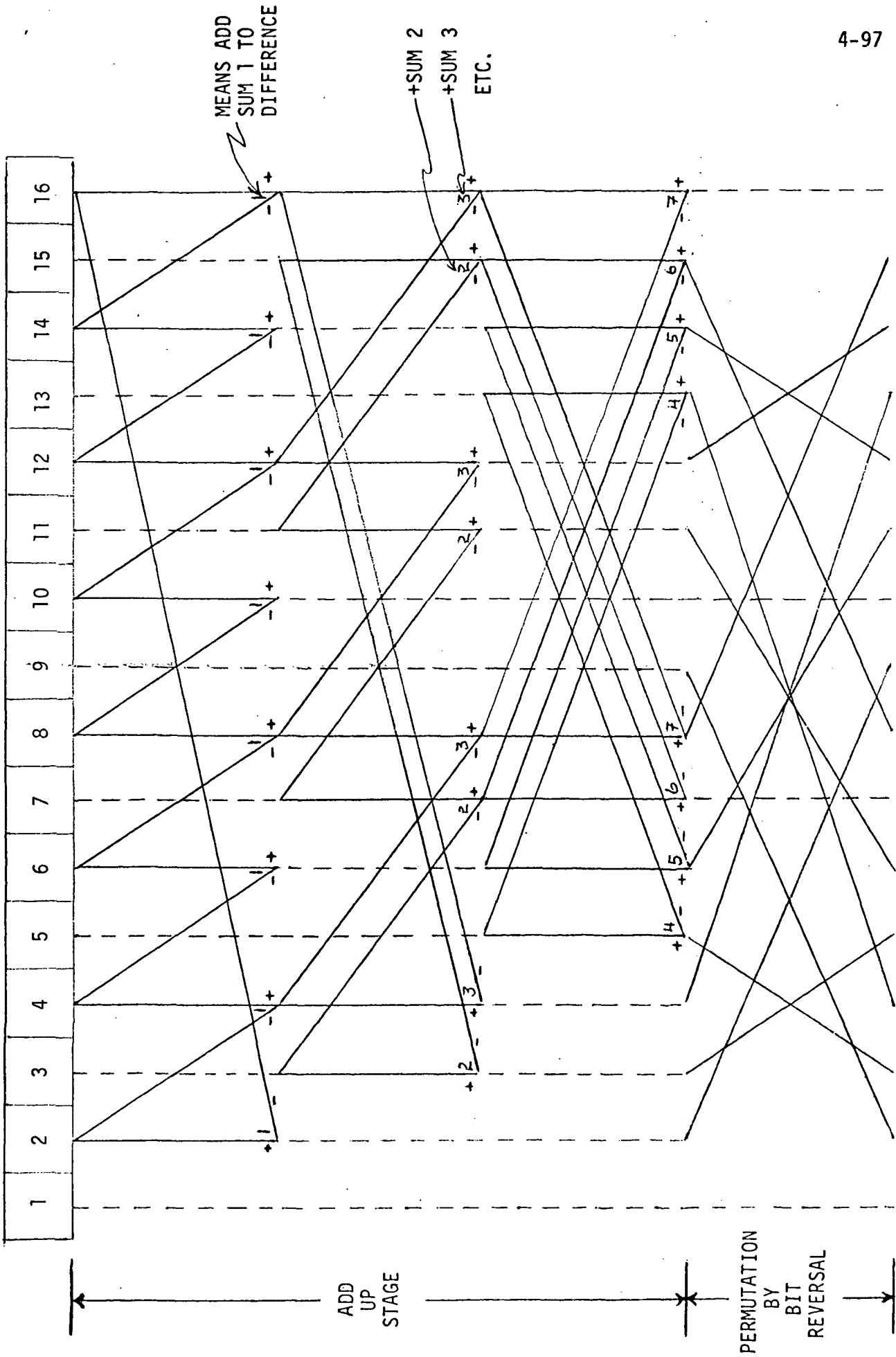


Figure 4.4.4(2) FFT Data Flow Chart

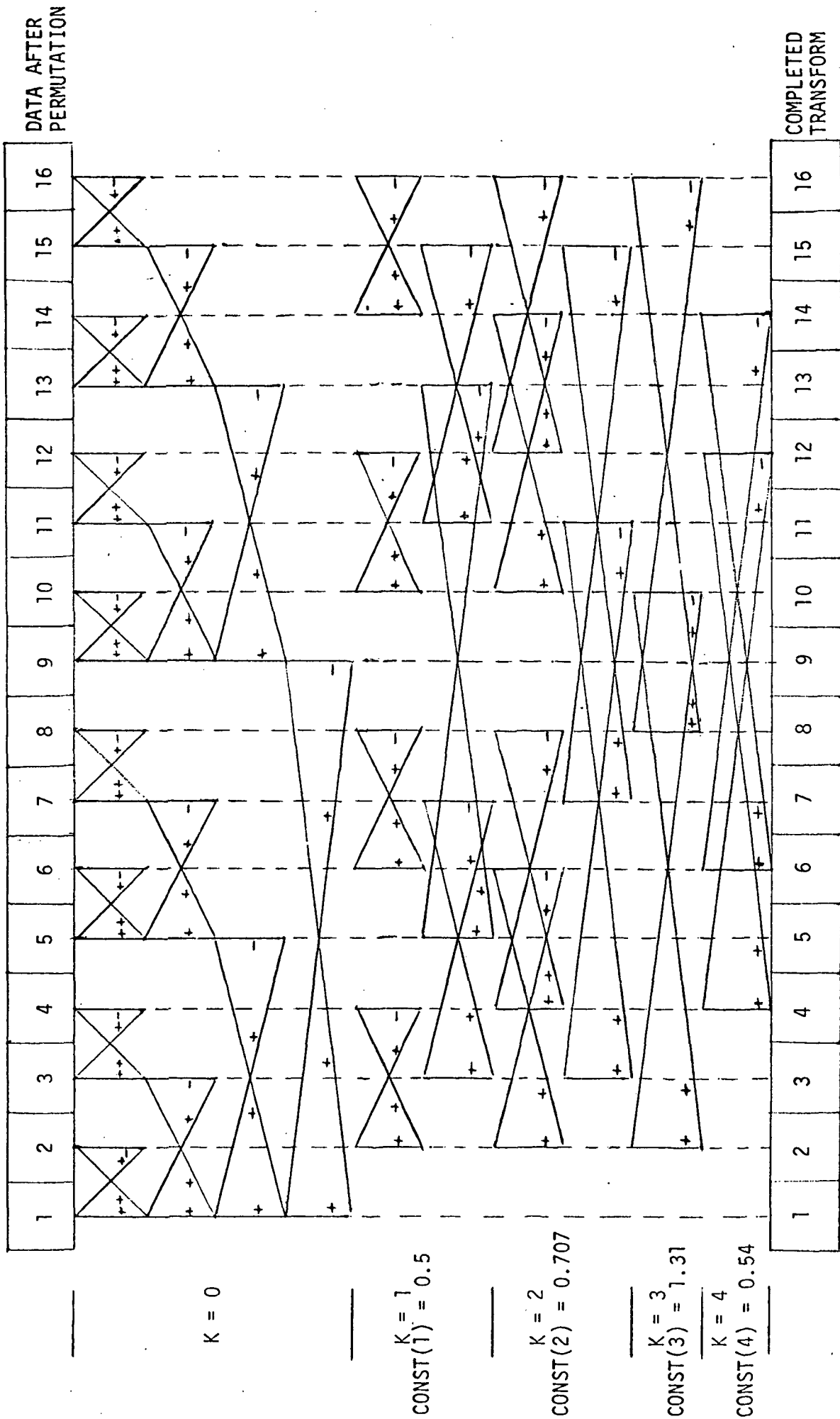


Figure 4.4.4(2) (continued)

* Multiply term by $(0,1) \cdot \text{CONST}(K)$.

break down the operations of the program. Secondly, a program written in FORTRAN (with certain hard-to-implement instructions like DO omitted) can be reduced to minimum complexity and debugged on a logical level before assembly language programming begins. Thus, one is assured of the integrity of the logic before proceeding to assembly language programming. The 8080A fast Fourier transform subroutine (FFT80) requires 1102 bytes of ROM.

After the FORTRAN framework is decided upon, compilation to assembly language can begin. However, assembly by a FORTRAN compiler would be quite unsatisfactory. Very great increases in efficiency and speed can be achieved by taking advantage of the architecture of the 8080A CPU. For example, instead of storing commonly used program variables in memory, which would require time consuming memory read/write operations, these values can be retained and operated upon using internal registers. It is, of course, occasionally necessary to free registers for other uses, which is achieved by pushing and popping the variables on the stack. The total time required for these operations is small compared to the time required for repeated memory accesses.

It would have been preferable to use a word length of 16 bits in this program as it allows greater resolution of incoming signals and removes the fear of overflow errors. This would mean complex data values would be 32 bits long. Unfortunately, the 8080A's double word (16 bit) instruction set is quite limited and very time consuming, while separately manipulating each byte of a double word is even more inefficient. The only reasonable approach then is to use 8-bit values, thus trading resolution for speed. In a few intermediate steps of the program it is necessary to expand to double words, such as in manipulation of the variable SUM in the add up stage of the program. SUM is a mean value which first contains a sum of values and then is divided by the number of values summed. Before division, SUM can easily grow to a value in excess of eight bits in length, so the word length is increased. After division, of course, eight bits is sufficient since an average cannot be larger than the largest of the averaged values.

The 8080A does not provide hardware multiplication, and conventional software multiplication is quite time consuming especially considering the non-integer factors involved. Thus, an alternate multiplication technique has been developed. From the development in Appendix B, it is seen that the weighting factors are pure imaginary and have the values:

$$w_k = 0.5 \operatorname{cosecant}(2\pi k/N)j, \quad k \neq 0, N/2 .$$

In Program C(1), $2k$ is defined $M1$ and N is defined $M2$. That is, $w_k = 0.5 \operatorname{cosecant}(\pi M1/M2)j$. The sequence $M1/M2$ is:

$$M1/M2 = m/2^n ,$$

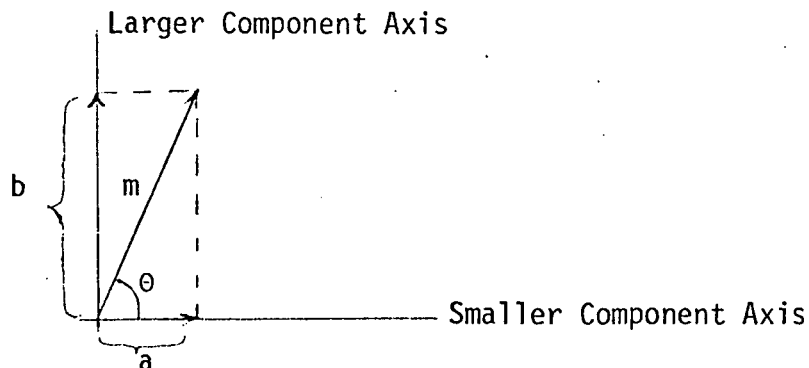
where $m = 1, 3, 5, \dots, 2^{n-1}-1$ for each $n = 1, 2, 3, \dots, (\log_2 N)-1$ (where $N =$ number of points of the transform). For a 64-point transform, $M1/M2$ is the sequence $1/2, 1/4, 1/8, 3/8, 1/16, 3/16, 5/16, 7/16, 1/32, 3/32, 5/32, 7/32, 9/32, 11/32, 13/32, 15/32$. A 16-point transform would require only the sequence $M1/M2 = 1/2, 1/4, 1/8, 3/8$. Table 4.4.4(1) delineates the ratios and consequent values of w_k .

The shifts listed in the table are used by the special multiplication subroutine MULCON (TEMP,K). MULCON multiplies by shifting the value in TEMP right or left a number of times and adding or subtracting the shifted value with the original value of TEMP. The number of shifts and the decision as to whether the resulting shifted value is to be added to or subtracted from TEMP is stored in a look-up table (called TABLE in the program). K is the index for TABLE. For example, suppose $K = 10_8$ when MULCON is called. $M1/M2 = 3/8$ and $w_k = 0.54$. The value in TEMP is to be multiplied by 0.54, thus $\text{TABLE}(10_8) = -1,+5$. TEMP is duplicated in TMPROD and shifting proceeds on TMPROD. The 1 value in TABLE indicates that TEMP is to be shifted right one time, yielding 0.5 TEMP , and the minus sign means that the shifted value is to be subtracted from TMPROD. Recalling that $\text{TMPROD} = \text{TEMP}$ originally, we now have $\text{TMPROD} = \text{TMPROD} - \text{TEMP}/2^1 = 0.5 \text{ TEMP}$. Next, TEMP is shifted right five times and the shifted value is added to TMPROD, which equals 0.5 TEMP from the last operation. Five

shifts yield 0.031 TEMP and $\text{TMPROD} = \text{TMPROD} + \text{TEMP}/2^5 = 0.5 \text{ TEMP} + 0.031 \text{ TEMP} = 0.53 \text{ TEMP}$. From Table 4.4.4(1) it can be seen that the exact answer is 0.54; the error is less than 0.01 TEMP . This is usually less than round-off error when working with integer arithmetic.

Upon completion of the FFT portion of the program, the array DATA, which initially held the time domain data, now contains the Fourier series coefficients. These transform values, however, are complex and for the purposes of Bayes classification, only magnitudes are of interest. The normal procedure for finding the magnitude of some value $(a+jb)$ is to take the square root of the sum of the squares of the coefficients, that is, $\sqrt{a^2 + b^2}$. To speed up this computation for a microprocessor application, a method is used which does not require the squaring and square-root routines; only the division routine is required. This method is described in the following paragraph.

A complex number can be visualized as a vector of magnitude m having two orthogonal components. Since m is always positive, the signs of these components are irrelevant; only the absolute values need be considered. The absolute value of the smaller component is defined here as a and the larger as b , so m is graphed.



It is clear that for these conditions, $45^\circ \leq \theta \leq 90^\circ$. What is needed is some function, $C(a,b)$, such that $a \cdot C = m$, that is, some function that relates the smaller component to the magnitude. It is apparent that $a \cdot \sec(\theta) = m$, but this is a function of θ . The next step is to express θ as a function of a and b . Of course, $\theta = \arctan(b/a)$, so the above expression becomes:

$$a \cdot \sec[\arctan(b/a)] = m, \quad 1 \leq b/a < \infty.$$

TABLE 4.4.4(1)
MULTIPLICATION (MULCON) CONSTANTS

$K_{10}(K_8)$	M1/M2	w_K^*	Right Shift	Left Shift
1 (2)	1/2	0.500j	-1,0	
2 (4)	1/4	0.707j	-2,-5	
3 (6)	1/8	1.307j	+2,+4	
4 (10)	3/8	0.541j	-1,+5	
5 (12)	1/16	2.563j	+1,+4	Shift left but do not add to original
6 (14)	3/16	0.900j	-4,-5	
7 (16)	5/16	0.601j	-1,+3	
8 (20)	7/16	0.510j	-1,0	
9 (22)	1/32	5.101j	+3,0	+2
10 (24)	3/32	1.722j	+1,+2	
11 (26)	5/32	1.061j	+4,0	
12 (30)	7/32	0.788j	-2,+5	
13 (32)	9/32	0.647j	-1,+3	
14 (34)	11/32	0.567j	-1,+4	
15 (36)	13/32	0.523j	-1,+5	
16 (40)	15/32	0.502j	-1,0	

$$*w_K = 0.5 \operatorname{cosecant}(\pi M1/M2)j$$

Thus, $C = \text{secant}[\arctan(b/a)]$. Now the calculation of C for each b and a would be as difficult as the RMS method, but this calculation is not necessary. Instead, a "window" technique is used, whereby a value C is associated with a certain range of values of b/a . In the actual program, the value $8b/a$ is calculated using a standard division routine as this preserves more significant digits in the quotient.

Because it is desired to perform the effective "multiplication" of $C \cdot a$ by successive additions of a , integral values of C are desired. Further, since the value $0.5a$ can be easily found and stored, then C can take the values 1.5, 2, 2.5, 3, ..., 9, 9.5.

