NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

Canada

THE UNIVERSITY OF ALBERTA

DIGITAL SIGNAL PROCESSOR FOR BIOMEDICAL APPLICATIONS

by

DANIEL AHENE YAW HAGAN

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH

IN PARTIAL FULFILMENT OF REQUIREMENTS FOR THE DEGREE

OF MASTER OF SCIENCE

DEPARTMENT OF ELECTRICAL ENGINEERING

EDMONTON, ALBERTA

FALL, 1988

ISBN 0-315-45703-1

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: DANIEL AHENE YAW HAGAN

TITLE OF THESIS: DIGITAL SIGNAL PROCESSOR FOR BIOMEDICAL APPLICATIONS

DEGREE: MASTER OF SCIENCE

YEAR THIS DEGREE GRANTED: FALL, 1988

DANIEL

(Student's signature)

Student's permanent address:

2B - 9010 112 ST.
EDMONTON AB
T6G 2C5

DATED 2ND SEPT. 1988.

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Reasearch for acceptance, a thesis entitled DIGITAL SIGNAL PROCESSOR FOR BIOMEDICAL APPLICATIONS submitted by DANIEL AHENE YAW HAGAN in partial fulfilment of the requirements for the degree of MASTER OF SCIENCE IN ELECTRICAL ENGINEERING.

M. P. Wenda

(Supervisor)

Date: Sept. 2/88

ABSTRACT

This thesis presents a digital signal processor for use in biomedical applications. The hardware is based on an International Business Machines Personal Computer Model XT (IBM PC XT) compatible expansion card. An Intel 8088 microprocessor, operating at 4.77 Mhz, is used as the central processing unit. The signal processor runs concurrently with the IBM PC XT in a multiprocessor configuration. The two systems share a common memory. The common memory is arranged as two banks with the IBM PC XT addressing one bank and the signal processor addressing the other. The banks may be switched around under software control.

Software is written for both the IBM PC XT and the signal processor. The IBM PC XT software handles the data acquisition, the data display, and the user interface. Data may be acquired either from a data acquisition card or from a disk file. Both input and output data may be displayed graphically or as numbers. The high resolution mode of the enhanced graphics adapter is used. The user interacts with the IBM PC XT software through a series of menu screens.

The signal processor handles the numeric computations. It performs the direct Fourier Transform, the inverse Fourier Transform, the convolution and the correlation functions. These routines are developed around the Fast Fourier Transform algorithm. A subprogram does the Fast Fourier Transform computation. The subprogram operates on data in 2's complement integer format. With data re-ordering, a 1024-point Fast Fourier Transform computation takes 180 milliseconds on an IBM PC AT running at 6 MHz. Both the convolution and the correlation functions use one complex array to hold the input and the output data. The two functions are realized with one direct Fourier Transform subprogram, one sorting subprogram, and one inverse Fourier Transform subprogram. The execution time of these functions for a 1024-point data is approximately 380 milliseconds.

ACKNOWLEDGEMENT

I would like to thank my sponsors, the Association of Universities and Colleges of Canada, for their financial support throughout the MSc. programme. I would also like to thank my supervisor, Dr. Nelson Durdle, for his encouragement and help in preparing this thesis. Finally, I would like to thank Mr. Kees Denhartogh for his assistance in drawing the circuit diagrams.

TABLE OF CONTENTS

List of Figures

List of Plates

1. INTRODUCTION

Digital signal processing has applications in the fields of biomedical engineering, telecommunications and geophysics. Numerous digital signal processors are available commercially. Their structures range from dedicated systems to general purpose computers, programmed to perform spectral, correlation and convolution analyses. Dedicated systems normally use special purpose integrated circuits to increase the data processing rate. A signal processor operating in real time requires high speed data processing units. High speed data processing units are usually made up of multiple arithmetic-logic units and their structures may be complex.

This thesis employs a general purpose microprocessor as the central processing unit of a digital signal processor. The general purpose microprocessor used is an Intel 8088. The choice of this microprocessor is influenced by the fact that the signal processor will operate in parallel with the IBM PC XT and the IBM PC XT uses the same processor as its central processing unit. Hardware interfacing between the two systems is relatively easier and the same software development tools will be used for both systems.

1.1 8088 Microprocessor

The Intel 8088 microprocessor is an 8-bit version of the 16-bit 8086 microprocessor. It is used as the central processing unit in the IBM PC XT. The 8088 registers are 16-bit wide but its external data bus is 8 bits wide. Two memory accesses are therefore required to read a 16-bit data. Its 8-bit data bus enables it to serve as replacement for the earlier Intel 8-bit microprocessors.

Its 20 address lines gives it a memory address space of one megabyte and an I/O address space of 64 kilobytes. The first 8 address lines are multiplexed onto the data lines. Latches and transceivers are therefore used to demultiplex the address and data lines.

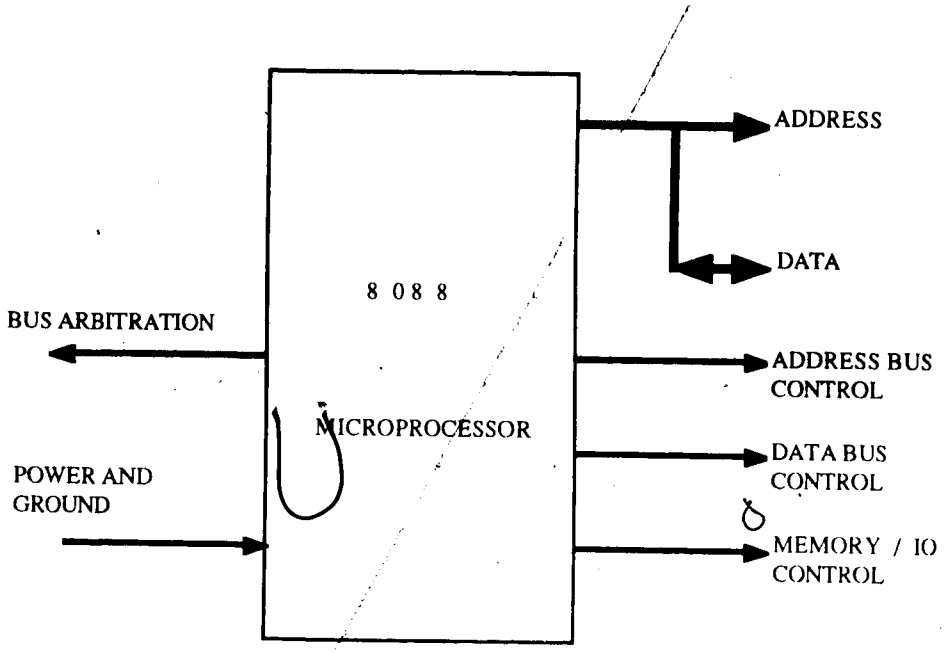


Fig. 1.1 8088 Functional diagram

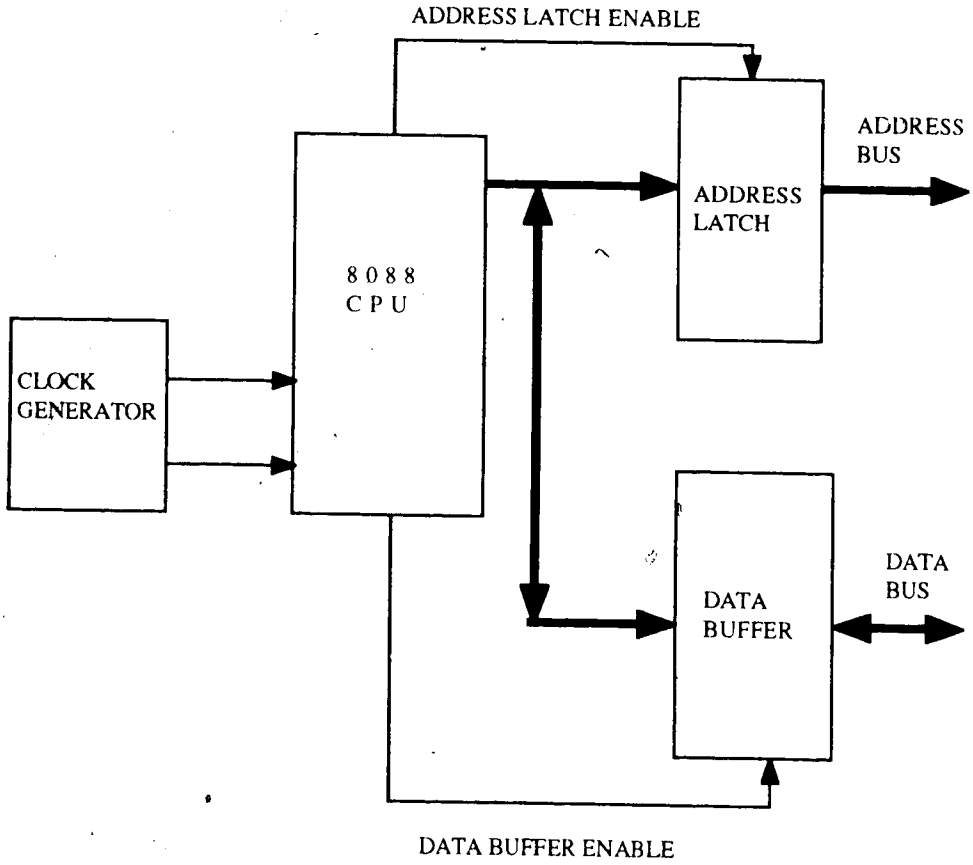


Fig. 1.2 8088 Address and data buffers

Fig. 1.1 shows a functional block diagram of the 8088 and Fig. 1.2 shows the address latches and the data transceivers.

The 8088 can operate in either a minimum mode or a maximum mode. The difference between the two modes is that a bus controller is needed to generate extra control signals in the maximum mode while the minimum mode requires no bus controller. Normally, the minimum mode configuration is used in a small system while the maximum mode is used in a big system. The 8088 has four general purpose registers, two index registers, and two base address pointers. It also has four system registers which are used to generate the 20-bit address.

1.2 Multiprocessor Systems

Multiprocessor systems enable data to be processed at a faster rate than uniprocessor systems. The 8088 can be used in a multiprocessor environment. A typical subsystem may contain some or all of the following units:

(a) a microprocessor circuit

(b) a numeric data processor

(c) an Input-Output processor and/or Direct Access Memory controller

(d) a local memory and/or local Input-Output devices

(e) a bus arbitration circuit

In addition the following system resources may be present:

(a) a global memory

(b) a global clock for synchronization purposes

(c) global Input-Output devices

A block diagram of the central unit of a typical subsystem is shown in Fig. 1.3

The bus arbitration circuit is always present in all the subsystems. One of the subsystems is designated the master controller. The master controller subsystem is used to

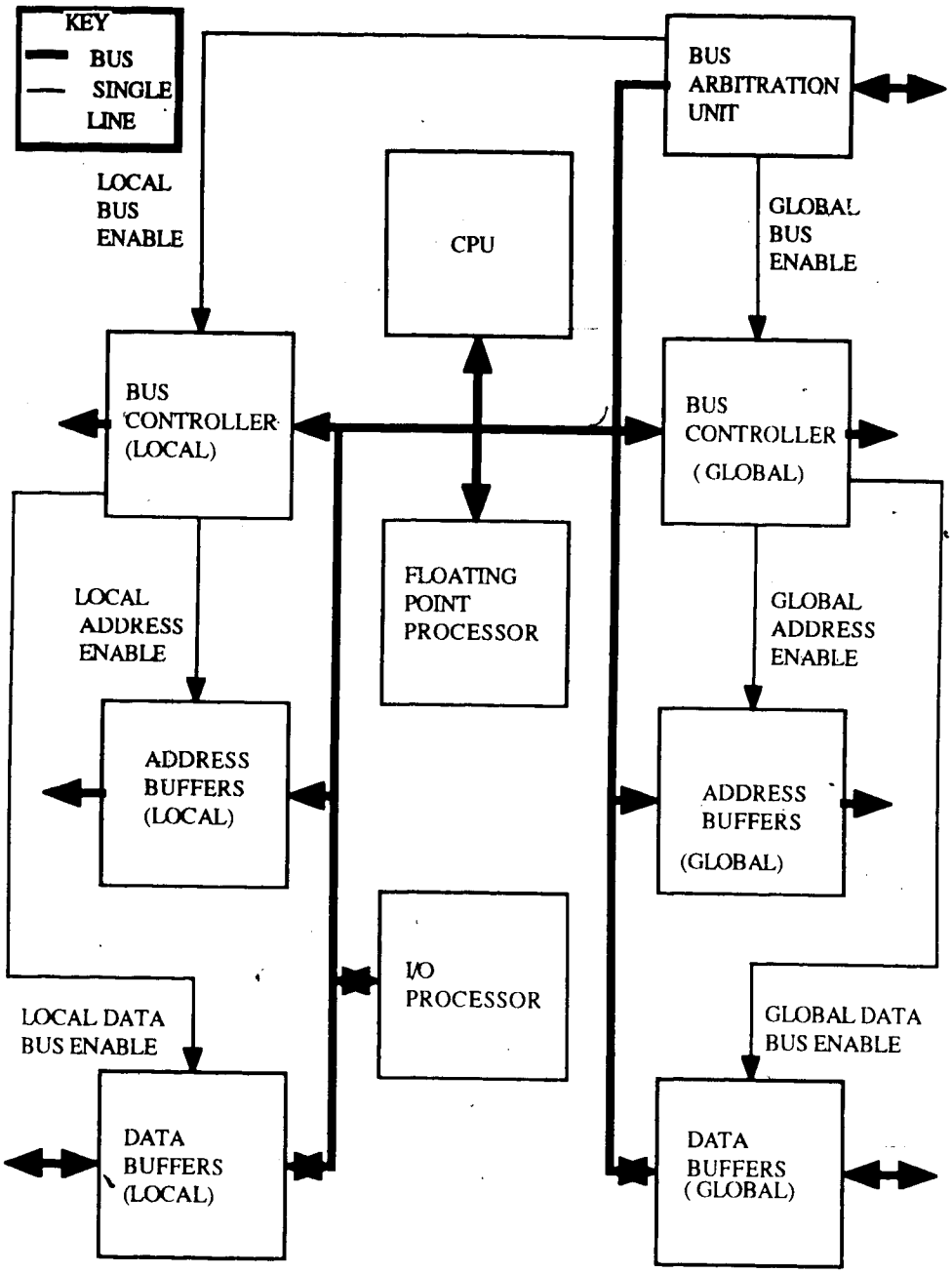


Fig. 1.3 A Subsystem in a multiprocessor environment

coordinate the activities of the remaining subsystems. Interaction between the subsystems is achieved via the bus arbitration circuits. Parameters are passed between the subsystems through the global memory.

The rate at which signals are processed is greatly increased with parallel processing. A multiprocessor system is one way of achieving parallel processing. Studies carried out show that the optimal number of subsystems for a multiprocessor system is about 3 or 4 [17]. The reason why increasing the number of subsystems does not show a corresponding increase in performance is because a significant amount of overhead is associated with the arbitration between the subsystems.

1.3 Digital Signal Processing

Digital signal processing has a number of advantages over analog signal processing. Some of the advantages are:

(a) both the input and the output data may be stored and retrieved very easily.

(b) computations may be repeated under different set of conditions.

(c) a number of display formats are available for both the input and the output data, e.g. graphical, numeric, etc.

The disadvantages of digital signal processing include the finite computational time and the errors due to the finite wordlength of the computer.

The processing element may perform a number of functions. These may include the following:

(a) spectral analysis

(b) signal recovery

(c) filtering

Since the analyses are usually time-consuming, a number of fast algorithms are used to reduce the computation time required. One of the popular algorithms is the Fast Fourier Transform algorithm [11] which is used to evaluate both the direct and the inverse

discrete Fourier Transforms. It can also be used to compute the convolution and the correlation functions.

The Fast Fourier Transform is easily realized in floating-point or fixed point arithmetic. A computer which has a floating-point arithmetic and transcendental functions capability normally require a very short program, in any scientific high level language, to implement the Fast Fourier Transform algorithm. In microprocessor-based systems, floating-point operations may either be too slow or be absent in the system. In such a situation, a choice has to be made between a digital signal processor chip or doing all the computations in 2's complement integer arithmetic. The latter is chosen for the Fast Fourier Transform algorithm implemented in this report. This choice is influenced by the fact that signed integers are more easily manipulated in 2's complement format.

A number of digital signal processing chips are available, the Texas Instruments TMS 320 family being a typical example. The TMS 32010 [3] accepts data in 16-bit 2's complement integer format. It has a 16-bit by 16-bit multiplier which yields a 32-bit product in 200 nanoseconds. This is an extremely fast multiplication time. Its instruction set and architecture are optimized for the common mathematical procedures that occur in digital signal processing.

For simple systems, the use of these application specific chips and their elaborate development systems is unacceptable. This report considers the use of a general purpose microprocessor to do digital signal processing in the PC DOS (the most popular operating system of the IBM personal computer) environment.

1.4 Biomedical Signals

Living tissues produce weak electrical signals [4]. The weak electrical signals may be amplified using an amplifier with a high gain. The amplified signal may be digitized using an analog-to-digital converter. The digital signal may then be processed by a digital

computer. The hardware interface between the digital computer and the signal source is called a data acquisition system.

In real time processing, the digital computer acquires the biomedical data, and process it. The computer displays the results on a screen or sends it to a plotter. A physician may then make some deductions from the results.

1.5 Objectives

The main objective of the thesis is to develop a digital signal processor for use in biomedical applications. The signal processor should have the following features:

(a) Signal processor should work in parallel with the IBM PC XT

(b) The signal processor should perform the computations; and the IBM PC XT should handle the acquisition and display of input / output data and the user interface.

(c) Software for both the IBM PC XT and the signal processor is to be optimized for both speed and size.

(d) The user interface is to be menu driven and very easy to use.

(e) Input/output data is to be displayed in color on a high resolution screen. The enhanced graphics display of the IBM PC is to be used.

(f) The user should be able to select the sampling rate of the data acquisition system.

(g) The user should be able to select the number of elements in a record of data.

(h) The signal processor should be able to do real time processing.

1.6 Design

A low frequency digital signal processor used for biomedical applications, is designed and tested. The IBM PC XT is used as the host processor and Input/Output

processor. The widespread use of the IBM PC makes it a good choice. The digital processor itself is

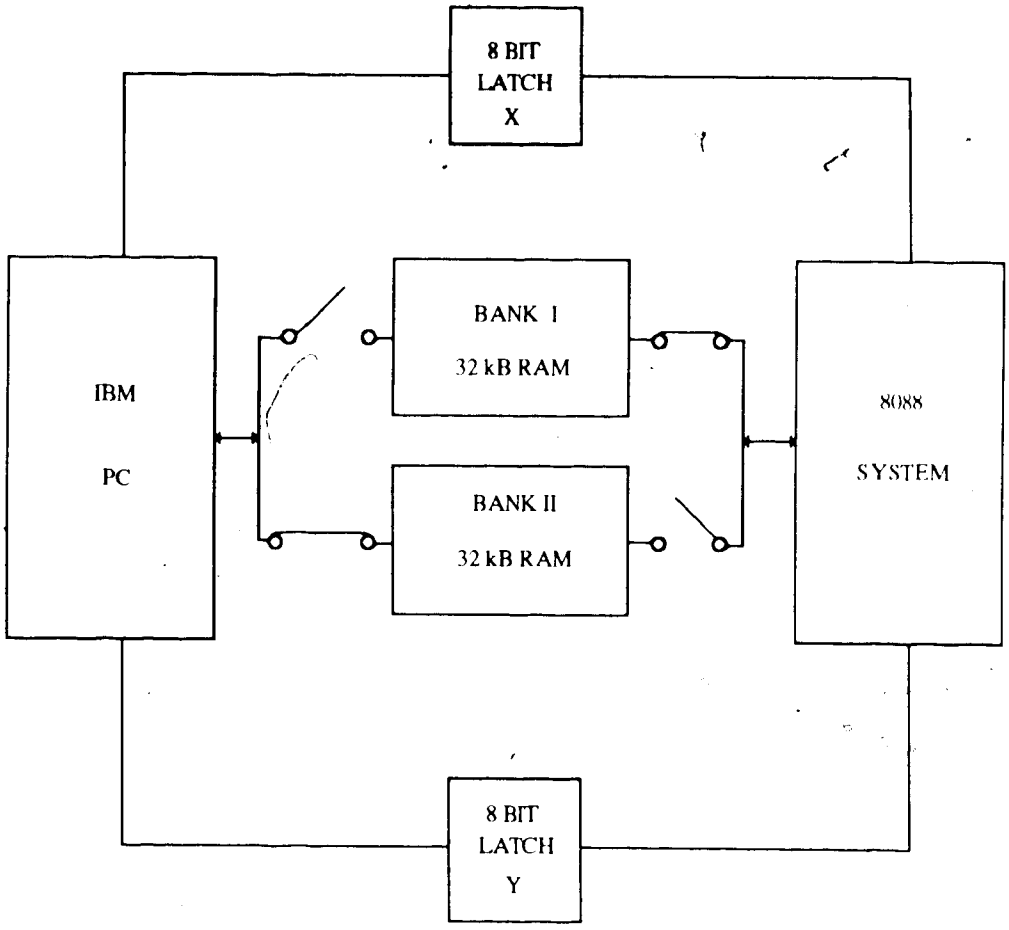


***Fig. 1.4** Block diagram of signal processor and the IBM PC XT interface*

designed on a board which fits into one of the expansion slots of the IBM PC XT. A block diagram of the signal processor and the IBM PC XT interface, is shown in Fig. 1.4.

The signal processor is subsequently referred to in this report as the coprocessor board. An Intel 8088 microprocessor is used as the central processing unit to simplify the interfacing between the coprocessor board and the IBM PC. The coprocessor board is configured as a subsystem in multiprocessor system, with the IBM PC being the other

subsystem. The shared resources are the global or shared memory and the Input/Output ports. The shared memory consists of two banks of read/write memory while the Input/Qutput ports consist of two 8-bit latches.

The coprocessor board has a local memory which acts as a writable control store and also as storage for local data. The control program is loaded into the control store through the global memory.

1.7 Thesis Organization

The hardware and the software for the two subsystems are discussed in the rest of the chapters. Chapter 2 gives a description of the coprocessor board hardware. The chapter starts by identifying the different modules on the coprocessor board. It then goes on to discuss the details of each module.

The software for the coprocessor board is described in chapter 3. The Fast Fourier Transform algorithm is examined and its implementation in integer format is explained. The use of the Fast Fourier Transform to compute the correlation and the convolution functions is also explained in chapter 3. The software routines are organized in modules. The more important modules are discussed in detail while the others are briefly mentioned. The interaction between main coprocessor routine and the main IBM PC routine is also mentioned in this chapter.

Chapter 4 deals with the IBM PC XT routines. Like the coprocessor subprograms the IBM PC XT subprograms are also organized into modules. The interaction between the different modules is mainly through the selection of a menu option. The IBM PC XT routines are extensively menu driven. The interface between the IBM PC XT routines are mentioned.

The testing and the performance of the system is discussed in chapter 5. Limitations on the acquisition and processing rates are also mentioned here. Chapter 6 has the conclusion of the report.

2. DIGITAL SIGNAL PROCESSOR HARDWARE

Digital signal processing using a single general purpose processor is usually time consuming because of the number of computations involved. The need to speed up processing has given rise to a number of special purpose digital signal processors. The architectures of these special purpose processors are optimized for the common mathematical operations that occur in digital signal computations. Typically, most of these processors have multiple arithmetic-logic units and in the case of highly specialized applications, constants are stored in a read-only memory. Data is coded in a manner which makes manipulations easier. Hardware design using these special purpose processors usually require expensive development systems. Moreover, not all operating environments support these development systems.

The choice of hardware for the design under consideration is governed by the following:

(a) off-the-shelf integrated circuit components

(b) the widespread use of the IBM PC and its DOS operating system

(c) the availability of expansion slots on the IBM PC.

The IBM PC XT should have an enhanced graphics adapter card and a corresponding monitor. The first 640 kilobyte memory address space of the IBM PC XT usually has read/write memory installed. In order to make this hardware configuration possible, this read/write memory should be reduced to 512 kilobyte. This allows memory address space to be available for the coprocessor board hardware. The vacant memory address space is actually a common memory to both the IBM PC XT and the coprocessor board. Further discussion on the common memory is given later.

The coprocessor board is designed to fit into one of the expansion slots of the IBM PC XT. The board is an Intel 8088 microprocessor-based system. A block diagram of the

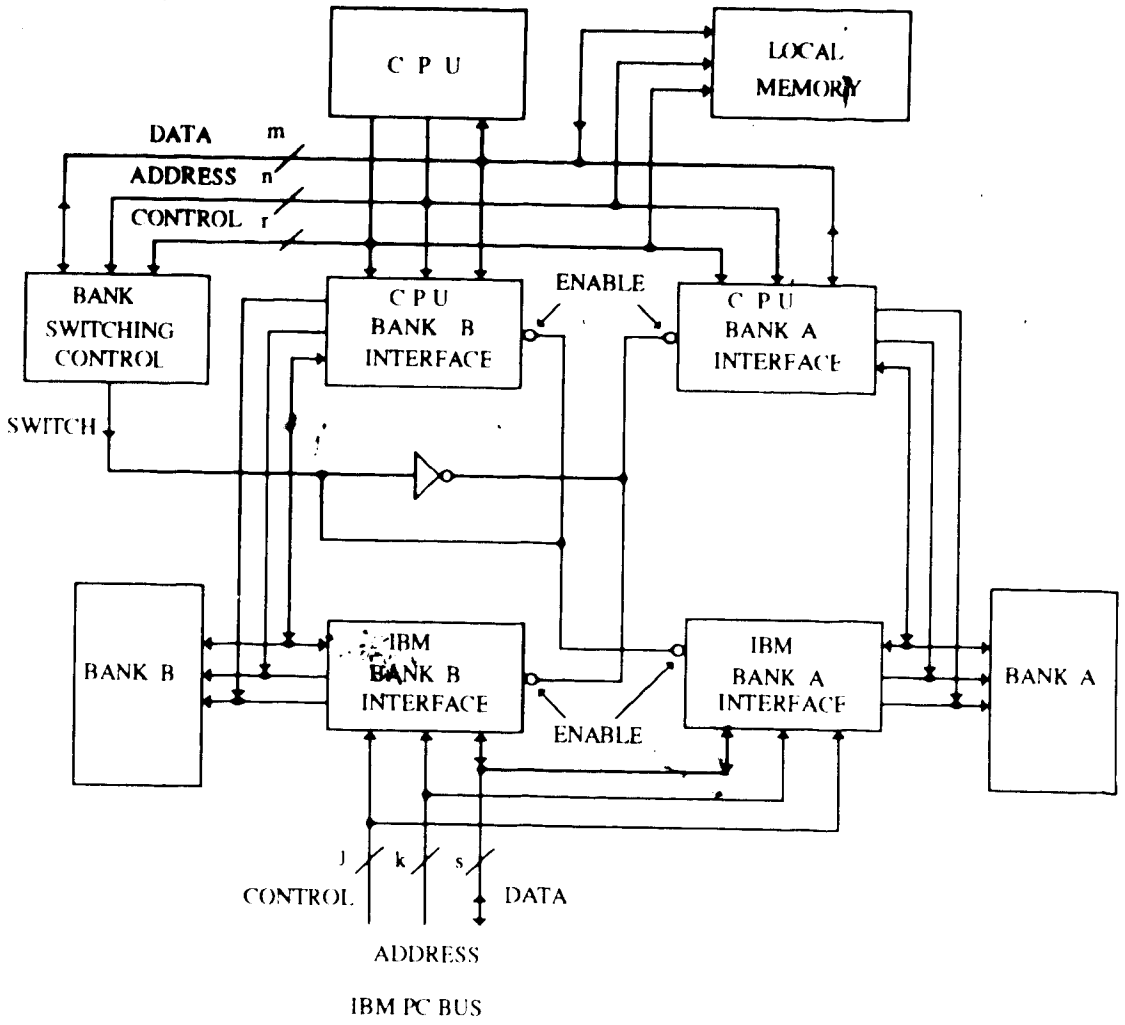


***Fig. 2.1** Block Diagram of coprocessor board*

coprocessor board is shown in Fig. 2.1 and Fig. 2.2. The subcircuits are that of the CPU, the local memory, the bank switching control, the banks interface, and the I/O.

2.1 CPU Subcircuit

The CPU subcircuit is made up of the following:

a. the microprocessor circuit,

b. the microprocessor reset circuit, and

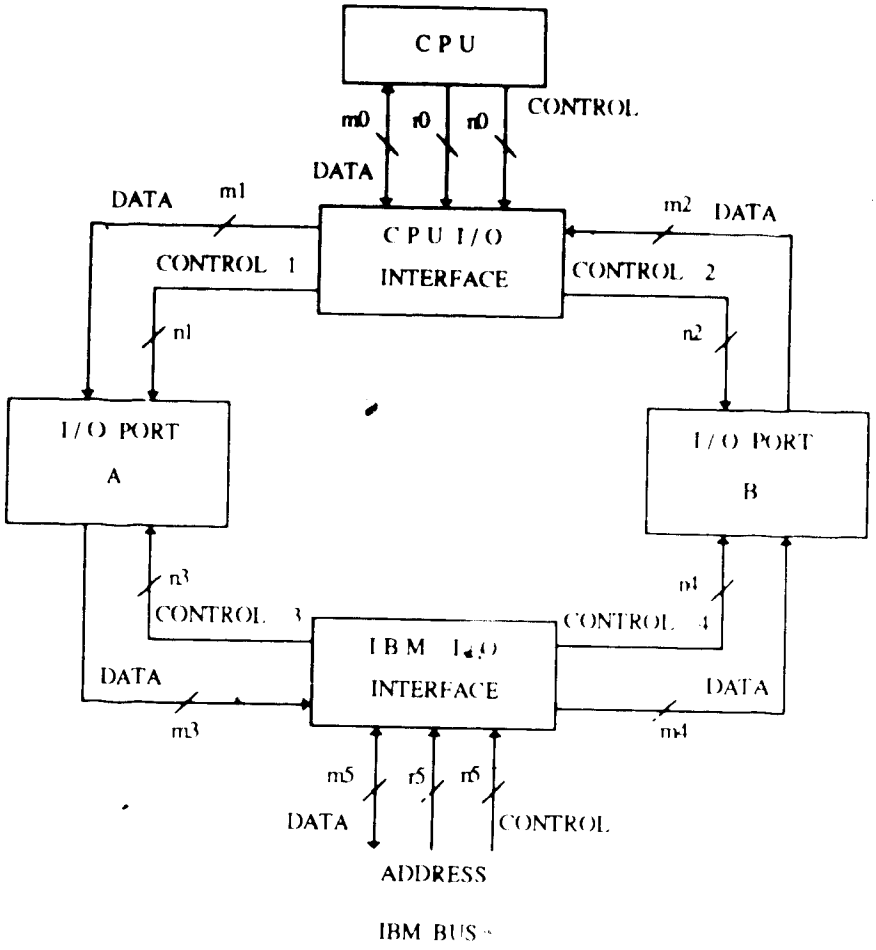


Fig. 2.2 I/O circuit for the coprocessor board.

c. the address decoding circuit.

2.1.1 Microprocessor Circuit

At the heart of the coprocessor board is an 8088 5Mhz microprocessor configured to operate in the minimum mode. It receives its 4.77Mhz clock signal from the 8284 clock generator. The address lines are latched and buffered with OCTAL latches while the data

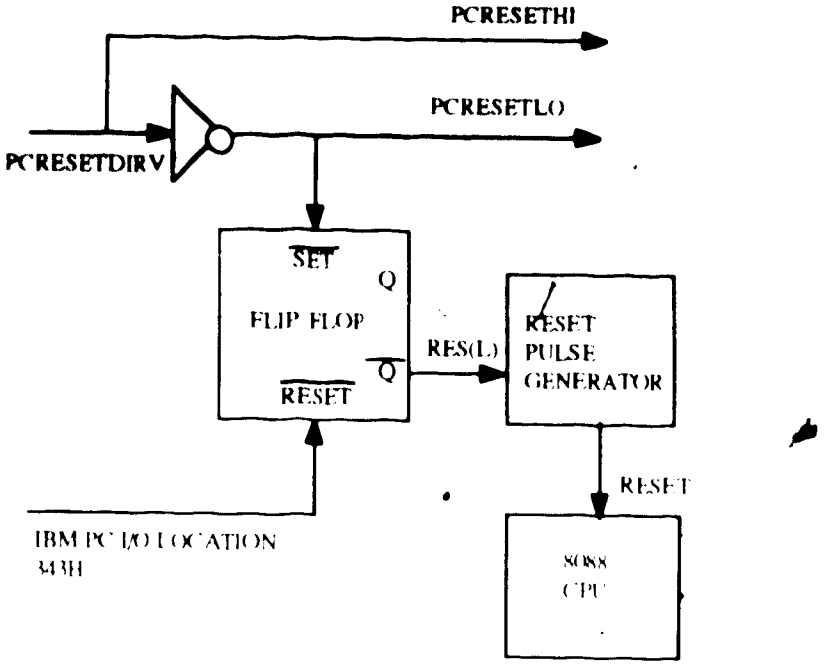


Fig. 2.3 Block diagram of the reset circuit

lines are buffered and demultiplexed from the lower address lines with an octal transceiver. A diagram of the microprocessor circuit is shown in Fig. A.1. A block diagram of the coprocessor reset circuit is shown in Fig. 2.3. The detailed reset circuit diagram is shown in Fig. A.2. The PCRESETDRIV signal comes from the IBM PC XT I/O channel. The derived signal PCRESETHI is used to clear all devices which have active high clear inputs while the derived signal PCRESETLO is used to clear all devices which have active low clear inputs. The D flip-flops falls in the latter category. Hence, at power up, the signal RES(L) is active. The RES(L) feeds the reset input of the 8284 clock generator. The reset output of the 8284 clock generator in turn feeds the reset input of the 8088 microprocessor. Therefore, at power up, the microprocessor is placed in a reset state.

A write by the IBM PC XT to I/O location 343H makes RES(L) inactive. The inactive RES(L) causes the 8284 clock generator to remove the active reset signal from the coprocessor CPU. In other words, the coprocessor CPU is removed from its reset state immediately after power up when the IBM PC XT writes to I/O location 343H.

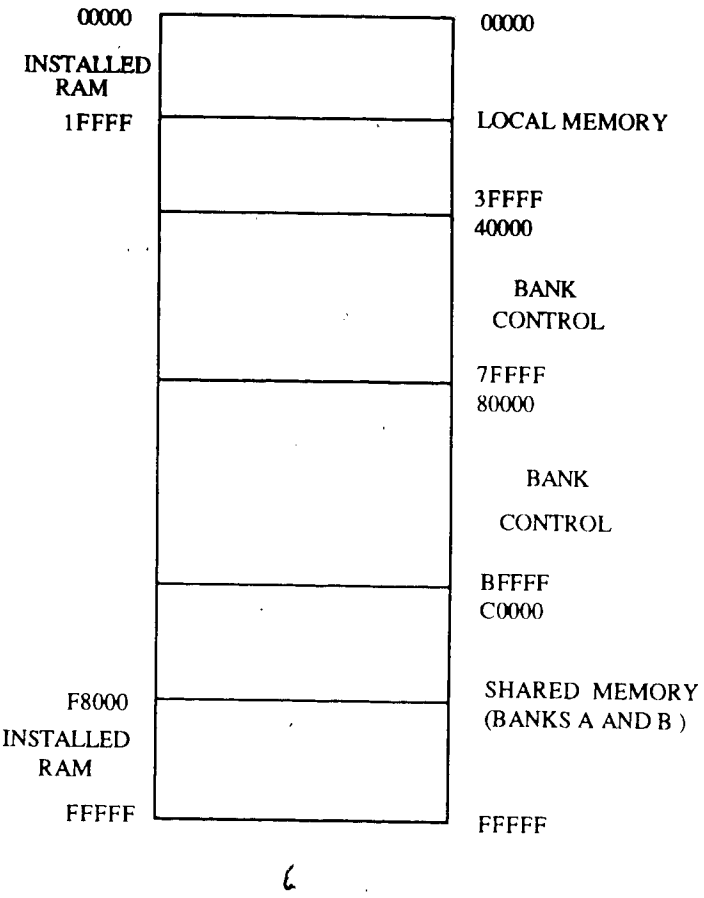
Three control lines are buffered with octal buffers. They are the read line (*RD), the write line (*WR) and the IO/*M line. The buffering of the control lines and the address decoding for the coprocessor board is shown in Fig. A.3.

One half of a dual 2-line to 4-line decoder is used to divide the entire memory address space of the 8088 microprocessor on the coprocessor into four equal blocks. The first block is allocated to the local memory of the coprocessor board while the last block is allocated to shared memory. In such a small system partial address decoding is the optimal choice.

The shared memory is made up of two banks of read/write memory. It is further discussed in section 2.3. The decoder outputs for the second and third memory blocks are used to generate control signals for the shared memory banks. The memory address map for the coprocessor is shown in Fig. 2.4.

2.2 Local memory

The local memory circuit is shown in Fig. A.4. The local memory made up of 8 kB of read/write memory occupying the first 8 kB memory address space of the 8088 on the coprocessor board. The 8 kB is enough to hold all the local variables and the code for the coprocessor board. Since four 2kB x 8 static RAMS are used, further memory address decoding is done with the second half of the decoder used for the main address decoding. The signal lines *SELC1 - *SELC4 in Fig. A.4 are the final chip select lines for the local memory.



NOTE - address lines A17, A16, and A15 do not take part in address decoding, hence multiple reflections in address space

***Fig. 2.4** Memory map for coprocessor board.*

2.3 Shared Memory

The shared or common memory for the IBM PC XT and the coprocessor board is configured such that, while the IBM PC XT is addressing bank A, the coprocessor board is addressing bank B and vice versa. However, the banks occupy different memory address spaces in the two systems. The IBM PC XT expects continuous memory in blocks of 128 kB, 256 kB, 512 kB, or 640 kB. During power-on-self-test (POST), a memory test is

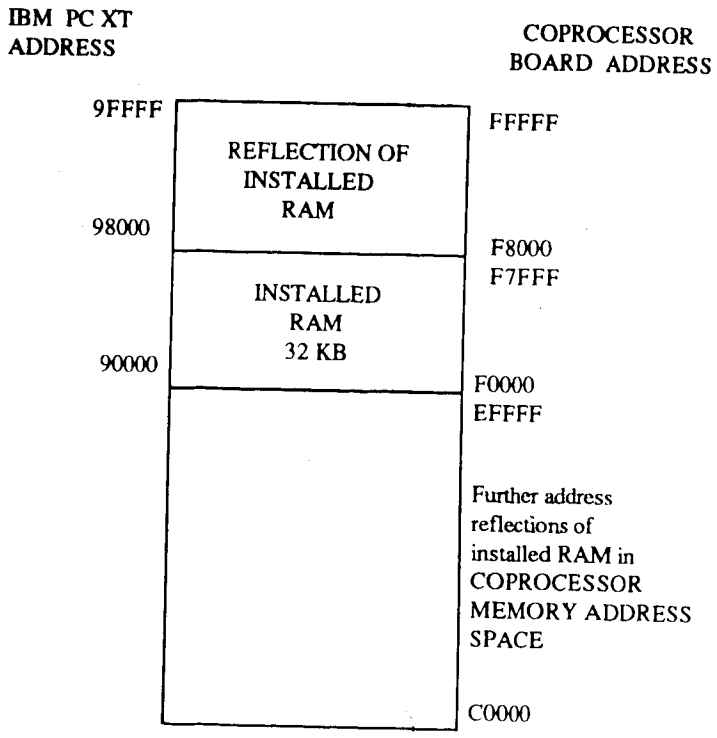


Fig. 2.5 *Shared memory -- memory map.*

done to determine the total amount of read/write memory. The read/write memory should occupy he first 640 kB memory address space. i.e. 00000 to 9FFFF hex. In order to ensure the smooth booting of the IBM PC XT, only the first 512 kB is declared, leaving a 128-kB memory address space for the shared memory. This shared memory address space occurs at memory address 80000 hex to 9FFFF hex. Only the memory address space 90000 hex to 9FFFF hex is used. Leaving the memory address space 80000 hex to 8FFFF hex free forces the IBM PC XT to end the memory test after the first 512 kB. The advantages of not declaring the 32 kB of shared memory to the IBM PC XT are:

(a) it ensures the smooth booting of the computer ;

(b) it prevents the operating system from taking control of the shared memory.

The IBM PC XT memory address decoding circuit is shown in Fig. A.5. The shared memory map is shown in Fig. 2.5. Each bank is made up of 32 kB of read/write memory and is implemented with a single 32 kB x 8 static RAM. The base address for the shared memory is 90000 hex on the IBM PC XT. This corresponds to a base address of F8000 on the coprocessor board. Memory test on the shared memory may be done as part of the hardware initialization of the coprocessor board.

2.3.1 Shared Memory Interfaces

A set of four memory-to-processor interfaces are required to implement the shared memory scheme. The four interfaces are:

(a) IBM PC XT-to-bank A

(b) IBM PC XT-to-bank B

(c) coprocessor board-to-bank A

(d) coprocessor board-to-bank B

At any given time, only two of the interfaces are active, the other two being tri-stated. The possible pairs are (a) and (d) or (b) and (c). Each interface is made up of three 74LS541 octal buffers and one 74LS245 octal transceiver. The octal buffers are used to buffer the address and control lines while the octal transceiver is used to buffer the data lines. The four buffers are shown in Figs. A.5, A.6, A.7, A.9, A.10 and A.11

2.3.2 Bank switching

The bank switching circuit is shown in Fig. A.8. A block diagram of the switching circuit is shown in Fig. 2.6. The PCRESETLO signal is derived from the IBM PC XT power-on-reset line while the IBMLOC3 refers to the decoded address of an IBM

PC XT I/O location. The PCRESETLO signal clears the flip-flop after power-on-reset. That means the signal *BUFSEL1 is always active after power up. A subsequent write to dedicated memory location causes the signal *IBMLOC3 to be active and the flip-flop is reset. For proper operation of the coprocessor board, this set-reset sequence may only occur once after power up. Subsequent control of the flip-flop output is achieved by making either *BUFSWITCHIN1 or *BUFSWITCHIN2 active.

These two signals come from the two remaining outputs of the coprocessor main memory address decoder. The bank switching is controlled by the coprocessor board. A write to any location within the 40000 hex to 7FFFF hex memory address space of the coprocessor board makes *BUFSWITCHIN1 active and a corresponding write to any location within the memory address space 80000 hex to BFFFF hex makes *BUFSWITCHIN2 active. Using the middle half of the memory address space to perform the switching is justified in such a small system where cost is to be kept low. Using only two memory or I/O locations to do the switching will require extra decoding.

The two output signals *BUFSEL1 and *BUFSEL2 control the four interfaces, with each signal controlling two of the interfaces. *BUFSEL1 controls the IBM PC XT-to-bank A interface and the coprocessor board-to-bank B interface and the *BUFSEL2 controls the IBM PC XT-to-bank B interface and the coprocessor board-to-bank A interface. Each bank will belong to only one of the two systems at any given time. The only possible problem is that of the IBM writing to a bank when that bank is in the process of being switched. However, that may be taken care of by means of a suitable protocol between the IBM PC XT and the coprocessor board. Fig. 2.1 shows the four interfaces and a simplified control signal controlling the two banks. The detail circuit diagrams for the four interfaces are shown in Figs. A.5, A.6, A.7, A.9, A.10, and A.11.

The shared memory scheme cuts down on the amount of time spent on moving blocks of data and also allows concurrency in the processing of data. While the IBM PC

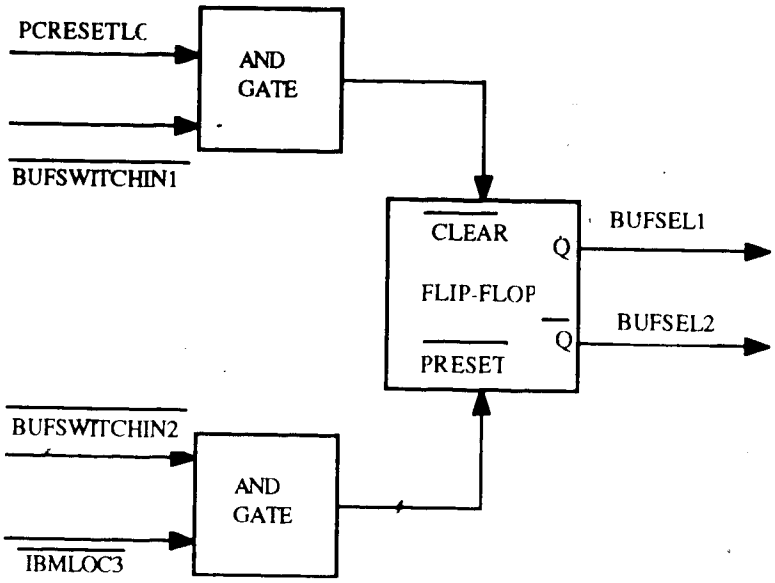


Fig. 2.6 Block diagram of the bank switching circuit

XT is moving data into one memory bank, the coprocessor may be processing data stored in the other bank.

2.4 I/O Address Decoding

Four locations in the IBM PC XT I/O address space are used to establish the necessary protocol between the IBM PC XT and the coprocessor board. The base address is switch-selectable and conveniently set at 340 hex. The circuit for selecting the base address and the decoding circuit for the four I/O locations are shown in Figs. A.12 and A.13

The first I/O location addresses an I/O port. The I/O port is implemented with four 4-bit D-type registers. They are arranged as two 8-bit unidirectional registers. The IBM PC XT can write to the first port but can only read from the second port. Similarly, the

coprocessor can write to the second port but can only read from the first. Here again, care must be taken to avoid stale data. An invalid data may result if one processor tries to read a port whilst, simultaneously, the other processor is writing to the same port. The I/O port arrangements are shown in Figs. A.15 and A.16.

The second and third IBM PC XT I/O locations are used for interrupt control and will be discussed later. The last location is used to negate the signal that goes to the reset input of the reset pulse generator. A write to this I/O location also causes the initial switching of the memory banks.

2.5 Interrupt Circuit

As part of the protocol between the two systems each processor must be able to interrupt the other. The interrupt circuit enables the two processors to accomplish this. The IBM PC XT microprocessor interrupts the other processor by writing to an I/O location equal to the I/O base address plus 2. This sends a non-maskable interrupt to the coprocessor CPU. Upon receiving the interrupt, the coprocessor CPU resets the non-maskable interrupt signal so that the next interrupt will be recognized. The non-maskable interrupt signal is reset by an I/O write to absolute I/O location 01 hex.

In much the same way, the coprocessor board interrupts the IBM PC XT microprocessor by an I/O write to absolute I/O location 02 hex and the IBM PC XT resets its interrupt signal by writing to I/O location base address + 1.

The coprocessor board is constructed using an IBM PC XT type expansion card and it fits into one of the expansion slots of the IBM PC XT.

2.6 Data Acquisition Hardware

A data acquisition card which fits into another expansion slot of the IBM PC XT is used to acquire data. It can be configured to respond to either memory or I/O addresses. It

uses sixteen consecutive memory or I/O locations for operational control and the base address may start anywhere in either the memory address space or the I/O address space.

It has two 12-bit digital-to-analog converters, a sixteen channel analog-to-digital converter, a three-port parallel interface and an advanced programmable timer/counter. The voltage range of each of the digital-to-analog converters and also the analog-to-digital converter is programmable.

Some options are hardware programmed by means of switches and jumpers. The other options are programmed under software control. This is achieved by writing bit patterns to various control ports. The option selected for this set-up is analog voltage range of -10 volts to +10 volts.

3. DIGITAL SIGNAL PROCESSOR SOFTWARE

The IBM PC XT and the coprocessor board perform different tasks during signal analysis but the tasks are performed concurrently. The IBM PC XT is mainly responsible for the acquisition and the display of data while the coprocessor board does the computations.

3.1 System Initialization

At power up, the coprocessor CPU is in the reset state and all flip-flops are cleared. When the signal analysis program is executed, a memory test is carried out on the memory bank in the address space of the IBM PC XT. If the memory test is successful, the IBM PC XT transfers the code (to be executed by the coprocessor board) and the local data to the shared memory bank. The IBM PC XT then pulls the coprocessor CPU out of its reset state and switches banks at the same time by writing to I/O location 343 hex.

The processor on the coprocessor board then moves the entire code to its local memory and transfers control to its main program. This program performs the necessary initializations and waits for a command from the IBM PC XT processor. The IBM PC XT may then place data and other information in the new bank and issue the necessary command. Protocol is established through interrupts and the I/O port. Fig. 3.1 shows the sequence of events that take place after power up.

3.2 Subsystems Interaction

Both the IBM PC XT and the coprocessor keep track of the bank identity as the banks are switched so that, in the event of an error in switching, the IBM PC XT will report on the error. Other status information such as the successful mathematical

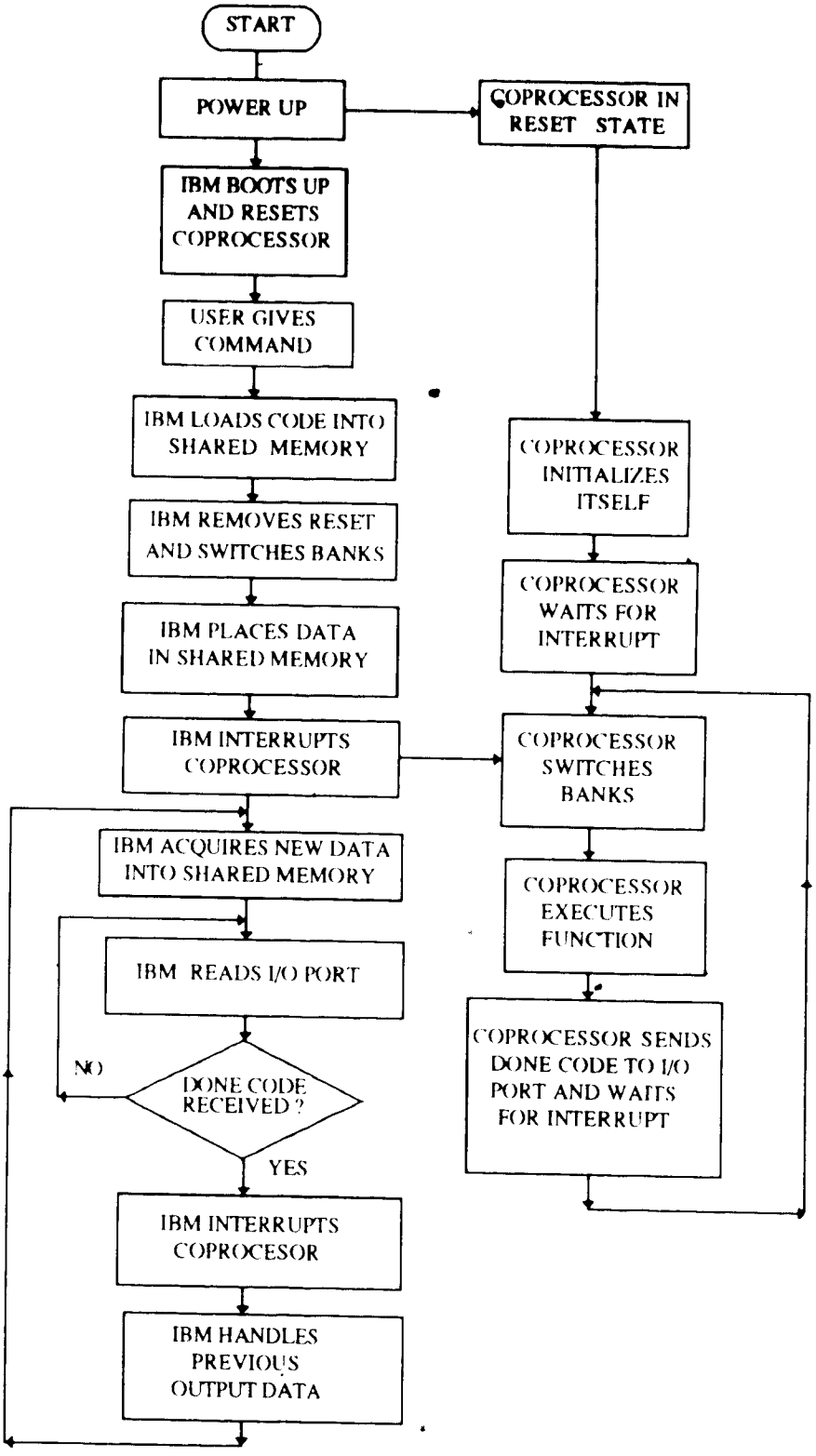


Fig. 3.1 Flow chart of IBM and Coprocessor interaction.

computation, divide by zero error and overflow are also maintained in the shared memory. In order to prevent corrupt data from appearing in each other's memory address space, the coprocessor does the actual switching while the IBM PC XT decides when the switching should take place.

After initialization, the coprocessor sits in a loop waiting for a flag to be set in its local memory. This flag is set by a non-maskable interrupt service routine. The non-maskable interrupt is generated by the IBM PC XT after it has placed valid data and function code in the shared memory bank in its memory address space. On receiving the valid flag, the coprocessor switches the banks so as to have access to the data and the function code. The coprocessor then uses a jump table to execute the selected function. It then places the results back in the shared memory bank in its address space. It also reports on the errors (if any) to the same bank. It once again waits for another flag to be set in its local memory before switching banks again.

While the coprocessor is executing the function, the IBM PC XT may be acquiring new data into the shared memory bank in its address space. For high data acquisition rates, both processors may finish their respective tasks about the same time or the IBM PC XT may finish earlier. The results may be handled in a variety of ways. It may either be moved into a very large buffer in the IBM PC XT local memory or viewed directly. The former is more efficient but the user may decide otherwise. Both the input data and the output may be stored on disk or displayed in graphical format.

3.3 Software Development

Executable files created under most operating systems are relocatable. On giving a command, the loader uses the relocation information in the executable file to assign absolute addresses where necessary. The true executable code is then loaded into available memory. The assigning of absolute addresses is more complicated in a processor with a

segmented memory address space. In order to get around the problems created by this situation, the Microsoft macro assembler version 5.00 and the Microsoft C compiler version 5.00 were used to develop the software. The assembler enables code to be aligned on a 256-byte boundary. Alignment on a 256-byte boundary is called page alignment. Page alignment forces the linker and the loader to preserve all addresses that fall within the same 64-kilobyte segment. Absolute addresses may be referred to but no data or code is generated by the linker. The data variables for the software are found in four memory address spaces. The four memory address spaces are :

(a) local memory of the coprocessor : (00000 hex to 1FFFF hex)

(b) shared memory as it appears to the coprocessor (F8000 hex to FFFFF hex)

(c) shared memory as it appears to the IBM PC XT (90000 hex to 97FFF hex)

(d) local memory of the IBM PC XT (00000 hex to 7FFFF hex)

Most variables and subprograms have at least two aliases. For example a variable may be called N in one address space and NQ in another address space but the physical location of this variable is exactly the same in both cases. In fact the layout for the shared memory templates for both the IBM PC XT and the coprocessor board are identical. The only difference between the two templates is the different base addresses.

The coprocessor software was written in Intel 8088 assembly language and was optimized for both speed and size. Most of the subprograms for the IBM PC XT were written in the C language. The rest were written in Intel 8088 assembly language.

3.4 Coprocessor Computations

The following computations are performed by the coprocessor on data received from the snared memory:

(a) direct Fourier Transform

(b) inverse Fourier Transform

(c) periodogram

(d) autocorrelation

(e) crosscorrelation

(f) linear convolution

(g) cyclic convolution

Filtering was not considered in this thesis because the absence of a floating point processor on the coprocessor board would make the calculation of the filter coefficients very slow with the floating-point emulator library. If the coefficients are obtained by some other means, they may be scaled to lie within the interval -16384 and 16383 and the linear convolution function may be used. This results in a finite impulse response (FIR) filtering[16].

Integer arithmetic is used throughout since the floating point emulator is too slow to carry out even a 1024-point Fast Fourier Transform.

3.4.1 Fast Fourier Transform

The Fast Fourier Transform forms the basis of most of the computations done by the coprocessor. Cooley and Turkey's algorithm is used [11]. This algorithm works for a set of data points whose number of elements can be expressed as an integer power of 2. The input data is reordered through bit reversal of the data index so that the output data will appear at the right position. The results are also stored back to the same location as the input data (in-place) to cut down on the amount of memory required for the operation. The direct discrete Fourier Transform is given by the relation

$$X(m) = \sum_{n=0}^{N-1} x(n) * W^{(m*n)} \qquad (1)$$

where $W = \cos(2\Pi / N) - j \sin(2\Pi / N)$ (2)

Unlike other integer Fast Fourier Transform routines that have a fixed value for N this routine can handle any number of data points provided it is an integer power of 2. The limit of 1024 is chosen because of memory limitation since memory is needed to store both the data and other constants. One set of constants is a sine table with 1024 entries. The sine table is generated from the relation

$$16384 * \sin(2\Pi\ i / N) + 0.5 \quad (3)$$

for $0 < i < 1024$. The value of

$$W\ (m * n) \quad (4)$$

is read from the sine table. This saves a significant amount of time since only the computation of the appropriate index and a memory access are required. Each entry in the sine table is stored as a 16-bit signed integer. The reason for scaling the sine values by 16384 is to prevent possible overflow of intermediate or final results, otherwise a higher value may have been used.

Each data point x(n) is represented as a 32-bit signed integer. However, in order to avoid possible overflow each input data point must lie within the interval -32767 and 32767. With these limits on the input data, the highest possible output value will occur when a unit step of amplitude 32767 is processed. In that case, the dc value will be:

32767 * 1024 (or approximately 33 550 000)

which is still representable as a 32-bit signed integer. In order to prevent intermediate results from overflowing, intermediate quantities are stored as 64-bit variables. They are then rescaled to the usual 32-bit representation. The rescaling is necessary because of the scaled sine and cosine values used.

The rescaling is done as follows:

Let $a = a1 * (2^{16}) + a0$

be a 32-bit signed integer with a0 being the least significant 16-bit word and a1 the most significant 16-bit word; and

$$b = b0$$

be a 16-bit signed integer. Then

$$c = a * b$$

$$= a1 * b0 * (2^{16}) + a0 * b0 .$$

Let d be the rescaled value.

$$d = c / 16384$$

$$= 4 * a1 * b0 + (a0 * b0) / 16384$$

Each term $x(n) * \cos(\emptyset)$ or $x(n) * \sin(\emptyset)$ is evaluated using the above procedure, the division by 16384 accounting for the scale factor used to generate the sine and cosine values.

Let the total number of entries in the sine table be NTOTAL. Since the cosine waveform has a 90 degree phase shift from the sine waveform, each cosine index into the sine table is offset from the corresponding sine index by NTOTAL / 4. The problem is then reduced to selecting the appropriate sine index and adding the offset to obtain the correct cosine index.

In an N-point radix 2 Fast Fourier Transform computation, it is observed that the computation is carried in stages. If

$$N = 2^M$$

then the stages are from L = 1 to L = M. For the sake of completeness, the FORTRAN code is reproduced in Fig. 3.2 [11].

```
PI      =  3.14159                    ( i)
DO 20 L =  1, M                       ( ii)
LE      =  2 ** L                     ( iii)
```

```
      LE1      =   LE / 2                                          ( iv)
      U        =   (1.0,0.0)                                       ( v)
      W        =   CMPLX(COS(PI / LE1 ), SIN((PI * SIGN) / LE1))   ( vi)
      DO 20 J  =   1, LE1                                          ( vii)
      DO 10 I  =   J, N, LE                                        (viii)
      IP       =   I + LE1                                         ( ix)
      T        =   X(IP) * U                                       ( x)
      X(IP)    =   X(I) - T                                        ( xi)
10    X(I)     =   X(I) +                                          ( xii)
20    U        =   U * W                                           (xiii)
```

***Fig. 3.2** Fast Fourier Transform Routine*

Statements (vi) and (xiii) are combined to generate a sine and cosine index into the sine table from the expressions:

sine index = (NTOTAL * J) / LE1

cosine index = sine index + (NTOTAL / 4)

The advantage of using the table is that the two complex manipulations in lines (vi) are (xiii) are replaced with two simple integer arithmetic expressions and two memory accesses. Clearly, the savings in time is enormous. The main disadvantages of using the sine table are :

(a) a large amount of memory is required to hold the sine table (2 kB in this case).

(b) the table will have to be regenerated if data points greater than the current number of entries in the table are to be handled.

3.4.2 Direct and Inverse Fourier Transforms

The direct discrete Fourier Transform is given by the expression

$$X(m) = \sum_{n=0}^{N-1} x(n) * W^{(SIGN * m * n)} \qquad (5)$$

where SIGN is equal to unity. The inverse discrete Fourier Transform is given by the expression

$$x(n) = \frac{1}{N} \sum_{n=0}^{N-1} X(m) * W^{(SIGN * m * n)} \qquad (6)$$

and SIGN has a value of -1.

The summation part of both (5) and (6) are carried out by the same routine. A second routine is then called to do the division in (6). The results are accurate to three significant numbers, the maximum obtained with the 16-bit representation of the sine table entries. For input data values whose maximum absolute value is less than 100, the errors arising from the integer manipulations make the results very unreliable. Ideally, the input values may be scaled to within the range [-10 000 , 10 000], although the actual limits are -32767 to +32767. Maximum accuracy is obtained when the input data values are scaled within the actual limits.

Re-ordering a 1024-point input data array and carrying out a direct discrete Fourier Transform on an IBM PC AT operating at a clock speed of 6Mhz takes about 180 milliseconds.

3.4.3 Correlation

An assumption is made about the nature of the input data points in computing the correlation function. That is, there are two sets of real valued data points to be correlated. Let the two real valued data points be x(n) and y(n). Also let

X(m) = DFT(x(n))

Y(m) = DFT(y(n))

z(n) = x(n) + j y(n)

Z(m) = DFT(z(n))

Sxx = (| X(m) | ** 2) / N

Sxy = (X(m) * CONJUGATE(Y(m)) / N

Rxx = autocorrelation function

Rxy = crosscorrelation function

then

Sxx(m) = DFT(Rxx(k))

and

Sxy(m) = DFT(Rxy(k))

Sxx is called the auto power spectrum [16] and Sxy is called the cross power spectrum [16]. To compute the correlation function requires two direct Fast Fourier Transforms and one inverse Fast Fourier Transform. However, it is possible to operate on two sets of real valued inputs in a single step. Unscrambling the results will take less time than carrying out two Fast Fourier Transforms in a serial manner.

The x(n) values are read directly into the real parts of z(n) and the y(n) values are read directly into the imaginary parts of z(n). A direct Fast Fourier Transform is then carried out on z(n) to give Z(m). If Z(N) = Z(0), then it can be shown [1] that

X(m) = 0.5 * (Z(m) + Z(N - m))

and

Y(m) = -0.5 * j * (Z(m) - Z(N - m))

where j is the complex number operator.

Instead of unscrambling and then forming the appropriate complex product, the required product is formed from the real and imaginary parts of Z(m). The result is stored back to the same location to await the inverse Fast Fourier Transform. This last stage yields the correlation function Rxx or Rxy. This method is faster than the direct approach to calculating the correlation function, the speed advantage occurring for N greater than 30. It also cuts down on the memory requirements by about two thirds.

The set of data points cannot exceed 512 since each set of N data points is zero padded to 2 * N points and the limit for the Fast Fourier Transform routine is 1024 points. To avoid overflows, the final results are scaled by a factor of (N^2). The unscaled values may therefore be obtained by multiplying each output value by (N^2). In most situations, the relative values are enough since errors are always inherent in the data source and the computational machinery. In this system, for example, a 12-bit analog-to-digital converter is being used. If the analog-to-digital converter is operated in the differential mode, then signals whose amplitude is less than 1 / 4095 of the full range voltage cannot be represented. The sine table used in the Fast Fourier Transform subprogram has entries which are accurate to three significant figures. These are the two major sources of error. Another source of error is the discarding of the remainder after an integer division.

3.4.4 Convolution

It is usual to carry out cyclic convolution on periodic signals and linear convolution with aperiodic signals. However, it is difficult to classify real time signals since given a sufficiently long time an apparent aperiodic signal may turn out to be periodic. Hence both linear and cyclic convolutions are considered. The procedure for evaluating the convolution function is almost the same as that for the correlation functions.

The are differences are :

(a) Sxx is not used

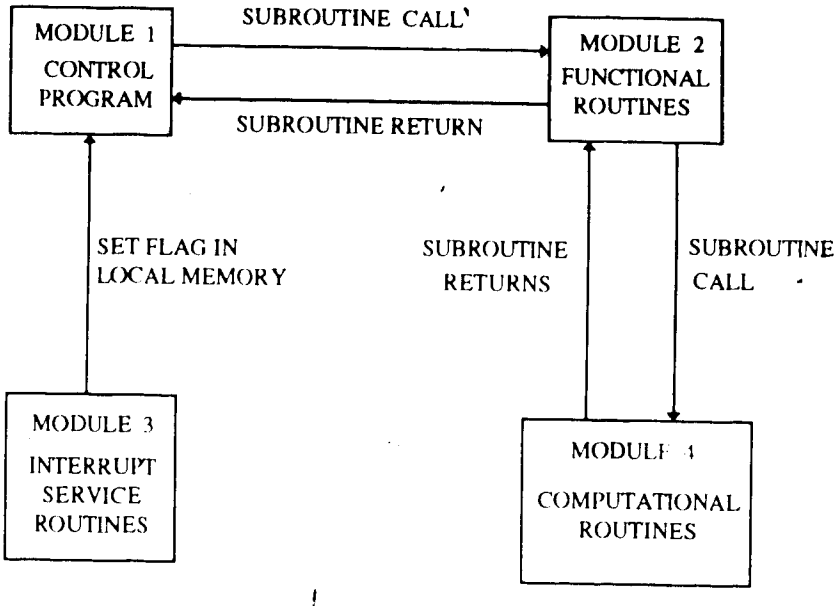


Fig. 3.3 Interaction between coprocessor modules

(b) Sxy is now given by $Sxy = (X(m) * Y(m)) / N$

The linear convolution is also limited to 512 points for each set of data since the data points are also zero padded. However, the cyclic convolution requires no zero padding and can therefore handle up to 1024 points for each set of data. The actual subprograms are discussed below.

3.5 Coprocessor Programs

The coprocessor routines are made up of four modules. Fig. 3.3 shows the interaction between the four modules. Module 1 is the control program for the coprocessor. At any given time, it calls one of the module 2 routines. The flow charts for the control program are shown in Figs. 3.1 and 3.4.

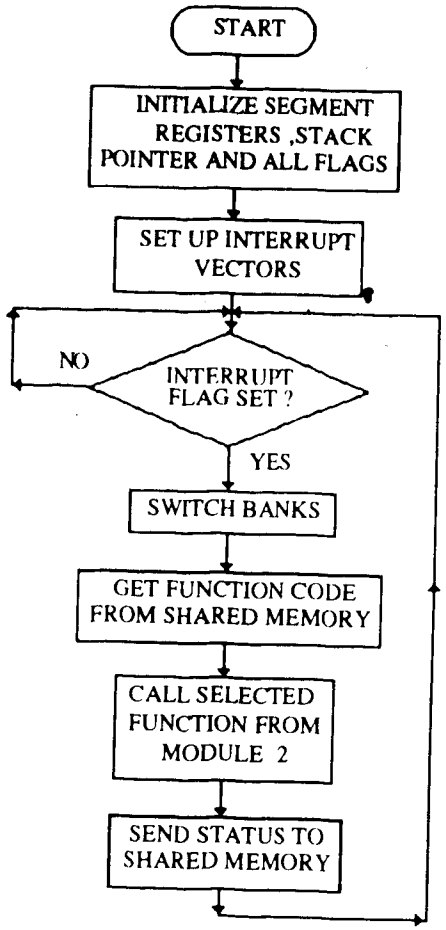


Fig. 3.4 Flow chart of coprocessor module 1

3.5.1 Module 2 Routines

Module 2 is made up of a number of independent functional subprograms. Each of these subprograms calls at least one of the subprograms from module 3. The flow chart of the coprocessor module 2 is shown in Fig. 3.5. The flow charts for the subprograms are shown in Fig. 3.6.