What must now be found is the range of values of b/a for which a particular value of C will be chosen. It seems reasonable, where the possible values of C are 2, 2.5, 3, 3.5, etc., that those values of b/a which yield a $2.25 \leq C < 2.75$ should be assigned the value $C = 2.5$, or where $2.75 \leq C < 3.25$, then $C = 3$. Using this method, the absolute error in the resulting magnitude is less than 0.25 times the smallest component. The magnitude, therefore, is an approximation; however, this approximation is adequate for most purposes.

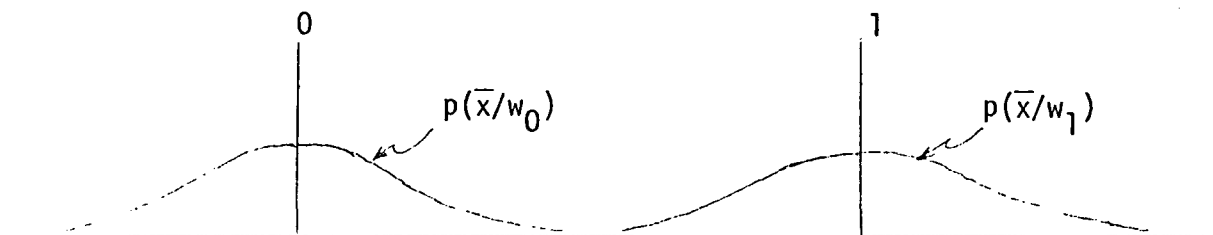
The FFT program composes what can be called the preprocessor part of the classification scheme. It receives the incoming data and finds in it certain parameters or features of interest, in this case frequency components. The next step is to extract those features which can be used to assign the data to a particular class. For FFT80, feature extraction merely means taking certain pre-determined complex frequency components, finding their magnitudes, and storing these magnitudes in a storage array, called MTUDE. The contents of the MTUDE array become the input data to the pattern recognizer stage of the program.

4.4.4.2 The Bayes Classifier - The classification scheme investigated uses the Bayes decision rule. In order to understand how this process works, it is helpful to visualize a space in which every pattern (input value) is a point in space. Thus, the space has as many dimensions as the pattern. For example, suppose the three low frequency components

X_1 , X_2 , and X_3 are being used to classify a signal. Then for each frame pattern, \bar{x} has three components as seen in Figure 4.4.4(1). Each point, $X(F_2, F_3, F_4)$, represents a pattern in three-space. If, after a sufficient number of trials (points plotted), the points are observed to cluster in groups, the Bayes decision rule can be applied. The clustered groups are called classes and some decision must be made as to what constitutes a class boundary. Once the boundaries are defined, each succeeding pattern will be classified according to which side of a boundary it falls upon. The Bayes method does not assure that misclassifications will not be made, but it does assure a minimum average loss in classification; that is, it assures statistical optimization [11]. The average loss in assigning a pattern \bar{x} to a class j given m classes is

$$r_j(\underline{x}) = \sum_{i=1}^m L_{ij} p(\bar{x}/w_i) p(w_i)$$

where $p(\bar{x}/w_i)$ is the probability density function of w_i , $p(w_i)$ is the probability of the occurrence of the class w_i and L_{ij} is a "loss matrix" which assigns a loss of zero for correct classifications and a loss of one for misclassifications. A pattern is assigned to the class offering the smallest average loss. Consider a simple example. Suppose we are receiving a signal that has been corrupted by Gaussian noise. The signal is made up of 1's and 0's, so the actual received signal has a probability density



while $p(w_0)$ is the probability of a 0 being sent and $p(w_1)$ is the probability of a 1 being sent. A reliable classifier can be implemented only if the two density curves do not significantly overlap.

In separating seismic signals into normal (background noise) and abnormal (seismic activity) classes, based on spectral composition, several things must be known. The values of $p(\text{normal})$ and $p(\text{abnormal})$ are not critical and can be arbitrarily set to $p(\text{normal}) = 0.9999$ and $p(\text{abnormal}) = 0.0001$. However, calculating values for the probability densities of normal and abnormal classes is more difficult. To an extent these are functions of the location of the seismic sensor. A PDCP located near a highway, for example, would have a noise spectrum and probability density different from one located in a more remote area. The factors affecting the probability density of seismic spectral components are not well understood and there is a dearth of information on the subject [12]. But perhaps the greatest problem encountered in the use of the Bayes classifier on spectral components of signals is that the signals of both classes are largely impulsive in nature. Impulses, when transformed, exhibit all frequencies equally, and, seismic and noise signals, while not pure impulses, are similarly sharp, fast-changing time functions. The problem then is that any given combination of frequencies that can be extracted from PDCP processing will be similar for both noise and seismic signals. That is, their probability density curves overlap to an extent that they cannot be reliably separated.

It is possible, however, to use the Bayes classifier to separate patterns with high frequency (impulsive-type) components from those with only low frequency components. This would enable noise and seismic signals to be differentiated from calm background activity; but if this were all that was required, a simple high pass filter would be the obvious choice.

While the use of Bayes classification of spectral data cannot be removed from consideration, it is suggested that to be of practical value very high resolution (multiple point) transforms are required. In addition, PDCP's must be trained individually with data taken from the sites on which they will be placed.

The implementation of a fast Fourier transform subroutine on an 8080A microcomputer requires the addition of 1.1K of memory. It will execute 16 point transforms in 39 ms, 32 point transforms in 96 ms, and 64 point

transforms in 228 ms at a CPU clock rate of 2 MHz. This demonstrates the feasibility of performing sophisticated and complex mathematical functions remotely on PDCP. With the advent of new microprocessors with increased speed, indirect addressing, and hardware multiply/divide, such as the Texas Instruments TMS 9900, multipoint FFT subroutines will become practical.

REFERENCES

1. Korn, G. A., Minicomputers for Engineers and Scientists, McGraw-Hill, New York, New York, 1973.
2. "Intel 8080 Microcomputer Systems User's Manual," Intel Corporation, July 1975.
3. Lathi, B. P., Communication Systems, John Wiley & Sons, Inc., New York, New York, 1965, pp. 89-91.
4. Hogg, G., "A Data Compression Primer," X-Document X-521-65-320, NASA - Goddard Space Flight Center, Greenbelt, Maryland, August 1965.
5. Kostopulos, G. K., Digital Engineering, John Wiley & Sons, Inc., New York, New York, 1975, pp. 309-311.
6. Schoep, V. R., "ERTS/GOES Convertible Data Collection Platform," Ball Brothers Research Corporation, Final Report for U. S. Geological Survey EROS Program Office Contract 14-08-0001-1390, 1974.
7. Lichfield, E. W. and Carlson, N. E., "Tropical Wind, Energy Conversion and Reference Level Experiment (TWERLE)," Engineering Design Report No. 2, National Center for Atmospheric Research, November 1974.
8. Coates, J. L., "The Nimbus-F Random Access Measurement System (RAMS)," IEEE Transactions on Geoscience Electronics, Vol. GE-13, No. 1, January 1975, pp. 18-27.
9. Rader, C. M. and N. M. Brenner, "A New Principle for Fast Fourier Transformation," Unpublished notes, 1972.
10. Brigham, Oran E., The Fast Fourier Transform, Englewood Cliffs, New Jersey: Prentice-Hall, 1974, p. 175.
11. Tou, J. T. and R. C. Gonzalez, Pattern Recognition Principles, Reading, Massachusetts: Addison-Wesley Publishing Co., 1974, p. 113.
12. Bath, Markus, Introduction to Seismology, New York: Halsted Press, 1973, pp. 249-251.

5. PROGRAMMABLE DATA COLLECTION PLATFORM DEVELOPMENT SYSTEM

The University of Tennessee programmable data collection platform development system includes an Intel 8080 based special-purpose micro-computer, a video display terminal, a cassette bulk storage device, and supportive system software.

5.1 UT PDCP DEVELOPMENT SYSTEM HARDWARE

A major objective of the PDCP study is to conceive and describe a programmable data collection platform. To aid in achieving this goal, the UT PDCP development system using the Intel 8080 microprocessor has been built. The UT PDCP is designed to serve as a developmental tool which will aid in determining the feasibility of using a microprocessor to perform all current DCP tasks as well as additional tasks which are not possible with contemporary DCP designs. The UT PDCP includes the capability of simulating the transmitted data messages for typical Landsat, GOES, TIROS-N and TWERLE data collection platforms. In addition, the UT PDCP is interfaced to a video display terminal and keyboard permitting user intervention with the various PDCP demonstration programs. The UT PDCP is not intended to reflect a choice of microprocessors for an actual PDCP; this choice will depend on many factors which are described in Section 6.

The basic hardware system consists of eight 11.43cm x 16.51cm (4.5" x 6.5") printed circuit cards manufactured by Pro-Log Corporation. There are three 4K RAM cards, two 2K ROM cards, one 32-line input card, one 32-line output card, and the CPU card. The eight Pro-Log MPS components cards supply 32 parallel input and output lines for interfacing simulated sensors, the system memory (12K RAM and up to 4K ROM), and the microprocessor chip and support logic. To complete the UT PDCP, four printed circuit cards have been added to the basic Pro-Log system. The additional cards include a dual serial I/O card, an analog interface card, a cassette analog interface and baud-rate timing card, and a utility card containing hardware system start-up, a baud-rate clock and current loop

interface for the video terminal, and single step control logic. All printed circuit cards plug into a small 12.7cm x 25.4cm (5" x 10") rack with a wire-wrapped back plane. The rack is mounted inside an attractive gray cabinet. Power is provided by a small external power supply. The front panel switches are minimal as control of the UT PDCP is intended to be from keyboard commands issued from the video terminal. Jacks for interconnection of the video terminal and cassette recorder are on the rear panel. Simulation of several DCP functions is provided by LED's mounted on the front panel. All connections between the cassette recorder, video display and the UT PDCP are via plug-in cables for ease of interconnection of the three system components. The only additional hardware needed to completely demonstrate the capabilities of the UT PDCP system are a laboratory dual-sweep oscilloscope and a frequency counter with period measurement capability. A description of the operating procedure for these instruments is included in the System Operation Manual. The following sections describe in detail the individual components of the UT PDCP system. A block diagram of the major components of the UT PDCP system hardware is depicted in Figure 5.1(1). Figure 5.1(2) presents a more detailed view of the PDCP microcomputer system components.

5.1.1 Basic Pro-Log System

An 8080 based microprocessor system was selected for the UT PDCP. There are several reasons for this choice. The 8080 microprocessor has an eight bit word size which should be optimal for the PDCP application. Extensive software support packages including resident text editors, assemblers, simulators, BASIC, FOCAL, and FORTRAN are available for the 8080. In addition, cross-assemblers, a cross-simulator, and a FORTRAN cross-compiler are also available. A few recent microprocessors have extensive software support available now; however, at the beginning of this study only the 8080A had extensive software support. Due to the short period allotted for this study, extensive software support was required to enable development of a large number of PDCP programs.

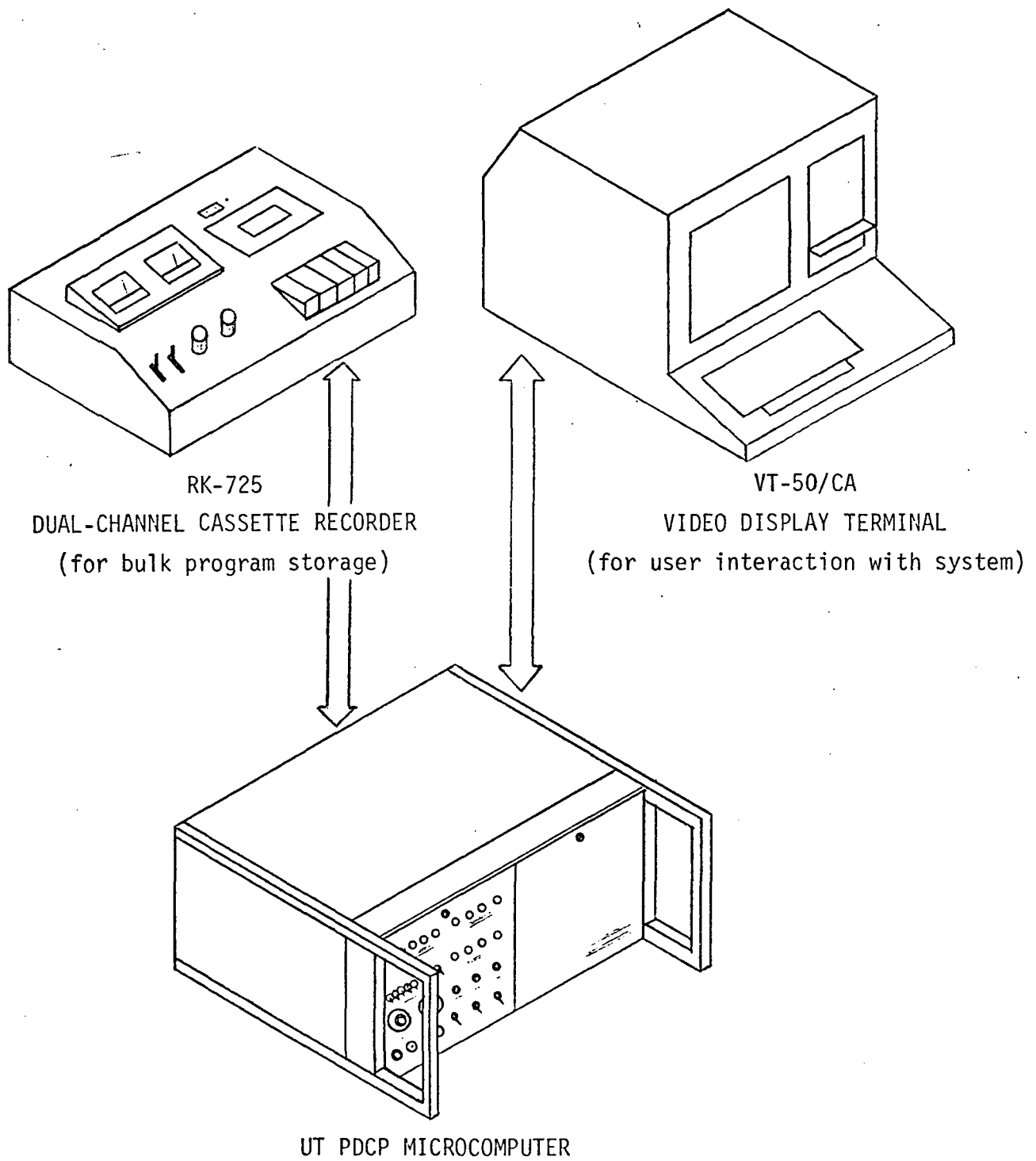


Figure 5.1(1) Major Components of the UT PDCP DEVELOPMENT SYSTEM.

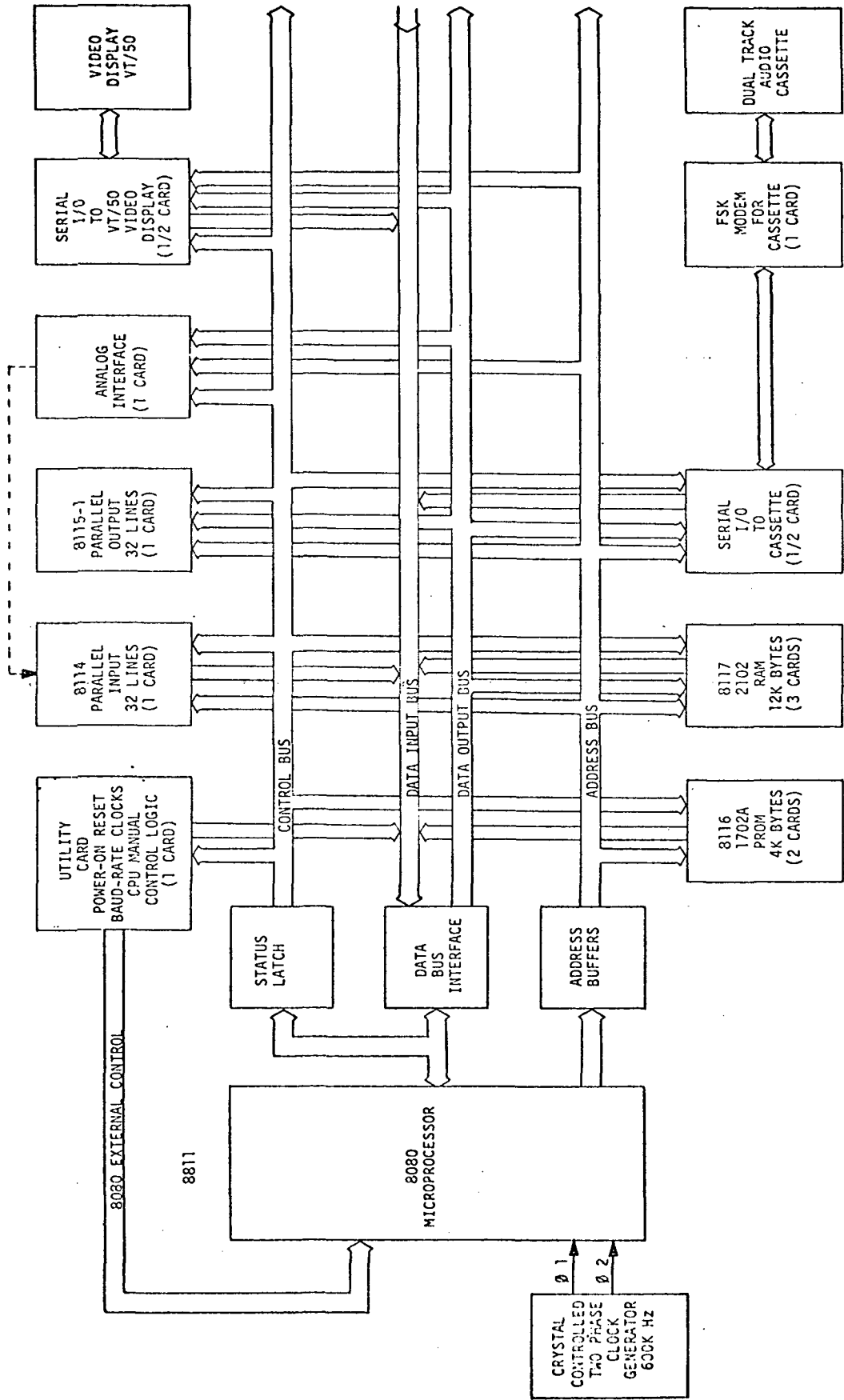


Figure 5.1(2) UT PDCP Microcomputer Development System Block Diagram.

The 8080 based Pro-Log Corporation Microprocessor System (MPS) card components offer a stripped-down, cost-effective basic system. With the addition of four specially designed cards, a powerful PDCP development system is obtained. Other companies were interviewed as a source of basic system hardware; however, on the basis of economics and availability, a decision was made to use a set of eight basic cards made by Pro-Log Corporation and add to them four specially designed cards to create a special purpose microcomputer. Perhaps a better PDCP development system could have been designed starting with only a microprocessor chip set and no commercial cards. However, due to the short time span of this study, the ideal approach had to be compromised. Thus, the Pro-Log MPS cards provide the basic microcomputer system functions, and four add-on cards were designed to form a powerful PDCP development system.

5.1.1.1 Central Processing Unit Card - The major component of the Pro-Log system is the 8811 Central Processing Unit (CPU) card. The CPU card contains an Intel 8080 microprocessor, a two-phase clock, a power-up reset circuit, and data, address, memory, and I/O control logic. A few timing signals needed for special functions were not brought out to the edge connector on the stock CPU card. Thus, these signals were either brought out on spare edge connector pins or generated externally on the utility card (see Section 5.1.3). To facilitate accurate timing for PDCP transmitter demonstration programs, the crystal in the clock circuit for the CPU was changed from 5.0 MHz to 4.8 MHz. The 4.8 MHz clock provides a basic $1.66\overline{66}$ microsecond (μsec) state time for all instructions. This state time was chosen to provide an integer number of states between phases of a Manchester encoded data stream transmitted at the typical DCP rates (100, 400, or 5000 bits per second). Note that a $1.66\overline{66}$ μsec state time is over three times slower than the nominal 500 nanosecond (nsec) state time for a full speed 8080 system. The slower state time also permits the use of low speed, inexpensive memory without the need for wait-cycle generation circuitry.

5.1.1.2 Random-Access Memory Cards - There are three 8117 Random-Access Memory (RAM) cards in the basic Pro-Log system which

provide a total of 12K of RAM. Each RAM card provides 4K by eight bits of static RAM (2102 type, 1.0 μ sec access time). An actual PDCP would not require such a large amount of RAM. The UT PDCP is provided with 12K bytes of RAM to facilitate program development and to provide a storage area for the text of the extensive comments which accompany the PDCP demonstration software. The 12K of RAM permits all the programs of the PDCP demonstration package to be resident simultaneously with room for additional programs.

Although the access time of the 2102 type RAM's used on the three 8117 RAM cards is faster (1.0 μ sec compared to 1.6666 μ sec state time) than the basic state time of the CPU, the frequency of the system clock is limited by the access time of the read-only memory (ROM). A wait-cycle generator circuit could have been added to obtain maximum overall system speed; however, different cycle times for RAM, ROM, and non-memory referencing instructions would have complicated software timing routines. Maximum system speed is not needed to simulate all DCP functions performed by current DCP's. The three 8117 RAM cards are assigned absolute memory addresses 0-12K.

5.1.1.3 Read-Only Memory Cards - The two 8116 Read-Only Memory (ROM) cards provide non-volatile program storage. A powerful system monitor program resides in slightly less than 3.5K of ROM. General purpose binary math subroutines occupy the remaining 0.5K of ROM. Both ROM cards accept up to eight 256 x 8 bit Intel 1702A type PROM's. Thus, each card can accommodate up to 2K of ROM. To keep costs low and still meet maximum data rate specifications, 1.7 μ sec maximum access time 1702A ROM's are used.

The ROM's are ultraviolet-erasable memories and are programmed with the department's Intellec 8/Mod 80 microcomputer. The two 8116 ROM cards have been assigned the 16-20K absolute memory addresses.

5.1.1.4 Latched Parallel Output Card - Thirty-two parallel latched output lines are provided by a Pro-Log 8115-1 Output Card. The parallel output card is organized as four 8-bit parallel output ports. Execution of an "OUT" instruction causes the eight bits in the accumulator to be

latched into a particular eight-bit output port depending on the port address specified by the "OUT" instruction. The output lines are TTL compatible with a fanout of 10. The four 8-bit parallel output ports are assigned the absolute output port addresses 0-3. Output port 0 is normally used for PDCP data output. Bit 0 of port 0 is normally a serial data output for PDCP transmitter formatted data. Other bits of port 0 provide scope data and synchronization signals as well as front panel LED data displays. Output port 1 is normally used to drive PDCP status displays and PDCP transmitter control signals. For example, four bits are used to drive the front panel PDCP mode displays (Landsat, GOES, TIROS-N, or TWERLE mode). Other bits of output port 1 represent signals to control platform functions such as platform power on and off, RF power on and off, data enable, and data integrator initial conditions control. Presently, output port 2 controls the cassette read/write flag LED, while port 3 is brought out to the rear panel for transmitter timing verification.

An additional parallel output port is provided on the dual serial I/O card (see Section 5.1.2.1) to control the cassette recorder interface.

5.1.1.5 Parallel Input Card - The last member of the Pro-Log MPS components card family is the 8113, 32-line parallel input card. Like the parallel output card, the input card is organized as four 8-bit parallel ports. The parallel input card provides parallel data input. The input ports are assigned absolute addresses 0-3, the same as the parallel output ports. The eight-bit data switch register on the front panel is connected to input port 0.

The switch register serves as simulated parallel sensor data. The eight-bit byte loaded on the data switch register is input to the accumulator of the CPU upon execution of the "IN 0" instruction. The remaining three parallel input ports provide additional simulated sensor inputs.

The dual serial I/O card also has a parallel input port (see Section 5.1.2.1) which is used to input status of the cassette recorder interface, the digital-to-analog converter, and the two Universal Asynchronous

Receiver Transmitter (UART) chips used to interface the serial data of the video display terminal and the cassette recorder.

5.1.2 Special Purpose Interface Cards

The eight 8080 based Pro-Log MPS components cards provide the basic functions of a microcomputer: CPU, memory, and parallel input/output. Four special-purpose interface cards have been added to the basic Pro-Log system to create a powerful special-purpose microcomputer system capable of simulating all the tasks performed by present DCP designs [see Figure 5.1(2)].

The dual serial I/O card was developed to interface the serial data format of the video display terminal to the parallel bus structure of the CPU. The second serial interface provides serial/parallel conversions for the cassette recorder interface to and from the CPU bus. A parallel input and output port is included on the dual serial I/O card to input status from the serial interfaces and cassette recorder and to output control signals to the cassette interface card and the external frequency counter.

The analog interface card provides simulated sensor input of analog data. The card contains an eight-bit digital-to-analog (DAC) converter, a comparator, a voltage-to-frequency converter and the necessary handshake logic to permit software analog-to-digital conversion.

The cassette interface card uses a frequency shift keyed (FSK) technique to store and retrieve data from an inexpensive cassette recorder. A subharmonic of the baud rate clock used to clock data to the cassette is also recorded. Thus, on playback, the subharmonic of the baud rate clock is used to phaselock the baud rate clock decoding the data. This technique permits large speed fluctuations from the cassette with no effect on the error rate of the data.

The general purpose utility card provides several special functions including hardware system start-up; baud rate clock and current loop interface for the video terminal; and single step control logic. A more detailed description of the four special purpose cards is provided below.

5.1.2.1 Dual Serial I/O Card - The dual serial I/O card and the other three special-purpose cards were designed to be directly compatible with the address and data-bus structure of the Pro-Log 8811 CPU card. The bus inputs and outputs are buffered so that the CPU bus driving capability is preserved. Two serial input and two serial output ports are implemented on the dual serial I/O card as well as a parallel input and output port.

The major component of the dual serial I/O card is a MOS, LSI device called a Universal Asynchronous Receiver Transmitter (UART). The UART performs serial-to-parallel and parallel-to-serial data conversion as well as providing status information on the states of the input and output buffers included in the device. The UART can be programmed to receive and transmit five to eight bit words, with or without parity and with one or two stop bits. The UART interfacing the video terminal is hardwire programmed for an eight-bit word, one stop bit and no parity. The second UART interfaces the serial data format of the cassette recorder and is programmed under software control via the parallel output port included on the dual serial I/O card.

The parallel output port also controls the motor of the cassette deck. Also provided on the dual serial I/O card is an eight-bit parallel input port used to input UART status. Port absolute-address decoding is included on the card, making it a stand-alone card except for baud rate clocks needed to drive the UART's. A standard Pro-Log single serial interface card would have required connection to the Pro-Log parallel input and output cards due to inadequate buffering of their serial interface card and the lack of address decoding. The specially designed dual serial I/O card not only provides two serial I/O ports and all the CPU handshaking required, but it is independent of the parallel I/O cards thus freeing all 64 parallel I/O lines for special purpose use.

The serial I/O data absolute port address assignment for the video display terminal is port 7. Input status and output cassette control and UART programming has been assigned to port 6.

5.1.2.2 Analog Interface Card - The analog interface card contains an eight-bit CMOS digital-to-analog converter (DAC), a comparator, a voltage-to-frequency converter, and CPU handshake logic. The eight-bit DAC and comparator are used in a software successive approximation analog-to-digital conversion technique. The voltage-to-frequency converter provides a second, less expensive technique for analog-to-digital conversion. The frequency output is counted with a software frequency counter routine resulting in a digital conversion of the analog input. The software routines for the successive approximation analog-to-digital conversion (ADC) and the voltage-to-frequency to digital conversion are included in PROM.

5.1.2.3 Cassette Analog Interface and Baud Rate Clock Card - The audio interface to and from the cassette recorder is provided by the cassette analog interface and baud rate clock card. Serial data from the cassette serial data output port of the dual serial I/O card is converted to AFSK tones of 4,500-Hz (space tone) and 5,500-Hz (mark tone). The AFSK signal is created on the cassette analog card and is recorded on the right channel of the cassette recorder. In playback, the recovered AFSK tones from the right channel are converted back to serial TTL compatible levels with an EXAR-210 FSK demodulator integrated circuit. The recovered serial data from the cassette analog card is then applied to the dual serial I/O card's cassette serial input port.

Simultaneous recording of a 600-Hz tone on the left channel is provided by a baud rate timing circuit on the cassette analog and baud rate timing card. A master clock frequency of 57,600-Hz is divided down to 600-Hz. The divider chain provides optional baud rates of 300, 600, 1200, and 1800 baud. The 1200 baud clock was selected for reliable, fast data recording.

The master 57,600-Hz clock is derived from two sources. In record, a 555 Timer IC operating as a fixed frequency astable oscillator supplies

the master clock frequency. In playback, the recovered 600-Hz subharmonic tone is used to phase-lock a VCO to 96 times the recovered 600-Hz tone. The synthesized recovered baud rate clock provides timing for the serial decoding of recovered data. Since the recovered baud rate timing suffers approximately the same timing errors as the recovered data, overall serial data decoding is relatively immune to timing errors due to the speed variations of an inexpensive cassette recorder.

5.1.2.4 General-Purpose Utility Card - This card contains:

1. Hardware monitor start-up circuit.
2. Baud rate clock for the video display terminal.
3. TTY current interface for the video display terminal.
4. Single step and other control logic.

A single push button switch is used to activate a hardware start-up circuit. In addition, a power-up circuit on the CPU card also activates the start-up circuit so that application of power starts the system.

The dual serial I/O card provides TTL logic level serial data. The utility card interfaces the standard 20mA current loop of the video display terminal to the TTL logic levels of the dual serial I/O card. A baud rate clock for the video terminal is included on the utility card providing switch selectable standard baud rates.

Synchronous run, wait, and single step logic is provided for hardware control of the UT PDCP. In addition, a special "slow-run" circuit is provided to permit slow execution of a program. This feature is particularly useful for demonstrating PDCP transmitter routines.

5.1.3 Miscellaneous Hardware Components of the UT PDCP Development System

The eight Pro-Log MPS components cards and the four special purpose interface cards plug into a small 12.7cm x 25.4cm (5" x 10") rack with

wire-wrapped edge connectors. The cards are powered by three voltages (+5, +12, and -9 volts) obtained from two commercial short-circuit proof power supply modules which were made specifically for the Pro-Log MPS components cards. The analog interface card also utilizes a -12 volt power supply which was added to the system.

An attractive gray cabinet houses the small rack containing the 12 system cards. The AC power supply is external and connected via a long flexible cable. The front panel of the cabinet contains a minimum number of switches and displays in an effort to simplify use of the UT PDCP system. A set of eight switches provides a fixed value data input source and is useful in simulating a sensor data source. The value toggled on the data switch register is input to the accumulator of the CPU upon execution of an "IN Ø" instruction. Other switches on the front panel include wait, single step, slow-run, start, power-on, and cassette motor-on. The function of these switches is explained in the System Operation Manual. Several LED's are included on the front panel to simulate special functions being performed by PDCP software routines. Finally, BNC-type jacks located on the front panel are used to display signals on an external oscilloscope, and one jack can be connected to an external frequency counter to provide timing verification of PDCP functions. The LED indications and signals appearing at the BNC jacks are explained in detail in the System Operation Manual.

The rear panel of the UT PDCP contains a jack for interconnection of the cassette recorder. Another connector provides interconnection to the video-display terminal. A baud rate select switch for the serial teletype interface also appears on the rear panel. The terminal baud rate switch allows connection of a hard copy teleprinter (such as an ASR-33) in place of the video terminal in the event hard copy is desired.

User interaction with the UT PDCP system is provided by a Digital Equipment Corporation VT-50/CA video display and data entry terminal. The VT-50/CA communicates via a full-duplex serial TTY current loop. A TTY current loop interface is included on the general-purpose utility card. Full duplex operation is maintained to achieve maximum versatility of the VT-50/CA. The VT-50/CA provides an 80-character per line, 12 line

alpha-numeric display and a multimode ASCII two-key rollover keyboard.

The connector feeding the VT-50/CA could be used to operate any other ASCII TTY 20mA current-loop device such as the industry standard low-speed ASR-33. Most applications for the UT PDCP system are best suited to a high-speed video terminal with cursor control like the DEC VT-50/CA; however, hard copy of program listings and data can be obtained using an ASR-33 teletype if desired.