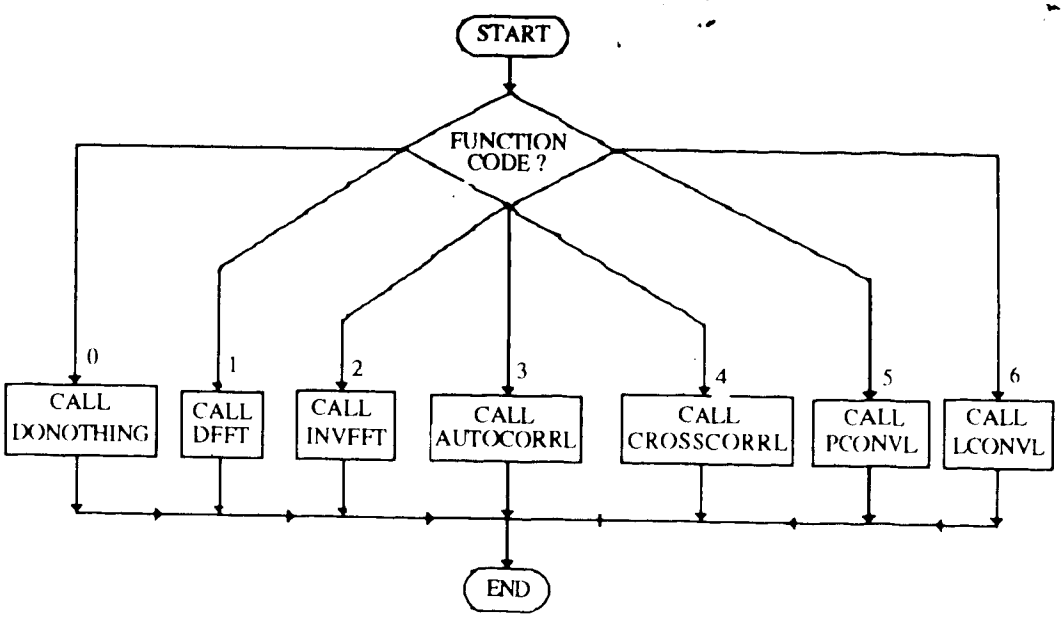


Fig. 3.5 Flow chart of coprocessor module 2

The list of module 2 routines are :

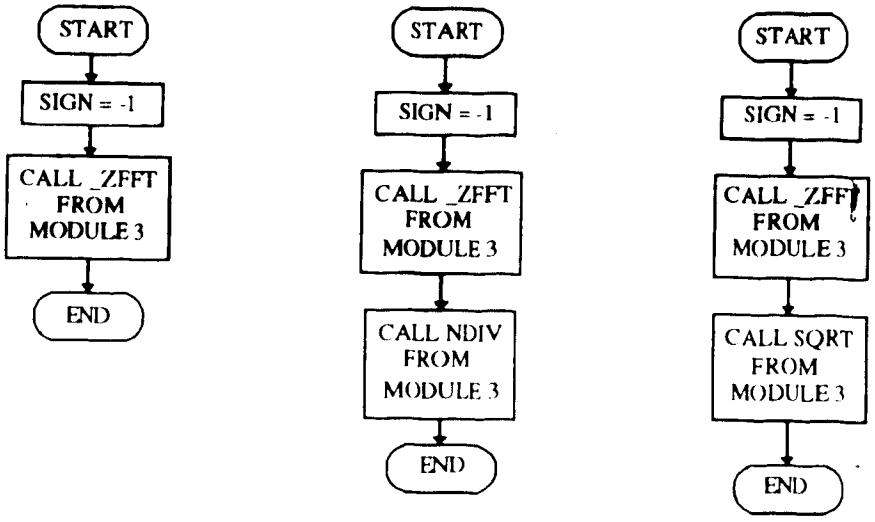
Routine name	*Purpose*
_DFFT	direct discrete Fourier Transform
_INVFFT	inverse discrete Fourier Transform
_PGRAM	power spectrum
_AUTOCORRL	autocorrelation
_CROSSCORRL	crosscorrelation
_PCONVL	cyclic convolution
_LCONVL	linear convolution

The common characteristics of all the module 2 routines are :

N = number of data points

M = $\log_2$ (N)

XREAL[] = real part of input/output data



(a) _DFFT (b) _INVFFT (c) _PGRAM

***Fig. 3.6** Flow charts of coprocessor module 2 subprograms*

XIMAG[] = imaginary part of input/output data

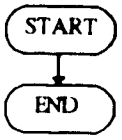
All the routines call _ZFFT from module 3. _DFFT calls no other routine. _INVFFT calls NDIV from module 3. All the correlation and the convolution routines call NDIV and SORT. With the exception of the cyclic convolution routine, these routines also call PAD0 from module 3.

3.5.2 Fast Fourier Transform Routine (ZFFT)

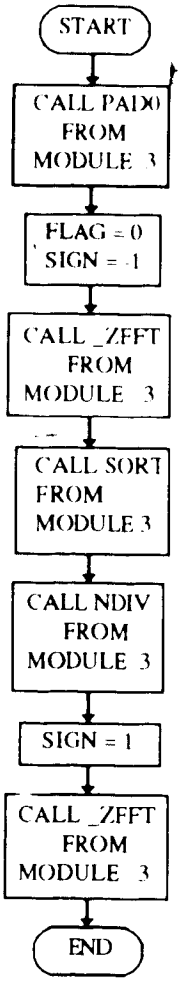
The most important routine in module 3 is the Fast Fourier Transform routine, _ZFFT. A flow chart of _ZFFT is shown in Fig 3.7. Its main characteristics are :

input data : XREAL[], XIMAG[]

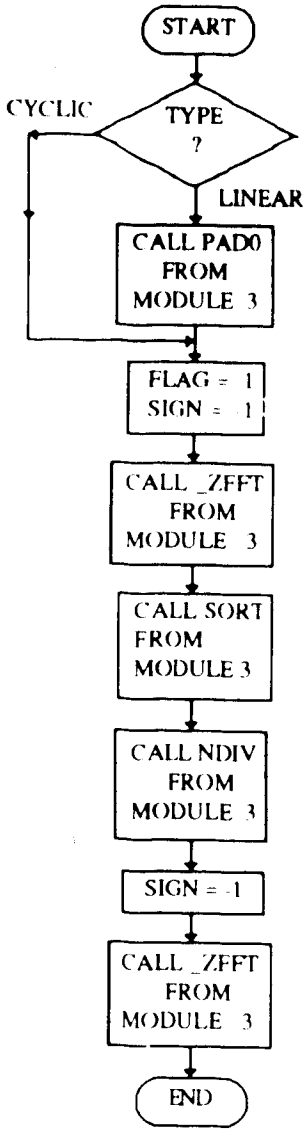
number of data points : N



(d) DONOTHING



(e) AUTO- and CROSS CORRELATION



(f) LINEAR and CYCLIC CONVOLUTION

Fig. 3.6 cont'd

M	=	$\log_2(N)$
SIGN	=	-1 for direct discrete Fourier Transform
SIGN	=	1 for inverse discrete Fourier Transform
output data	:	XREAL[], XIMAG[]
subroutine called	:	MFT

The subroutine _MFT computes the product of two 32-bit signed integers and then divides the result by 16834. Other routines in module 3 are SORT and PAD0. PAD0 pads N data points with N zeros and sets the value of N to 2 * N. SORT computes the functions Sxx(m) and Sxy(m) from Z(m) as discussed in an earlier section.

3.5.3 Module 4 Routines

Module 4 is made up of three interrupt service routines. They are NMISER, OVERFLOW, AND DIVBY0ERR. As the name of each one suggests, the NMIser routine is called whenever a non-maskable interrupt is generated. It sets a flag in the local memory of the coprocessor to indicate to the coprocessor that a non-maskable interrupt has occurred. The coprocessor monitors this flag. It then switches banks when the flag is set and it has finished with its current task.

The OVERFLOW and DIVBY0ERR routines serve as traps for accumulator overflow and divide by zero errors. Each sets an arithmetic error flag when an arithmetic error occurs.

At the end of a function call from module 2, the control program moves the arithmetic flag to a location in the shared memory. After bank switching, the IBM PC XT can read this flag . The flag setting will determine whether the function from module 2 was successfully executed.

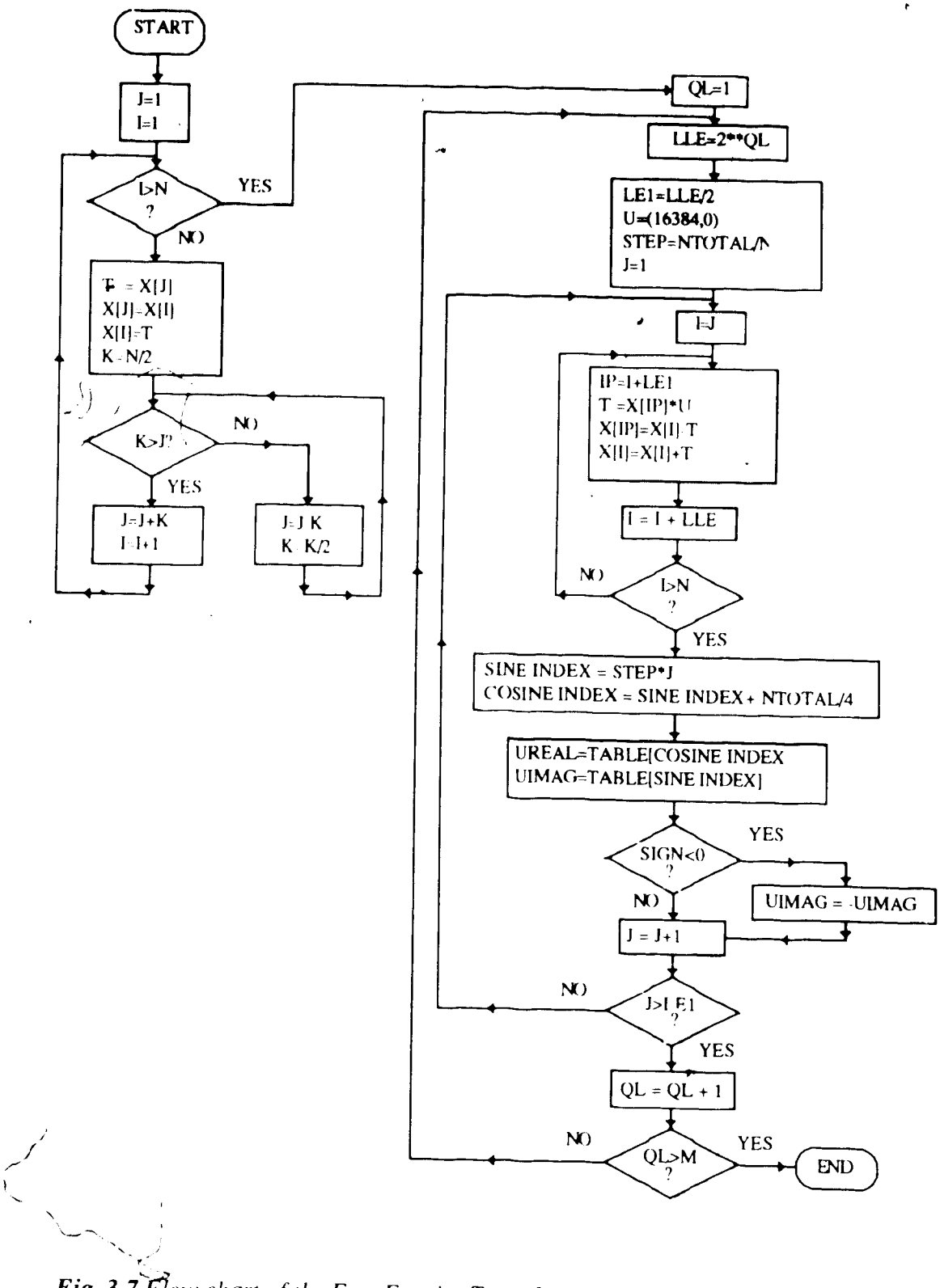


Fig. 3.7 Flow chart of the Fast Fourier Transform subprogram _ZFFT

4. IBM PC XT ROUTINES

The IBM PC XT routines developed are made up of five modules, the first being MODULE 0 and the last being MODULE 4. Fig. 4.1 shows the interaction between the five modules. Although MODULE 5 appears on the diagram, it is actually an extension of MODULE 2. MODULE 5 handles the filter routines which are not yet implemented. However, provisions have made for the filter routines in both the IBM PC XT modules and the coprocessor modules.

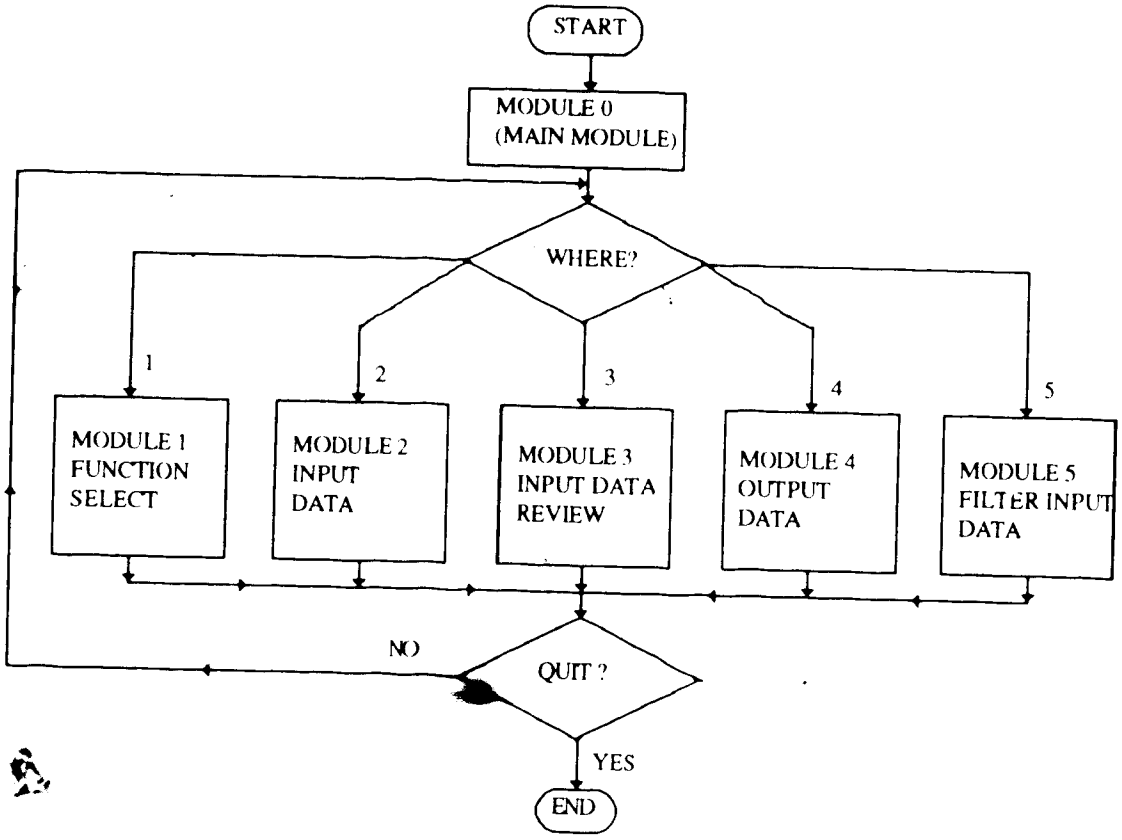


Fig. 4.1 IBM PC XT routines - interaction between modules.

Each module, except the first, is made up of a main routine and a number of subroutines. The routines in a particular module work together to fulfill a particular task. MODULE 0 initializes the coprocessor board, the shared memory, the IBM PC XT local variables, and the data acquisition system. An infinite loop following the initialization enables the other modules to be called, one at a time. A flow chart of MODULE 0 is shown in Fig. 4.2. The IBM PC XT initializes the coprocessor board by loading the code to be executed by the coprocessor into the shared memory and removing the reset signal from the reset pin of the microprocessor on the coprocessor board. Next, the sine table used by the ZFFT routine (one of the coprocessor routines) is then loaded into both banks of memory. Each bank is then given an identification number and all flags are set to their inactive states.

The data acquisition system is initialized by setting a default sampling frequency of 1000 Hz and a default analog-to-digital channel address of 0. All the timers are also stopped.

4.1 Menu Display and Option Selection

All the modules have the same basic structure. Each has a main routine that calls other routines. Each has a screen text which makes up the menu display. The user then selects an option from the menu display by means of the cursor and the <ENTER> keys. The screen display for all the modules is performed by the routine called "menu". This routine selects the screen text for a particular module and then displays it on a text screen. The "menu" routine then draws a box around the displayed text.

The default option selected is "QUIT". This is highlighted. The "UP" and "DOWN" cursor keys are used to highlight the other options. A highlighted option is selected with the <ENTER> key. Selecting an option with the <ENTER> key also terminates the "menu" program. The selected option is returned to the calling program in a coded form. A flow chart for the "menu" routine is shown in Fig. 4.3.

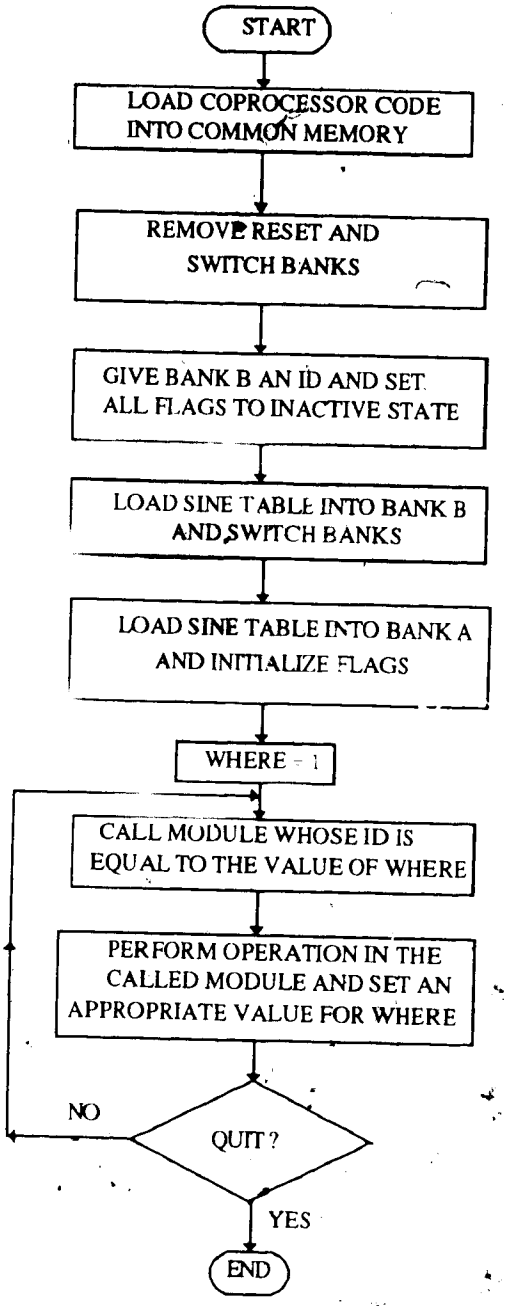


Fig. 4.2 Flow chart of IBM PC MODULE 0

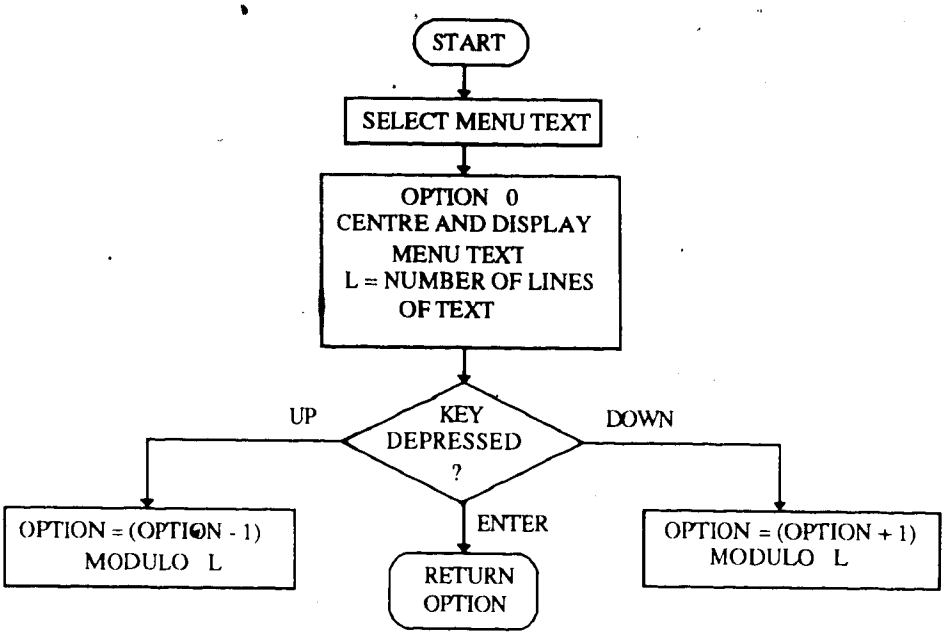


Fig. 4.3 Flow Chart of "menu" Routine.

4.2 MODULE 1

The main program of MODULE 1 first calls the "menu" routine described above. If the user decides not to abort the program, then three flags are set, based on the option selected by the user. The flags are

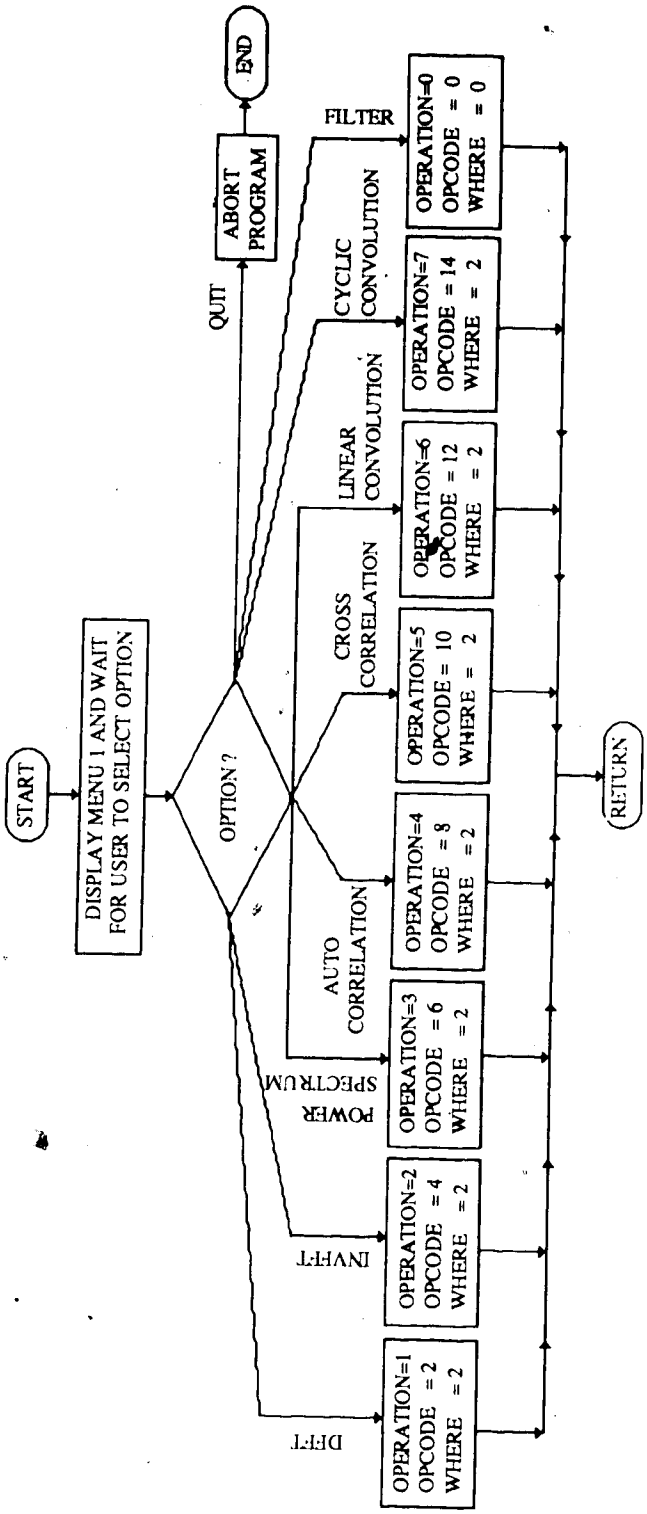
(a) OPERATION

(b) OPCODE

(c) WHERE

A flow chart of MODULE 1 is shown in Fig. 4.4. The OPERATION flag is used by the other modules to select the function to be performed. The functions, their OPERATION flags and OPCODES are shown below:

function	*OPERATION FLAG*	*OPCODE*
QUIT	-	-
DO NOTHING	0	0



4.4 Flow chart of IBM PC software module 1

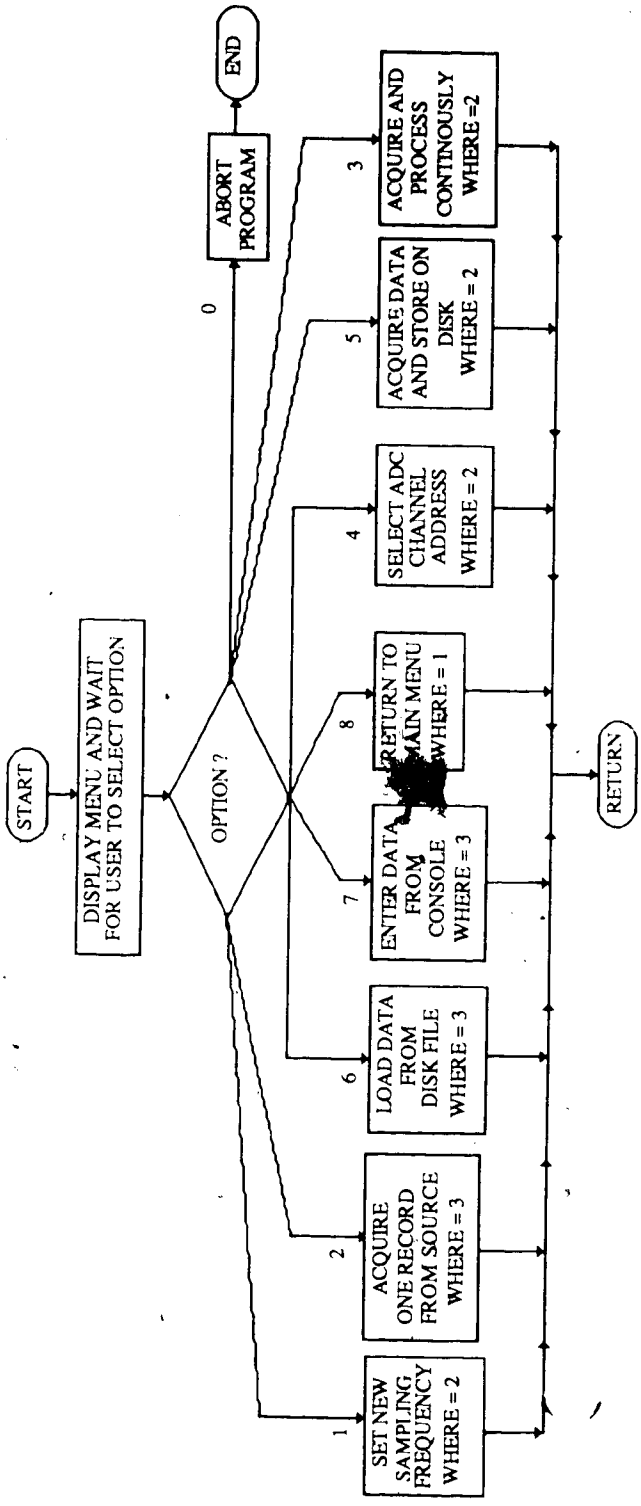


Fig. 4.5 Flow chart of IBM PC module 2

DFFT	1	2
INVFFT	2	4
PGRAM	3	6
AUTOCORRELATION	4	8
CROSSCORRELATION	5	10
LINEAR CONVOLUTION	6	12
CYCLIC CONVOLUTION	7	14
FIR LOW PASS FILTER	8	16
FIR HIGH PASS FILTER	9	18
FIR BAND PASS FILTER	10	20
IIR LOW PASS FILTER	11	22
IIR HIGH PASS FILTER	12	24
IIR BAND PASS FILTER	13	26

The OPCODE flag is an offset into a jump table. The jump table is used by the coprocessor main routine to select the function to be performed. The flag WHERE is used for communication between the modules. Its value is used by the main IBM PC program to call one of the five modules.

4.3 MODULE 2

MODULE 2 is the input data module. It allows the user to input data from a number of sources. MODULE 2 starts with the main program calling the "menu" routine described in section 4.1. The options available to the user in MODULE 2 are:

(a) abort the program

(b) set a new sampling frequency

(c) select analog-to-digital converter channel address

(d) acquire a data record from the data acquisition system

(e) continuous processing

(f) acquire a long data record and store it in a disk file

(g) load a record of data from a disk file

h) enter a record of data from the console

(i) go to module 1

Each option is serviced by a routine. The outlines of the routines are given below.

4.3.1 NEWFREQ

The routine NEWFREQ accepts the sampling frequency Fs a as double precision number from the keyboard. It then calls the routine SETFREQ. The routine SETFREQ selects the appropriate 16-bit control word for timer #1 of the counter/timer on the data acquisition system board. It also calculates the 16-bit value required by the load register 1 of the timer. This 16-bit value is stored in the variable COUNTVAL. A flow chart of the SETFREQ routine is given in Fig. 4.6.

$$\text{TIMER COUNT} = \text{COUNTVAL} = (\text{Fsrc} / \text{Fs}) - 1$$

There are five possible frequency sources for each of the timers. These frequencies are derived from from the standard 1.00 Mhz clock on the data acquisition system board. Frequency divisions of 1, 10, 100, 1000, 10000, 16, 256, 4096, and 65536 are possible. The SETFREQ selects a frequency source so as to give a high TIMER COUNT value.

4.3.2 NEWADCCHNL

The analog-to-digital converter on the data acquisition system board has 16 channel in the single-ended mode and eight in the differential input mode. A channel address must be sent to the board before starting any conversion. The routine NEWADCCHNL accepts the ADC

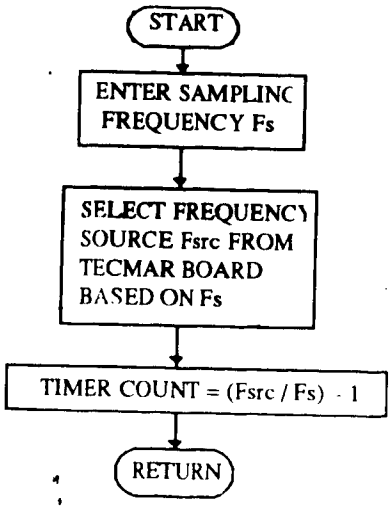


Fig. 4.6 Flow chart of IBM PC routine SET_FREQ

channel address from the keyboard and stores it as the variable CHANNEL. This variable is used by the routine which reads the ADC output.

4.3.3 ACQUIRE

The ACQUIRE routine acquires a record of data from the data acquisition system. A flow chart of this routine is shown in Fig. 4.7 The record is read into the variables XQREAL[] and XQIMAG[] so that it can be processed by one of the functional routines on the coprocessor board. The record length cannot exceed 1024 points. The routine first prompts the user to enter the number of data points. It then calls the routine INIT_TECMAR to initialize the data acquisition system. The routine START_TIMER starts timer #1 after loading the value of COUNTVAL into load register 1.

The data is acquired through timer interrupts and the interrupt handler is called READADC. The routine READADC first acknowledges the timer interrupt and then outputs the ADC channel address to the appropriate port. It then initiates a software start

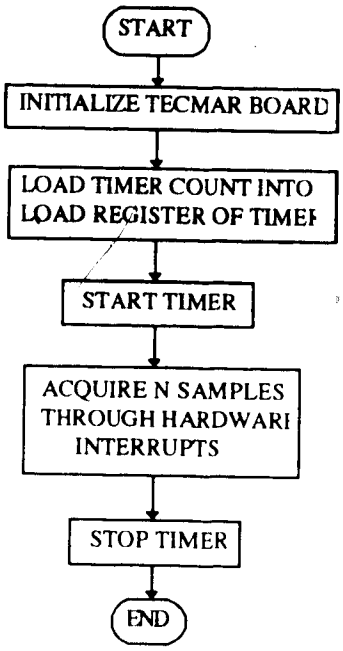


Fig. 4.7 Flow chart of IBM PC routine ACQUIRE

of conversion and reads the ADC output when conversion is complete. The current number of data points is held in the variable DATACOUNT. If this number not greater than the required number of data points, then the ADC output is stored in XQREAL[DATACOUNT] and the value of DATACOUNT is increased by one. Otherwise the output of the ADC is simply ignored.

When the required number of samples have been acquired, the timer is placed in a hold state until the next acquisition.

4.3.4 ACQPROCESS

The usefulness of the coprocessor board becomes evident when data is to be acquired and processed continuously. The routine which handles the continuous processing is called ACQPROCESS for lack of a better name. A flow chart of this routine is shown in Fig. 4.8. A record is acquired from the data acquisition system into shared memory

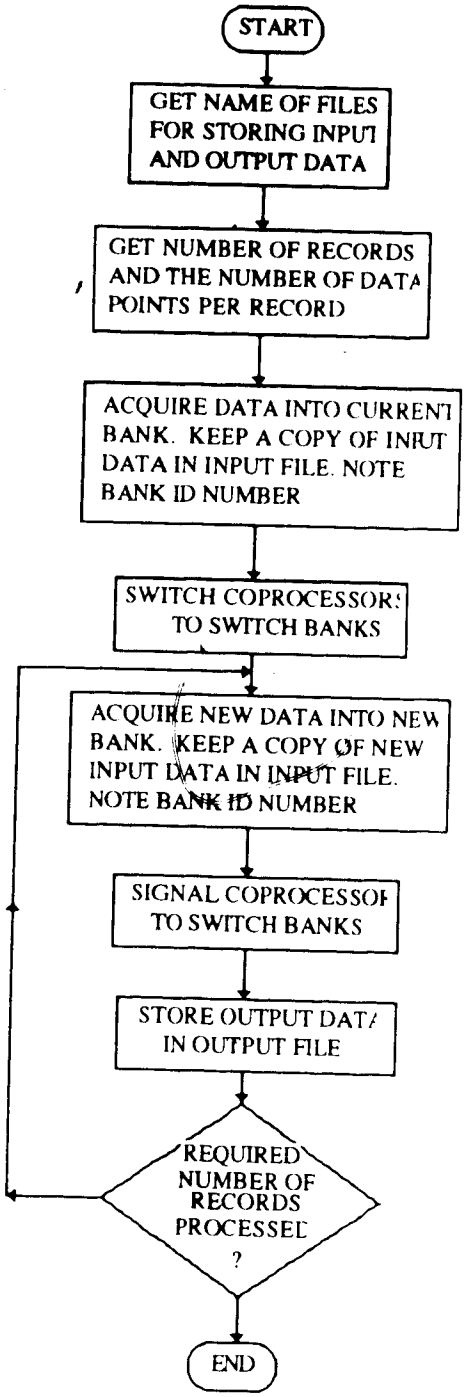


Fig. 4.8 Flow chart of ACQPROCESS

and a copy of it is stored on a disk file. The coprocessor is then signalled to process the data and the IBM PC XT proceeds to acquire a new record into a new bank, once again keeping a copy of the new record in the same input file.

When both systems have finished with their respective tasks the banks are switched. While the coprocessor is working on the next record, the IBM PC XT stores the output the previous record into an output file and then proceeds to fill the shared memory with a new record. The cycle repeats until the required number of records have been processed. The user can examine both the input and the output files later.