Several cassette recorders were tested for suitability as a bulk program storage device. Of all the recorders tested, the Lafayette RK-725 stereo cassette deck exhibited the best frequency response (approximately twice the frequency response of all other units tested). Since a speaker and high-power audio amplifier are not needed, a cassette deck is preferable to a cassette player. The high-frequency data audio tones for the cassette interface take advantage of the high-frequency response of the Lafayette RK-725. Higher frequency AFSK tones reduce jitter from the phase locked loop demodulator. Also, having two channels permits excellent isolation between the AFSK tones and the baud rate subharmonic tone. No filter is required to separate the tones as would be the case for a monaural recorder. Frequency jitter on playback (a common problem of inexpensive recorders) was not substantial and as discussed in Section 5.1.2.3, a 600-Hz baud rate subharmonic tone recording is used to eliminate timing errors in decoding the data.

In summary, the hardware for the UT PDCP development system consists of a special purpose 8080-based microcomputer, a DEC VT-50/CA video terminal, and a cassette recorder and interface for program storage.

5.2 UT PDCP DEVELOPMENT SYSTEM SOFTWARE PACKAGE

To develop a large number of PDCP demonstration programs in the short time period of this study, software support must be extensive. At the beginning of this research period, the only microprocessor family possessing substantial software support was the Intel 8080 microprocessor

family. Since then, several companies have second-sourced the 8080 including improved versions. Several software firms have also added software support. Intel's user's library contains many useful programs, including a Macro Cross Assembler which runs on the DEC PDP-11 minicomputer. Dr. Steve Olsen of the University of Utah supplied a paper tape of his cross assembler, and all of the PDCP programs written during the research period were assembled on the department's PDP-11/40 using a modified version of Dr. Olsen's cross assembler.

Microcomputer software is often developed using the hexadecimal or octal number systems. The PDCP programs developed on the PDP-11/40 use an octal format whereas standard Intel 8080 software is provided in a hexadecimal format. In addition, non-programmers may not be familiar with either hexadecimal or octal and, thus, would prefer to work in decimal. To facilitate the development of PDCP programs a multiradix system monitor was developed for the UT PDCP development system. This monitor will execute commands in octal, hexadecimal, or decimal. For example, programs can be listed in any of the three radices.

The PDCP resident system monitor provides several key functions for the operation of the UT PDCP system. Examination and modification of memory contents, cassette storage and retrieval of bulk data, radix conversions, and program execution commands with debugging techniques are among the many system monitor functions which have been incorporated under this study. In addition, a special keyboard assembler command permits programming of the UT PDCP directly from the keyboard using standard Intel instruction mnemonics. Finally, memory contents may be displayed symbolically using a special list symbolic command. The PDCP resident monitor resides in non-volatile ROM and is available immediately on system start-up. The two 8116 Pro-Log PROM cards contain the entire PDCP system monitor. The System Operation Manual support software section describes in detail the resident monitor software and commands.

The system monitor requires about 3.5K of the available 4K of PROM. The remaining 0.5K of PROM contains general purpose PDCP subroutines. Having these often used subroutines resident in non-volatile memory

simplifies source program writing as the routines are simply called when needed instead of repeating the source code of the entire subroutine. Source listings and explanations of these PDCP subroutines are described in the System Operation Manual support software section.

6. FUTURE PDCP SYSTEMS

The last five years have witnessed the birth, development, and application of the microprocessor. It is difficult to accurately predict what will happen during the next five years in this field. Semiconductor manufacturers are already working on third-generation microprocessors. The designers of data collection system platforms should have available to them a prediction or reasonable projection of the future characteristics and capabilities of the microprocessor if a programmable data collection platform is to be considered. The purpose of this section of the report is to provide a method of microprocessor selection and to furnish the designer with a microprocessor capability forecast for the next five-year period.

6.1 EVALUATION OF AVAILABLE MICROPROCESSORS

The research proposal for this contract projected the source-destination matrix [1] for evaluation of microprocessor instruction sets. In this technique, an instruction is considered as the transfer of data from a selected source to a selected destination. By constructing a matrix in which the sources are in rows and the destination in columns, the instruction set and functional operations of the microprocessor can be depicted in a concise format. However, the source-destination matrix has two significant disadvantages. First, no accepted methods exist for evaluating or establishing a performance measurement for the microprocessor from the source-destination matrix such that a comparison of different microprocessors can be made. Secondly, the source-destination matrix does not integrate other system characteristics into the evaluation.

Present trends in system evaluation are toward the development of classification and performance measures which integrate all the operating characteristics of a system. For a microprocessor system, this includes system constraints, hardware organization, memory hierarchy, software structures, and application directorates. In fact, most of these considerations can be represented by the following functional form:

$$\text{Performance Measure} = \Delta f(\text{system constraints,} \\ \text{application constraints,} \\ \text{software constraints}) .$$

System constraints include power, weight, size, and speed considerations, while application constraints focus on the specific computational tasks to be executed. Thus, application constraints more clearly define the capability of various microprocessors to meet computational requirements. Software constraints are imposed by the failure to provide software support for the system. For instance, new applications are difficult to program and implement for a microprocessor system with no cross assembler, editor, or simulator.

This research has directed effort toward generating performance measures relating to system, application, and software constraints. A systematic procedure for generating a performance measure is implemented by constructing a system for matrices relating these constraints to the different microprocessors. Consider the following matrix form:

Constraints	Importance Weighting Matrix	μP_1	μP_n
System	\bar{W}_1	\bar{R}_{11}	\bar{R}_{n1}
Application	\bar{W}_2	\bar{R}_{12}	\bar{R}_{n2}
Software	\bar{W}_3	\bar{R}_{13}	\bar{R}_{n3}

\bar{W}_i is a weighting matrix associated with the importance of a particular constraint; whereas, \bar{R}_{ji} is a rating or measure matrix for μP_j (the j th microprocessor) which evaluates the capability of the microprocessor to satisfy the conditions imposed by the constraint. The performance measure, PM_j , for μP_j is given by

$$\begin{aligned}
 PM_j & \stackrel{\Delta}{=} [\bar{W}]' [\bar{R}_j] \\
 & = [\bar{W}_1 \ \bar{W}_2 \ \bar{W}_3]' \begin{bmatrix} \bar{R}_{j1} \\ \bar{R}_{j2} \\ \bar{R}_{j3} \end{bmatrix} \\
 & = \sum_{i=1}^3 \bar{W}_i \bar{R}_{ji} .
 \end{aligned}$$

Within each general constraint there are numerous conditions for which the measure matrix, \bar{R}_{ji} , must be established. For instance, consider the submatrix formed for the general system constraints in Table 6.1(1). A weighting factor, \bar{W}_{1i} , is associated with the importance of the i th parameter. In evaluating a given microprocessor, a measure (r_{j1i}) of the ability of the microprocessor to satisfy the parameter requirement can be established. Similar submatrices are formed for the applications constraints of Table 6.1(2) and the software constraints of Table 6.1(3).

To provide an example of this method for microprocessor evaluation, a comparison of the INTEL 8080A and RCA COSMAC microprocessors is developed. In generating each measure matrix, the difference in performance measures is the dominant consideration. The absolute values of the measures would be adjusted as additional information about other microprocessor families is included. Each measure value is established on a scale from 1 to 100. The larger measure values indicate the microprocessor to be more suitable in satisfying the parameter constraints.

If a large number of microprocessors are included in the evaluation, the system lends itself to computer implementation for bookkeeping purposes. A computer with higher-order language capability and containing matrix multiplication features such as APL is ideally suited for the bookkeeping task. Changes in absolute value of the measures can be easily entered and the performance measures can be quickly recomputed.

TABLE 6.1(1)
SYSTEM CONSTRAINTS

Parameter	Weighting Matrix	Measure Matrix
	\bar{w}_j	\bar{r}_{j1}
Power Consumption	w_{11}	r_{j11}
Number of Power Supplies	w_{12}	r_{j12}
Weight	w_{13}	r_{j13}
Size	w_{14}	r_{j14}
Cost	w_{15}	r_{j15}
Minimum System Complexity	w_{16}	r_{j16}
Availability of MSI and LSI Support Logic	w_{17}	r_{j17}
Second Sourcing	w_{18}	r_{j18}

TABLE 6.1(2)
APPLICATION CONSTRAINTS

Parameter	Weighting Matrix \bar{W}_2	Measure Matrix \bar{R}_{j2}
INSTRUCTION SET		
Arithmetic Inst.		
ADD, SUB, AND, EX-OR, OR	W_{21}	r_{j21}
Multiply Divide	W_{22}	r_{j22}
Addressing Modes	W_{23}	r_{j23}
Subroutine Linkage	W_{24}	r_{j24}
Bit Manipulation	W_{25}	r_{j25}
I/O Operations	W_{26}	r_{j26}
Conditional Inst.	W_{27}	r_{j27}
MICROPROCESSOR ORGANIZATION		
Accumulators, Working Registers	W_{28}	r_{j28}
Hardware, Software Stack	W_{29}	r_{j29}
Control	W_{2A}	r_{j2A}
Interrupt Structure Software, Trap Vector	W_{2B}	r_{j2B}
BENCH MARK PROGRAMS		
Program Storage	W_{2C}	r_{j2C}
Execution Time	W_{2D}	r_{j2D}

TABLE 6.1(3)
SOFTWARE CONSTRAINTS

Parameter	Weighting Matrix	Measure Matrix
	\bar{W}_3	\bar{R}_{j3}
Resident Assembler	W_{31}	r_{j31}
Resident Editor	W_{32}	r_{j32}
Cross Assembler	W_{33}	r_{j33}
Simulator	W_{34}	r_{j34}
PL-1	W_{35}	r_{j35}
FORTRAN	W_{36}	r_{j36}
BASIC	W_{37}	r_{j37}
Other High-Level Languages	W_{38}	r_{j38}

The system measure matrices, \bar{R}_{I1} and \bar{R}_{R1} , from Table 6.1(1) for the INTEL 8080A and the RCA COSMAC is determined to be

$$\bar{R}_{I1} = \begin{bmatrix} r_{I11} \\ r_{I12} \\ r_{I13} \\ r_{I14} \\ r_{I15} \\ r_{I16} \\ r_{I17} \\ r_{I18} \end{bmatrix} = \begin{bmatrix} 20 \\ 40 \\ 40 \\ 40 \\ 60 \\ 50 \\ 75 \\ 90 \end{bmatrix} \quad \text{and} \quad \bar{R}_{R1} = \begin{bmatrix} r_{R11} \\ r_{R12} \\ r_{R13} \\ r_{R14} \\ r_{R15} \\ r_{R16} \\ r_{R17} \\ r_{R18} \end{bmatrix} = \begin{bmatrix} 95 \\ 100 \\ 90 \\ 60 \\ 55 \\ 80 \\ 50 \\ 50 \end{bmatrix}$$

In an attempt to justify a few of the measure values, the following comments are provided. Notice that $r_{I11} = 20$ and $r_{R11} = 95$. The justification of this difference is simple. COSMAC consumes an order of magnitude less power than the 8080A. Next, $r_{I12} = 40$ and $r_{R12} = 100$. The 8080A requires three power supplies while COSMAC requires only one unregulated supply. On the other hand, $r_{I17} = 75$ and $r_{R17} = 50$. INTEL provides support hardware in a family of integrated circuits which is superior to that provided by RCA at this time. There are similar considerations for establishing each measure value in \bar{R}_{I1} and \bar{R}_{R1} .

Next, consider the system constraints performance measure, $SPM_j = \bar{W}_1' \bar{R}_{j1}$. If the weighting matrix, \bar{W}_1 , is chosen as unity,

$$SPM_I = \frac{\bar{W}_1' \bar{R}_{I1}}{\sum W_{1n}} = 60.00$$

and

$$SPM_R = \frac{\bar{W}_1' \bar{R}_{R1}}{\sum W_{1n}} = 82.86,$$

where the product $\bar{W}_1' R_{ji}$ has been normalized to a value of 100 for the ideal microprocessor by dividing the result by the sum of the system constraint weights, $\sum W_{1n}$.

COSMAC makes a somewhat better showing; however, a more realistic weighting matrix for DCP applications is

$$\bar{W}_1 = \begin{bmatrix} W_{11} \\ W_{12} \\ W_{13} \\ W_{14} \\ W_{15} \\ W_{16} \\ W_{17} \\ W_{18} \end{bmatrix} = \begin{bmatrix} 100 \\ 5 \\ 3 \\ 1 \\ 10 \\ 8 \\ 8 \\ 15 \end{bmatrix}.$$

W_{11} is weighted more heavily than all other factors combined since power consumption is the most important consideration. Among the remaining system constraints, the availability of second sources for the microprocessor is the most important since this provides some protection against long delivery delays and premature removal of the product from the market. $W_{15} = 10$, indicating cost is not as important when comparing microprocessor-based systems. This weight could become increasingly more important if a microprocessor-based PDCP was being compared to a hardwired DCP.

Using the more realistic choice of weights for the weighting matrix, the system performance measures become

$$SPM_I = 35.90$$

and

$$SPM_R = 84.47 .$$

The superiority of COSMAC to satisfy system constraints in PDCP applications becomes more evident.

Generating the measure matrix, \bar{R}_{j2} , for the application constraints of Table 6.1(2) yields the following two matrices:

$$\bar{R}_{I2} = \begin{bmatrix} r_{I21} \\ r_{I22} \\ r_{I23} \\ r_{I24} \\ r_{I25} \\ r_{I26} \\ r_{I27} \\ r_{I28} \\ r_{I29} \\ r_{I2A} \\ r_{I2B} \end{bmatrix} = \begin{bmatrix} 100 \\ 0 \\ 70 \\ 95 \\ 40 \\ 50 \\ 90 \\ 80 \\ 80 \\ 80 \\ 90 \end{bmatrix} \quad \text{and} \quad \bar{R}_{R2} = \begin{bmatrix} r_{R21} \\ r_{R22} \\ r_{R23} \\ r_{R24} \\ r_{R25} \\ r_{R26} \\ r_{R27} \\ r_{R28} \\ r_{R29} \\ r_{R2A} \\ r_{R2B} \end{bmatrix} = \begin{bmatrix} 100 \\ 0 \\ 70 \\ 60 \\ 35 \\ 80 \\ 75 \\ 65 \\ 40 \\ 80 \\ 50 \end{bmatrix}$$

The measures relating to bench mark programs are not included since programs for COSMAC have not been sufficiently developed to provide a realistic comparison. Note that $r_{I22} = 0 = r_{R22}$. Neither microprocessor provides a hardware multiply or divide at the present time. Since the 8080A has superior subroutine linkage, $r_{I24} = 95$ and $r_{R24} = 60$. On the other hand, COSMAC provides for better I/O communications; therefore, $r_{I26} = 50$ and $r_{R26} = 80$. The stack operations of the Intel 8080A are far superior to those of COSMAC, resulting in $r_{I29} = 80$ and $r_{R29} = 40$.

Considering the importance of the measure in the application constraints of PDCPs, the weighting matrix, \bar{W}_2 , is determined to be

$$\bar{W}_2 = \begin{bmatrix} W_{21} \\ W_{22} \\ W_{23} \\ W_{24} \\ W_{25} \\ W_{26} \\ W_{27} \\ W_{28} \\ W_{29} \\ W_{2A} \\ W_{2B} \end{bmatrix} = \begin{bmatrix} 100 \\ 20 \\ 75 \\ 75 \\ 75 \\ 65 \\ 55 \\ 50 \\ 60 \\ 60 \\ 20 \end{bmatrix} .$$

The comparison of the application constraints performance measure, $APM_j = \frac{\bar{W}_2' \bar{R}_{j2}}{\sum W_{2n}}$ for the two microprocessor yields

$$APM_I = \frac{\bar{W}_2' \bar{R}_{I2}}{\sum W_{2n}} = 74.77$$

and

$$APM_R = \frac{\bar{W}_2' \bar{R}_{R2}}{\sum W_{2n}} = 65.88 .$$

These results indicate that the Intel 8080A is a better choice in satisfying application constraints.

Finally, the measure matrices for the software constraints of Table 6.1(3) are determined.

$$\bar{R}_{I3} = \begin{bmatrix} r_{I31} \\ r_{I32} \\ r_{I33} \\ r_{I34} \\ r_{I35} \\ r_{I36} \\ r_{I37} \\ r_{I38} \end{bmatrix} = \begin{bmatrix} 90 \\ 90 \\ 90 \\ 90 \\ 60 \\ 50 \\ 90 \\ 50 \end{bmatrix}$$

and

$$\bar{R}_{R3} = \begin{bmatrix} r_{R31} \\ r_{R32} \\ r_{R33} \\ r_{R34} \\ r_{R35} \\ r_{R36} \\ r_{R37} \\ r_{R38} \end{bmatrix} = \begin{bmatrix} 80 \\ 80 \\ 95 \\ 90 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

At the present time, INTEL provides better software support than RCA, especially in compiler design. However, compiler design is less important in PDCP applications than the availability of a cross assembler and simulator programs. Thus, the weighting matrix, \bar{W}_3 , for software constraints is determined to be

$$\bar{W}_3 = \begin{bmatrix} W_{31} \\ W_{32} \\ W_{33} \\ W_{34} \\ W_{35} \\ W_{36} \\ W_{37} \\ W_{38} \end{bmatrix} = \begin{bmatrix} 75 \\ 75 \\ 90 \\ 80 \\ 30 \\ 10 \\ 0 \\ 1 \end{bmatrix} .$$

The software constraints performance measure for the two microprocessors is

$$SOPM_I = \frac{\bar{W}_3' \bar{R}_{I3}}{\sum W_{3n}} = 86.29$$

and

$$SOPM_R = \frac{\bar{W}_3' \bar{R}_{R3}}{\sum W_{3n}} = 76.87 .$$

The software support performance measure of the 8080A is not really that superior to COSMAC.

Linearly combining the results generated from these considerations, the overall performance measure for the two microprocessors is evaluated to be

$$PM_I = \frac{\bar{W}' \bar{R}_I}{\sum W_i} = 65.65$$

and

$$PM_R = \frac{\bar{W}' \bar{R}_R}{\sum W_j} = 75.74 .$$

The RCA 1802's performance measure is only ten percentage points higher than the INTEL 8080's. However, in PDCP applications, system constraints will generally be considerably more important than either application or software constraints. A typical importance weighting matrix for the PDCP application is

$$\bar{W} = \begin{bmatrix} 100 \\ 40 \\ 20 \end{bmatrix}$$

The overall performance measures for the 8080A and 1802 microprocessors are

$$PM_I = 51.92$$

and

$$PM_R = 78.87 .$$

The 1802 now has a clear performance advantage of nearly 27 percentage points.

The attempt here is not to conclusively state which of the available microprocessors is better suited for a microprocessor-based PDCP, but rather, to emphasize this technique as a method for evaluating and comparing microprocessors for suitability in PDCP applications. Performance measures derived using this technique are primarily intended to aid in choosing between two or more microprocessors which are known to meet the basic constraints of the application. Invalid results can be obtained if one attempts to make an "apples to oranges" type comparison by failing to eliminate machines with unacceptable characteristics before proceeding with the performance evaluation. For example, a bipolar

microprocessor should be eliminated on the basis of unacceptable power consumption even though program execution time would be much shorter than for a CMOS or NMOS microprocessor. This limitation of the performance evaluation procedure is not a significant handicap since the initial screening process is relatively easy to accomplish. The final screening, which is much more difficult to perform, can be simplified by applying the performance evaluation procedure developed in this section.

6.2 TECHNOLOGY FORECAST

In selecting whether to include the programmable feature in a data collection platform system, the designer should have available any projections which indicate the characteristics and capabilities of semiconductor devices for a period covering the next five to ten years. While these predictions are difficult to obtain, reasonably accurate models can be developed which yield satisfactory results over this time span.

There are a number of methods of technology forecasting [2] which will lead to the development of a set of performance characteristics of components during a particular period in the future. These different methods can be reduced to two basic classes of forecasting techniques. One method, called trend forecasting, involves the development of a mathematical model which utilizes past history as a guide for the projection in the future. The second utilizes the judgement of experts in the field to predict the changes in future technology. This is termed intuitive forecasting.

There are essentially two components which will have the greatest effect upon the design of a programmable data collection system. One component is the microprocessor chip itself, and the other is the memory associated with the microprocessor. Already microprocessors are beginning to appear with clock, input/output buffers and control logic on a single chip. There is even some attempt to include a small memory area on the microprocessor substrate making a true single-chip microcomputer.

Trend forecasting has predicted that the average computer add time will decrease by one order of magnitude in a decade [3]. Five years from now, one could expect the average computer add time to be one-third that of the present day average add time. The same performance improvement can be expected in microprocessors. The average present day microprocessor has an add time of two microseconds. In five years, the average add time is expected to be 670 nanoseconds.

Semiconductor memories have taken a more dramatic reduction in chip area, power, and cost requirements. The area of the memory cell is decreasing at the rate of one order of magnitude in seven years. Some experts believe that a 128K-bit memory chip could be available in 1980. Therefore, it seems reasonable to predict that a microprocessor with a 600-nanosecond add time and 8K of eight-bit words could be fabricated on a single chip within five years. The cost of such a device will be in the \$25 range. With powerful computational capabilities such as these becoming available in the not too distant future, the programmable data collection platform should be a viable element in data collection systems.

The most promising low power technologies for use on future PDCPs appear to be silicon-on-sapphire CMOS (SOS), closed cell COS/MOS logic (C^2L), and integrated injection logic (I^2L). I^2L is a bipolar circuit design technique which significantly increases the density of bipolar circuits and permits operation at any point along a constant speed-power product line spanning several decades of propagation delay and power consumption. Thus, low power operation can be achieved at the cost of increased propagation delay or speed can be improved if higher power consumption is acceptable. Although future production of low power microprocessor systems using I^2L is feasible, most manufacturers are currently concentrating I^2L development efforts in the areas of linear-to-digital interface circuits and combinations of linear and digital processing on a single chip. Many I^2L designs use a level conversion circuit to provide a simple interface between the I^2L I/O signals and standard TTL logic. A CMOS to I^2L interface should also be feasible since most new CMOS designs are capable of driving a TTL load. Thus,

I^2L circuits may provide LSI combinations of digital and analog peripheral functions for future microprocessor based PDCPs.

The standard bulk CMOS process has recently been improved by two different techniques which both have excellent potential for providing future advances in microprocessor systems. Commercial products using SOS and C^2L technologies are already available. SOS integrated circuits are manufactured using many of the standard bulk CMOS processing procedures. The main distinction between SOS and bulk CMOS is that a sapphire substrate is utilized in SOS processing to reduce load capacitance and improve circuit density. Propagation delay is also reduced. High speed, low power, SOS memories which are compatible with the 1802 microprocessor are currently manufactured by RCA. These devices exhibit lower power consumption than bulk CMOS devices for operating speeds above approximately 1KHz but consume more power than bulk CMOS devices in static operation.

The most recent advancement in CMOS processing is the closed cell COS/MOS, or C^2L , process developed by RCA for the 1802 microprocessor. C^2L is a circuit design technique that permits a common source structure for 200 to 300 transistors. The transistor's gate electrode forms a closed circle which provides gate termination and eliminates the need for guardbands. As a result, circuit densities are approximately the same as for SOS. C^2L also results in a higher transconductance-to-drain capacitance ratio. In conjunction with a self-aligned silicon gate which reduces Miller capacitance, C^2L offers significant speed improvements over bulk CMOS. Additional refinements in the C^2L and SOS technologies are expected to provide continued improvements in microprocessors and associated devices which exhibit the low power consumption required by the PDCP application.

REFERENCES

1. Nichols, J., "The Source-Destination Matrix for Instruction Set Presentation," IEEE Spring Computer Conference, San Francisco, California, February 25-26, 1975, pp. 61-63.
2. Bright, J. R., Editor, Technological Forecasting for Industry and Government, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, pp. 146-147, 1968.
3. Wintz, et al., "Study of On-Board Processing for Earth Resources Data," Wintek Corp., Lafayette, Indiana, NASA - Ames Contract No. NASA 8327, September 1975.

APPENDIX A

BINARY MATH PACKAGE

APPENDIX A

BINARY MATH PACKAGE

```

000000
000000
000000
000001
000001
000002
000002
000003
000003
000004
000004
000005
000006
000006
000007
000007
000010
000010
000011
000011
000012
000012
000014
000014
000015
000015
000016
000016
000017
000021
000021
000022
000022
000023
000023

          345
          176
          201
          167
          043
          176
          210
          167
          043
          176
          316
          167
          043
          176
          316
          167
          341
          311

SHORT MULTI-PRECISION ADD SUBROUTINE
;
; THIS SUBROUTINE PERFORMS BINARY ADDITION ON TWO UNSIGNED
; BINARY NUMBERS. THE 16 BIT NUMBER IN REGISTER PAIR BC IS
; ADDED TO THE 32 BIT NUMBER STORED IN MEMORY LEAST SIGNIFI...
; CANT BYTE FIRST BEGINNING AT THE LOCATION SPECIFIED BY THE
; CONTENTS OF HL. THE 32 BIT SUM IS RETURNED LEAST SIGNIFI...
; CANT BYTE FIRST BEGINNING IN THE MEMORY LOCATION ADDRESSED
; BY HL. ALL MACHINE STATUS IS PRESERVED EXCEPT FOR THE ACC
; AND FLAGS. ON RETURN, CARRY FLAG (CY) = 1 INDICATES
; AN OVERFLOW OCCURED. THE SUBROUTINE REQUIRES 20 BYTES
; OF ROM, USES 4 BYTES OF STACK STORAGE (RAM) AND EXECUTES IN
; 141 STATES. A SHORT FORM NOTATION FOR THE PRECEDING EX...
; PLANATION IS GIVEN BELOW:
;
; (BC) + (MCHLD) ----> MCHLD  CY=1 INDICATES OVERFLOW
; 16 BITS + 32 BITS = 32 BITS  ACCUMULATOR AND FLAGS
; DESTROYED. 141 STATES, 20 BYTES ROM, AND 4 BYTES STACK.
SMADD:
    PUSH    H           ; SAVE STATUS OF HL
    MOV     A,M         ; GET LS BYTE OF SUM
    ADD    C           ; FORM LS BYTE OF SUM
    MOV    M,A         ; STORE LEAST SIG BYTE OF SUM
    INX    H           ; INCREMENT ADDRESS POINTER
    MOV    A,M         ; GET 2ND BYTE OF SUM
    ADC    B           ; FORM SECOND BYTE OF SUM
    MOV    M,A         ; STORE SECOND BYTE OF SUM
    INX    H           ; INCREMENT ADDRESS POINTER
    MOV    A,M         ; MOVE NEXT BYTE TO A
    ACI    O           ; FORM THIRD BYTE OF SUM
    MOV    M,A         ; STORE THIRD BYTE OF SUM
    INX    H           ; INCREMENT ADDRESS POINTER
    MOV    A,M         ; MOVE NEXT BYTE TO A
    ACI    O           ; FORM MOST SIGNIFICANT BYTE OF SUM
    MOV    M,A         ; STORE MS BYTE OF SUM
    POP    H           ; RESTORE HL
    RET

```

Program A(1) Short Multiprecision Add Subroutine.

DOUBLE PRECISION ADD WITH MEMORY SUBROUTINE

```

; (BC) + (MCHL) -> MODE; CY=1 INDICATES OVERFLOW
; 16 BITS + 16 BITS = 16 BITS ... ACCUMULATOR AND FLAGS ARE
; DESTROYED. 80 STATES, 11 BYTES ROM, AND 2 BYTES STACK.
;
DADDM:

```

000050	MOV	A, M	; GET LS BYTE FROM MEMORY
000050	ADD	C	; ADD LS BYTES
000051	STAX	D	; STORE LS BYTE SUM IN MEMORY
000052	INX	H	; INCREMENT MEMORY POINTERS
000053	INX	D	
000054	MOV	A, M	; GET MS BYTE FROM MEMORY
000055	ADC	B	; ADD MS BYTES
000056	STAX	D	; STORE SUM OF MS BYTES IN MEMORY
000057	DCX	H	; RESTORE HL TO ORIGINAL VALUE
000060	DCX	D	; RESTORE DE TO ORIGINAL VALUE
000061	RET		; RETURN

176
201
022
043
023
176
210
022
053
033
311

Program A(3) Double Precision Add With Memory Subroutine.

```

000063
000063
000063
000064
000064
000065
000065
000066
000066
000067
000067
000070
000070
000071
000071
000072
000072
000073
000073
000074
000074
000075
000075
000075

171
226
022
043
023
170
236
022
053
033
311

DOUBLE PRECISION SUBTRACT WITH MEMORY SUBROUTINE
;
; (BC) ... (MCHLD) ... (MCDE) CY=1 INDICATES UNDERFLOW
; 16 BITS ... 16 BITS = 16 BITS ... ACCUMULATOR AND FLAGS
; DESTROYED ... 99 STATES, 11 BYTES ROM, 2 BYTES STACK.
;
DSUBM:
MOV A,C ; GET LS BYTE
SUB M ; SUBTRACT LS BYTES
STAX D ; STORE RESULT IN MEMORY ADDRESSED BY DE
INX H ; INCREMENT MEMORY POINTER
INX D ; INCREMENT DE POINTER
MOV A,B ; GET NEXT BYTE
SBB M ; SUBTRACT MEMORY
STAX D ; STORE RESULT IN MEMORY ADDRESSED BY DE
DCX H ; RESTORE HL TO ORIGINAL VALUE
DCX D ; RESTORE DE TO ORIGINAL VALUE
RET ; RETURN

```

Program A(4) Double Precision Subtract With Memory Subroutine.


```

DOUBLE PRECISION DIVIDE SUBROUTINE
;
; THIS ROUTINE EXPECTS A 32-BIT NUMBER IN HL AND DIVIDES
; IT BY THE 16-BIT NUMBER IN BC. THE QUOTIENT IS RETURNED
; IN DE WHILE THE REMAINDER IS RETURNED IN HL. THE ORIGINAL
; DIVISOR IS SAVED AND RETURNED IN BC. THE QUOTIENT IS
; ROUNDED-OFF. CY=1 IMPLIES ROUND-OFF OCCURRED.
;
; 64 BYTES ROM + 7 BYTES FROM DPMULT ROUTINE, 8 BYTES STACK USAGE
; AND 3645-3456 MACHINE STATES.
;
DPDIV:
MVI A,17 ; PRESET BIT SHIFT COUNT
DPDIV1: PUSH PSW ; SAVE BITS SHIFT COUNT
CALL DFGSUB ; DOUBLE PRECISION SUBTRACT (HL-BC)
XCHG ; HL <----> DE
CALL DFGHLC ; SHIFT HL LEFT THRU CARRY
XCHG ; HL <----> DE
CALL DFGHLC ; SHIFT HL LEFT THRU CARRY
POP PSW ; GET BITS SHIFT COUNT
DCR A ; DECREMENT SHIFT COUNT
JNZ DFDIV1 ; NOT 16TH SHIFT, LOOP
PUSH B ; SAVE DIVISOR
PUSH H ; SAVE REMAINDER
MOV L,C ; DIVISOR TO HL
MOV H,D
CALL DFGHRC ; DIVIDE DIVISOR BY TWO
MOV B,H ; DIVISOR/2 ----- BC
MOV C,L
POP H ; RESTORE REMAINDER
PUSH H ; SAVE REMAINDER AGAIN
CALL DFGSUB ; REMAINDER-(DIVISOR/2) ----> HL
JNC DFDIV2 ; FIRST FRACTION < .5 THEREFORE DON'T ROUNDOFF
INX D ; ELSE ROUND-OFF QUOTIENT