4.3.5 ACQTODISK

The ACTODISK routine enables the user to acquire record of any length from the data acquisition system and store it on disk. This is the only routine which has no limit on the number of samples per a record. The acquired data may be edited and processed later.

4.3.6 LOAD

The LOAD routine loads data from a disk file and stores it the shared memory for later processing. The data may be in a variety of formats, depending on the operation to be performed. A flow chart of the LOAD routine is shown in Fig. 4.9. The number of data points may be specified in the input file or may be entered from the keyboard. The user is prompted by the program to select one of the two options. If the number of data points is specified in the input file, it is the first number in the file. The number of data points cannot exceed 1024.

4.3.7 ENTER_DATA

The ENTER_DATA routine is a variation of the LOAD routine. Instead of reading the input data from a disk file, it accepts the data from the keyboard. It is useful for

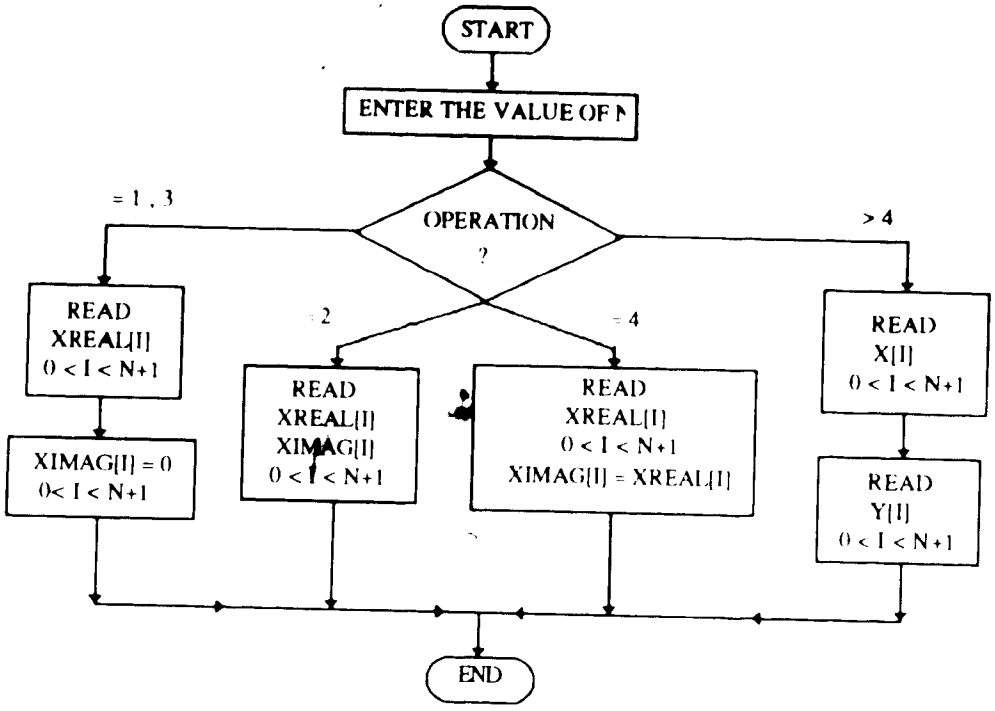


***Fig. 4.9** Flow chart of IBM PC subprogram LOAD*

debugging purposes.

Valid input data operations from MODULE 2 enable MODULE 0 to transfer control to MODULE 3, the input data review and processing module. Transitions which may result in the system crashing are not allowed by the program.

4.4 MODULE 3

Fig. 4.10 shows the flow chart of MODULE 3. The options available to the user in this module are:

(a) abort the program

(b) display input data on the console

(c) modify the input data

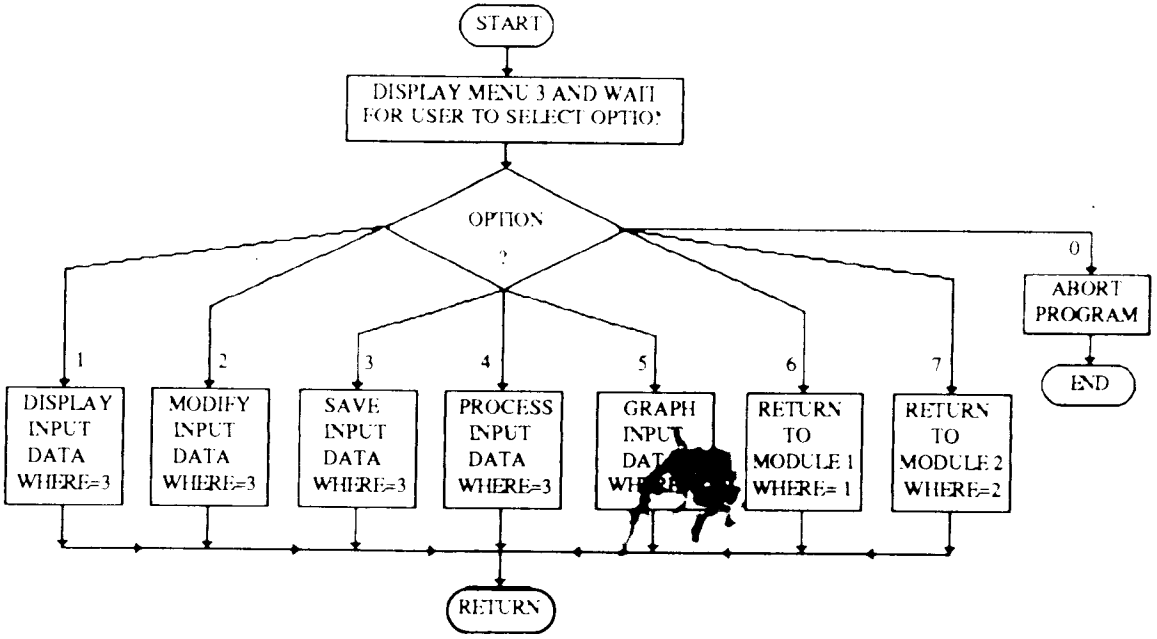


Fig. 4.10 Flow chart of IBM PC XT module 3

(d) save input data on a disk file

(e) process input data

(f) display the input data graphically

(g) go to module 2

(i) go to module 1

The routine DISPLAYIN simply sends the input data to the standard screen while the routine SAVEIN saves the same input data on a disk file. The SAVEIN routine saves the input data in a format that can be read by the LOAD routine described in section 4.3.6. The MODIFYIN routine enables very limited editing to be done on the input data. The PROCESS_DATA routine simply sends a signal to the coprocessor to process the input data and return results to the IBM PC XT. The user option to display the input data graphically is very complicated.

This option first enables a suitable title to be selected for the graph. It also sets a flag to indicate whether the data is one of the following:

(a) complex

(b) a single array

(c) two single arrays

It then calls the graphic routine to do the plotting.

4.5 MODULE 4

The MODULE 4 routines are similar to the MODULE 3 routines. The only differences between the two modules are:

(a) The MODULE 3 routines operate on input data while the MODULE 4 routines operate on output data.

(b) The format of the output data is usually different from the format of the input data.

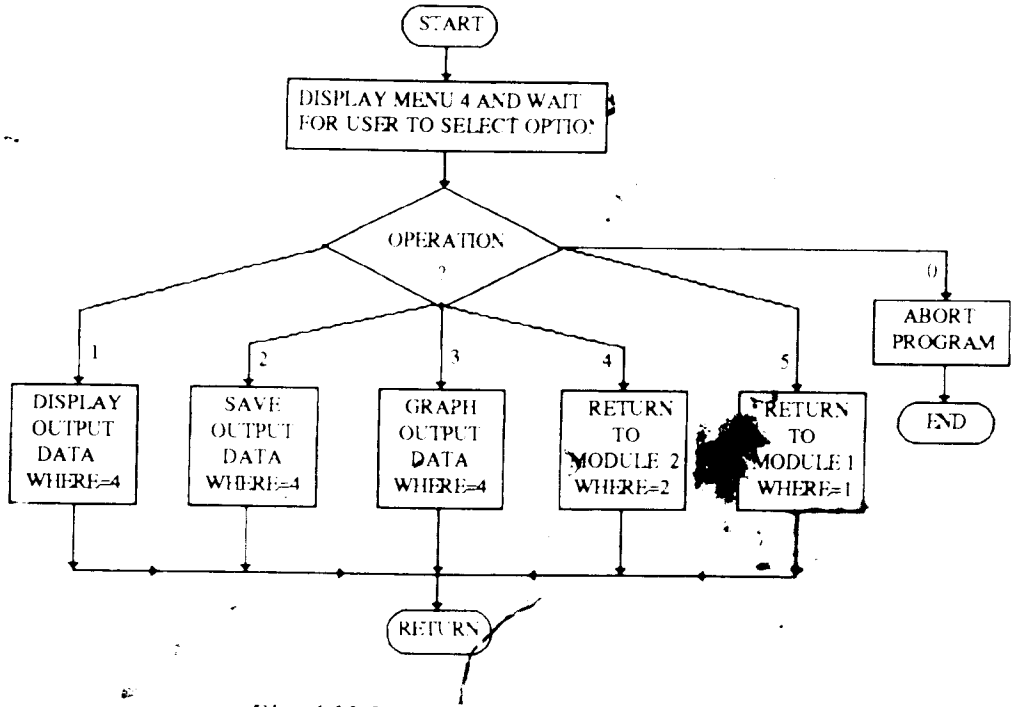


Fig. 4.11 Flow chart of IBM PC module 4

A flow chart of the MODULE 4 is shown in Fig. 4.11. The options available to the user are:

(a) abort the program

(b) display output data on the console

(c) save output data on a disk file

(d) display the output data graphically

(e) go to module 2

(f) go to module 1

4.6 MODULE 5

MODULE 5 is an extension of MODULE 2. It provides a menu text display and option selection feature for the filter routines. A flow chart of MODULE 5 is shown in Fig. 4.12. The options available to the user are:

(a) FIR LOW PASS FILTER

(b) FIR HIGH PASS FILTER

(c) FIR BAND PASS FILTER

(d) IIR LOW PASS FILTER

(e) IIR HIGH PASS FILTER

(f) IIR BAND PASS FILTER

The routines associated with these options are not yet implemented but templates have been provided for them throughout the entire package. The actual routines may be implemented in assembler if a suitable method of calculating the coefficients is determined.

4.7 GRAPHIC DISPLAY

There are three main routines which make up the graphic display routines. They are:

(a) GETabsMAX

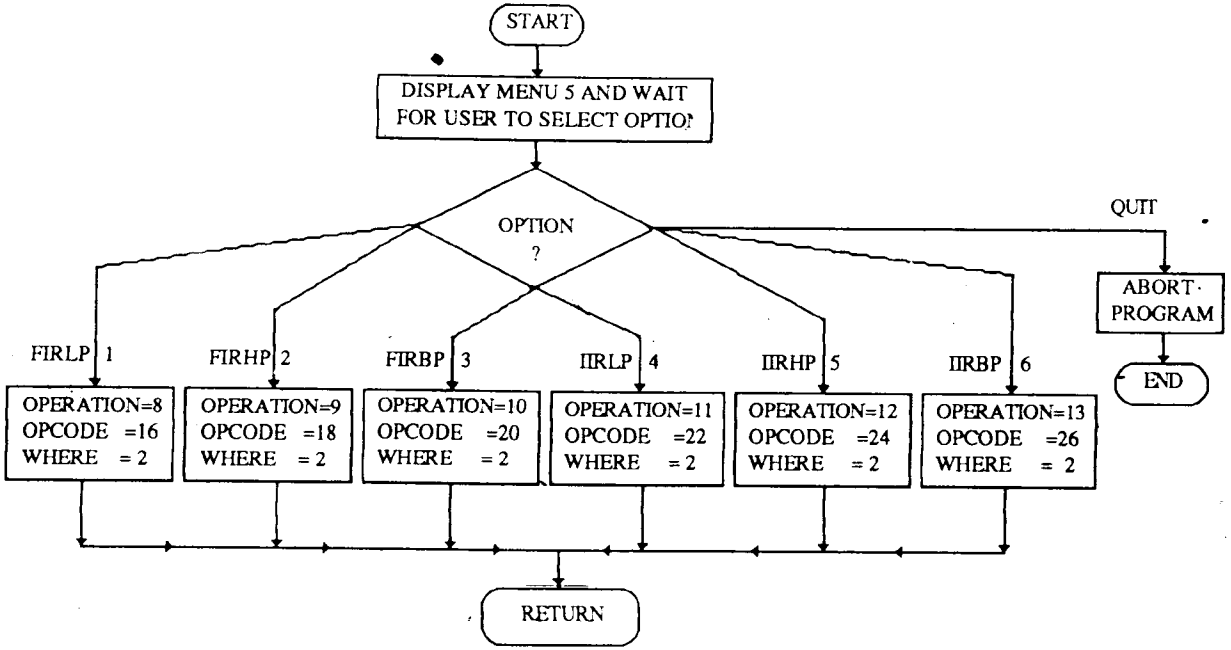


Fig. 4.12 Flow chart of IBM PC XT module 5

(b) GRAPH

(c) SCALE_DATA

(d) PLOThiN

These three routines are supplemented by a number of lower level routines from the Microsoft C version 5.00 graphics library. The GETabsMAX routine returns the maximum absolute value of a set long integers. For complex data points, the routine is used to obtain the maximum absolute value of the real and imaginary parts separately. The overall absolute maximum value is then easily determined from the two values.

The SCALE_DATA routine scales all the data points so that each of them lies within the closed interval [-100, 100]. The routine GRAPH acts as the main routine for the graphic display routines. It calls the other three routines.

4.7.1 GRAPH

A flow chart of the routine is shown in Fig. 4.13. This routine first sets the display mode to high resolution graphics mode (enhanced graphic mode 16). The screen is then partitioned into a series of overlapping colors. The Microsoft C graphics library has two coordinate systems for the graphic screen - the physical and the logical coordinate systems.

The physical coordinate system has a fixed origin at the upper left corner of the screen. The logical coordinate system has a variable origin and this origin may be located anywhere on the screen. The logical coordinate system is used throughout the graphic display routines.

The logical origin is set. The axes are then drawn and labeled. The graph is given a title and the data points scaled. The scaling of the data is performed by the routine SCALE_DATA. The routine PLOThiN is called to plot the points. On return from PLOThiN the screen is returned to the text mode.

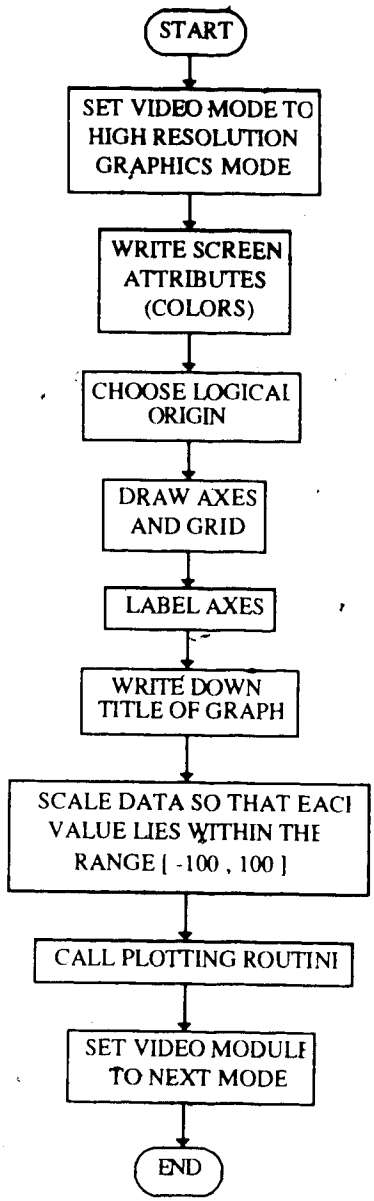


Fig. 4.13 Flow chart of graph plotting subprogram

4.7.2 PLOThiN

Fig. 4.14 shows the flow chart of the PLOThiN. For data points less than 100, the entire graph fits on the screen. For large number of data points, only the first 125 points are displayed on the screen. The cursor keys may be used to scroll the graph to the left or right. The <ENTER> key terminates the scroll process and Q quits the PLOThiN routine returns control to the GRAPH routine.

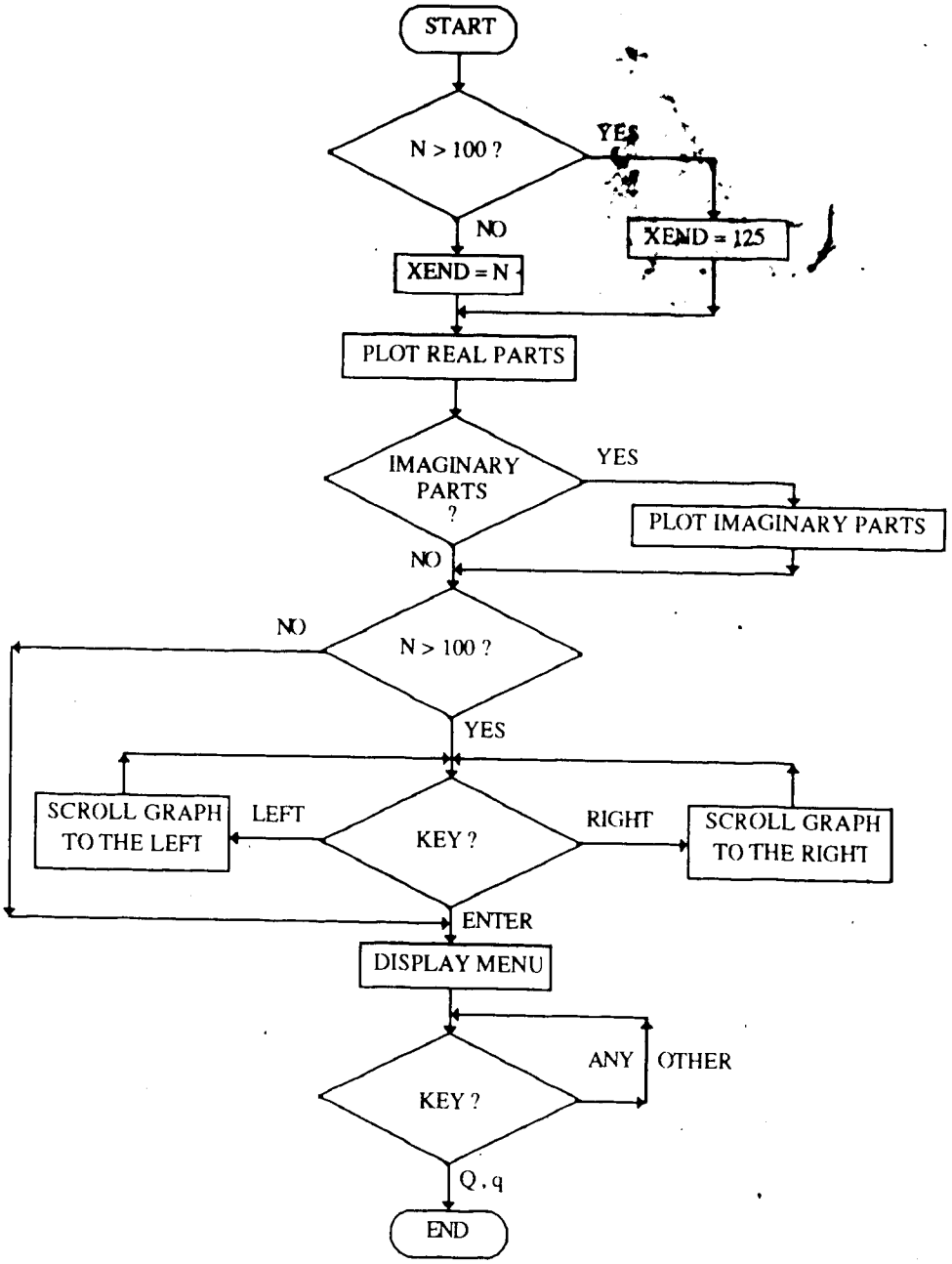


Fig. 4.14 Flow chart of PLOThiN

5. RESULTS

The IBM PC software was written first and tested. Each subprogram of all of the four modules was tested to ensure that it performed correctly. at all times. Next, the control program of each module was tested to ensure that it calls the correct subprogram at any given time. The user interface was then tested and the problems associated with unflushed keyboard buffer was eliminated.

The second phase of the tests involved the hardware of the signal processor. The common memory banks located on the circuit board of the signal processor was first tested. This was followed by the testing of the central processing unit of the signal processor. Finally, the testing of the hardware interrupt mechanism and the input/output port of the signal processor was done.

The third phase of the tests involved the signal processor software.The subprograms of the signal processor were initially tested by running each of them in the local memory of the IBM PC . They were then tested again in the local memory of the signal processor.

The fourth phase of the testing involved the timer/counters of the data acquisition system. The tests consisted of programming the control registers of the timer/counters to generate the correct sampling pulses. The analog-to-digital converter was tested to ensure it worked correctly.

5.1 IBM PC XT Software Tests

Module 0 of the IBM PC software is the control program for all the other modules. It calls module 1 when the program is run.

Module 1 displays a menu when it is called. This menu is shown in Plate 1. Like all the other menus, the menu text appears in black and it is surrounded by a light cyan box.

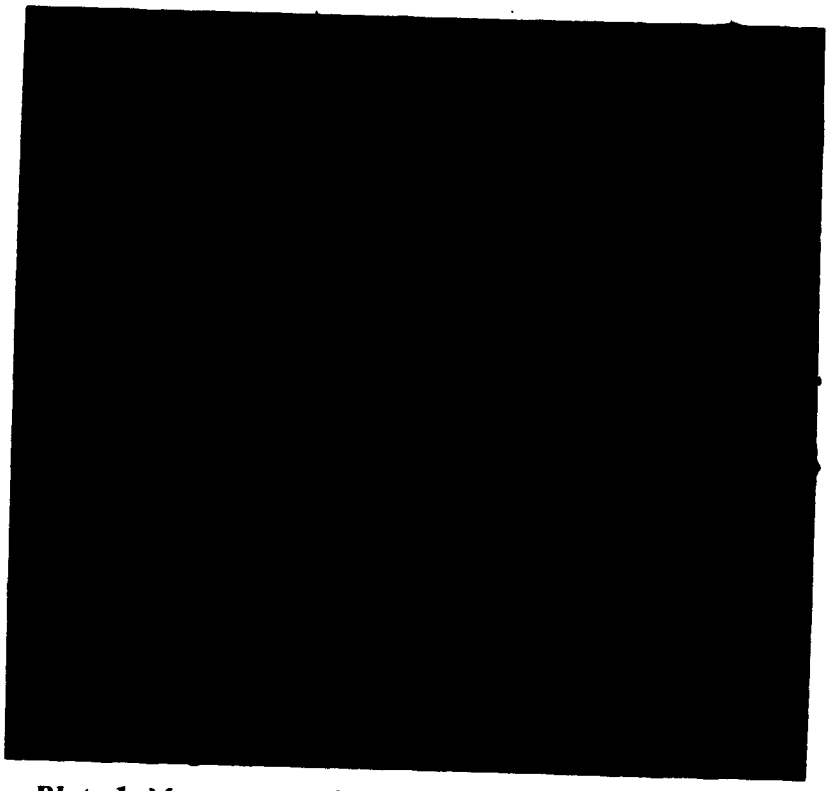


***Plate 1**. Menu screen for IBM PC module 1*

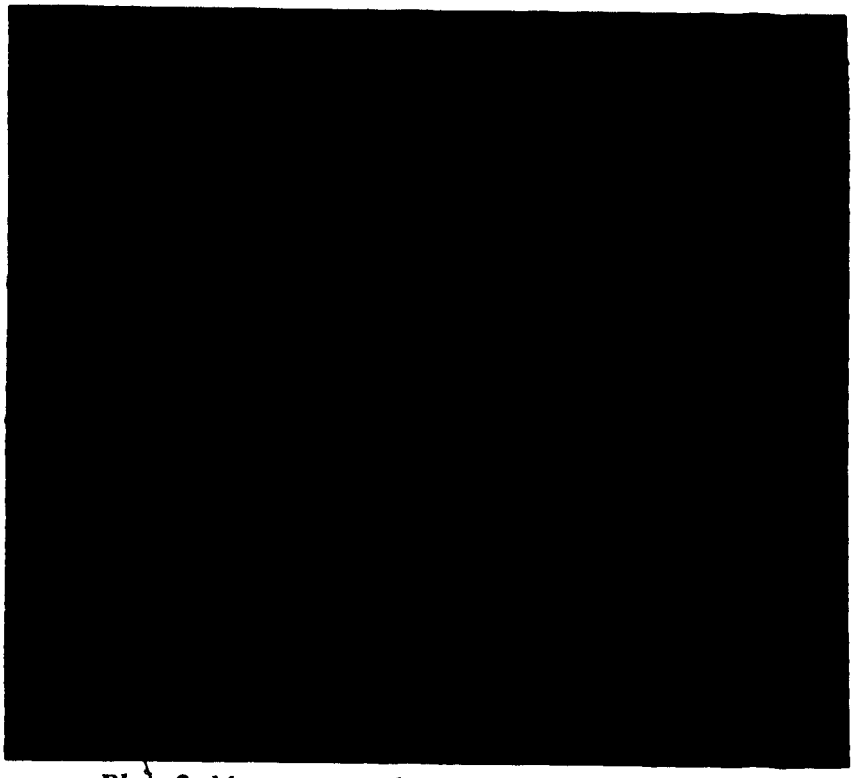


Plate 2. Menu screen for IBM PC module 2

The background of the menu screen is black. The picture in Plate 1 was taken off the screen of an enhanced color monitor.

The options available to the user in the first menu are shown in Plate 1. One of the options is highlighted in red. The "UP" and "DOWN" arrow keys enable the user to change of the highlighted option. A highlighted option is selected by pressing the <ENTER> key on the keyboard. The screen blinks and the menu associated with module 2 of the IBM PC software is displayed on the screen.

The menu associated with module 2 of the IBM PC software is shown in Plate 2. The options enable data to be acquired from either a disk file or from the data acquisition system. Data can only be read from a disk file if the a valid data file exists in the current directory. Before the data acquisition system can be used, the user has to set the sampling frequency and the analog-to-digital converter channel address. These values may be set only or changed from time to time. A sampling frequency between 10000 Hz and 0.002 Hz may be selected. The upper limit of 10000 Hz gives the data acquisition system time to convert the analog data to digital form and store it in common memory. The analog-to-digital converter has 16 channels but for input signals that have both positive and negative values, the channels are paired. The valid channel numbers for this application are 0 - 7.

The menu options in Plate 2 that begin with "ACQUIRE" obtains data from the data acquisition card and each of them assumes that a suitable sampling frequency and analog-to-digital channel address have already been selected. Once data has been acquired, the user has the option of applying a Hanning window [16] to the input data. Windowing the input data reduces the leakages due to the discontinuities in the input function waveform[15]. After the input data has been acquired, the menu associated with the IBM PC module 3 is displayed.

The menu associated with IBM PC module 3 is shown in Plate 3. This menu enable the user to display the input data on the screen, save it on a disk, or process it. The input

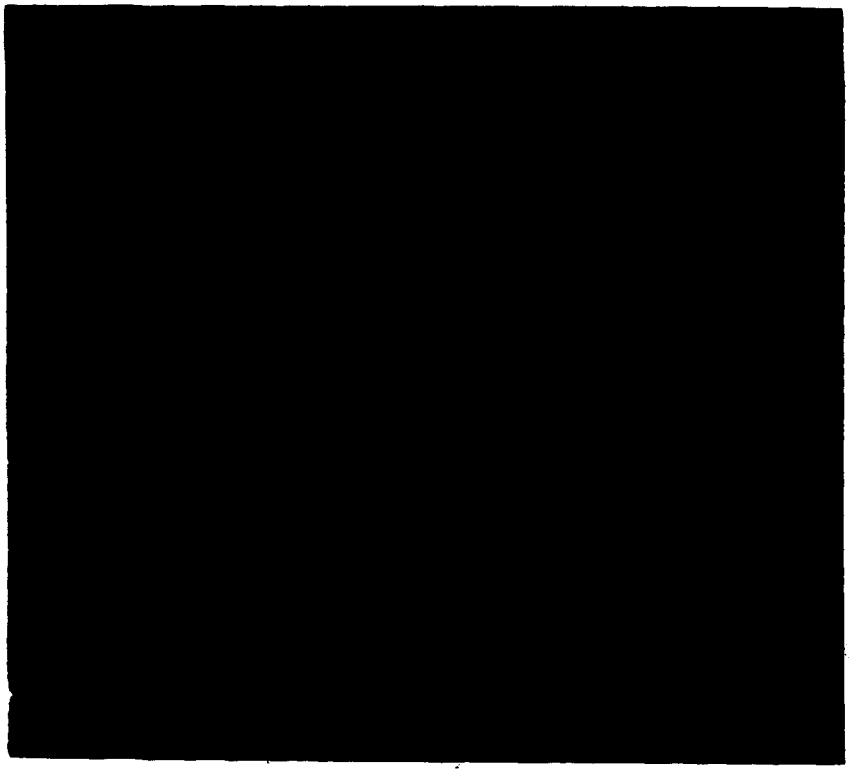


Plate 3. Menu screen for IBM PC module 3

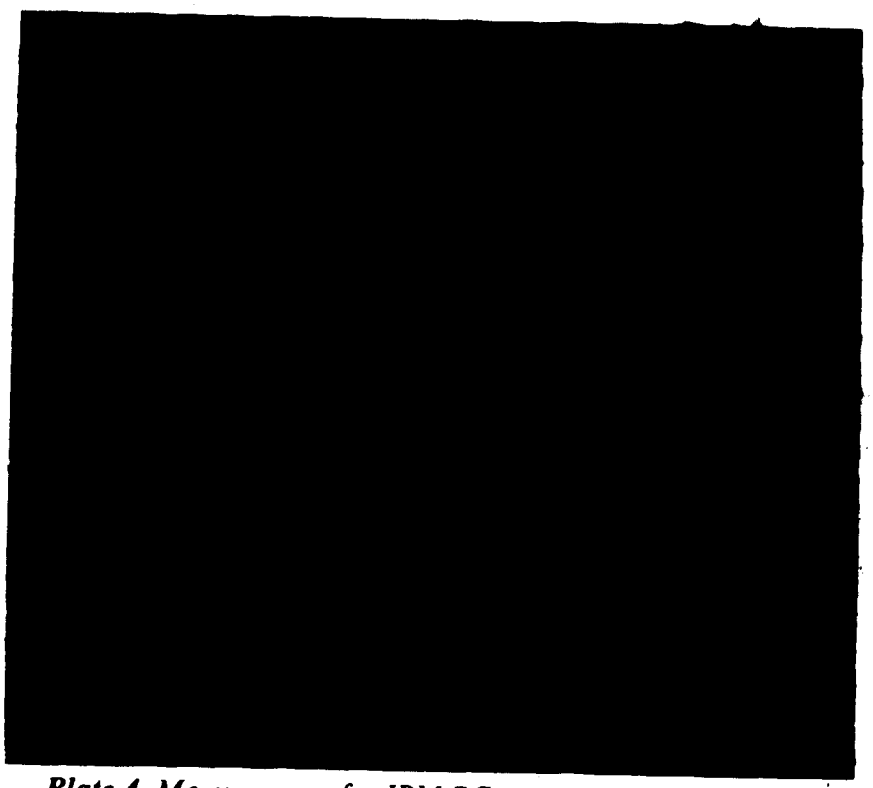


***Plate 4**. Menu screen for IBM PC module 4*

data may be displayed graphically or as numbers. For large data sizes, sufficient data to fill the screen is displayed and the user makes use of the cursor and the <ENTER> keys to display the rest of the data. Prompts on the screen tells the user exactly what to do.

The selection of the "PROCESS INPUT DATA" option enable the transition from the IBM PC module 3 to IBM PC module 4 to take place.

The menu associated with IBM PC module 4 is shown in Plate 4. This menu enable the user to save the output data or display it on the screen. The data may be displayed graphically or as numbers. The user may return to the menu associated with the IBM PC module 2 by selecting the "INPUT NEW DATA" option.

The user may abort the program from any of the 4 menus by selecting the "QUIT" option.

5.2 Graphic Display

The low level graphic routines from the Microsoft C graphics library make the graphic screen update fast and scrolling is achieved with virtually no flicker. The text on the graphics screen has good resolution. The white background makes the other colors stand out clearly and the high resolution mode enable a lot of information to be displayed on the screen.

Both the input and the output data may be displayed graphically. Plate 5 shows the graphic screen of the input data for a direct Fourier Transform. The screen has a white background and light cyan grid lines. The grid lines enable values to be read from the graph more easily. The two axes are drawn in red and labelled in light magenta. The numbers on the vertical axis in terms of the percentage of the absolute maximum value of either the real parts and the imaginary parts of the data. The numbers on the horizontal axis refer to either the number of sample periods (time domain data) or the harmonics (frequency domain data). The title of the graph and the value of the absolute maximum value of the data appear

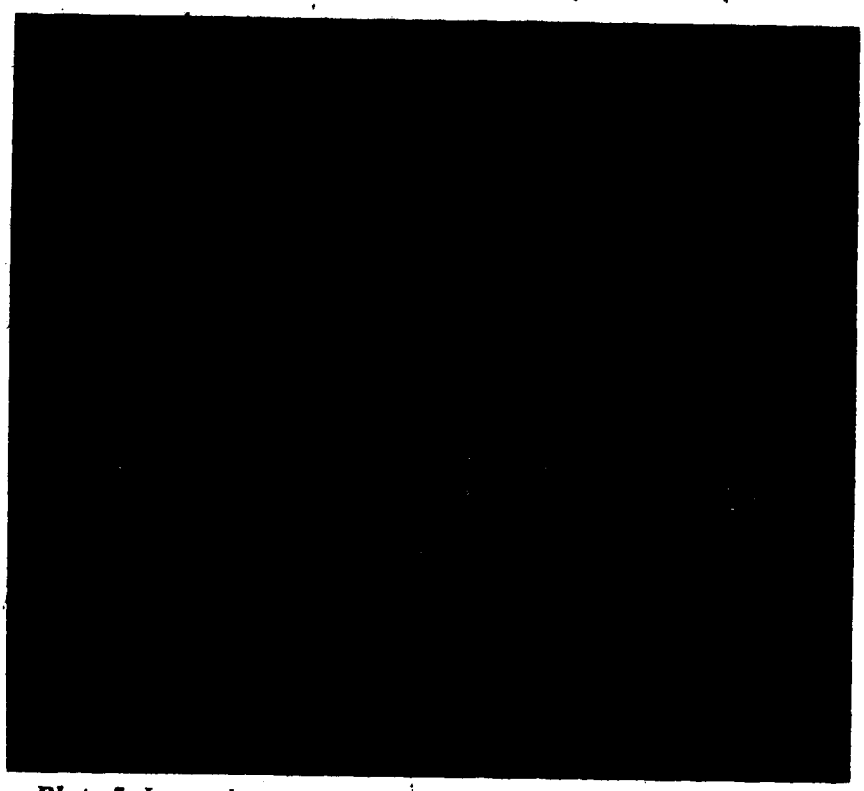


***Plate** 5. Input data - Direct Fourier Transform*

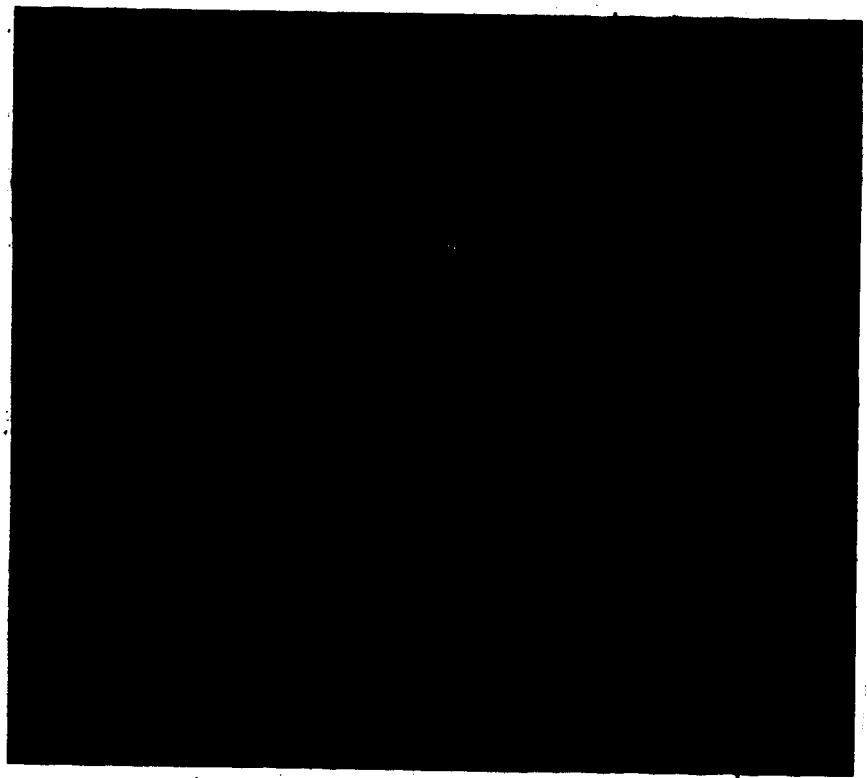


***Plate 6**. Output data - Direct Fourier Transform*

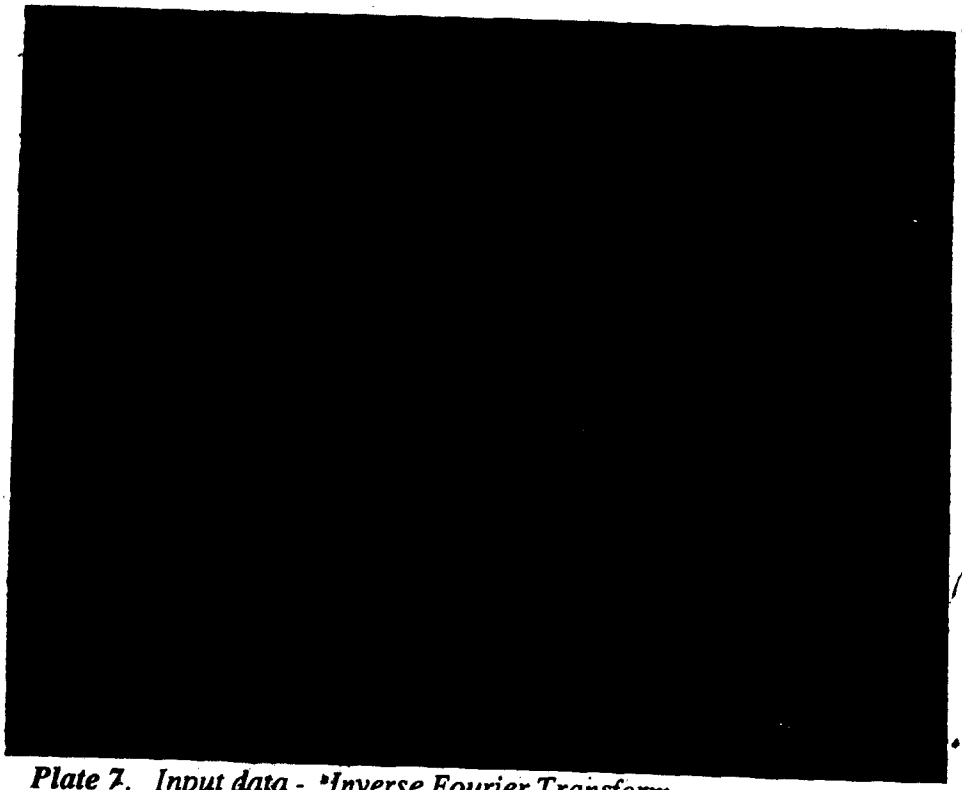


Plate 7. *Input data - Inverse Fourier Transform*

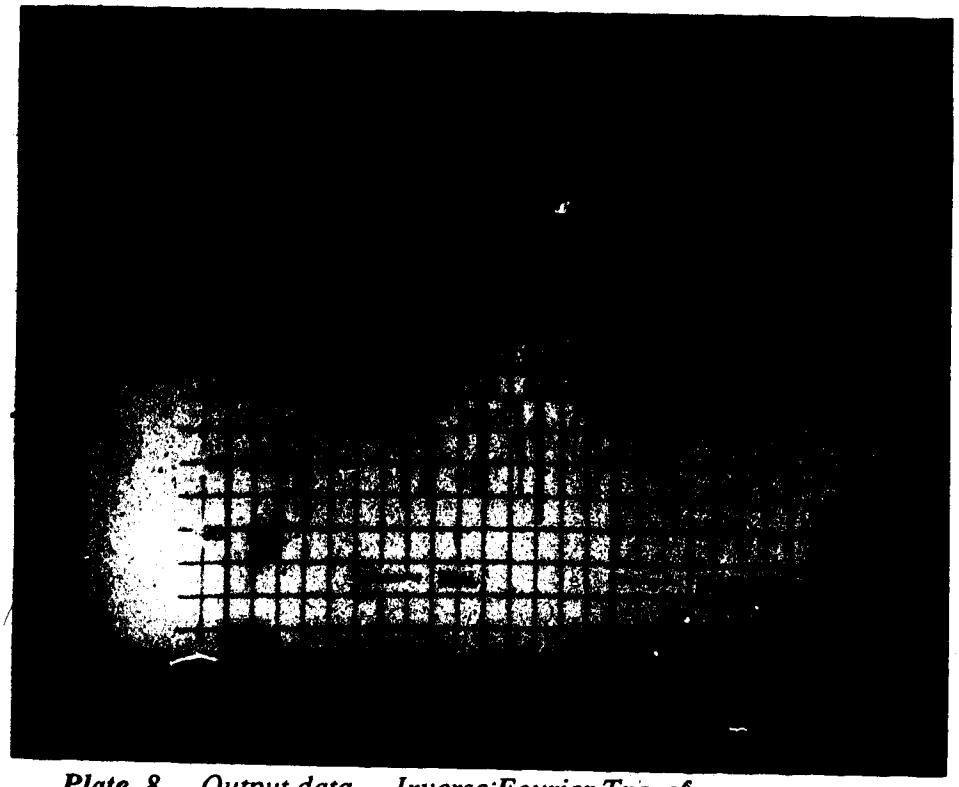


Plate 8 . Output data - Inverse Fourier Transform

in light magenta. User instructions for scrolling the display appear in green. The real part pf the data is plotted with red vertical bars and the imaginary part of the data is plotted with blue vertical bars. In order to ensure a clear display, the imaginary values are slightly displaced from the true positions reduce the possibility of the real and imaginary parts overlapping.

The input data for the direct Fourier Transform was generated from the expression

$$3276.7\ (\ 2\sin(\phi) + \sin(5\phi) + \sin(15\phi) + 2\sin(40\phi) + \sin(65\phi) + \sin(80\phi)\ ).$$

The output data from the direct Fourier Transform is shown in Plate 6. Since the sine function is an odd function all the output data values are imaginary[16]. Only the first 125 harmonics are shown in Plate 6. Plate 6 clearly shows the expected harmonics.

The graphic display for a 64-point inverse Fourier Transform is shown in Plate 7. and its output data is shown in Plate 8. For data points less than 125, a small menu will appear to the extreme left of the screen.. Pressing the key "Q" or "q" ends the display. For data points greater than 128, the small menu to the extreme left of the screen appear only after the user has ended scrolling by pressing the <ENTER> key.

The input data for the power spectrum display was generated from the same function as that used for the direct Fourier Transform. The input data for the power spectrum is shown in Plate 9 and the corresponding data is shown in Plate 10. To prevent overflow, the output data is scaled down by a factor of N^2, where N is the number of data points. In this example, N = 512. Comparing the output data of the absolute maximum of the direct Fourier Transform in Plate 6 and that for the power spectrum in Plate 10, it was observed that

maximum absolute maximum for the direct Fourier Transform $= 1.667 \times 10^6$

maximum absolute maximum for the power spectrum $= 10.732 \times 10^6$

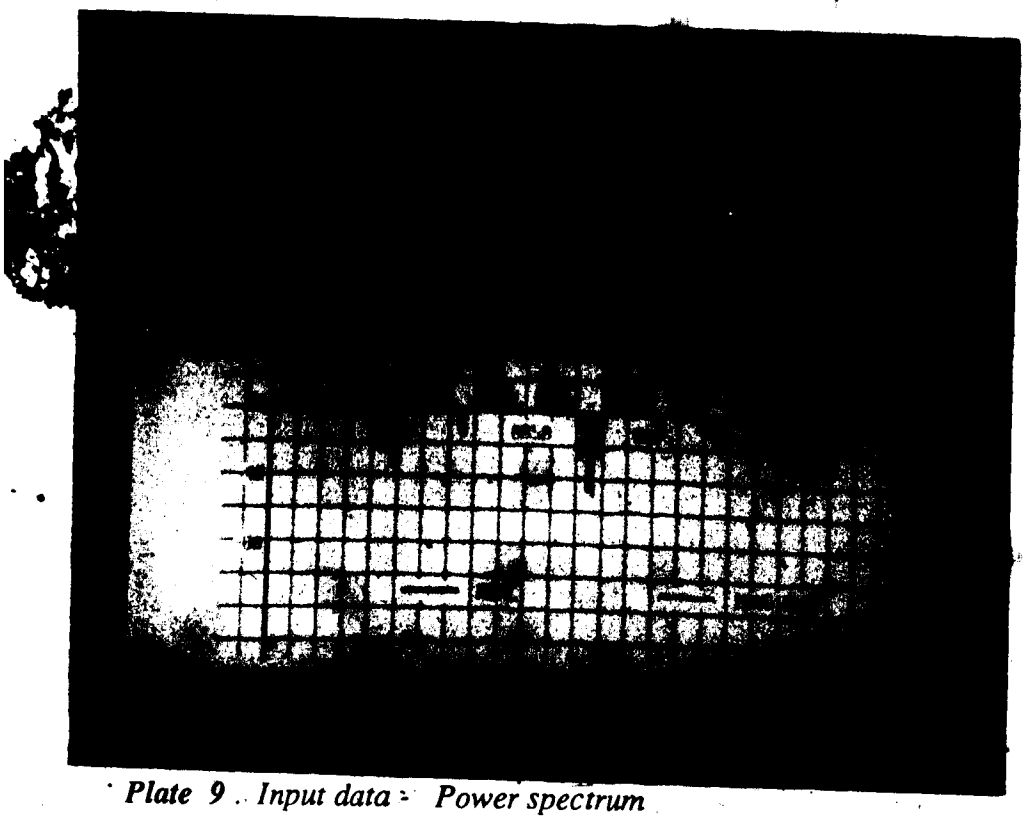


***Plate 9** .. Input data :- Power spectrum*

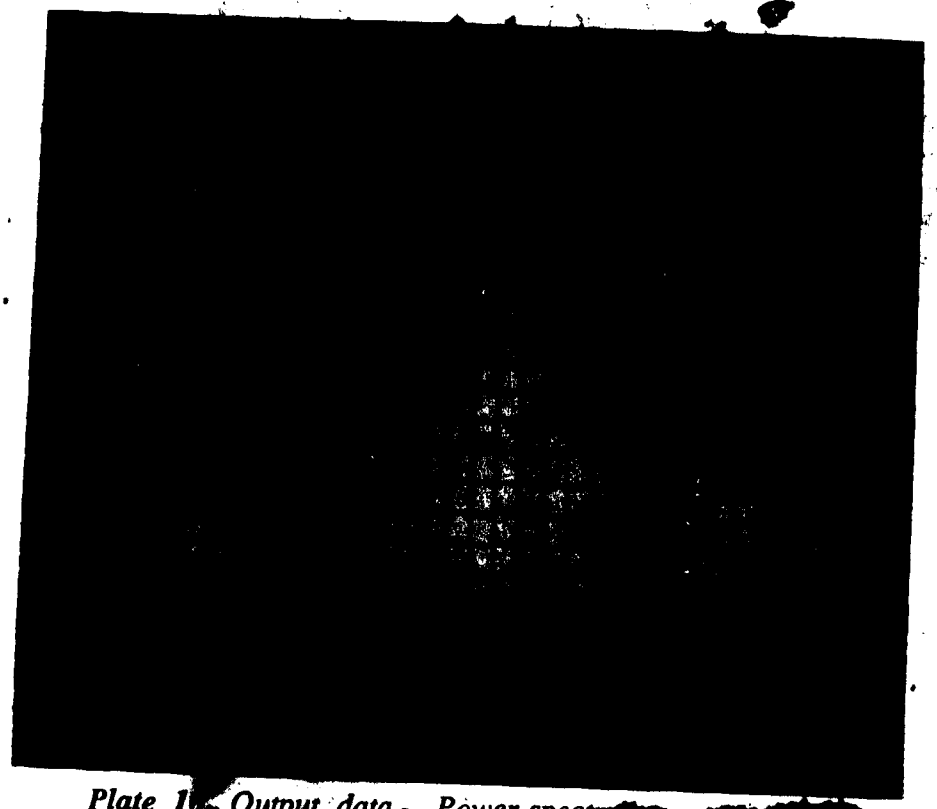


Plate 1. Output data - Power spectr

Plate 11. Input data - Autocorrelation

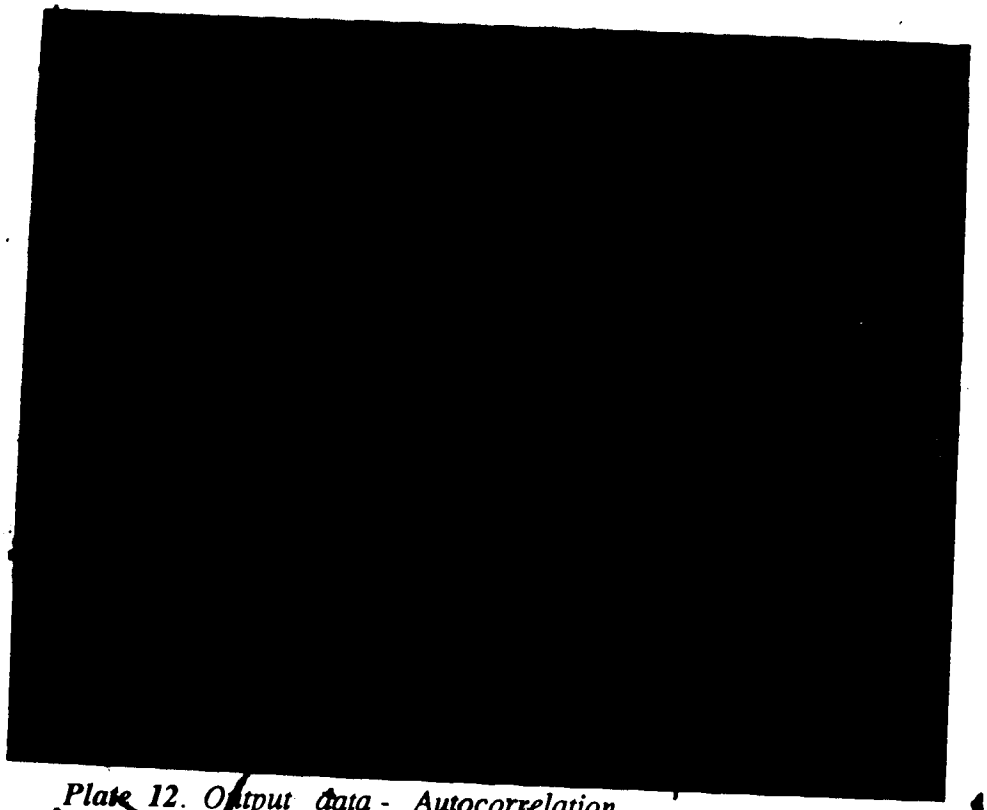


Plate 12. Output data - Autocorrelation

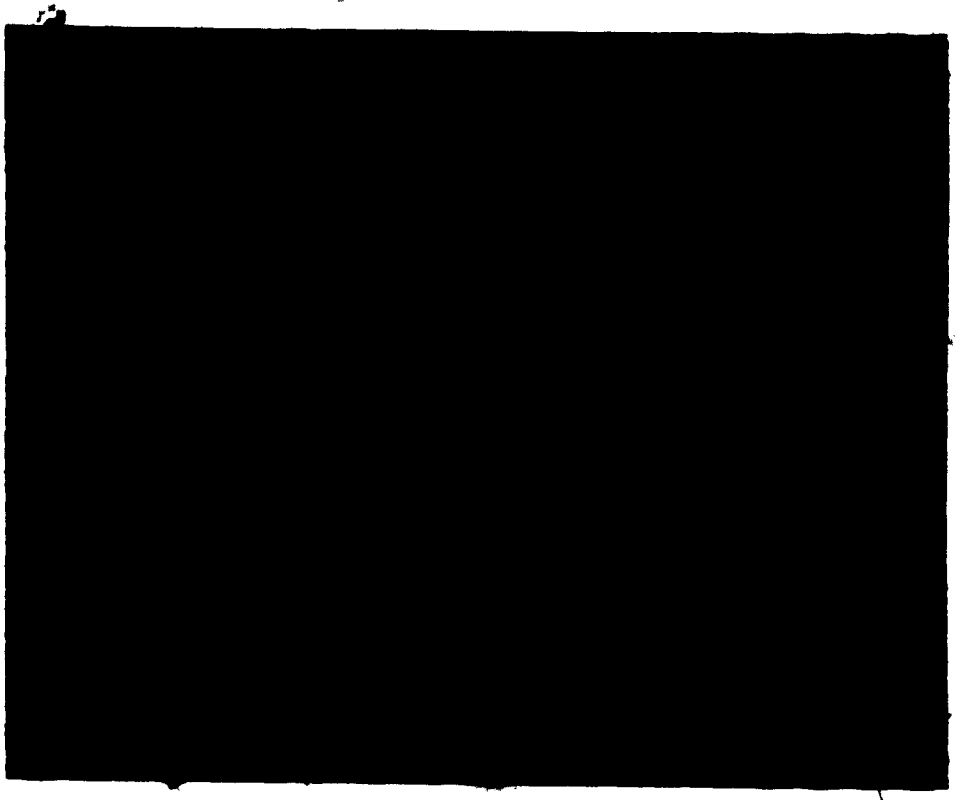


Plate 13 . Input data - Crosscorrelation

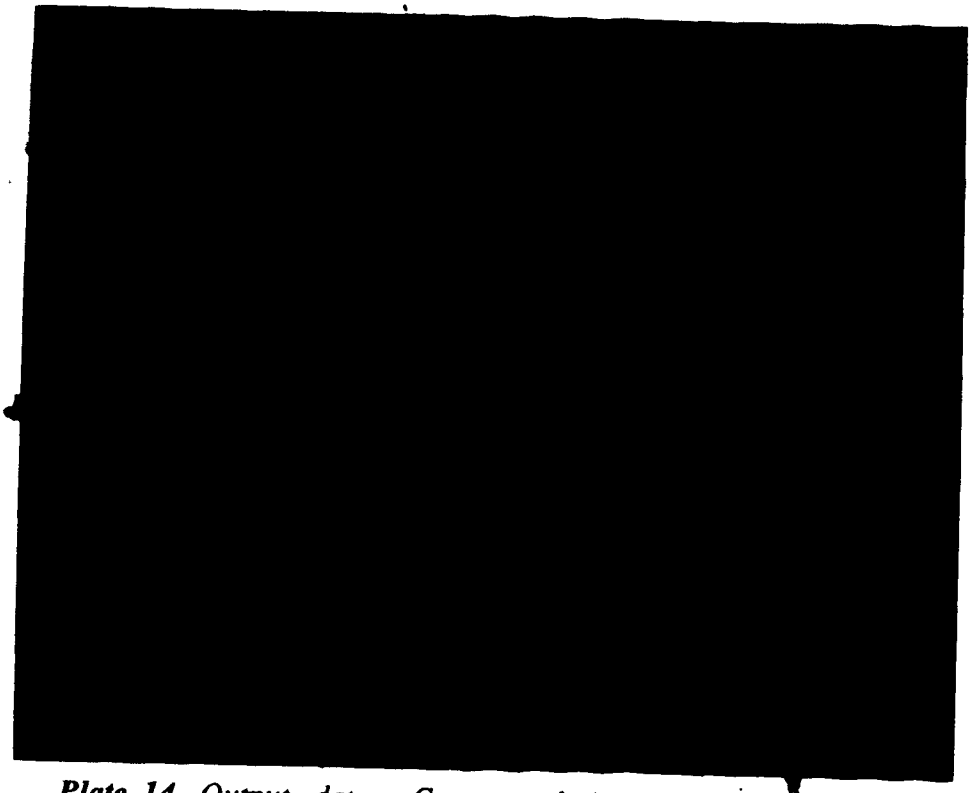


Plate 14. *Output data - Crosscorrelation*

Plate 15. Input data - Linear convolution

Plate 16. *Output data - Linear convolution*

and $(1.667 \times 10^6)^2 / 512^2 = 10.732 \times 10^6$

The above check can only be made if the output data has only real or only imaginary values. If the output data is made up of non-zero real and imaginary parts, then the magnitude of the signal at one frequency may be used for the comparison. The second check for the power spectrum output in this example is that, the relative magnitudes if the output signal is the same as the relative magnitudes of the output data signals for the direct Fourier Transform in Plate 6.

The graphic screens for the input and output data for the correlation and convolution functions are shown in Plates 11 - 16.

5.3 Coprocessor Board Tests

A logic analyzer, an oscilloscope and a digital multimeter were the main tools used to debug the coprocessor hardware. The configuration of the coprocessor dictated that with each stage of testing, the board be plugged into one of the expansion slots of the IBM PC XT, the test carried out, and the board be unplugged for fault correction. The test procedure was thus very slow.

The first tests were memory tests on the two banks of memory. Several test patterns were used to ensure that all memory locations were good. Upon reset, bank A appeared in the memory address space of the IBM PC XT. After successfully testing bank A, the banks were switched by writing to I/O location 343 hex. Bank B was then tested.

The second stage of testing was to verify that the microprocessor on the coprocessor board picked its first instruction from the location FFFF0 hex in its memory address space when its reset signal was removed. An infinite loop was placed at this location. With only 16 locations to the end the memory address space, no exotic set of instructions could be executed from these locations. However, a long jump to a location further away enabled the coprocessor to execute more instructions.

Some of the tasks performed by the coprocessor at this stage were:

(a) performing a memory test on its local memory

(b) switching the banks

(c) reading and writing to the I/O port

The local memory test was carried out in the following manner. The coprocessor first stored a test pattern in one local memory location. The test pattern is then read back by the coprocessor and stored in shared memory. This was done for all the local memory locations. After that, the coprocessor switched banks to enable the IBM PC XT to examine the shared memory locations that mapped onto the local memory.

Having successfully tested the local memory, the coprocessor was made to execute code from its local memory. The IBM PC XT first loaded the coprocessor code into the shared memory and pulled the coprocessor out of its reset state. The coprocessor then transferred the rest of the code to its local memory and finally transferred control to the code in the local memory.

The actual coprocessor routines (except the control program) were tested in the local memory of the IBM PC XT with input data whose outputs were predictable. Typical input data used were sampled sine, cosine and square waves. The control programs together with the other routines were then loaded into the local memory for the final test.

5.4 Performance

Normally, separate variables hold the input and the output data. For a number of computational algorithms, this situation is unavoidable. The in-place Fast Fourier Transform algorithm uses the same variable to hold both input and output data. The memory required to hold data is cut in half. Since each data point requires 8 bytes to hold both the real and the imaginary parts, a set 1024 data elements requires 8 KB of memory.

The sine table requires 2 KB memory. Another 10 KB has been set aside for the filter routines. The infinite impulse response filter routines have both present and past inputs in the filter difference equations, hence the need to use variables other than XREAL an XIMAG to hold the data. The total memory usage comes up to 20 KB. That leaves 12 KB for future use.

5.4.1 Fast Fourier Transform

The execution speed of the this routine was measured under the following conditions:

(a) input data is reordered

(b) 1024 data points

(c) direct Fast Fourier Transform

(d) IBM PC AT running at 6 Mhz

Execution time = 180 milliseconds

The execution time is very good for a general purpose processor with a limited number of general purpose registers and running at a relatively low speed. A 1024-point floating-point Fast Fourier Transform running on a Compaq Corporation personal computer takes about 5 seconds[15]. The correlation and convolution functions have very fast execution times. Only one direct and one inverse Fast Fourier Transforms are required. One complex array is required for the data. The conventional definition of these functions require three real arrays for the data.

Most integer Fast Fourier Transform routines require a fixed number of data points. The ZFFT routine has no such restriction. This unique feature of the ZFFT routine enable it to handle any number of data points, provided it is an integer power of 2 and also less than or equal to 1024. The 16-bit 2's complement input data format is sufficient for most applications.

5.4.2 Data Acquisition System

The data acquisition system is set up so that the terminal count of timer #1 on the TECMAR board "LABMASTER" causes a hardware interrupt on the IRQ2 line on the I/O channel of the IBM PC XT. The time interval between initiating a hardware interrupt and the IBM PC XT executing the first instruction of the interrupt service routine is 61 clock cycles. With a 210 nanosecond clock period, that amounts to 12.8 microseconds.

A typical conversion time for the analog-to-digital converter set up for -10V to +10V operation is 35 microseconds. If it is assumed that the interrupt service routine takes 2.2 microseconds, then a minimum of 50 microseconds is required in between samples. That gives a maximum sampling rate of 20 kHz. To eliminate analog-to-digital data overrun and also to enable more processing to be done by the interrupt service routine, a maximum sampling rate is 10 kHz is assumed.

5.4.3 Inter-record Gap

The absence of an I/O processor on the IBM PC XT limits its use in real time processing. With hard disk access times greater than 30 milliseconds and console I/O being even slower, fast changing signals cannot be tracked. If some gap between one record and the next can be tolerated, then real time processing can easily be done. The presence of the coprocessor board is to introduce concurrency into the acquisition and processing of the signals and therefore cut down on gap between one record and the next. The continuous processing routine discussed in the previous chapter accomplishes the concurrency between the acquisition and the processing of data.

5.5 User Information

The entire software package is made up of three source files and two assembler language source files. The files aré HSIG.H, SIG1.C, SIG2.C, ZFT1.ASM and ZFT2.ASM. HSIG.H is actually a C header file. The C source files SIG1.C and SIG2.C are compiled separately. Similarly, the two assembler language files are assembled separately. The four object files are then linked into a single executable file and called SIG.EXE.

Before using the package, it is assumed that the coprocessor is in one of the expansion slots of the IBM PC XT and the TECMAR board "LABMASTER' is in another slot. It is further assumed that hardware interrupts are triggered by the terminal count of timer #1 on the TECMAR board and that hardware interrupts occur on the IRQ2 line.

6. CONCLUSION

A low frequency signal processor for use in biomedical applications has been designed. The board fits into one of the expansion slots of the IBM PC XT. The signal processor operates concurrently with the IBM PC XT to increase the rate at which data records are be processed.

The IBM PC XT and the signal processor share a common memory. This common is made up two banks of read/write memory. While the IBM PC XT is addressing one bank, the signal processor is addressing the other. The banks may be switched around by the CPU of the signal processor. The signal processor switches the banks after it has completed its current task and has received a non-maskable interrupt from the IBM PC XT. Information is exchanged between the two systems through the shared memory.

Software was written for both the IBM PC XT and the signal processor. The IBM PC XT software is made up of number of subprograms which enable the IBM PC XT to acquire, store and display input and output data. Data may be acquired either from a data acquisition card or from a disk file. The acquired data is stored in shared memory and a copy may be kept in a disk file. Input and output data may displayed graphically on a high resolution graphics monitor. The IBM enhanced graphics display system is used.

The user interface is also part of the IBM PC XT software. The user interacts with the rest of the software through a number of menus. The "UP" and "DOWN" cursor keys are used to highlight a menu option and the "<ENTER>" key is used to select the option. The screen is updated after selecting an option. The updated screen may be a new menu or a screen with prompts for the user. The prompts enable the user to supply some information to the program.

The coprocessor software performs the numerical computations. The IBM PC XT puts data into the shared memory and sends a non-maskable interrupt to the central

processing unit of the signal processor. Upon receiving the interrupt, the signal processor switches banks and fetches the function code from the shared memory. It then performs the function specified by the function code. The following functions are available:

(a) direct discrete Fourier Transform

(b) inverse discrete Fourier Transform

(c) power spectrum

(d) auto-correlation

(e) cross-correlation

(f) linear convolution

(g) cyclic convolution

All the mathematical functions listed above depend on the Fast Fourier Transform algorithm. This approach results in good computation times[15]. A 1024-point Fast Fourier Transform computation is done in about 180 milliseconds on an IBM PC AT running at 6 MHz. The "in-place" Fast Fourier Transform algorithm[11] reduces the total memory required for the computations. The memory requirements for the convolution and the correlation functions are reduced by a third. In addition, the convolution and the correlation functions require only two calls to the Fast Fourier Transform subprogram and one call to a unscrambling subprogram . In the absence of the sorting program, three calls to the Fast Fourier Transform subprogram are required. The unscrambling subprogram, however, require only N complex multiplications for N data points, while the Fast Fourier require $N * \log_2 N$ multiplications for the same number of data points.

The outstanding features of the thesis are:

(a) software is very easy to use

(b) software is optimized for both speed and size

(c) sampling frequency and the number of samples may be selected by the user.

(d) the signal processor is capable of real time processing

(e) the input and out data are displayed in color on a high resolution graphic screen.

List of References

1. AHMED, N., NATARAJAN, T. **Discrete-time Signals and Systems**, Reston Publishing Company, 1983.

2. CADZOW, J. A. **Discrete-time Systems - An Introduction with Interdisciplinary Applications**, Prentice-Hall, 1983.

3. CAMDEL, E. R. ""Monolithic Digitral Signal Processing Microcomputer [illegible]tecture", **IEEE International Workshop on Computer Systems** [illegible]**nization**, 1983.

4. CLYNES, M., MI[illegible] **Biomedical Engineering Systems, Inter-University Electronics Series**, Vol. 10, McGRAW-HILL BOOK COMPANY, 1970.