```

Program A(7) Double Precision Divide Subroutine.


```

000347 CALL DFGHLC ; SHIFT HL LEFT
000348 XCHG
000349
000350 CALL DFGHLC ; SHIFT HL LEFT
000351 CALL DFGSUB ; SUBTRACT PARTIAL ROOT WITH O1
000352
000353 XCHG ; ADDED FROM PARTIAL DIFFERENCE LEFT IN HL
000354 ; PUT PARTIAL DIFFERENCE BACK IN DE
000355
000356 POP B ; GET PARTIAL ROOT BACK AND SHIFT COUNT
000357 MOV A,B ; SHIFT IN NEXT BIT OF ROOT
000358
000359 RAL
000360 MOV B,A ; SAVE PARTIAL ROOT
000361 DCR C ; DECREMENT SHIFT COUNT
000362
000363 JNZ SORT1 ; NOT DONE, DO AGAIN
000364
000365 XRA A ; CLEAR CARRY AND ACC AND CHECK FOR ROUND-OFF
000366
000367 PUSH B ; SAVE PARTIAL ROOT
000368 MOV L,B ; PARTIAL ROOT---->L
000369 MOV H,A ; O---->H
000370 CALL DFGHLC ; SHIFT 0 INTO PARTIAL ROOT
000371
000372 STC ; SET CARRY
000373
000374 CALL DFGHLC ; SHIFT 1 INTO PARTIAL ROOT TO CREATE R01
000375 MOV B,H ; PUT R01 INTO BC
000376 MOV C,L
000377 XCHG ; GET PARTIAL DIFFERENCE FROM DE
000378 CALL DFGHLC ; SHIFT 00 INTO HL
000379 CALL DFGHLC
000380 CALL DFGSUB ; SUBTRACT HL--BC IF POSSIBLE
000381 POP B ; RESTORE PARTIAL ROOT
000382
000383 JNC SORT2 ; NO CARRY IMPLIES NO ROUND OFF REQUIRED
000384
000385 INR B ; ELSE ROUND OFF UPWARD
000386
000425

```

Program A(8) (continued)

000426	MOV	A,B	, FINISHED, PUT ROOT IN A
000427	POP	H	, RETORE STATUS
000430	POP	D	
000431	POP	B	
000432	RET		, RETURN

Program A(8) (continued)

MULTI-BYTE BINARY ADDITION AND SUBTRACTION SUBROUTINES
 ;
 ; SUBROUTINES MADD AND MSUB PERFORM MULTI-BYTE, UNIGNED BINARY
 ; ADDITION AND SUBTRACTION ON TWO NUMBERS A AND B. THE TWO
 ; NUMBERS TO BE ADDED (OR SUBTRACTED) ARE STORED LOW-ORDER
 ; BYTE TO HIGH-ORDER BYTE BEGINNING AT (BC) AND (HL) RES-
 ; PECTIVELY. NUMBER B IS THE SUBTRAHEND FOR SUBTRACTION.
 ; THE RESULT IS STORED LOW-ORDER BYTE TO HIGH-ORDER
 ; BYTE IN THE LOCATION FORMERLY OCCUPIED BY THE
 ; NUMBER A. REGISTER E SPECIFIES THE NUMBER
 ; OF BYTES IN EACH NUMBER. EXECUTION TIME
 ; IS GIVEN BY: T=(E)56 + 25 STATES
 ; WHERE E IS THE # BYTES PER NUMBER
 ;

```

000000 MADD:      PUSH    PSW          ; SUBROUTINE ... MULTI-BYTE ADD
000001          PUSH    B           ; SAVE CPU STATUS
000002          PUSH    D
000003          PUSH    H
000004          XRA    A           ; CLEAR ACCUMULATOR AND CARRY BIT
000005          LDAX  B           ; LOAD BYTE OF FIRST NUMBER TO A
000006          ADC   M           ; ADD WITH CARRY BYTE OF SECOND NUMBER
000007          STAX  B           ; STORE SUM IN (BC)
000008          DCR   E           ; ADDITION FINISHED IF E=0
000009          JZ    EXIT        ; JUMP IF FINISHED
000010          INX   B           ; OTHERWISE POINT BC TO NEXT BYTE OF 1ST NUMBER
000011          INX   H           ; ALSO POINT HL TO NEXT BYTE OF 2ND NUMBER
000012          JMP  MADD+5      ; ADD NEXT TWO BYTES
000013
000014          147      000
000015          303      005      000
    
```

```

000021 MSUB:      PUSH    PSW          ; START OF MULTI-BYTE SUBTRACT ROUTINE
000022          PUSH    B           ; SAVE CPU STATUS
000023          PUSH    D
000024          PUSH    H
000025          XRA    A           ; CLEAR ACCUMULATOR AND CARRY BIT
000026          365
000027          305
000028          325
000029          345
000030          257
    
```

Program A(9) Multibyte Binary Addition and Subtraction Subroutines.

000026	LDAX	D	; LOAD BYTE OF MINJEND TO A
000026	012		
000027	CBX	M	; SUBTRACT BYTE OF SUBTRAHEND FROM A
000027	236		
000030	STAX	B	; STORE DIFFERENCE IN I(BC)
000030	002		
000031	DCR	E	; ALL DONE IF E=0
000031	025		
000032	JZ	EXIT	; JUMP IF FINISHED
000032	147		
000035	INX	D	; OTHERWISE POINT (BC) TO NEXT BYTE
000035	000		
000036	INX	H	; AND POINT (HL) TO NEXT BYTE
000036	043		
000037	JMP	MSUB+5	; SUBTRACT NEXT TWO BYTES
000037	303		
		026	000

Program A(9) (continued)

```

MULTI BYTE COMPARE SUBROUTINE
;
; SUBROUTINE MCOMP COMPARES TWO MULTI BYTE UNSIGNED BINARY NUMBERS
; A AND B WHICH ARE STORED LOW-ORDER BYTE TO HIGH-ORDER BYTE
; BEGINNING AT C0C0 AND C1D0 RESPECTIVELY. REGISTER E
; SPECIFICS THE NUMBER OF BYTES IN EACH NUMBER. THE
; RESULT OF THE COMPARISON IS RETURNED IN REGISTER
; D. FOR A<B, D=2. FOR A<=B, D=1. FOR A=B, D=0.
; TIMING HAS BEEN ADJUSTED SO THAT EXECUTION
; TIME DEPENDS ONLY ON THE NUMBER OF BYTES
; IN EACH NUMBER. EXECUTION TIME T IS:
; T = 110xE + 102 STATES WHERE E =
; NUMBER OF BYTES PER NUMBER
;
;
; START OF MULTI BYTE COMPARE ROUTINE
; SAVE MACHINE STATUS

```

ADDRESS	INSTRUCTION	OPERANDS	OPERATION	STATUS
000042	PUSH	FSW		
000042				
000042	PUSH	B		
000043				
000043	PUSH	D		
000044				
000044	PUSH	H		
000045				
000045	MVI	D,0		
000046				
000046	LDAX	B		
000050				
000050	CMP	M		
000051				
000051	JC	ALTM		
000052				
000052	JNZ	AGTM		
000055				
000055	MVI	A,0		
000060				
000060	CMP	D		
000062				
000062	JNZ	AGTM+5		
000063				
000063	MVI	D,0		
000066				
000066	JMP	AGTM+6		
000070				
000070				

Program A(10) Multibyte Compare Subroutine.

000073	ALTM:	MVI	D, 1		; AND SO SET D=1
000074		JMP	AGTM+2		
000075		MVI	D, 2		; AND SO SET D=2
000100	AGTM:	MOV	A, M		; DELAY 7 STATES. NEXT THREE INSTRUCTIONS
000102		DLY10			; ALSO PROVIDE DELAY TO EQUALIZE EXECUTION TIMES
000103		MOV	A, M		; DELAY 7 MORE STATES
000105		DLY10			
000106		DCR	E		; COMPARISON COMPLETE IF E=0
000107		JZ	MC1		; JUMP IF THROUGH
000114		INX	H		; OTHERWISE POINT (HL) TO NEXT BYTE OF NUMBER A
000115		INX	B		; POINT (BC) TO NEXT BYTE OF NUMBER B
000116		JMP	MCMP+6		; COMPARE NEXT TWO BYTES
000121	MC1:	POP	H		; RESTORE MACHINE STATE EXCEPT FOR
000122		MOV	B, D		; FLAG REGISTER E
000123		POP	D		
000124		MOV	D, B		
000125		POP	B		
000126		POP	PCW		
000127		RET			; RETURN TO CALLING POINT
000128					

Program A(10) (continued)

```

MULTI BYTE MEMORY-TO-MEMORY MOVE
; SUBROUTINE MOVE DUPLICATES A BLOCK OF MEMORY BEGINNING AT <BC> ;
; IN A BLOCK OF MEMORY BEGINNING AT <HL>. REGISTER E SPECIFIES ;
; THE NUMBER OF BYTES IN THE BLOCK. EXECUTION TIME T = ;
; T = 49RE + 9I STATES WHERE E = NUMBER OF BYTES IN BLOCK. ;

MOVE:          ; STARTING POINT OF MOVE BLOCK ROUTINE
              ; SAVE CPU STATUS
              PUSH PSW
              PUSH B
              PUSH D
              PUSH H
              LDAX B          ; LOAD BYTE OF DATA BLOCK
              MOV M,A        ; MOVE TO NEW MEMORY LOCATION
              DCR E          ; MOVE COMPLETE IF E=0
              JZ EXIT        ; RETURN IF THROUGH
              INX B          ; OTHERWISE POINT (BC) TO NEXT BYTE
              INX H          ; POINT (HL) TO NEXT BYTE OF NEW BLOCK
              JMP MOVE+4    ; MOVE THE NEXT BYTE

000130
000130
000130
000131
000131
000132
000132
000133
000133
000134
000134
000135
000135
000136
000136
000137
000137
000142
000142
000143
000143
000144
000144

          365
          305
          325
          345
          012
          167
          035
          312
          000
          003
          043
          303

          147
          000
          134
          000

```

Program A(11) Multibyte Memory-to-Memory Move Subroutine.

APPENDIX B

A NEW PRINCIPLE FOR FAST FOURIER TRANSFORMATION

APPENDIX B

A NEW PRINCIPLE FOR FAST FOURIER TRANSFORMATION

by

C. M. Rader, N. M. Brenner

Let $\{a_n\}$ be a sequence of $N = 2^m$ data, whose Discrete Fourier Transform is $\{A_k\}$. Let $W = \exp(-j2\pi/N)$. Present FFT algorithms are derived from an equation like (1) or its dual*

$$A_k = \text{DFT}\{a_{2n}\} + W^k * \text{DFT}\{a_{2n+1}\} . \quad (1)$$

Each of the DFT's in (1) is a DFT of a half-length data sequence, and can be expressed as two still shorter DFT's. After m such stages of simplification an algorithm is evident which requires $5(N \log_2 N)$ operations to execute.

This note presents an alternative to Equation (1) which may similarly be applied iteratively to itself leading to an FFT algorithm. The new algorithm which so results has the peculiarity that none of the multiplying constants is complex. Its advantages would therefore seem to be most pronounced in systems for which multiplications are most costly.

*Equation (1) is too specialized, leading to radix-2 algorithms. It portrays the essence of the derivation, however.

Derivation

Let $\{b_n\}$ and $\{c_n\}$ be the sequences

$$\begin{aligned} b_n &= a_{2n} \\ c_n &= a_{2n+1} - a_{2n-1} + Q \end{aligned} \quad n = 0, 1, \dots, N/2 - 1 \quad (2)$$

where

$$Q = \frac{2}{N} \sum_{n=0}^{\frac{N}{2}-1} a_{2n+1}$$

The $\frac{N}{2}$ points DFT's of these sequences are $\{B_k\}$ and $\{C_k\}$. It is helpful to consider the sequence $\{d_n\}$ and its DFT, $\{D_k\}$ defined by

$$d_n = a_{2n+1} \quad n = 0, 1, \dots, N/2 - 1 \quad (3)$$

so that $\{B_k\}$ and $\{D_k\}$ are the DFT's in Equation (1).

C_k and D_k can be simply related. Since Q is a constant, it appears in C_k in only one term, C_0 . Furthermore, C_0 is exactly equal to D_0 . For other values of k , C_k can be expressed by the circular shifting theorem for DFT.

$$\begin{aligned} C_k &= D_k(1 - W^{2k}) \\ &= W^{-k} D_k (W^k - W^{-k}) \end{aligned} \quad k = 1, 2, \dots, \frac{N}{2} - 1 \quad (4)$$

Hence,

$$W^k D_k = C_k / (W^k - W^{-k}) \quad k \neq 0 \quad (5)$$

and we can rewrite Equation (1)

$$A_k = B_k + C_k / (W^k - W^{-k})$$

$$k = 1, 2, \dots, \frac{N}{2} - 1, \frac{N}{2} + 1, \frac{N}{2} + 2, \dots, N - 1 \quad (6a)$$

$$\left. \begin{aligned} A_0 &= B_0 + C_0 \\ A_{N/2} &= B_{N/2} - C_{N/2} \end{aligned} \right\} \quad (6b)$$

Since W^k and W^{-k} are complex conjugates, their difference is twice the imaginary part, e.g.,

$$W^k - W^{-k} = -2j \sin 2\pi k/N$$

so that (6a) may be written

$$A_k = B_k + \frac{1}{2}j \csc\left(\frac{2\pi}{N} k\right) C_k \quad k \neq 0, N/2 \quad (6a)$$

Equations (6a) and (6b) are the replacements for Equation (1) promised. A fast Fourier Transform algorithm based on (6a) and (6b) requires only multiplication by pure imaginary constants.

Comments

$\frac{N}{2}Q$ could have been added to the output terms $A_0, A_{N/2}$ rather than to each of $\frac{N}{2}$ input terms as in Equation (2) but this would have made the substitution of Equation (2) recursively into itself, to produce an FFT algorithm, a difficult matter. In hardware implementation, e.g., "pipelines" this may not be a consideration.

If, in Equation (2) we had used $a_{2n+1} + a_{2n-1}$, the minus sign would change to + in Equations (4), (5), and (6a), thus changing $\frac{1}{2}j \csc(2\pi k/N)$ to $\frac{1}{2} \sec(2\pi k/N)$. The exceptional cases for k would then be $k = N/4, 3N/4$ and these would involve $\pm j$, the only non-real operation encountered. We judge that the secant form offers no advantages over the cosecant form.

Whereas the constants W^k used in Equation (1) have unity magnitude, $\csc(2\pi k/N)$ can get very large. Therefore, small computation errors can lead to large output errors. Experience verifies that this is the case. We recommend that the method be modified if more than 8192 points are used. A conventional factoring can reduce a DFT to shorter DFT's and each of these can be computed by the cosecant method.

Substantial savings in multiplications can be made in the conventional FFT by deriving the algorithm in a higher radix. The method proposed here can also be developed for radices other than two. The $(p-1)$ -sequences to be formed are of the form $c_n^{(q)} = a_{pn+q} - a_{pn-q} + Q_q$. We have no idea whether computational savings can result from higher radix methods.