5. CUSHMAN, R. H. "Sophisticated Development Tool Simplifies DSP Chip Programming", **EDN**, September, 1983.

6. ELDER IV, J. F., MAGAR, S. "Single Chip Approach to Digital Signal Processing", presented at **Wescon**, 1983.

7. IBM PC XT TECHNICAL REFERENCE MANUAL.

8. JERMANN, W. H. **Programming 16-bit Machines-The PDP 11, 8086 and M68000**, Prentice-Hall, 1986.

9. JOURDAIN, R. **Programmers Probelm Solver for the IBM PC, XT and AT**, Prentice-Hall, 1986.

10. MACRO ASSEMBLER FOR THE MS-DOS OPERATING SYSTEM-Programmer's Guide, Microsoft Corporation,1987.

11. McGILLEN, C. D., COOPER, G. R. **Continuous and Discrete Signal System Analysis**, Holt, Rinehart and Winston, 1984.

12. OSBORNE, A. KANE, G. **Osborne 16-Bit Microprocessor Handbook**, Osborne/McGraw-Hill, 1981.

13. PURDUM, J., LESLIE, T. C. **C Standard Library**, Que Corporation, 1987.

14. RAMIREZ, R. W. **The FFT Fundamentals and Concepts**, Prentice-Hall, 1985.

15. SEWARDS, A. **Introduction to d.s.p., Electronics and Wireless World**, Vol. 94, n.1630, August, 1988.

16. STANLEY, W. D., DOUGHERTY G. R., DOUGHERTY, R. **Digital Signal Processing**, Reston Publishing Company, 1984.

17. YU-CHENG, L., GIBSON, G. A. **Microcomputer Systems-The 8086/88 Family**, Prentice-Hall, 1986.

APPENDIX I Coprocessor Circuit Diagrams

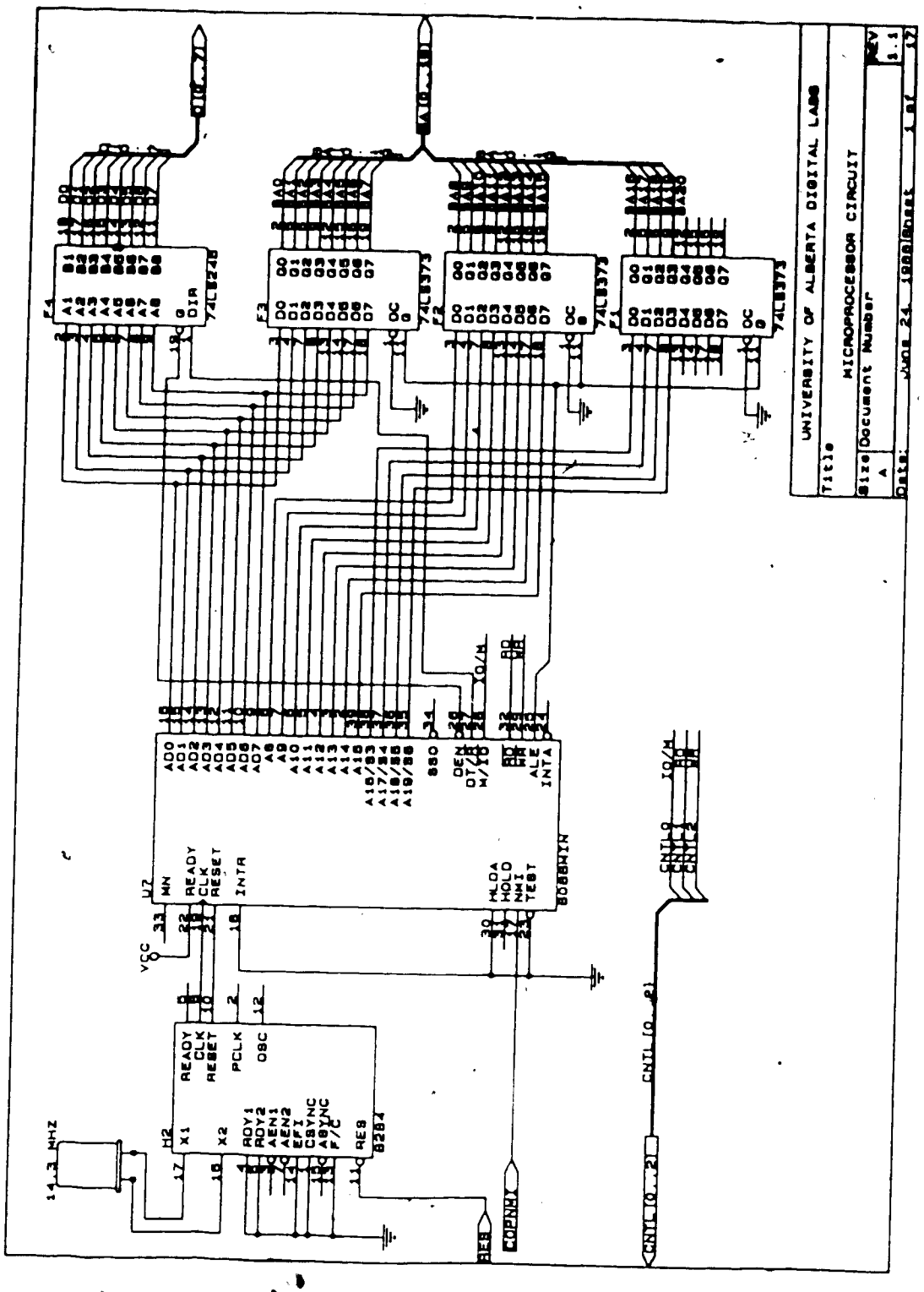


Fig. A.1 Microprocessor Circuit

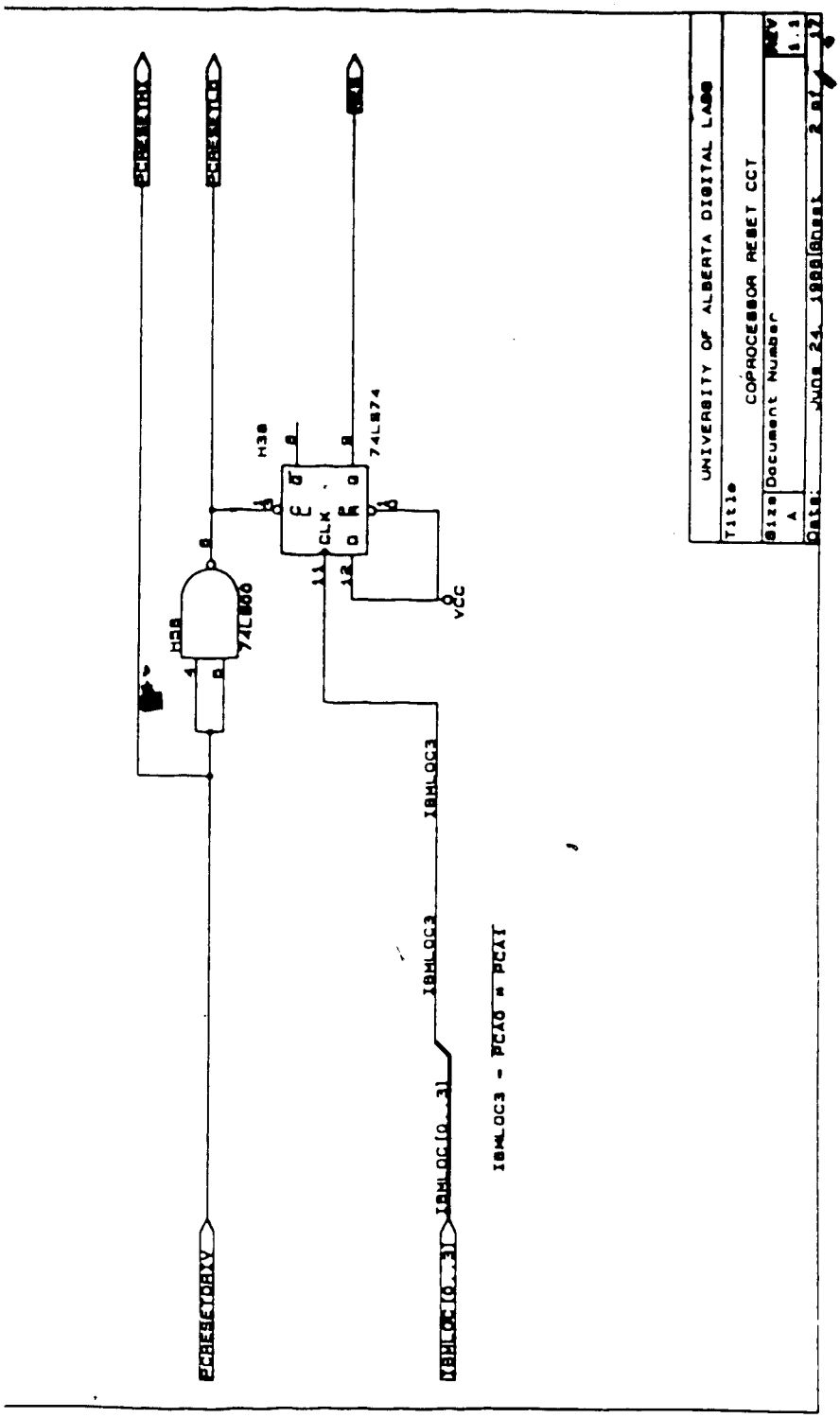


Fig. A.2 Coprocessor reset circuit

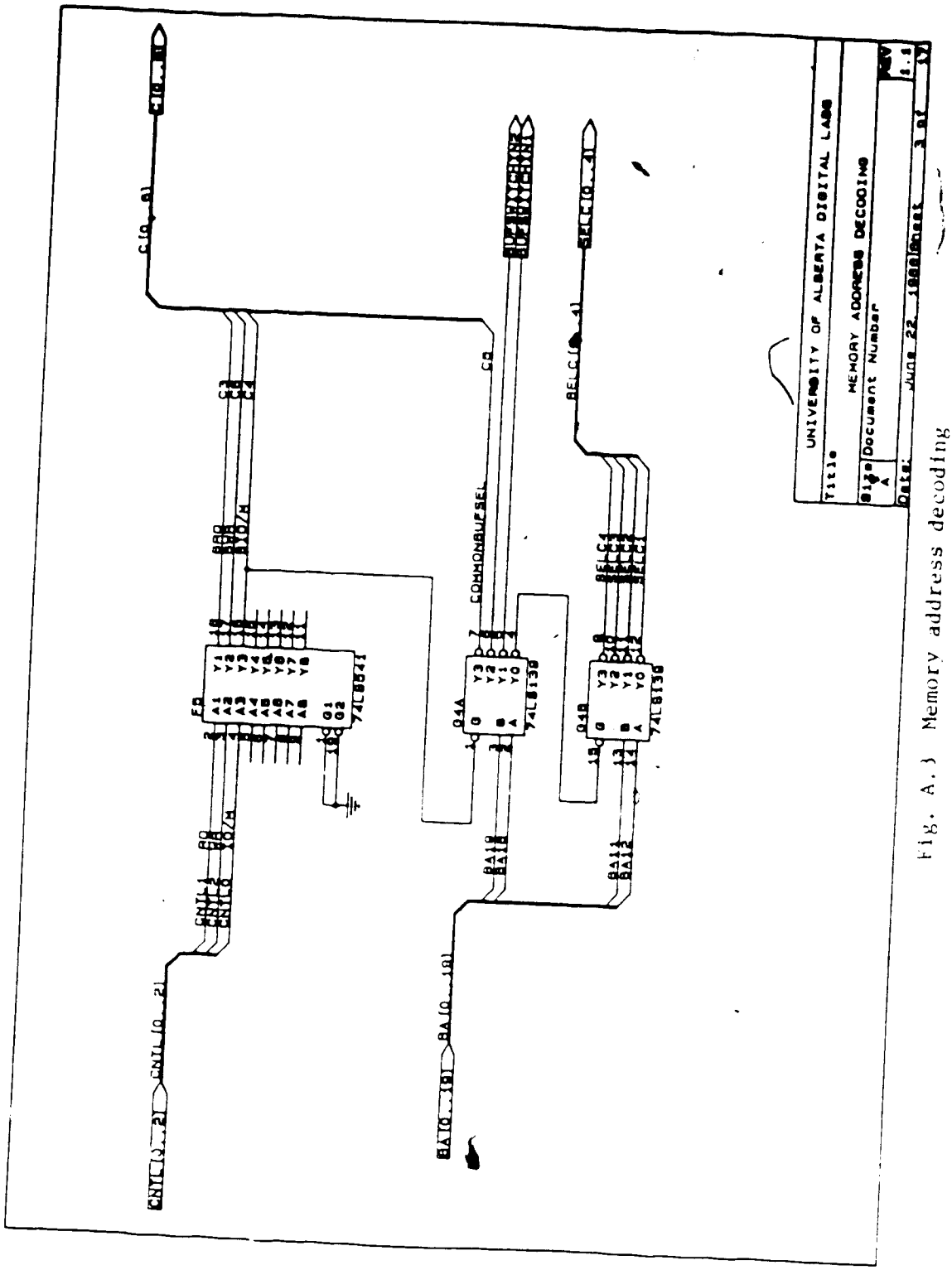


Fig. A.3 Memory address decoding

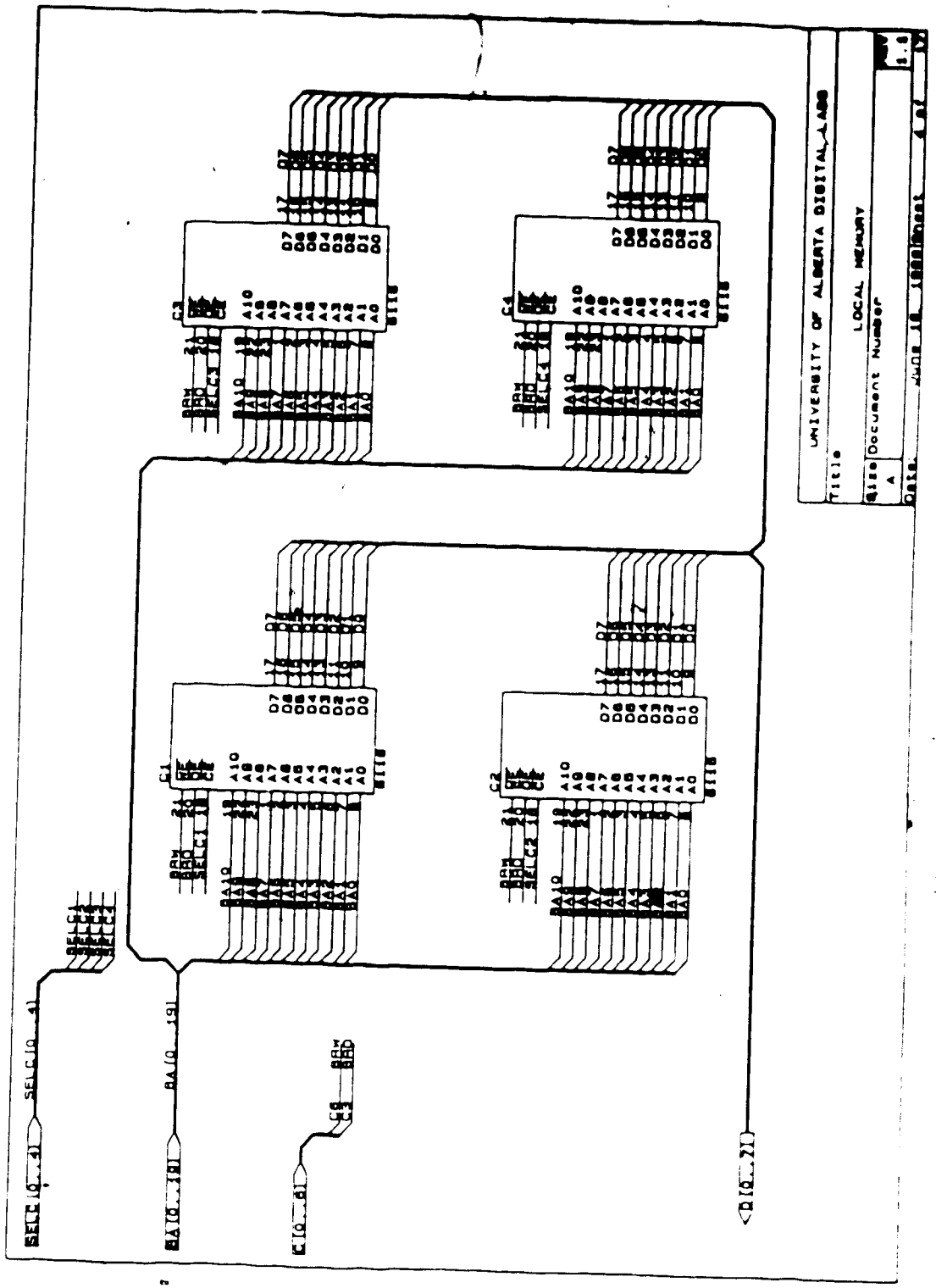


Fig. A.1 Local memory

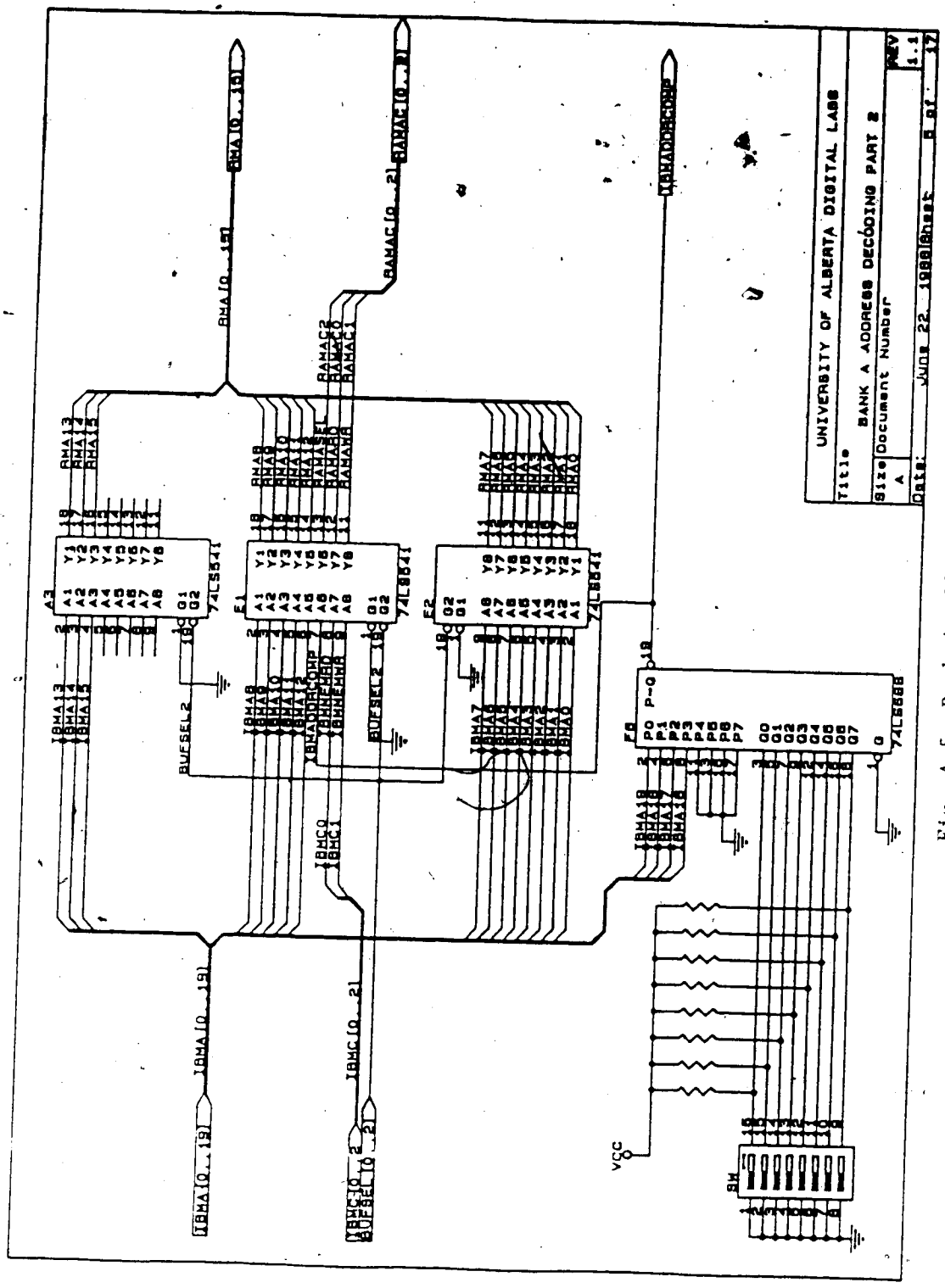


Fig. A.5 Bank A address decoding part 2

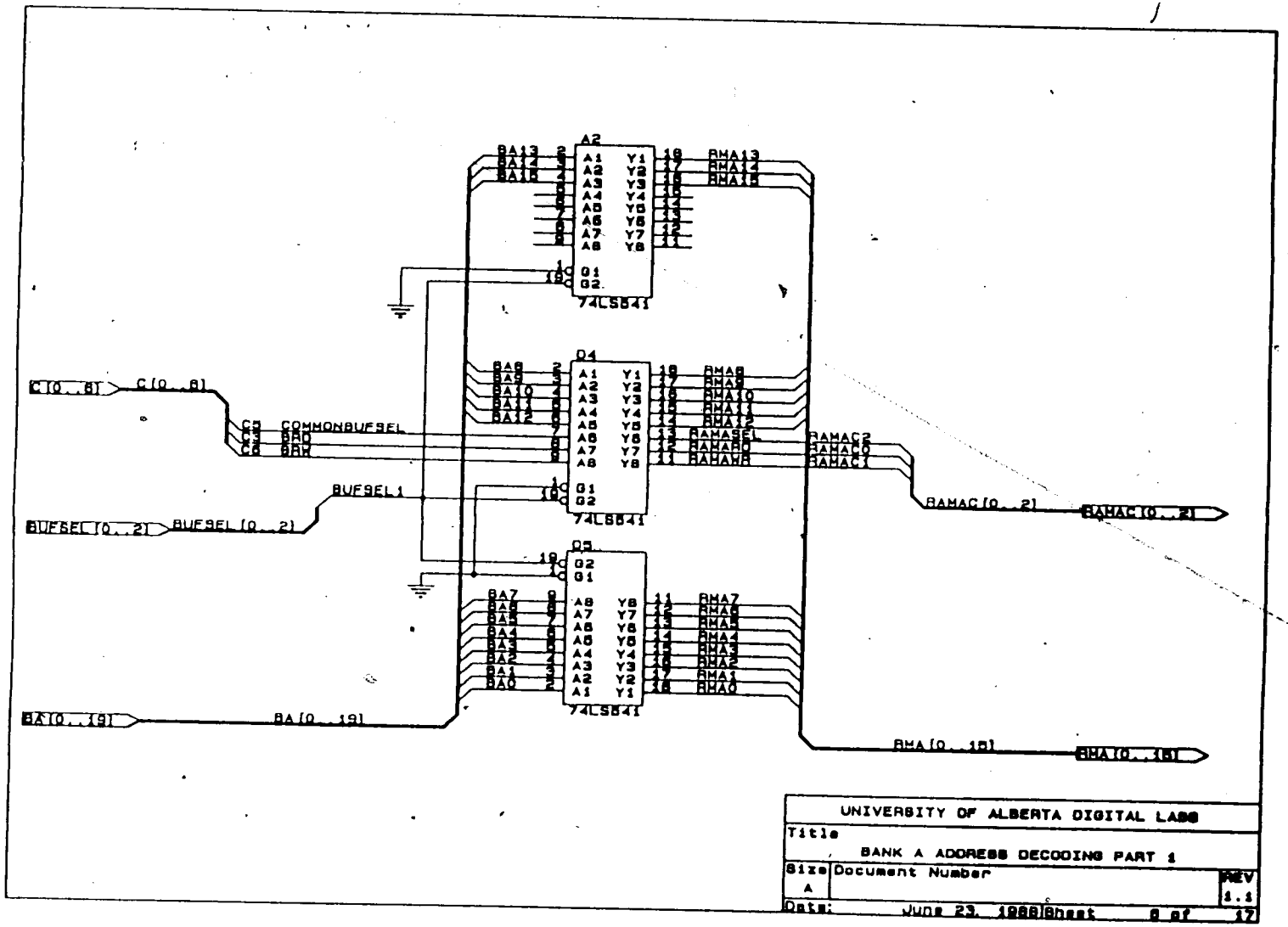


Fig. A.6 Bank A address decoding part 1

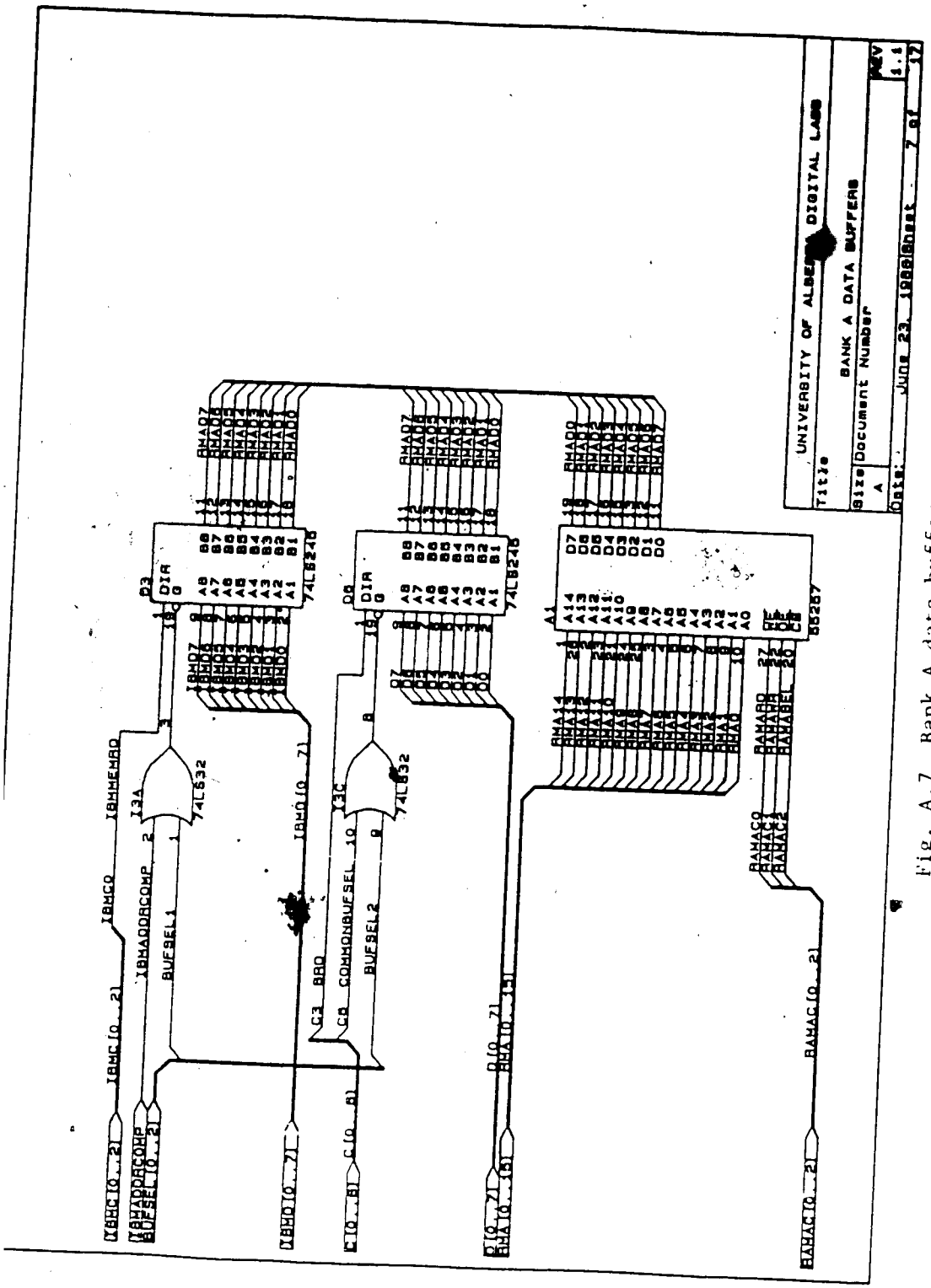


Fig. A.7 Bank A data buffers

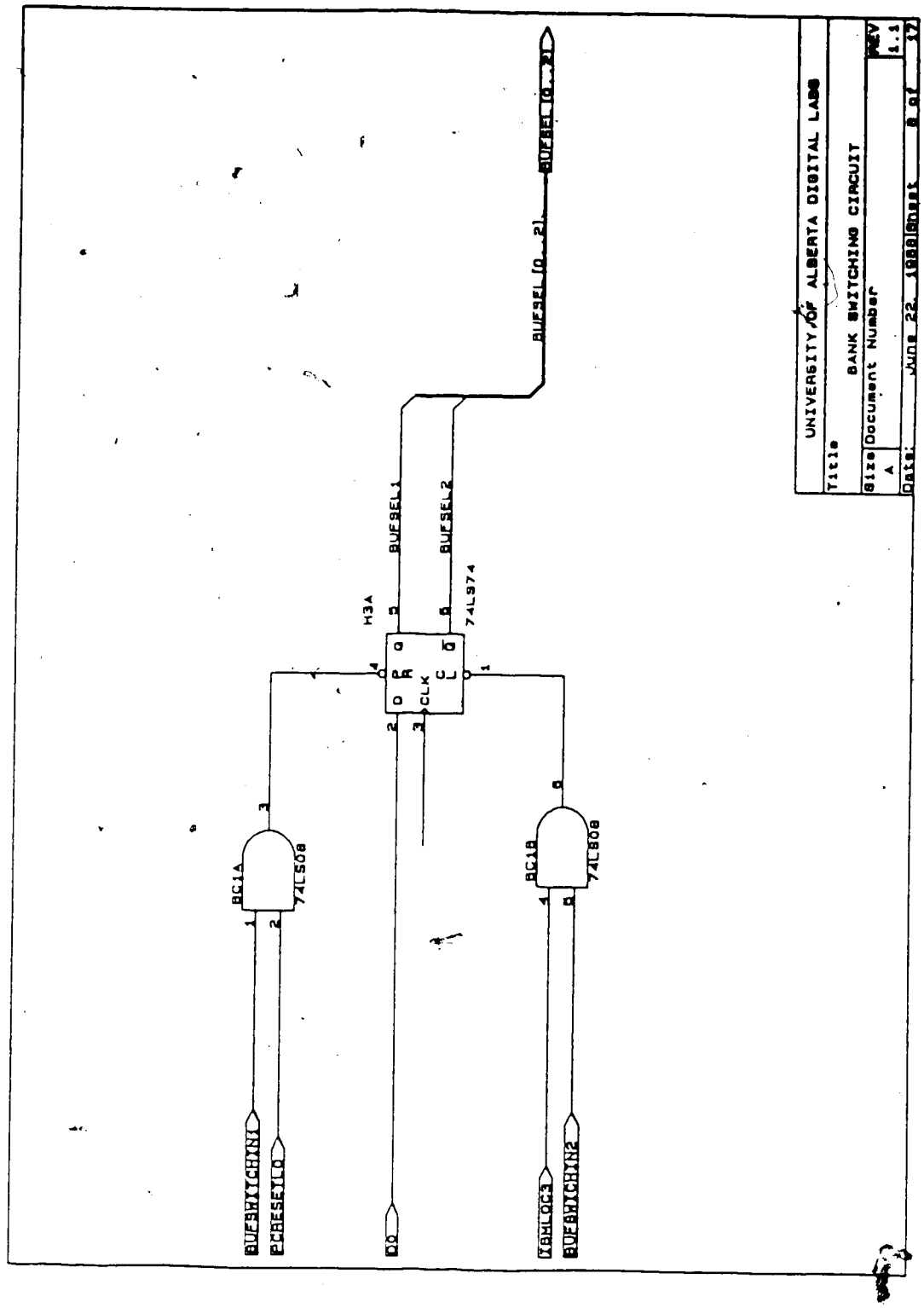


Fig. A.8 Bank switching circuit

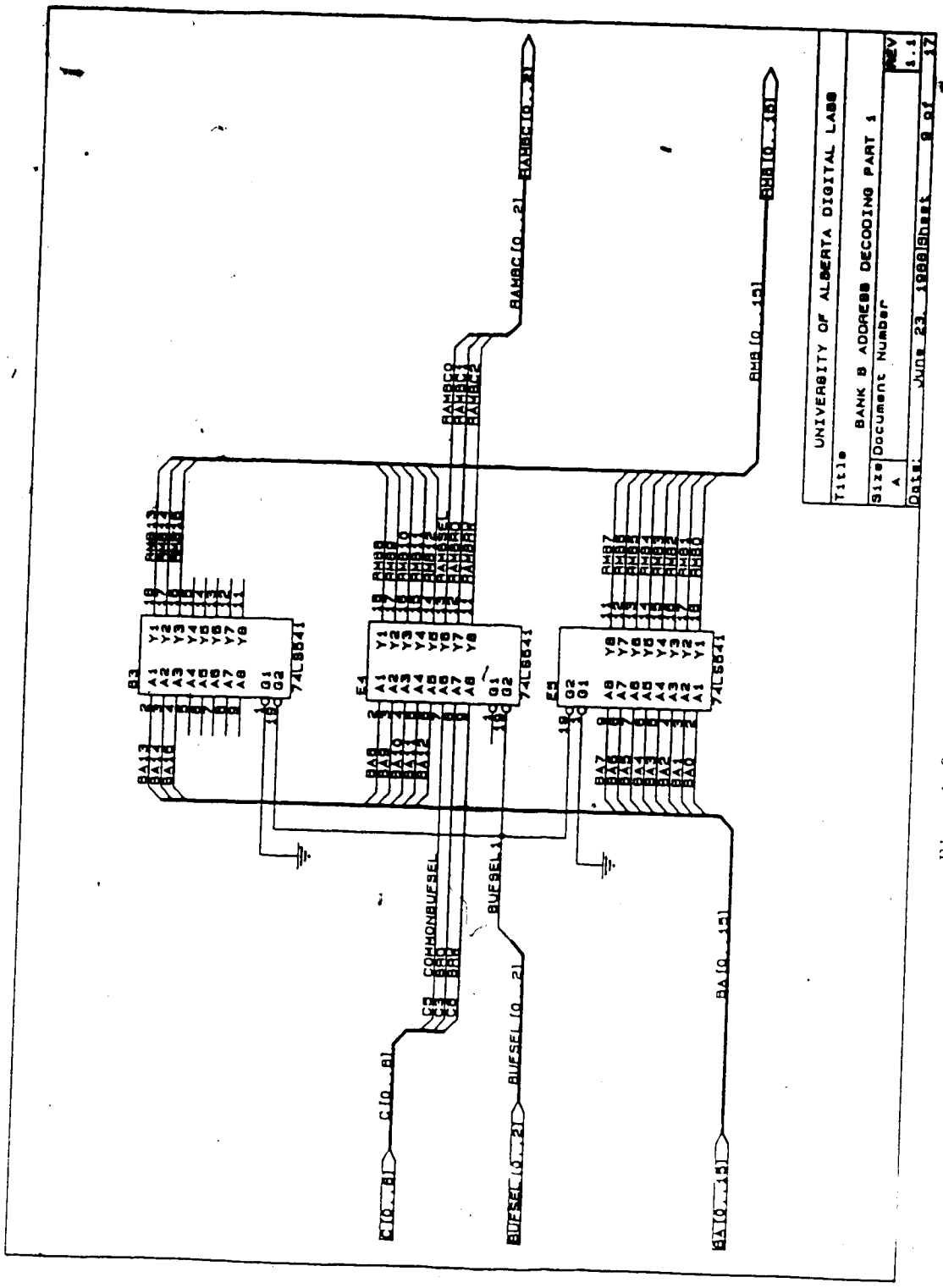


Fig. A.9 Bank B address decoding part 1

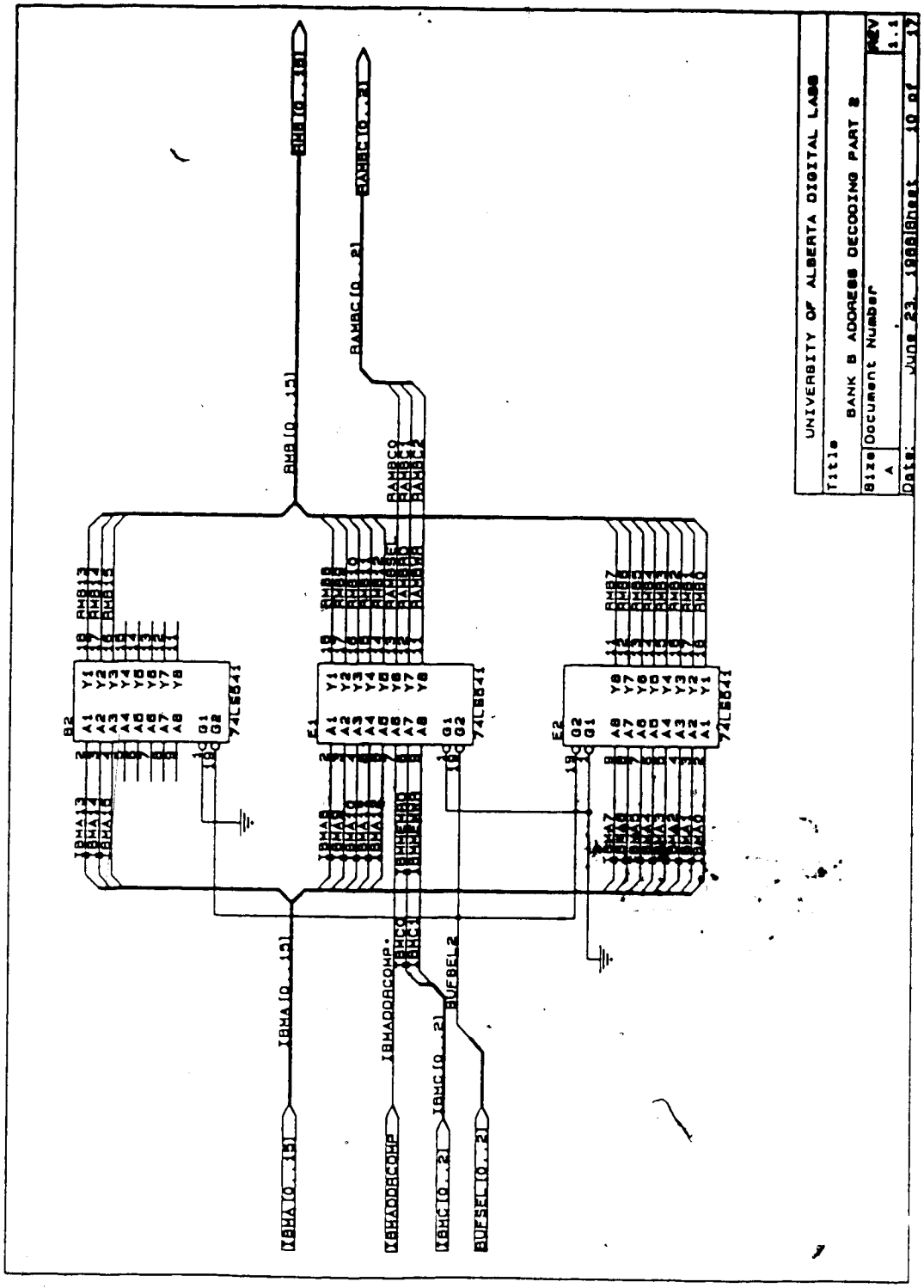


Fig. A.10 Bank B address decoding part 2

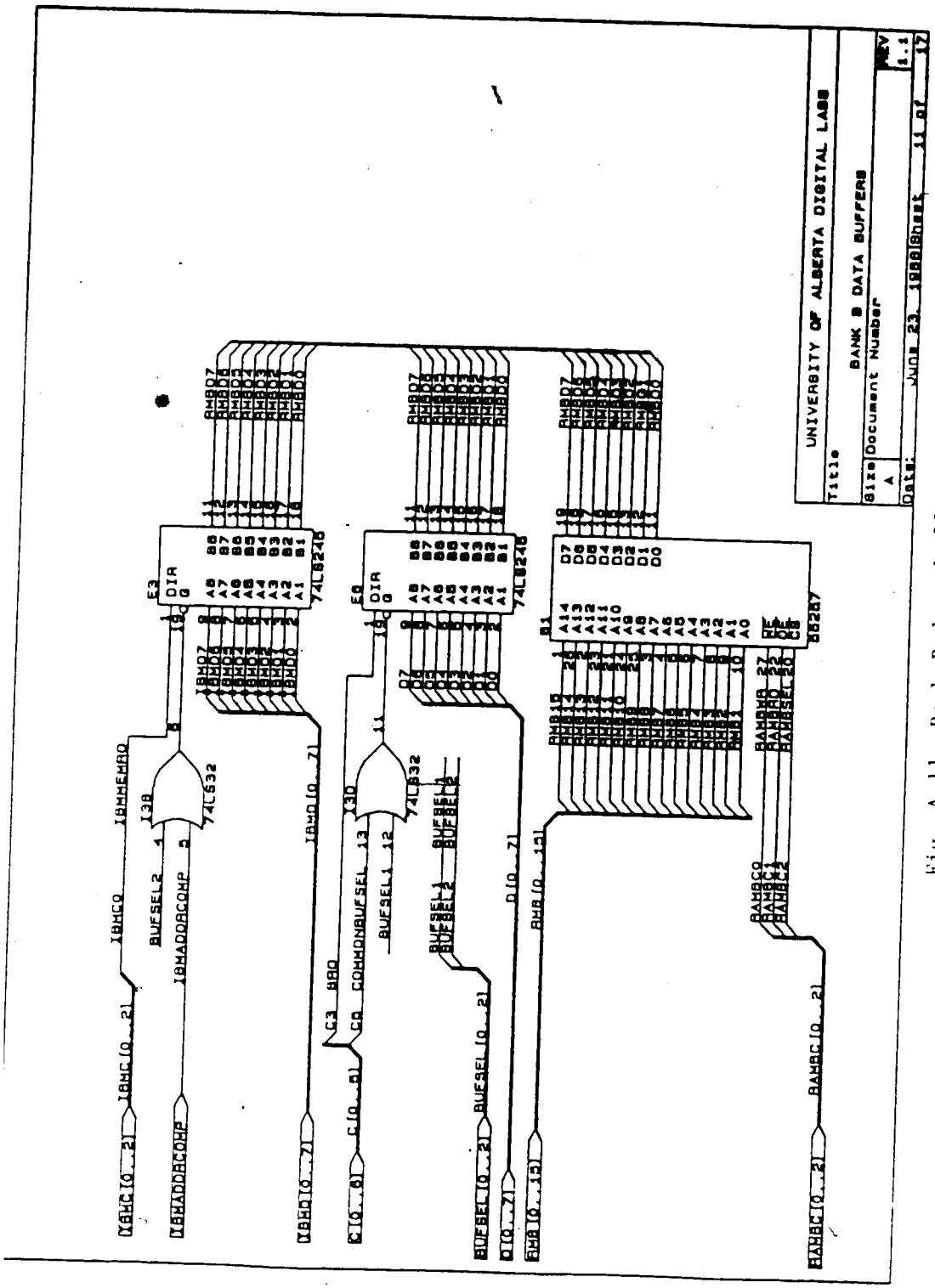


Fig. A.11 Bank B data buffers

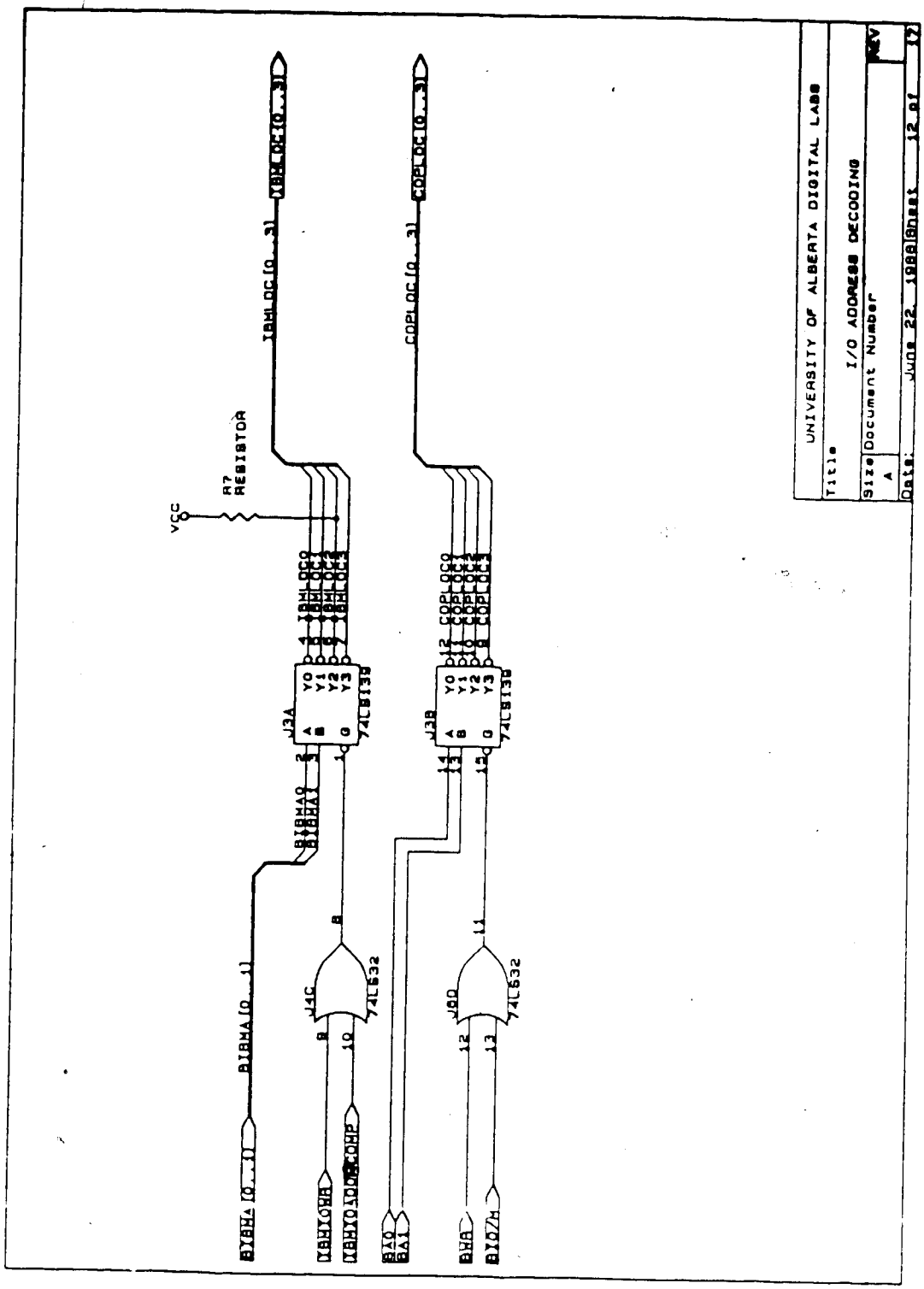


Fig. A.12 I/O address decoding

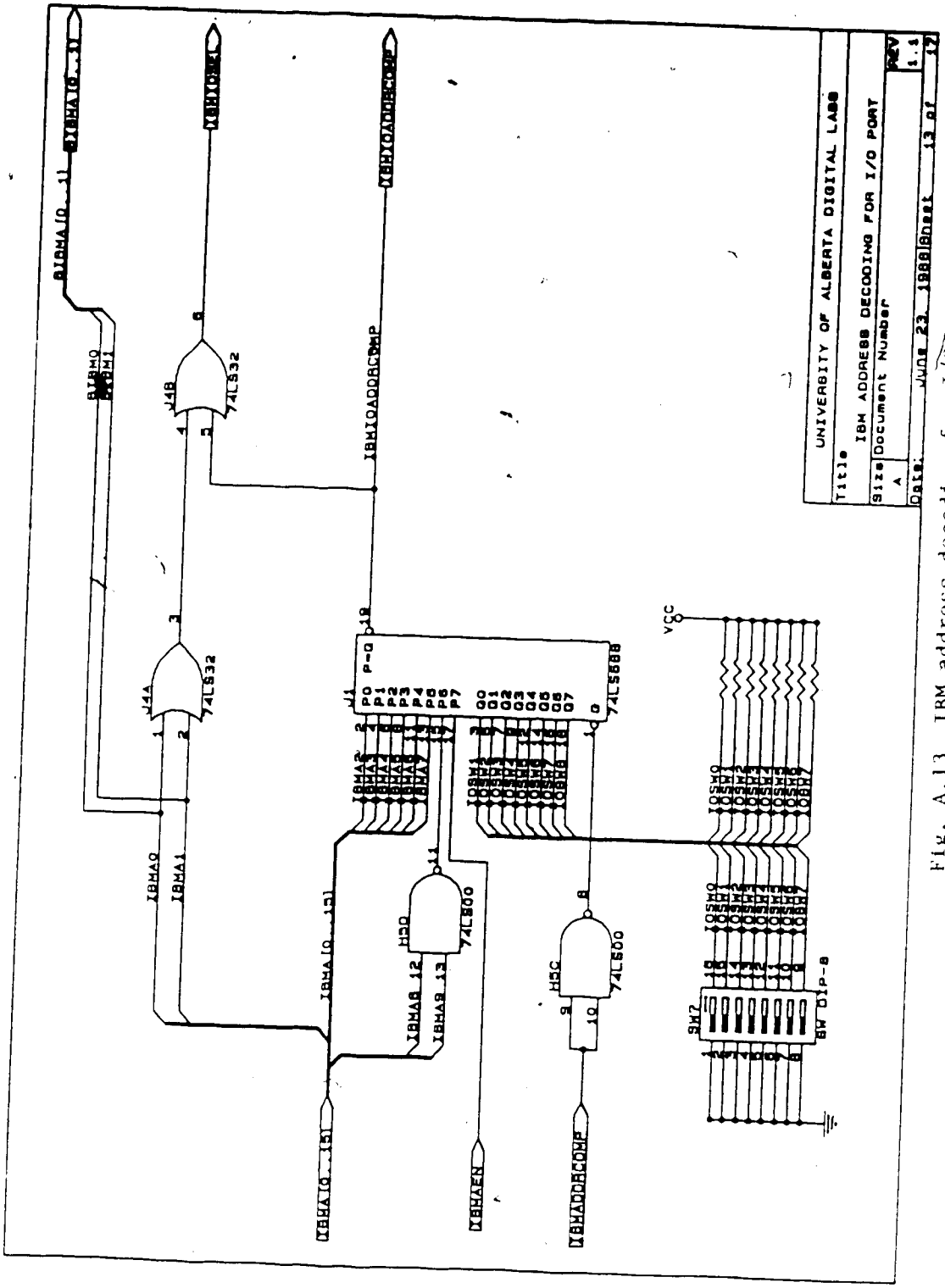


Fig. A.13 IBM address decoding for I/O port

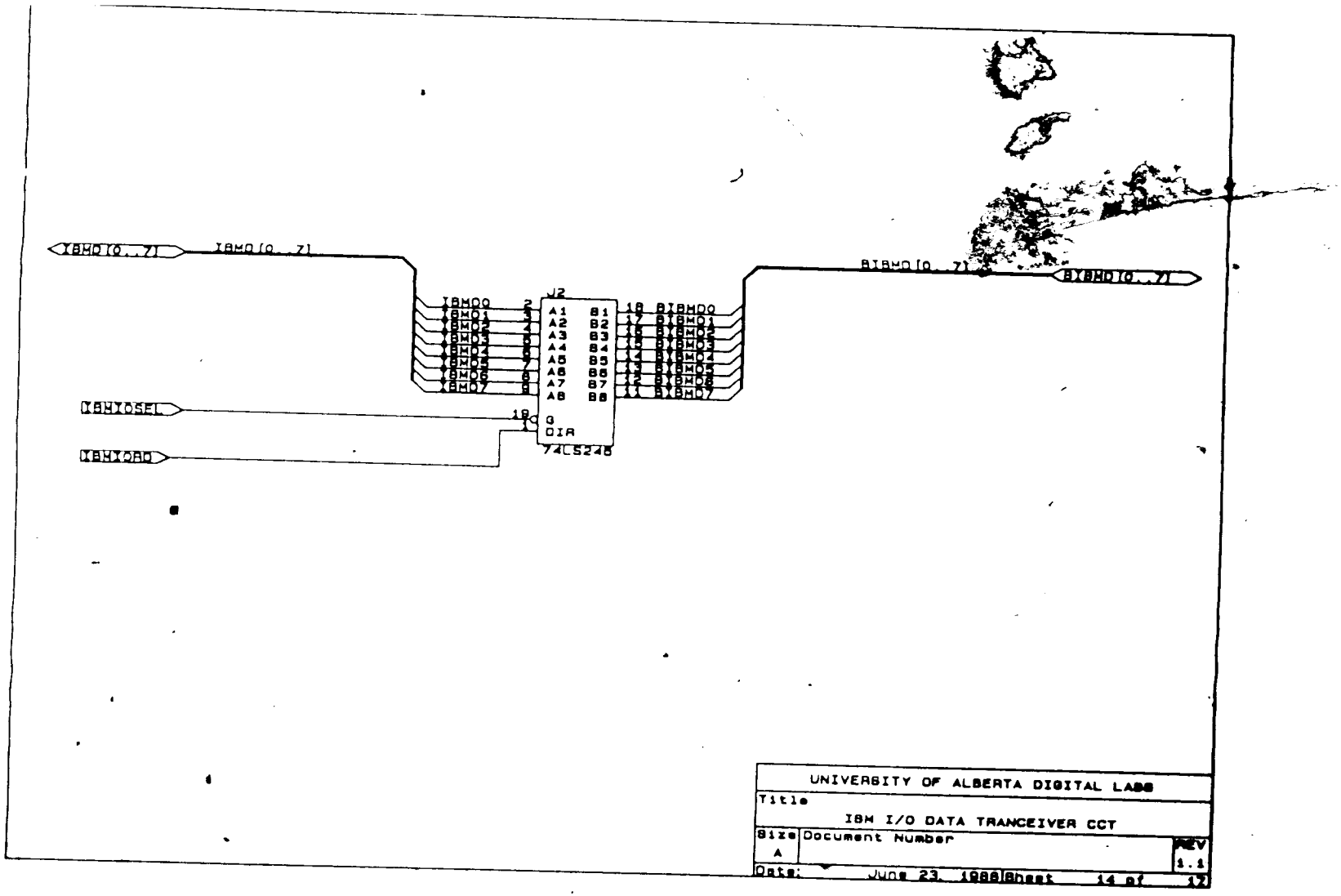


Fig. A.14 IBM I/O Data Transceiver Circuit

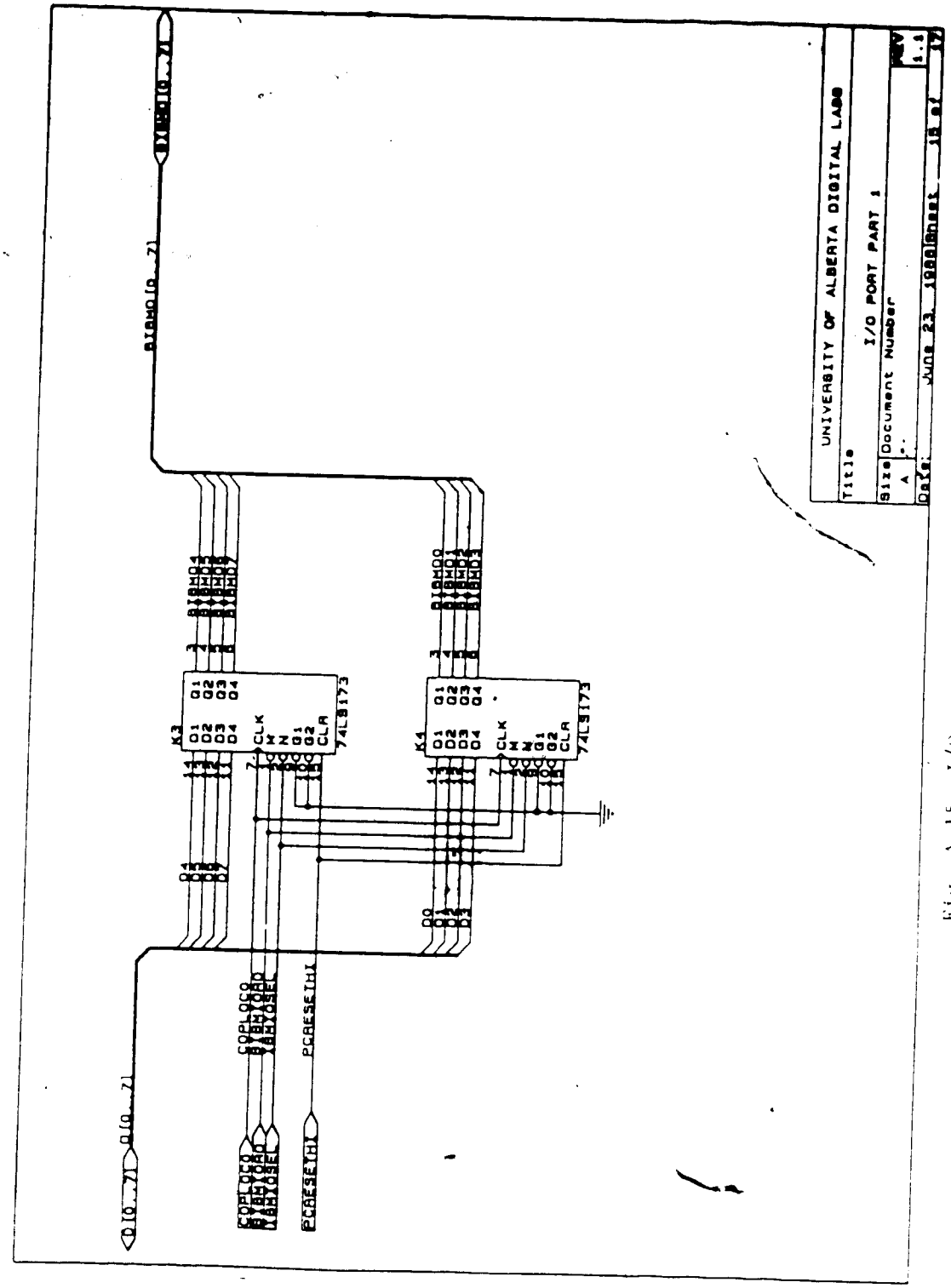


Fig. A.15 I/O port part 1

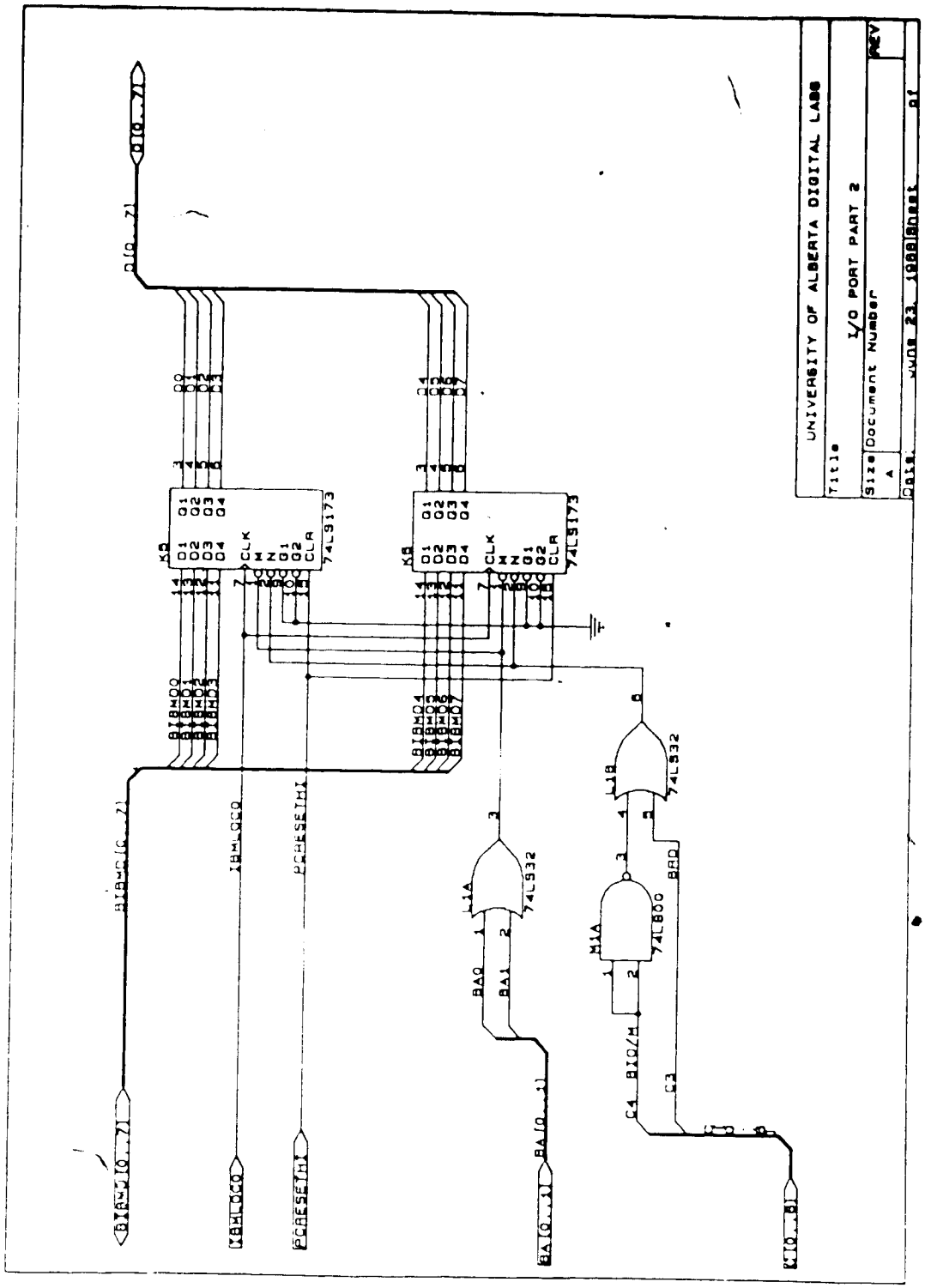


Fig. A.16 I/O port part 2

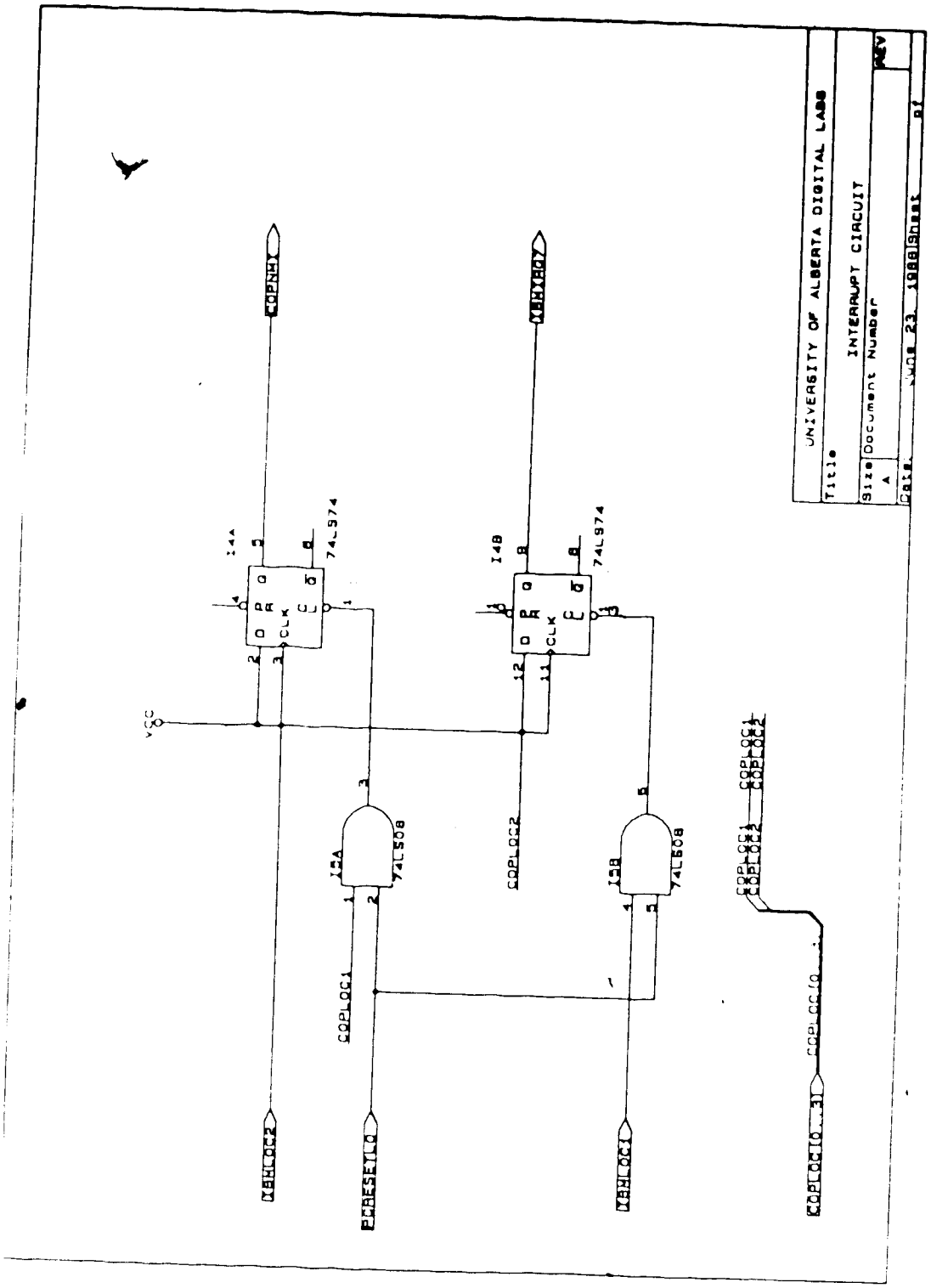


Fig. A.17 Interrupt Circuit

APPENDIX II SOFTWARE LISTINGS

A. IBM SOFTWARE PART I

```
/*************************************
 *    HSIG.H                         *
 *************************************/

#include  <stdio.h>
#include  <stddef.h>
#include  <stdlib.h>
#include  <string.h>
#include  <dos.h>
#include  <conio.h>
#include  <bios.h>
#include  <graph.h>
#include  <math.h>

#define  A_BANK              0x0000
#define  B_BANK              0xFFFF
#define  INTPRESENT          0x5A5A
#define  STATUSOK            0xA5A5
#define  ZERODIVERR          0x4444
#define  OVFLERR             0x6666
#define  DONE                0xCC

#define  OPDONOTHING         0
#define  OPDFFT              2
#define  OPINVFFT            4
#define  OPPGRAM             6
#define  OPAUTOCORRL         8
#define  OPCROSSCORRL        10
#define  OPLCONVL            12
#define  OPPCONVL            14
#define  OPFIRLP             16
#define  OPFIRBP             18
#define  OPFIRHP             20
#define  OPIIRLP             22
#define  OPIIRBP             24
#define  OPIIRHP             26

void COP(void)    ;
void RES(void)    ;
void NMIRQST(void) ;

extern int  far  QTABLE[]  ;        /*  TABLE IN S9000     */
extern int  far  VTABLE[]  ;        /*  TABLE IN CD_DATA   */
extern int  far  NQ        ;        /*  N  IN S9000        */
extern int  far  MQ        ;        /*  M  IN S9000        */
extern long far  XQREAL[]  ;        /*  XREAL IN S9000     */
extern long far  XQIMAG[]  ;        /*  XIMAG IN S9000     */
```

```
extern int far QIRQ7FLAG ;
extern int far QIRQ7CNT ;
extern int far QOPCODE ;
extern int far QOPSTATUS ;
extern int far QBANKNO ;          /* QBANKNO = BANKID */

extern int far VBANKNO ;          /* VBANKNO = BANKID */
extern int far VIRQ7FLAG ;
extern int far VIRQ7CNT ;


/*****************************************************
 *TECMAR BOARD INFORMATION*
 *****************************************************/

#define SOC              0
#define DAC0LSB          TECMAR + 0
#define DAC0MSB          TECMAR + 1
#define DAC1LSB          TECMAR + 2
#define DAC1MSB          TECMAR + 3
#define CNTLADDR         TECMAR + 4
#define MUXADDR          TECMAR + 5
#define SOCADDR          TECMAR + 6
#define TIMERINTACK      TECMAR + 7

#define STATUSADDR       TECMAR + 4
#define ADCLSB           TECMAR + 5
#define ADCMSB           TECMAR + 6

#define TIMERDATA        TECMAR + 8
#define TIMERCNTL        TECMAR + 9

#define PIADA            TECMAR + 12
#define PIADB            TECMAR + 13
#define PIADC            TECMAR + 14
#define PIACNTL          TECMAR + 15

struct byte
    {
      unsigned char LO ;
      unsigned char HI ;
    } ;

typedef union tecdata
    {
      struct byte BYTE ;
      unsigned int WORD ;
    } tecdata ;
```

```
#define SYSCALL          0x10
#define SYSINT           0x21

#define XRANGE1          32767
#define XRANGE2          327670000

#define TIMErange1       256
#define TIMErange2       512
#define TIMErange3       1024

#define FREQrange1       256
#define FREQrange2       512
#define FREQrange3       1024

#define DEFAULT          -1
#define TRUE             1
#define FALSE            0
#define TCURSOROFF       0x2020

#define UP               72
#define DOWN             80
#define LEFT             75
#define RIGHT            77
#define ENTER            28

typedef enum OPTIONS1
    {
     QUIT1,
     DIRECT,
     INVERSE,
     PERIODOGRAM,
     AUTOCORRELATION,
     CROSSCORRELATION,
     LCONVOLUTION,
     PCONVOLUTION,
     FILTER
    } OPTIONS1 ;

typedef enum OPTIONS2
    {
     QUIT2,
     NEWSAMPLINGFREQ ,
     NEWADCCHANNEL   ,
     ACQUIREDATA     ,
     ACQUIREPROCESS  ,
     ACQUIRETODISK   ,
     LOADDATA,
     ENTERDATA,
     MAINMENU
    } OPTIONS2 ;

typedef enum OPTIONS3
```

```
    {
     QUIT3,
     DISPLAYINPUT,
     MODIFYINPUT,
     SAVEINPUT,
     PROCESSDATA,
     GRAPHINPUT,
     MAINmenu
    } OPTIONS3 ;

typedef enum OPTIONS4
    {
     QUIT4,
     DISPLAYOUTPUT,
     SAVEOUTPUT,
     GRAPHOUTPUT,
     NEWDATA,
     mainMENU
    } OPTIONS4 ;

typedef enum OPTIONS5
    {
     QUIT5,
     FIRLP,
     FIRHP,
     FIRBP,
     IIRLP ,
     IIRHP ,
     IIRBP ,
     MAIN5
    } OPTIONS5 ;

typedef enum OPTIONS6
    {
     QUIT6,
     DISPLAYOLDOUT,
     SAVEOLDOUT,
     GRAPHOLDOUT,
     MAIN6
    } OPTIONS6 ;

extern  long far pointRE[1026]
         pointIM[1026]      ;

extern  long far XN[2052]
         YN[2052]

typedef struct mnuAtr
    {
      int fgNormal, fgSelect, fgBorder ;
```

```
        long bgNormal, bgSelect, bgBorder ;
        int  centered ;
        char     nw[2], ne[2], se[2], sw[2], ns[2], ew[2] ;
    } mnuAtr ;

int   MENU1(void)                    ;
int   MENU2(void)                    ;
int   MENU3(void)                    ;
int   MENU4(void)                    ;
int   MENU5(void)                    ;
int   MENU6(void)                    ;
int   MENU(int row,int col,char * *items,struct mnuAtr menus) ;
void  box(int row,int col,int hi,int wid,struct mnuAtr menus) ;
void  itemize(int row,int col,char *str,int len)          ;

int   LOAD(int OP)                   ;
void  SAVEIN(int OP)
void  SAVEOUT(int OP)
void  ENTER_DATA(int OP)
void  MODIFY_DATA(int OP)
void  DISPLAYIN(int OP)
void  DISPLAYOUT(int OP)
long  GETabsMAX(long *data)
void  PROCESS_DATA(enum OPTIONS1 choice)     ;
unsigned  cursor(unsigned value)

long  GETabsMAX(long *DATA )         ;
void  GRAPH(int OP,int OUTDATA,char *GTITLE,char *XAXISLABEL)      ;
void  SCALE_DATA(int CHOICE, int IMAGINARY )              ;
void  PLOTlowN(int IMAGINARY,int XSPACING,short COLOR1,short COLOR2) ;
void  PLOThiN(char *GTITLE, char *XAIXSLABEL, int IMAGINARY,int
        XSPACING,short COLOR1,short COLOR2)  ;
void  DRAW1_INIT(char *GTITLE, short COLOR1, short COLOR2)       ;
void  DRAW2_INIT(char *BBBB,char *CCCC,char *DDDD,char *EEEE,char
        *XAXISLABEL)  ;
```

```
        frexp( (double) NQ, &MQ ) ;
        MQ -= 1 ;
        QOPSTATUS = 0 ;

        VBANKNO = QBANKNO ;
        NMIRQST() ;

} /********** END OF PROCESS_DATA **********/

unsigned cursor(value)
unsigned value;
{
     union REGS inregs, outregs;
     int ret;

     inregs.h.ah = 3;       /* Get old cursor */
     inregs.h.bh = 0;
     int86(0x10,&inregs,&outregs);
     ret = outregs.x.cx;
     inregs.h.ah = 1;       /* Set new cursor */
     inregs.x.cx = value;
     int86(0x10,&inregs,&outregs);
     return(ret);
```