APPENDIX C

8080A FAST FOURIER TRANSFORM PROGRAM


```

000002 041 351 004 MVI M,LOG2N
000005 046 004
000007 041 351 004
000010 041 351 004
000012 041 351 004
000013 041 351 004
000014 041 351 004
000016 041 351 004
000017 041 351 004
000018 041 351 004
000019 041 351 004
000020 041 351 004
000021 041 351 004
000022 041 351 004
000023 041 351 004
000024 041 351 004
000025 041 351 004
000026 041 351 004
000027 041 351 004
000028 041 351 004
000029 041 351 004
000030 041 351 004
000031 041 351 004
000032 041 351 004
000033 041 351 004
000034 041 351 004
000035 041 351 004
000036 041 351 004
000037 041 351 004
000038 041 351 004
000039 041 351 004
000040 041 351 004
000041 041 351 004
000042 041 351 004
000043 041 351 004
000044 041 351 004
000045 041 351 004
000046 041 351 004
000047 041 351 004
000048 041 351 004
000049 041 351 004
000050 041 351 004
000051 041 351 004
000052 041 351 004
000053 041 351 004
000054 041 351 004
000055 041 351 004
000056 041 351 004
000057 041 351 004
000058 041 351 004
000059 041 351 004
000060 041 351 004
000061 041 351 004
000062 041 351 004
000063 041 351 004
000064 041 351 004
000065 041 351 004
000066 041 351 004
000067 041 351 004
000068 041 351 004
000069 041 351 004
000070 041 351 004
000071 041 351 004
000072 041 351 004
000073 041 351 004
000074 041 351 004
000075 041 351 004
000076 041 351 004
000077 041 351 004
000078 041 351 004
000079 041 351 004
000080 041 351 004
000081 041 351 004
000082 041 351 004
000083 041 351 004
000084 041 351 004
000085 041 351 004
000086 041 351 004
000087 041 351 004
000088 041 351 004
000089 041 351 004
000090 041 351 004
000091 041 351 004
000092 041 351 004
000093 041 351 004
000094 041 351 004
000095 041 351 004
000096 041 351 004
000097 041 351 004
000098 041 351 004
000099 041 351 004
000100 041 351 004

```

Program C(1) (continued)


```

000154 LXI H,PREV+1
000154 SUB M ;A ← REAL(TEMP-PREV)
000157 LXI H,TEMP+1
000160 MOV M,A ;HL POINTS TO IMAG(TEMP)
000163 DCX H ;A ← IMAG(TEMP)
000164 MOV A,M
000165 LXI H,PREV
000166 SUB M
000171 LXI H,TEMP
000172 MOV M,A
000175 ;PREV=DATA(J)
000175 LPHLD DJVAL
000176 SHLD PREV
000201 ;DATA(J)=TEMP
000204 LPHLD TEMP
000204 PUSH D
000207 XCHG
000210 LPHLD DJADDR
000211 MOV M,D
000214 INX H
000215 MOV M,E
000216 POP D
000217 ;J=J+NZ
000217 MOV A,D
000220 ADD E
000221 MOV C,A
000222 ;IF (J-N3=1, LE, 0) GO TO 20
000222 DCR A
000223 CPI N3
000224 JM $20
000226 JZ $20
000231

```

Program C(1) (continued)

```

000231 012 SUM=SUM/2#*K
000234 PUSH E
000235 PUSH D
000236 LHL D
000237 XCHG
000240 LHL D
000241 MOV B,H
000242 MOV C,L
000243 LXI H,K
000244 MOV H,M
000245 DCR H
000246 JMP OTLOOP
000247 MOV A,B
000248 ANA A
000249 RRC
000250 MOV B,A
000251 MOV A,C
000252 RAR
000253 MOV C,A
000254 MOV A,D
000255 ANA A
000256 RRC
000257 MOV D,A
000258 MOV A,E
000259 RAR
000260 MOV E,A
000261 JMP LOOP25
000262 XCHG
000263 SHLD SUM
000264 OTLOOP: XCHG
000265 SHLD SUM
000266
000267
000268
000269
000270
000271
000272
000273
000274
000275
000276
000277
000278
000279
000280
000281
000282
000283
000284
000285
000286
000287
000288
000289
000290
000291
000292
000293
000294
000295
000296
000297
000298
000299
000300
000301

```

Program C(1) (continued)

```

000301 042 020 005 MOV H, B
000304 140 MOV L, C ; HL ← IMAG(SUM)/2**K
000305 151 SHLD SUM+2
000306 042 022 005 POP D
000311 321 POP B ; RESTORE REGISTERS
000312 301
; J=1
000313 110 MOV C, B
000314 325 ; 30 DATA(J)-DATA(J)+SUM
; 30: PUSH D ; SAVE N1 & N2
000315 041 020 005 LXI H, SUM
000320 176 MOV A, M ; A ← REAL(SUM)
000321 365 PUSH PSW ; STORE A
000322 041 040 005 POP PSW ; RESTORE A
000325 175 ADD H ; A ← REAL(DATA(J)+SUM)
000326 201 MOV D, A ; SAVE REAL RESULT
000327 157 MOV A, L ; A ← IMAG(DATA(J))
000330 322 LXI H, SUM+2
000333 044 026 005 ADD M ; A ← IMAG(DATA(J)+SUM)
000334 042 026 005 LHD DJADDR ; HL ← DATA+(J)
000337 176 MOV M, D
000340 043 INX H
000341 156 MOV M, A
000342 147 MOV M, A
000343 042 034 005 POP PSW
000346 361 ADD H ; A ← REAL(DATA(J)+SUM)
000347 204 MOV D, A ; SAVE REAL RESULT
000350 127 MOV A, L ; A ← IMAG(DATA(J))
000351 175 LXI H, SUM+2
000352 041 022 005 ADD M ; A ← IMAG(DATA(J)+SUM)
000355 206 LHD DJADDR ; HL ← DATA+(J)
000356 052 026 005 MOV M, D
000361 162 INX H
000362 043 MOV M, A
000363 167

```

Program C(1) (continued)

```

000364 POP D ;RESTORE N1 & N2
000364
000365 ;J=J+N2
000365 MOV A,C
000366 ADD E ;A C= J+N2
000366 MOV C,A
000367
000370 ;IF (J-N3<1, LE, 0) GO TO 30
000370 DCR A
000370 CPI N3
000371 JM S30
000372 JZ S30
000376 ;25 1=1+1
000376 INR B ;2BYTES/CMPLX. VALUE
00401 INR B
00401 INR B
00402 ;IF (1-N2, LE, 0) GO TO 15
00402 MOV A,B
00403 CMP E
00404 JM S15
00405 JZ S15
00410 ;30 N1=N2
00410 MOV D,E
00413 ;GO TO 10
00413 JMP S10
00414 ; PERMUTATION BY BIT REVERSAL
00414 ; REGISTER B HOLDS THE VARIABLE I
00414 ; REGISTER C HOLDS THE VARIABLE J
00414 ; REGISTER D HOLDS THE VARIABLE N1
00414 ; REGISTER E HOLDS THE VARIABLE N2
00417 ;40 I=1
00417 MVI B,2
00421 ;J=1
00421 MOV C,B
00422 ;45 IF (I-J, GE, 0) GO TO 50
00422 MOV A,B
00423 CMP C
00424 JF S50

```

Program C(1) (continued)


```

000524 021
000525 076 040
000526 247
000527 037
000528 127
000529 171
000530 272
000531 372 164 001
000532
000533
000534
000535
000536
000537
000538
000539
000540
000541
000542
000543
000544
000545
000546
000547
000548
000549
000550
000551
000552
000553
000554
000555
000556
000557
000558
000559
000560
000561
000562
000563
000564
000565
000566
000567

; N1=(N0+1)/2
MVI A,N0+1
ANA A
RAR
MOV D,A
; IF (J=N1.LT.O) GO TO 60
MOV A,C
CMP D
JM 080
J=J-N1
SUB D
MOV C,A
; N1=N1/2
MOV A,D
ANA A
RAR
RAR
JNC 075
RAL
INR A
JMP 076
RAI
MOV D,A
; IF (N1-1.GE.O) GO TO 60
CFI S
JP 060
J=J+1
MOV A,C
ADD D
MOV C,A
; I=I+1
INR B
; A ← N0+1
; RESET CARRY
; A ← A/2
; RESET CARRY
; A ← A/2
; RESTORE VALUE
; A ← A+1
; RESTORE VALUE
; N1 IS IN A

```

Program C(1) (continued)

```

000567 004 INR B
000570 004 ; IF (I-N3-1, LT, 0) GO TO 45
000570 004 MOV A, B
000571 170 DCR A
000572 075 CPI N3
000573 376 JM 345
000575 372 022 001

;
;
; 2 POINT FOURIER TRANSFORMS & PHASE SHIFTS
; REGISTER B HOLDS THE VARIABLE I
; REGISTER C HOLDS THE VARIABLE J
; REGISTER D HOLDS THE VARIABLE N1
; REGISTER E HOLDS THE VARIABLE N2
; REGISTER H HOLDS THE VARIABLE M1
; REGISTER L HOLDS THE VARIABLE N0
; 190 K=0
; 190 LXI H, K
; 190 MVI M, 0
; M2=1
; 190 INX H
; M2 FOLLOWS K IN MEMORY
; 190 MVI M, 2
; 2 BYTES/ COMPLEX VALUE
; M1=0
; 190 MVI M, 0
; 100 IF (M2-N3-1, GT, 0) GO TO 150
; 100 PUSH H
; 190 LXI H, M2
; 190 MOV A, M
; A ← (M2)
; 190 DCR A
; 190 DCR A
; 190 CPI N3
; 190 POP H
; 190 JP MATUDE
; FFT FINISHED. FIND MAGNITUDES.
; IF (M1, EQ, 0) GO TO 110
; 190 MOV A, H
; 190 ANA A
; 190 JZ 3110
;
000600 041 351 004
000600 066 000
000605 043
000606 066 002
000610 046 000
000612 345
000613 041 352 004
000616 176
000617 075
000620 075
000621 376 037
000623 341
000624 362 142 003
000627 174
000630 247
000630 000631

```

Program C(1) (continued)

```

000631 312 243 001
;105 K=K+1
;105: PUSH H
000634 345 LXI H,K
000635 041 351 004 INR M
000640 064 INR M
000641 064 POP H
000642 341 ;110 NO=M1+1
;110: MOV A,H
000643 174 ADI 2
000644 306 002 MOV L,A
000646 157 ;N1=M2
;N1=M2
;110: PUSH H
;110: ;SAVE M1 & NO
000647 345 LXI H,M2
000650 041 352 004 MOV D,M
000653 126 POP H
000654 341 ;120 N2=2*N1
;120: MOV A,D
000655 172 ADD A
000656 207 MOV C,A
000657 137 ; IF(N1-NO-1.GE.0) GO TO 140
; IF(N1-NO-1.GE.0) GO TO 140
;120: MOV A,D
000660 172 DCR A
000661 075 CPI NO
000662 376 JF S140
000664 362 106 002 ; I3=NO
; I3=NO
;120: MOV A,L
000667 175 PUSH H
000670 345 LXI H,I3
000671 041 353 004 MOV M,A
000674 167 POP H
000675 341 ;125 I=I3
;125: PUSH H
;125: ; I3 <- NO
;125:
000676

```

Program C(1) (continued)

```

000676 LXI H,13
000677 MOV B,M
000702 POP H
000703
000704 J=1+N1
000705 MOV A,B
000706 ADD D
000707 MOV C,A
000708
000709 ;TEMP=DATA(J)
000710 PUSH H
000711 ;SAVE REGISTERS
000712
000713 CPXTEL DATA,C,DJADDR,DJVAL ;HL ← (DATA(J))
000714
000715 SHLD TEMP
000716 POP H
000717 ;RESTORE REGISTERS
000718
000719 ;IF(M1.EQ.0) GO TO 105
000720 MOV A,H
000721 ANA A
000722 PUSH H
000723 ;AFFECTS FLAGS
000724 PUSH D
000725 JZ S135
000726
000727 ;S132 TEMP=(O.1)*TEMP*TABLE(K)
000728 S132: PUSH B
000729 LHLD TEMP
000730 ;HL ← (TEMP)
000731 SUB A
000732 ;A ← 0
000733 SUB L
000734 ;L ← -IMAG(TEMP)
000735 MOV D,A
000736 ;DE ← TEMP*(O.1)
000737 MOV E,H
000738
000739
000740
000741
000742
000743
000744
000745
000746
000747
000748
000749
000750
000751
000752
000753
000754
000755
000756

```

```

000756 104 CALL MULCON ;TEMP=DEXTABLE(K)
000757 015 POP B
000758 002
000759 001
;135 DATA(J)=DATA(I)-TEMP
;135: LHL D TEMP ;HL ←- TEMP
;136: XCHG ;DE ←- TEMP
000760 052 024 005
000761 050
000762 041 040 005
000763 175
000764 200
000765 157
000766 022 001 002
000767 044
000768 042 030 005
000769 176
000770 043
000771 156
000772 147
000773 042 036 005
000774 174
000775 222
000776 001015
000777 001016
000778 001017
000779 001018
000780 001019
000781 001020
000782 001021
000783 001022
000784 001023
000785 001024
000786 162
000787 043
000788 160
000789 001021
000790 001031
000791 001032
000792 001033
000793 172
000794 001036
000795 204
000796 001037
;135: DATA(I)=DATA(I)+TEMP
;135: LHL D DIVAL ;HL ←- (DATA(I))
;136: MOV A,H
;137: SUB D ;A ←- REALL(DATA(A))-TEMP
;138: MOV H,A
;139: MOV A,L
;140: SUB E ;A ←- IMAGE(DATA(I))-TEMP
;141: MOV L,A ;HL ←- (DATA(I))-TEMP
;142: PUSH D
;143: XCHG
;144: LHL D DJADDR
;145: MOV M,D
;146: INX H
;147: MOV M,E
;148: POP D
;149: DATA(I)=DATA(I)+TEMP
;149: LHL D DIVAL ;HL ←- (DATA(I))
;150: MOV A,D
;151: ADD H
;152: MOV H,A

```

```

001040 MOV A,E
001040 173 ADD L
001041 209 MOV L,A
001042 157 XCHG
001043 350 LHL D,ADDR
001044 052 030 005 MOV M,D
001047 162 INX H
001050 043 MOV M,E
001051 163 POP D
001052 321 POP H
001053 341
001054 IF(MI.NE.O) GO TO 160
001054 174 MOV A,H
001055 247 ANA A
001056 302 142 002 JNZ $160
; I=I+NZ
001061 MOV A,B
001061 170 ADD C
001062 203 MOV B,A
001063 107
; IF(I-N3=1,LE.O) GO TO 130
001064 DCR A
001064 075 DCR A
001065 075 CPI N3
001066 376 037 JM $130
001070 372 304 001
; I30 NO=2*NO-1
; I30: MOV A,L
001073 175 ADD A
001074 207 SUI 2
001075 326 002 MOV L,A
001077 157
; N1=2*N1
001100 MOV A,D
001100 172 ADD A

```

; RESTORE REGISTERS

; A ← (M1)

; SET FLAG

; A ← 2*A

Program C(1) (continued)

```

001102 MOV D,A
001102
001103 ; GO TO 120
001103 JMP $120
001104 ; 140 M1=M1+2
001104 MOV A,H
001107 ADI 4
001107 MOV H,A
001111 ; IF(2*M1-M2, LE, 0) GO TO 105
001111 ADD A ; A=2*M1
001112 DCR A
001113 PUSH H
001114 LXI H,M2
001115 CMP M ; 2*M1-M2
001120 POP H
001121 JM $105
001122 ; 142 M1=1
001125 ; 144: MV1 H,2
001125 ; 145 M2=2*M2
001127 ; 145: PUSH H
001130 LXI H,M2
001130 MOV A,M
001133 ADD A
001134 MOV M,A
001135 POP H
001136 ; GO TO 100
001137 JMP $100
001137 ; 160 Y=N1+2*(1+13+NO)
001142 ; 160: PUSH H
001142 LXI H,13
001143 MOV A,M
001143 ; A C... 13
001146 POP H
001147 INR A
001150

```

Program C(1) (continued)


```

001150 074 INR A ;A ← I3+1
001151 074 SUB L ;A ← I3+1-NO
001152 225 ADD A ;A ← 2*(I3+1-NO)
001153 207 ADD D ;A ← N1+2*(I3+1-NO)
001154 202 SUB B ;A ← N1+2*(I3+1-NO)-1
001155 220 MOV B,A
001156 107 ; IF(I-13.GT.0) GO TO 130
                                PUSH H
001157 345 LXI H,I3
001160 041 CMP M ;A=1, (M)=I3
                                004
001163 276 POP H
001164 341 JZ $165
001165 312 JP $130
001170 362 ; I3=I3+N2
                                304 001
001173 345 $165: PUSH H
001174 041 LXI H,I3
                                004
001177 176 MOV A,M ;A ← I3
001200 203 ADD E ;A ← I3+N2
001201 167 MOV M,A
001202 341 POP H
                                IF (I3-N3-1.LE.0) GO TO 125
                                DCR A
001203 075 DCR A
001204 075 CPI N3
001205 376 JM $125
001207 372 ; GO TO 130
                                001
                                JMP $130
001212 303 ; SUBROUTINE MULCON(TEMP,K)
                                073 002
                                ; MULCON IS A SPECIAL ROUTINE FOR MULTIPLYING A NUMBER IN
                                ; 'TEMP' BY A CONSTANT. THIS IS ACCOMPLISHED BY A SHIFTING PROCESS
                                ; AND SUCCESSIVE ADDITION. FOR EXAMPLE, SUPPOSE CONSTANT=0.81.
                                ; SHIFTING 'TEMP' RIGHT TWO TIMES YIELDS 0.25*TEMP. IF THIS SHIFTED
                                ; VALUE IS THEN SUBTRACTED FROM THE ORIGINAL VALUE, 0.75*TEMP.

```



```

001264 207 ADD A ; A ← 4*REAL(TEMP)
001265 207 ADD D ; A ← A+REAL(TEMP)
001266 202 MOV B,A
001267 107 MOV A,E
001270 173 ADD A
001271 207 ADD A
001272 207 ADD D ; A ← 5*IMAG(TEMP)
001273 202 MOV C,A
001274 117 ;
001275 046 ; 220 NTERMS=3 ; H=NTERMS
001275 003 ; 225 NTERMS=NTERMS-1 ; 1ST LOOP=REAL, 2ND=IMAG
001277 045 DCR H ;
; IF (NTERMS.EQ.0) GO TO 250
; TKVAL=TABLE(K)
; PUSH H
LHLD TKADDR
MOV A,M ; A ← TABLE(K)
POP H
; IF (TKVAL.LT.0) GO TO 230
MOV L,A ; L ← TABLE(K), L=TKVAL
ANA A
JM 0230
; IF (TKVAL.EQ.0) GO TO 250
; TMPROD=TMPROD+TEMP/2**TKVAL
; PUSH H
MOV H,L ; PUT TKVAL IN H AND L
MOV A,D ; A ← REAL(TEMP)
DCR H ; H IS REAL LOOP COUNTER
JM CTNDA ; EXIT AFTER TKVAL LOOPS
CFI H-1 ; TEST FOR A=-1
001300 312 134 003
001300 312 134 003
001303 345
001304 052 032 005
001307 176
001310 341
001311 157
001312 247
001313 372 022 003
001316 312 134 003
001316 312 134 003
001321 345
001322 145
001323 172
001324 045
001325 372 353 002
001330 001330

```

Program C(1) (continued)


```

001416 341
001417
001417 303 277 002
; GO TO 225
; JMP 225
; 230 TIKVAL=COMPLEMENT (TIKVAL)
; 230: MOV A, L
; CMA
; MOV L, A
; TMRD=TMPROD-TEMP/2**TIKVAL
; PUSH H
; PUT TIKVAL IN H AND L
MOV H, L
MOV A, D
LOOPDD: DCR H
JM CTNUAA
CFI ...1
JNZ LM11A
MVI A, O
JMP CTNUAA
LM11A: ANA A
JP LM11
STC
RAR
JMP LOOPDD
CTNUAA: SUB B
MOV B, A
MOV A, E
LOOPEE: DCR L
JM CTNUBB
CFI ...1
JNZ LM22A
MVI A, O
JMP CTNUBB

```



```

001651 156      034      005      MOV      A, H      ; A ← REAL(DATA(N))
001652 147      ANA     A      ; AFFECT SIGN BIT
001653 042      JP      S300
001654 174      CMA     A
001655 247      INR     A
001656 362      MOV     C, A      ; A ← A
001657 057      MOV     A, L      ; A ← ABS(REAL(DATA(N)))
001658 074      ANA     A      ; TEST SIGN BIT
001659 117      JP      S310
001660 175      CMA     A
001661 247      INR     A
001662 362      MOV     L, A      ; L ← ABS(IMAG(DATA(N)))
001663 057      ANA     A
001664 074      MOV     A, L      ; TEST SIGN BIT
001665 157      JP      S310
001666 247      CMA     A
001667 057      INR     A
001668 074      MOV     L, A      ; L ← ABS(REAL(DATA(N)))
001669 157      ANA     A      ; TEST FOR BOTH VALUES = 0.
001670 247      JP      S315      ; IS ABS(IMAG)=0?
001671 302      CMP     C
001672 271      JNZ     S315
001673 302      MVI     H, 0
001674 046      MOV     L, H      ; BOTH VALUES 0
001675 154      JMP     S455      ; HL ← 0
001676 303      ;
001677 271      ; FIND LARGER VALUE. MOVE LARGER INTO HL, SMALLER INTO C.
001678 302      CMP     C
001679 046      ;
001680 154      ;
001681 303      ;
001682 271      ;
001683 302      ;
001684 046      ;
001685 154      ;
001686 303      ;
001687 271      ;
001688 302      ;
001689 046      ;
001690 154      ;
001691 303      ;
001692 271      ;
001693 302      ;
001694 046      ;
001695 154      ;
001696 303      ;
001697 271      ;
001698 302      ;
001699 046      ;
001700 154      ;
001701 303      ;
001702 271      ;
001703 302      ;
001704 046      ;
001705 154      ;
001706 303      ;
001707 271      ;
001708 302      ;
001709 046      ;
001710 154      ;
001711 303      ;
001712 271      ;
001713 302      ;
001714 046      ;
001715 154      ;
001716 303      ;
001717 271      ;
001718 302      ;
001719 046      ;
001720 154      ;
001721 303      ;
001722 271      ;
001723 302      ;
001724 046      ;
001725 154      ;
001726 303      ;
001727 271      ;
001728 302      ;
001729 046      ;
001730 154      ;
001731 303      ;
001732 271      ;
001733 302      ;
001734 046      ;
001735 154      ;
001736 303      ;
001737 271      ;
001738 302      ;
001739 046      ;
001740 154      ;
001741 303      ;
001742 271      ;
001743 302      ;
001744 046      ;
001745 154      ;
001746 303      ;
001747 271      ;
001748 302      ;
001749 046      ;
001750 154      ;
001751 303      ;
001752 271      ;
001753 302      ;
001754 046      ;
001755 154      ;
001756 303      ;
001757 271      ;
001758 302      ;
001759 046      ;
001760 154      ;
001761 303      ;
001762 271      ;
001763 302      ;
001764 046      ;
001765 154      ;
001766 303      ;
001767 271      ;
001768 302      ;
001769 046      ;
001770 154      ;
001771 303      ;
001772 271      ;
001773 302      ;
001774 046      ;
001775 154      ;
001776 303      ;
001777 271      ;
001778 302      ;
001779 046      ;
001780 154      ;
001781 303      ;
001782 271      ;
001783 302      ;
001784 046      ;
001785 154      ;
001786 303      ;
001787 271      ;
001788 302      ;
001789 046      ;
001790 154      ;
001791 303      ;
001792 271      ;
001793 302      ;
001794 046      ;
001795 154      ;
001796 303      ;
001797 271      ;
001798 302      ;
001799 046      ;
001800 154      ;
001801 303      ;
001802 271      ;
001803 302      ;
001804 046      ;
001805 154      ;
001806 303      ;
001807 271      ;
001808 302      ;
001809 046      ;
001810 154      ;
001811 303      ;
001812 271      ;
001813 302      ;
001814 046      ;
001815 154      ;
001816 303      ;
001817 271      ;
001818 302      ;
001819 046      ;
001820 154      ;
001821 303      ;
001822 271      ;
001823 302      ;
001824 046      ;
001825 154      ;
001826 303      ;
001827 271      ;
001828 302      ;
001829 046      ;
001830 154      ;
001831 303      ;
001832 271      ;
001833 302      ;
001834 046      ;
001835 154      ;
001836 303      ;
001837 271      ;
001838 302      ;
001839 046      ;
001840 154      ;
001841 303      ;
001842 271      ;
001843 302      ;
001844 046      ;
001845 154      ;
001846 303      ;
001847 271      ;
001848 302      ;
001849 046      ;
001850 154      ;
001851 303      ;
001852 271      ;
001853 302      ;
001854 046      ;
001855 154      ;
001856 303      ;
001857 271      ;
001858 302      ;
001859 046      ;
001860 154      ;
001861 303      ;
001862 271      ;
001863 302      ;
001864 046      ;
001865 154      ;
001866 303      ;
001867 271      ;
001868 302      ;
001869 046      ;
001870 154      ;
001871 303      ;
001872 271      ;
001873 302      ;
001874 046      ;
001875 154      ;
001876 303      ;
001877 271      ;
001878 302      ;
001879 046      ;
001880 154      ;
001881 303      ;
001882 271      ;
001883 302      ;
001884 046      ;
001885 154      ;
001886 303      ;
001887 271      ;
001888 302      ;
001889 046      ;
001890 154      ;
001891 303      ;
001892 271      ;
001893 302      ;
001894 046      ;
001895 154      ;
001896 303      ;
001897 271      ;
001898 302      ;
001899 046      ;
001900 154      ;
001901 303      ;
001902 271      ;
001903 302      ;
001904 046      ;
001905 154      ;
001906 303      ;
001907 271      ;
001908 302      ;
001909 046      ;
001910 154      ;
001911 303      ;
001912 271      ;
001913 302      ;
001914 046      ;
001915 154      ;
001916 303      ;
001917 271      ;
001918 302      ;
001919 046      ;
001920 154      ;
001921 303      ;
001922 271      ;
001923 302      ;
001924 046      ;
001925 154      ;
001926 303      ;
001927 271      ;
001928 302      ;
001929 046      ;
001930 154      ;
001931 303      ;
001932 271      ;
001933 302      ;
001934 046      ;
001935 154      ;
001936 303      ;
001937 271      ;
001938 302      ;
001939 046      ;
001940 154      ;
001941 303      ;
001942 271      ;
001943 302      ;
001944 046      ;
001945 154      ;
001946 303      ;
001947 271      ;
001948 302      ;
001949 046      ;
001950 154      ;
001951 303      ;
001952 271      ;
001953 302      ;
001954 046      ;
001955 154      ;
001956 303      ;
001957 271      ;
001958 302      ;
001959 046      ;
001960 154      ;
001961 303      ;
001962 271      ;
001963 302      ;
001964 046      ;
001965 154      ;
001966 303      ;
001967 271      ;
001968 302      ;
001969 046      ;
001970 154      ;
001971 303      ;
001972 271      ;
001973 302      ;
001974 046      ;
001975 154      ;
001976 303      ;
001977 271      ;
001978 302      ;
001979 046      ;
001980 154      ;
001981 303      ;
001982 271      ;
001983 302      ;
001984 046      ;
001985 154      ;
001986 303      ;
001987 271      ;
001988 302      ;
001989 046      ;
001990 154      ;
001991 303      ;
001992 271      ;
001993 302      ;
001994 046      ;
001995 154      ;
001996 303      ;
001997 271      ;
001998 302      ;
001999 046      ;

```

Program C(1) (continued)


```

001727 051  DAD  H
001730 051  DAD  H
001731 051  DAD  H
001732 053  XCHG
001733 171  MOV  A,C
001734 247  ANA  A
001735 302  JNZ  DIVIDE
001740 052  LLD  LARGER
001743 303  JMP  $450
001746 006  DIVIDE: MVI B,2
001747 006  MOV  A,D
001750 172  MOV  D,A
001751 127  MOV  A,E
001752 173  RAL
001753 027  MOV  E,A
001754 137  DCR  B
001755 005  JZ   $340
001756 312  MOV  A,D
001761 172  RAL
001762 027  SUB  C
001763 221  JNC  $330
001764 322  ADD  C
001767 201  JMP  $330
001770 303  MOV  A,E
001773 173  CMA
001774 057  ANA  A
001775 247  JM   $450
001776 372  PUSH PSW
002001 365  POP  PSW
002001 365  POP  PSW

; TEST FOR SMALLER VALUE = 0. IF SO, MAGNITUDE = LARGER VALUE.
; A ← SMALLER VALUE
; B ← LARGER VALUE
; AFFECT O FLAG
; HL ← LARGER VALUE
; HL ← LARGER VALUE
; HL ← LARGER VALUE
; ROTATE CARRY INTO E. ROTATE
; NEXT MOST SIG. BIT TO CARRY
; ROTATE MOST SIG. BIT TO
; HIGH ORDER QUOTIENT
; SUBTRACT DIVISOR. IF < HIGH
; ORDER QUOTIENT, GO TO $330
; ELSE, ADD BACK
; A ← QUOTIENT
; TEST SIGN BIT
; QUOTIENT > 1 BYTE, MAG = LARGER VALUE

```

Program C(1) (continued)

002037	372	320	004	JZ	S460	
002042	312	320	004	MOV	A,C	
002045	171			ADD	D	
002046	202			MOV	C,A	; C=2.5E
002047	117			JMP	S460	
002050	303	320	004	MOV	A,C	
002053	171			ADD	E	
002054	203			MOV	C,A	; C=3E
002055	117			MOV	A,B	
002056	170			CFI	29.	; SHOULD MAG BE > 3.5?
002057	376	035		JF	S390	
002061	362	102	004	CFI	25.	; SHOULD MAG BE = 3.5?
002064	376	031		JM	S460	
002066	372	320	004	JZ	S460	
002071	312	320	004	MOV	A,C	
002074	171			ADD	D	
002075	202			MOV	C,A	
002076	117			JMP	S460	
002077	303	320	004	MOV	A,C	
002102	171			ADD	E	
002103	203			MOV	C,A	; C=4E
002104	117			MOV	A,B	
002105	170			CFI	37.	; SHOULD MAG BE > 4.5?
002106	376	045		JF	S390	
002110	362	131	004	CFI	33.	; SHOULD MAG BE = 4.5?
002113	376	041		JM	S460	
002115	372	320	004	JZ	S460	
002120	312	320	004	MOV	A,C	
002123	171					

Program C(1) (continued)

000	004	0410:	MOV	A,C	
002207	171		ADD	L	
002210	203		MOV	C,A	;C=7E
002211	117		MOV	A,B	
002212	170		CFI	61	; SHOULD MAG BE > 7.5?
002213	376		JP	0430	
002215	362	004	CFI	57	; SHOULD MAG BE = 7.5?
002220	376		JM	0460	
002222	372	004	JZ	0460	
002225	312	004	MOV	A,C	
002229	171		ADD	D	
002231	202		MOV	C,A	
002232	117		JMP	0460	
002233	303	004	MOV	A,C	
002236	171		ADD	E	
002237	203		MOV	C,A	;C=8E
002240	117		MOV	A,B	
002241	170		CFI	70	; SHOULD MAG BE > 8.5E?
002242	376	106	JP	0440	
002244	362	004	CFI	66	; SHOULD MAG BE = 8.5E?
002247	376	102	JM	0460	
002251	372	004	JZ	0460	
002254	312	004	MOV	A,C	
002257	171		ADD	D	
002260	202		MOV	C,A	
002261	117		JMP	0460	
002262	303	004	MOV	A,C	
002265	171		ADD	E	
002266	203				

Program C(1) (continued)

002267	MOV	C, A	0C=9E
002267	MOV	A, B	
002270	CFI	76	SHOULD MAG BE = LARGER VAL?
002270	JP	\$450	
002271	CFI	74	SHOULD MAG BE = 2.5E?
002272	JM	\$460	
002272	JZ	\$460	
002273	MOV	A, C	
002274	ADD	D	
002274	MOV	C, A	
002275	JMP	\$460	
002276	LHLD	LARGER	HL ← LARGER VALUE (H ← 0)
002277	MOV	C, L	C ← MAGNITUDE OF LARGER VALUE
002278	LHLD	MAGS	
002279	MOV	M, C	ADDRESS IN MAGS
002280	RET		
002281	MAGS:	DS	2
002282	LARGER:	DS	2
002283	MITUDE:	DS	16
002284	K:	DS	1
002285	M2:	DS	1
002286	M3:	DS	1
002287	THE NEGATIVE NUMBERS IN THIS TABLE ARE MEANT TO BE 1'S COMPLEMENTED IN THE MULCON SUBROUTINE. THAT IS, THE NUMBER --2 IN THE TABLE, WHICH IS ASSEMBLED TO 1111110 IN BINARY, WILL BE COMPLEMENTED IN THE PROGRAM TO YIELD 0000001. THUS, --2 IN THIS TABLE WILL REALLY BE INTERPRETED AS --1 BY THE PROGRAM.		
002288	TABLE:	DB	0, 0
002289	DB	--2, 0, --3, --6, 2, 4	
002290	DB	--2, 5, 1, 4, --5, --6	
002291	Program C(1) (continued)		

```

002267 004
002270 376
002271 372
002272
002272 376
002273 000
002274 376
002275 000
002276 000
002277 000
002400
002400 001
002401 002
002402 004
002403 000
002404 375
002405 005
002406
002406 376
002407 000
002410 376
002411 004
002412 376
002413 005
002414
002414 376
002415 000
002416
002420
002424
002424
002430
002432
002434
002436
002440
002502

DB      ...2, 3, ...2, 0, 3, 0

DB      1, 2, 4, 0, ...3, 5

DB      ...2, 3, ...2, 4, ...2, 5

DB      ...2, 0

PREV:   DS      2
SUM:    DS      4
TEMP:   DS      2
DJADDR: DS      2
DIADDR: DS      2
TKADDR: DS      2
DJVAL:  DS      2
DIVAL:  DS      2
DATA:   DS      34
        END

```

Program C(1) (continued)