B . IBM SOFTWARE PART II

```
/************************
 *     SIG1.C             *
 ************************/

#include  "hsig.h"

char    *SCREEN1[] =
      {
         "QUIT",
         "DIRECT FOURIER TRANSFORM",
         "INVERSE FOURIER TRANSFORM",
         "PERIODOGRAM",
         "AUTOCORRELATION",
         "CROSSCORRELATION",
         "LINEAR CONVOLUTION",
         "CYCLIC CONVOLUTION",
         "FILTER",
         NULL
      } ;

char    *SCREEN2[] =
      {
         "QUIT",
         " ENTER  SAMPLING FREQUENCY",
         " ENTER  ADC CHANNEL ADDRESS",
         " ACQUIRE DATA FROM  D A S  ",
         " ACQUIRE AND PROCESS DATA FROM  D A S ",
         " ACQUIRE DATA AND STORE ON DISK",
         "LOAD INPUT DATA FROM DISK",
         "ENTER INPUT DATA FROM KEYBOARD",
         "RETURN TO MAIN  MENU",
         NULL
      } ;

char    *SCREEN3[] =
      {
         "QUIT",
         "DISPLAY INPUT DATA",
         "MODIFY INPUT DATA",
         "SAVE INPUT DATA",
         "PROCESS INPUT DATA",
         "GRAPH INPUT DATA",
         "RETURN TO MAIN MENU",
         NULL
      } ;

char    *SCREEN4[] =
      {
         "QUIT",
```

```
        "DISPLAY OUTPUT DATA",
        "SAVE OUTPUT DATA",
        "GRAPH OUTPUT DATA",
        "INPUT NEW DATA",
        "RETURN TO MAIN MENU",
        NULL
    } ;

char    *SCREEN5[] =
    {
        "QUIT",
        "FIR LOW PASS ",
        "FIR HIGH PASS ",
        "FIR BAND PASS ",
        "IIR LOW PASS ",
        "IIR HIGH PASS ",
        "IRR BAND PASS",
        "RETURN TO MAIN MENU",
        NULL
    } ;

char    *SCREEN6[] =
    {
        "QUIT",
        "DISPLAY PREVIOUS OUTPUT DATA",
        "SAVE PREVIOUS OUTPUT DATA",
        "GRAPH PREVIOUS OUTPUT DATA",
        "RETURN TO MAIN MENU",
        NULL
    } ;

mnuAtr  menuA =
    {
        0,          15,          4,
        3,          4,           3,
        TRUE,
        "\xDA",     "\xBF",      "\xD9",
        "\xC0",     "\xB3",      "\xC4",
    } ;

char MESSGA[] = { " FFT-BASED ROUTINES " } ;
char MESSGB[] = { "      UP       DOWN    <ENTER> SELECT" } ;

char MESSG1[] = { " FUNCTION  M E N U" };
char MESSG2[] = { " INPUT  D A T A  M E N U" } ;
char MESSG3[] = { " INPUT DATA REVIEW MENU" } ;
char MESSG4[] = { " OUTPUT DATA  M E N U" } ;
char MESSG5[] = { " FILTER  M E N U" } ;
char MESSG6[] = { " PREVIOUS RESULTS " } ;
```

```
LENA = sizeof(MESSGA) ;
LENB = sizeof(MESSGB) ;
LEN1 = sizeof(MESSG1) ;
LEN2 = sizeof(MESSG2) ;
LEN3 = sizeof(MESSG3) ;
LEN4 = sizeof(MESSG4) ;
LEN5 = sizeof(MESSG5) ;
LEN6 = sizeof(MESSG6) ;

FILE          *INfilePTR ,      *OUTfilePTR      ;
char          INfileNAME[40],    OUTfileNAME[40]  ;
long          XDATAMAX,          XDATAMIN,          XDATAabsMAX  ;
int           OPERATION,         pointMAX         ;
int           FIRST                               ;
unsigned char CHANNEL                             ;
double        Fs                                  ;
int           DATACOUNT                           ;
unsigned int  COUNTVAL                            ;
tecdata       TECDATA                             ;
tecstatus     TECSTATUS                           ;
teccntl       TECCNTL                             ;

/**************************************
 *  . MAIN PROGRAM      *
 **************************************/


int main()
{
  int WHERE ;     /* RETURN CODES FOR THE MENU ROUTINES */
                  /* 1 --> MENU1                        */
                  /* 2 --> MENU2                        */
                  /* 3 --> MENU3                        */
                  /* 4 --> MENU4                        */
                  /* 5 --> MENU5                        */



  Fs = 1000.0 ;
  CHANNEL = 0 ;
  FIRST = 1 ;
  WHERE = 1 ;

  COP() ;
  RES() ;
  INIT() ;

  QBANKNO = B_BANK ;
  VBANKNO = B_BANK ;
```

```
    for(i = 0 ; i < 1025 ; i++)
       QTABLE[i] = VTABLE[i] ;

    QOPCODE =OPDONOTHING ;
    NMIRQST() ;

    QBANKNO = A_BANK ;
    VBANKNO = A_BANK ;

    for(i = 0 ; i < 1025 ; i++)
       QTABLE[i] = VTABLE[i] ;

    QOPCODE = OPDONOTHING ;

    _settextcolor(5) ;
    _clearscreen(_GCLEARSCREEN);

    while(1)
    {
       switch(WHERE)
       {
          case 0:
               exit(0)          ;
          case 1:
               WHERE = MENU1()    ;    /* DISPLAY MENU1 */
               break              ;
          case 2:
               WHERE = MENU2()    ;    /* DISPLAY MENU2 */
               break              ;
          case 3:
               WHERE = MENU3()    ;    /* DISPLAY MENU3 */
               break              ;
          case 4:
               WHERE = MENU4()    ;    /* DISPLAY MENU4 */
               break              ;
          case 5:
               WHERE = MENU5()    ;    /* DISPLAY MENU5 */
               break              ;

          case 6:
               WHERE = MENU6()    ;    /* DISPLAY MENU6 */
               break              ;

          default:
               WHERE = MENU1()    ;    /* DISPLAY MENU1 */
               break              ;
       }
    }
    exit(0) ;
} /*************** END OF MAIN PROGRAM **********/

/*********************************************
```

```
*          M E N U 1              *
*****************************************/
int MENU1()
{
  int   WHERE;      /*  RETURN CODE */
  OPTIONS1 choice ;
  char *BUFFER ;

  _settextcolor(5) ;
  _clearscreen(_GCLEARSCREEN);
  _settextposition(2,40 - (LENA / 2) );
  _outtext(MESSGA);                /* MENU TITLE */

  _settextposition(4,40 - (LENB / 2) );
  _outtext(MESSGB);
  _settextposition(6,40 - (LEN1 / 2) );
  _outtext(MESSG1);
  choice = MENU(12,40,SCREEN1,menuA);   /* DISPLAY MENU AND */
  switch(choice)
  {
     case DIRECT :
           OPERATION = 1        ;
           QOPCODE = OPDFFT     ;
           WHERE = 2            ;
           break                ;
     case INVERSE :
           OPERATION = 2        ;
           QOPCODE =  OPINVFFT  ;
           WHERE = 2            ;
           break                ;
     case PERIODOGRAM :
           OPERATION = 3        ;
           QOPCODE = OPPGRAM    ;
           WHERE =  2           ;
           break                ;
     case AUTOCORRELATION :
           OPERATION = 4        ;
           QOPCODE = OPAUTOCORRL ;
           WHERE = 2            ;
           break                ;
     case CROSSCORRELATION :
           OPERATION = 5        ;
           QOPCODE = OPCROSSCORRL ;
           WHERE = 2            ;
           break                ;
     case LCONVOLUTION :
           OPERATION = 6        ;
           QOPCODE = OPLCONVL   ;
           WHERE = 2            ;
           break                ;

     case PCONVOLUTION :
```

```
                OPERATION = 7       ;
                QOPCODE = OPPCONVL  ;
                WHERE = 2           ;
                break               ;

        case FILTER :
                WHERE = 5           ;
                break               ;
        case QUIT1 :
                exit(0)             ;
                break               ;
        default:
                OPERATION = 0       ;
                QOPCODE = OPDONOTHING ;
                WHERE = 1           ;
                break               ;
    } ;

    QOPSTATUS = 0;
    return WHERE;

} /************** END OF MENU1 ************/

/*********************************************
 *          M E N U 2                *
 *********************************************/
int MENU2()
{
 int WHERE;     /* RETURN CODE */
 OPTIONS2 choice;
 char *BUFFER;

 _settextcolor(5);
 _clearscreen(_GCLEARSCREEN);
 _settextposition(2,40 - (LENA / 2) );   /* DISPLAY MENU TITLE */

 _outtext(MESSGA);
 _settextposition(4,40 - (LENB / 2) );
 _outtext(MESSGB);
 _settextposition(6,40 - (LEN2 / 2) );
 _outtext(MESSG2);
 choice = MENU(12,40,SCREEN2,menuA);     /* DISPLAY MENU    */
 switch(choice)
 {
   case NEWSAMPLINGFREQ :
          NEWFREQ()         ;
          WHERE = 2         ;
          break             ;

   case NEWADCCHANNEL :
          NEWADCCHNL()      ;
          WHERE = 2         ;
```

```
            break           ;

        case ACQUIREDATA :
            ACQUIRE()       ;
            WHERE = 3       ;
            break           ;

        case ACQUIREPROCESS :
            ACQPROCESS()    ;
            WHERE = 1       ;
            break           ;

        case ACQUIRETODISK :
            ACQTODISK()     ;
            WHERE = 2       ;
            break           ;

        case LOADDATA :
            N = LOAD(OPERATION) ;
            WHERE = 3       ;
            break           ;
        case ENTERDATA :
            ENTER_DATA(OPERATION) ;
            WHERE = 3       ;
            break           ;
        case MAINMENU :
            OPERATION = 0   ;
            WHERE = 1       ;
            break           ;
        case QUIT2 :
            exit(0)         ;
            break           ;
        default:
            OPERATION = 0   ;
            WHERE = 2       ;
            break           ;
    } ;
    return WHERE ;
} /************** END OF MENU2 ************/

/*********************************************
 *          M E N U 3                *
 *********************************************/
int MENU3()
{
    int   WHERE   ; /* RETURN CODE */
    int   OUTDATA ; /* OUTDATA = 1 IF OUTPUT DATA IS BEING DISPLAYED */
                    /* OUTDATA = 0 IF INPUT  DATA IS BEING DISPLAYED */
    OPTIONS3 choice ;
    char GTITLE[60], XAXISLABEL[60] ;
    char BUFFER[80] ;
```

```
_settextcolor(5) ;
_clearscreen(_GCLEARSCREEN);
_settextposition(2,40 - (LENA / 2) );    /* DISPLAY MENU TITLE */
_outtext(MESSGA);
_settextposition(4,40 - (LENB / 2) );
_outtext(MESSGB);
_settextposition(6,40 - (LEN3 / 2) );
_outtext(MESSG3);
choice = MENU(12,40,SCREEN3,menuA);      /* DISPLAY MENU AND */
switch(choice)
{
   case DISPLAYINPUT :
         DISPLAYIN(OPERATION)   ;
         WHERE = 3              ;
         break                  ;
   case MODIFYINPUT :
         MODIFY_DATA(OPERATION) ;
         WHERE = 3              ;
         break                  ;
   case SAVEINPUT :
         SAVEIN(OPERATION)      ;

         WHERE = 3              ;
         break                  ;
   case PROCESSDATA :
         PROCESS_DATA(OPERATION) ;
         WHERE = 4              ;
         break                  ;
   case GRAPHINPUT :
         OUTDATA = 0 ;
         switch(OPERATION)
         {
            case 1 :
                  strcpy(GTITLE," INPUT DATA  DIRECT FFT") ;
                  strcpy(XAXISLABEL," n ");
                  break          ;
            case 2 :
                  strcpy(GTITLE," INPUT DATA  INVERSE FFT") ;
                  strcpy(XAXISLABEL," m ");
                  break          ;
            case 3 :
                  strcpy(GTITLE," INPUT DATA  PERIODOGRAM ") ;
                  strcpy(XAXISLABEL," n ");
                  break          ;
            case 4 :
                  strcpy(GTITLE," INPUT DATA  AUTOCORRELATION ") ;
                  strcpy(XAXISLABEL," n ");
                  break          ;
            case 5 :
                  strcpy(GTITLE," INPUT DATA  CROSSCORRELATION ") ;
                  strcpy(XAXISLABEL," n ");
                  break          ;
```

```
            case 6 :
                  strcpy(GTITLE," INPUT DATA  LINEAR CONVOLUTION ") ;
                  strcpy(XAXISLABEL," n ");
                  break             ;

            case 7 :
                  strcpy(GTITLE," INPUT DATA  CYCLIC CONVOLUTION ") ;
                  strcpy(XAXISLABEL," n ");
                  break             ;
            case 8 :
                  strcpy(GTITLE," INPUT DATA  FIR LOW PASS FILTER") ;
                  strcpy(XAXISLABEL," n ");
                  break             ;

            case 9 :
                  strcpy(GTITLE," INPUT DATA  FIR HIGH PASS FILTER ") ;
                  strcpy(XAXISLABEL," n ");
                  break             ;
            case 10 :
                  strcpy(GTITLE," INPUT DATA  FIR BAND PASS FILTER ") ;
                  strcpy(XAXISLABEL," n ");
                  break             ;
            case 11 :
                  strcpy(GTITLE," INPUT DATA  IIR LOW PASS FILTER  ") ;
                  strcpy(XAXISLABEL," n ");
                  break             ;
            case 12 :
                  strcpy(GTITLE," INPUT DATA  IIR HIGH PASS FILTER ") ;
                  strcpy(XAXISLABEL," n ");
                  break             ;
            case 13 :
                  strcpy(GTITLE," INPUT DATA  IIR BAND PASS FILTER ") ;
                  strcpy(XAXISLABEL," n ");
                  break             ;
           default:

                 printf("\n FATAL ERROR ");
                 exit(0)            ;
                 break              ;
        }
        GRAPH(OPERATION, OUTDATA, GTITLE, XAXISLABEL) ;
        WHERE = 3           ;
        break               ;
case  MAINmenu  :
        OPERATION = 0       ;
        WHERE = 1           ;
        break               ;
case  QUIT3 :
        exit(0)             ;
        break               ;
default:
        OPERATION = 0       ;
```

```
            WHERE = 4          ;
            break              ;
    } ;
    return WHERE ;
    /*************** END OF MENU3 ************/

/*********************************************
 *          M E N U 4                *
 *********************************************/
int  MENU4()
{
   int   WHERE, OUTDATA ;
   OPTIONS4  choice ;

   char GTITLE[80], XAXISLABEL[40] ;
   char BUFFER[80] ;


   _settextcolor(5) ;
   _clearscreen(_GCLEARSCREEN);
   _settextposition(2,40 - (LENA / 2) );   /* DISPLAY MENU TITLE */
   _outtext(MESSGA);
   _settextposition(4,40 - (LENB / 2) );
   _outtext(MESSGB);
   _settextposition(6,40 - (LEN4 / 2) );
   _outtext(MESSG4);
   choice = MENU(12,40,SCREEN4,menuA);     /* DISPLAY MENU AND */

   QOPCODE = OPDONOTHING ;
   NQ = 2 ;
   MQ = 1 ;
   NMIRQST() ;
   while(VBANKNO != QBANKNO) ;

   switch(choice)
   {
      case  DISPLAYOUTPUT :
            DISPLAYOUT(OPERATION)   ;
            WHERE = 4          ;
            break              ;
      case  SAVEOUTPUT :
            SAVEOUT(OPERATION)     ;
            WHERE = 4          ;
            break              ;
      case  GRAPHOUTPUT :
            OUTDATA = 1 ;
            switch(OPERATION)
            {
               case  1 :
                     strcpy(GTITLE,"OUTPUT DATA  DIRECT FFT") ;
                     strcpy(XAXISLABEL," m ");
                     break              ;
```

```
case 2 :
        strcpy(GTITLE,"OUTPUT DATA  INVERSE  FFT") ;
        strcpy(XAXISLABEL," n ");
        break           ;
case 3 :
        strcpy(GTITLE,"OUTPUT DATA  PERIODOGRAM ") ;
        strcpy(XAXISLABEL," m");
        break           ;
case 4 :
        strcpy(GTITLE,"OUTPUT DATA  AUTOCORRELATION ") ;
        strcpy(XAXISLABEL," n");
        break           ;
case 5 :
        strcpy(GTITLE,"OUTPUT DATA  CROSSCORRELATION ") ;
        strcpy(XAXISLABEL," n");
        break           ;

case 6 :
        strcpy(GTITLE,"OUTPUT DATA  LINEAR CONVOLUTION ") ;
        strcpy(XAXISLABEL," n ");
        break           ;

case 7 :
        strcpy(GTITLE,"OUTPUT DATA  CYCLIC CONVOLUTION ") ;
        strcpy(XAXISLABEL," n ");
        break           ;
case 8 :
        strcpy(GTITLE,"OUTPUT DATA  FIR LOW PASS FILTER") ;
        strcpy(XAXISLABEL," n ");
        break           ;

case 9 :
        strcpy(GTITLE,"OUTPUT DATA  FIR HIGH PASS FILTER ") ;
        strcpy(XAXISLABEL," n ");
        break           ;
case 10 :
        strcpy(GTITLE,"OUTPUT DATA  FIR BAND PASS FILTER ") ;
        strcpy(XAXISLABEL," n ");
        break           ;
case 11 :
        strcpy(GTITLE,"OUTPUT DATA  IIR LOW PASS FILTER  ") ;
        strcpy(XAXISLABEL," n ");
        break           ;
case 12 :
        strcpy(GTITLE,"OUTPUT DATA  IIR HIGH PASS FILTER ") ;
        strcpy(XAXISLABEL," n ");
        break           ;
case 13 :
        strcpy(GTITLE,"OUTPUT DATA  IIR BAND PASS FILTER ") ;
        strcpy(XAXISLABEL," n ");
        break           ;
```

```
                default:
                        printf("\n FATAL ERROR ");
                        exit(0)         ;
                        break           ;
                }
                GRAPH(OPERATION, OUTDATA, GTITLE, XAXISLABEL) ;
                WHERE = 4               ;
                break           ;
        case NEWDATA :
                WHERE = 2               ;
                break           ;
        case mainMENU :
                OPERATION = 0           ;
                WHERE = 1               ;

                break           ;
        case QUIT4 :
                exit(0)         ;
                break           ;
        default:
                OPERATION = 0           ;
                WHERE = 4               ;
                break           ;
  } ;
  return WHERE ;
} /*************** END OF MENU4 ************/

/*********************************************
 *          M E N U 5                *
 *********************************************/
int MENU5()
{
  int   WHERE;      /*  RETURN CODE */
  OPTIONS5 choice ;
  char BUFFER[80] ;

  _settextcolor(5) ;
  _clearscreen(_GCLEARSCREEN);
  _settextposition(2,40 - (LENA / 2) );
  _outtext(MESSGA);                 /* MENU TITLE */
  _settextposition(4,40 - (LENB / 2) );
  _outtext(MESSGB);
  _settextposition(6,40 - (LEN5 / 2) );
  _outtext(MESSG5);
  choice = MENU(12,40,SCREEN5,menuA);   /* DISPLAY MENU AND */
  switch(choice)
  {
     case FIRLP :
          OPERATION = 8         ;
          QOPCODE = OPFIRLP     ;
          WHERE = 2             ;
          break         ;
```

```
        case FIRHP :
                OPERATION = 9       ;
                QOPCODE = OPFIRHP   ;
                WHERE = 2           ;
                break               ;
        case FIRBP :
                OPERATION = 10.     ;
                QOPCODE = OPFIRBP   ;
                WHERE = 2           ;
                break               ;
        case IIRLP :
                OPERATION = 11      ;
                QOPCODE = OPIIRLP   ;
                WHERE = 2           ;
                break               ;

        case IIRHP :
                OPERATION = 12      ;
                QOPCODE = OPIIRHP   ;
                WHERE = 2           ;
                break               ;

        case IIRBP :
                OPERATION = 13      ;
                QOPCODE = OPIIRBP   ;
                WHERE = 2           ;
                break               ;
        case QUIT5 :
                exit(0)             ;
                break               ;
        default:
                OPERATION = 0       ;
                QOPCODE = OPDONOTHING ;
                WHERE = 5           ;
                break               ;
    } ;

    QOPSTATUS = 0 ;
    return WHERE ;

} /*************** END OF MENU5 ************/


/*****************************************
 *          M E N U 6                    *
 *****************************************/
int MENU6()
{
    int WHERE, OUTDATA ;
    OPTIONS6 choice ;
```

```
char GTITLE[80], XAXISLABEL[40] ;
char BUFFER[80] ;

_settextcolor(5) ;
_clearscreen(_GCLEARSCREEN);
_settextposition(2,40 - (LENA / 2) );    /* DISPLAY MENU TITLE */
_outtext(MESSGA);
_settextposition(4,40 - (LENB / 2) );
_outtext(MESSGB);
_settextposition(6,40 - (LEN6 / 2) );
_outtext(MESSG6);
choice = MENU(12,40,SCREEN6,menuA);     /* DISPLAY MENU AND */
switch(choice)
{
   case DISPLAYOUTPUT :
         DISPLAYOUT(OPERATION)  ;
         WHERE = 6              ;
         break                  ;
   case SAVEOUTPUT :
         SAVEOUT(OPERATION)     ;
         WHERE = 6              ;
         break                  ;
   case GRAPHOUTPUT :
        OUTDATA = 1 ;
         switch(OPERATION)
         {
            case 1 :
                  strcpy(GTITLE,"OUTPUT DATA  DIRECT FFT") ;
                  strcpy(XAXISLABEL," n ");
                  break                 ;
             case 2 :
                  strcpy(GTITLE,"OUTPUT DATA  INVERSE  FFT") ;
                  strcpy(XAXISLABEL," n ");
                  break                 ;
             case 3 :
                  strcpy(GTITLE,"OUTPUT DATA  PERIODOGRAM ") ;
                  strcpy(XAXISLABEL," m");
                  break                 ;
             case 4 :
                  strcpy(GTITLE,"OUTPUT DATA  AUTOCORRELATION ") ;
                  strcpy(XAXISLABEL," n");
                  break                 ;
             case 5 :
                  strcpy(GTITLE,"OUTPUT DATA  CROSSCORRELATION ") ;
                  strcpy(XAXISLABEL," n");
                  break                 ;
             case 6 :
                  strcpy(GTITLE,"OUTPUT DATA  LINEAR CONVOLUTION ") ;
                  strcpy(XAXISLABEL," n ");
                  break                 ;

             case 7 :
```

```
                strcpy(GTITLE,"OUTPUT DATA  CYCLIC CONVOLUTION ") ;
                strcpy(XAXISLABEL," n ");
                break           ;
        case  8 :
                strcpy(GTITLE,"OUTPUT DATA  FIR LOW PASS FILTER") ;
                strcpy(XAXISLABEL," n ");
                break           ;

        case  9 :
                strcpy(GTITLE,"OUTPUT DATA  FIR HIGH PASS FILTER ") ;
                strcpy(XAXISLABEL," n ");
                break           ;
        case 10 :
                strcpy(GTITLE,"OUTPUT DATA  FIR BAND PASS FILTER ") ;
                strcpy(XAXISLABEL," n ");
                break           ;
        case  11 :
                strcpy(GTITLE,"OUTPUT DATA  IIR LOW PASS FILTER  ") ;
                strcpy(XAXISLABEL," n ");
                break           ;
        case  12 :
                strcpy(GTITLE,"OUTPUT DATA  IIR HIGH PASS FILTER ") ;
                strcpy(XAXISLABEL," n ");
                break           ;
        case  13 :
                strcpy(GTITLE,"OUTPUT DATA  IIR BAND PASS FILTER ") ;
                strcpy(XAXISLABEL," n ");
                break           ;

        default:
                printf("\n FATAL ERROR ");
                exit(0)         ;
                break           ;
        }
        GRAPH(OPERATION, OUTDATA, GTITLE, XAXISLABEL) ;
        WHERE =  6      ;
        break           ;
    case  MAIN6 :
        OPERATION = 0       ;
        WHERE = 1       ;

        break           ;
    case  QUIT6 :
        exit(0)         ;
        break           ;
    default:
        OPERATION = 0       ;
        WHERE = 6       ;
        break           ;
  } ;
  return WHERE ;
} /************** END OF MENU6 ************/
```

```
/*************************************************************************
 *    PUT MENU ON SCREEN                                                 *
 *    Starting <row> and <column>.                                       *
 *    Array of menu <items> strings.                                     *
 *    Global structure variable <menus> determines:                      *
 *         Colors of border, normal items, and selected item.            *
 *         Centered or left justfied.                                    *
 *         Border characters.                                            *
 * Returns number of item selected.                                      *
 *************************************************************************/
int MENU(row, col, items, menus)
int row, col;
char *items[];
mnuAtr menus ;
{
     int i, num, max = 2, prev, curr = 0, choice;
     int litem[25];
     long bcolor;
     char *BUFFER ;
     unsigned PREVIOUSCURSOR ;

     PREVIOUSCURSOR = cursor(TCURSOROFF);
     bcolor = _getbkcolor();
     /* Count items, find longest, and put length of each in array */
     for (num = 0; items[num]; num++) {
          litem[num] = strlen(items[num]);
          max = (litem[num] > max) ? litem[num] : max;
     }
     max += 2;
     if (menus.centered) {
          row -= num / 2;
          col -= max / 2;
     }
     /* Draw menu box */
     _settextcolor(menus.fgBorder);
     _setbkcolor(menus.bgBorder);
     box(row++,col++,num,max,menus);
     /* Put items in menu */
     for (i = 0; i < num; ++i) {
          if (i == curr) {
               _settextcolor(menus.fgSelect);
               _setbkcolor(menus.bgSelect);
          } else {
               _settextcolor(menus.fgNormal);
               _setbkcolor(menus.bgNormal);
          }

          itemize(row+i,col,items[i],max - litem[i]);
     }
     /* Get selection */
     for (;;) {
```

```
        switch ((_bios_keybrd(_KEYBRD_READ) & 0xff00) >> 8) {
            case UP :
                prev = curr;
                curr = (curr > 0) ? (--curr % num) : num-1;
                break;
            case DOWN :
                prev = curr;
                curr = (curr < num) ? (++curr % num) : 0;
                break;
            case ENTER :
                _setbkcolor(bcolor);
                cursor(PREVIOUSCURSOR);
                return(curr);
            default :
                continue;
        }
        _settextcolor(menus.fgSelect);
        _setbkcolor(menus.bgSelect);
        itemize(row+curr,col,items[curr],max - litem[curr]);
        _settextcolor(menus.fgNormal);
        _setbkcolor(menus.bgNormal);
        itemize(row+prev,col,items[prev],max - litem[prev]);
    }
} /************ END OF MENU ************/

/*********************************************************
 * Draw menu box.                                        *
 *     <row> and <col> are upper left of box.            *
 *     <hi> and <wid> are height and width.              *
 *********************************************************/
void box(row, col, hi, wid, menus)
int row, col, hi, wid;
mnuAtr menus ;
{
    int i;
    char temp[80];

    _settextposition(row,col);
    temp[0] = *menus.nw;
    memset(temp+1,*menus.ew,wid);
    temp[wid+1] = *menus.ne;
    temp[wid+2] = NULL;
    _outtext(temp);
    for (i = 1; i <= hi; ++i) {
        _settextposition(row+i,col);
        _outtext(menus.ns);
        _settextposition(row+i,col+wid+1);
        _outtext(menus.ns);
    }
    _settextposition(row+hi+1,col);
    temp[0] = *menus.sw;
    memset(temp+1,*menus.ew,wid);
```

```
        temp[wid+1] = *menus.se;
        temp[wid+2] = NULL;
        _outtext(temp);

    } /********* END OF MENU ***************/


/*************************************************************
 * Put an item in menu.                                      *
 *     <row> and <col> are left position.                    *
 *     <str> is the string item.                             *
 *     <len> is the number of blanks to fill.                *
 *************************************************************/
void itemize(row,col,str,len)
int row, col, len;
char str[];
{
     char temp[80];

     _settextposition(row,col);
     _outtext(" ");
     _outtext(str);
     memset(temp,' ',len--);
     temp[len] = NULL;
     _outtext(temp);
} /********** END OF ITEMIZE **************/

/*********************************************
 *          LOAD(OP)                         *
 * LOADS DATA FROM A FILE. THE DATA FORMAT   *
 * DEPENDS ON THE OPERATION BEING PERFORMED. *
 * IT IS A SINGLE ARRAY OF INTEGERS FOR THE  *
 * FOLLOWING OPERATIONS:                     *
 *              1. DIRECT FFT                *
 *              2. PERIODOGRAM               *
 *              3. AUTOCORRELATION           *
 * AND TWO SINGLE ARRAY OF INTEGERS FOR THE  *
 * FOLLOWING OPERATIONS:                     *
 *              1. CROSSCORRELATION          *
 *              2. CONVOLUTION               *
 * THE FINAL DATA FORMAT IS FOR INVERSE      *
 * FFT AND IS MADE UP A COMPLEX ARRAY OF     *
 * INTEGERS.                                 *
 *********************************************/
int LOAD(OP)
int OP;
{
  int i , KEYIN  ;
  char *BUFFER;

  _setvideomode(_TEXTC80 ) ;
/* _setbkcolor(_LIGHTGREEN);       */
```

```
_clearscreen(_GCLEARSCREEN);
printf("\n  ENTER NAME OF DATA FILE : ") ;
gets(INfileNAME);
if((INfilePTR =fopen(INfileNAME,"r")) ==0)
{
   printf("\n  CANNOT OPEN FILE  %s FOR READING \n",INfileNAME);
   exit(0);
} ;
printf("\n  IS THE NUMBER OF DATA POINTS SPECIFIED\n");
printf("\n  IN THE INPUT DATA FILE ? ");
KEYIN =  _bios_keybrd(_KEYBRD_READ) & 0x00FF ;
if( KEYIN == 'Y' || KEYIN == 'y' )
{
printf("\n    LOADING DATA,  WAIT...");
fscanf(INfilePTR," %d ",&N ) ;
}

else
{
   printf("\n  ENTER THE NUMBER OF DATA POINTS :")  ;
   NQ = atoi(gets(BUFFER)) ;
}
printf("\n   N    = %d \n", NQ) ;

if(NQ > 1024)
{
  printf("\n VALUE OF N TOO BIG \n") ;
  exit(0) ;
 }

switch(OP)
{
   case  1:
         for(i = 1; i < NQ + 1; i++)
         {
            fscanf(INfilePTR,"\n %ld ",&XQREAL[i]) ;
            XQIMAG[i] = 0   ;
          }
         break        ;
   case  2:
         for(i = 1; i < NQ + 1; i++)
            fscanf(INfilePTR,"\n %ld  %ld ",&XQREAL[i],&XQIMAG[i]) ;
         break        ;
   case  3:
   case  4:
         for(i = 1; i < NQ + 1; i++)
         {
            fscanf(INfilePTR,"\n %ld ",&XQREAL[i]) ;
            XQIMAG[i] = XQREAL[i]  ;
          }
         break    ;
   case  5:
```

```
        case 6:
        case 7:
        case 8:
        case 9:
        case 10:
        case 11:
        case 12:
        case 13:
              for(i = 1; i < NQ + 1; i++)
                 fscanf(INfilePTR,"\n %ld ",&XQREAL[i]) ;

              for(i = 1; i < NQ + 1; i++)
                 fscanf(INfilePTR,"\n %ld ",&XQIMAG[i]) ;
              break ;
        default:
              printf("\n  ERROR IN TYPE OF OPERATION \n");
              exit(0) ;

     }
     fclose(INfilePTR) ;
     return NQ ;
  } /*************** END OF LOAD **********/

  /*****************************************
   *        SAVEIN(OP)                     *
   * SAVES INPUT DATA IN FILE. THE DATA FORMAT *
   * DEPENDS ON THE OPERATION BEING PERFORMED. *
   * IT IS A SINGLE ARRAY OF INTEGERS  FOR THE *
   * FOLLOWING OPERATIONS:                 *
   *              1. DIRECT FFT      *
   *              2. PERIODOGRAM     *
   *              3. AUTOCORRELATION *
   * AND TWO SINGLE ARRAY OF INTEGERS FOR THE *
   * FOLLOWING OPERATIONS:                 *
   *              1. CROSSCORRELATION *
   *              2. CONVOLUTION     *
   * THE FINAL DATA FORMAT IS FOR INVERSE  *
   * FFT AND IS MADE UP A COMPLEX ARRAY OF    *
   * INTEGERS.                             *
   *****************************************/

  void SAVEIN(OP)
  int OP;
  {
     int i ;
     char *BUFFER;

     _setvideomode(_TEXTC80) ;
  /* _setbkcolor(_LIGHTMAGENTA);    */
     _clearscreen(_GCLEARSCREEN);
     printf("\n  ENTER NAME DATA FILE: ") ;
     gets(INfileNAME);
```

```
    if((INfilePTR =fopen(INfileNAME,"w")) ==0)
    {
       printf("\n  CANNOT OPEN FILE  %s FOR WRITING \n",INfileNAME);
       exit(0);
    } ;
    printf("\n    SAVING DATA,  WAIT...");
    putw(NQ, INfilePTR) ;
    switch(OP)
    {
       case  1:
       case  3:
       case  4:
             for(i = 1; i < NQ + 1; i++)
                fprintf(INfilePTR,"\n %ld ",XQREAL[i]) ;
              break        ;
       case  2:
              for(i = 1; i < NQ + 1; i++)
                fprintf(INfilePTR,"\n %ld   %ld ",XQREAL[i],XQIMAG[i]);
             break        ;
       case  5:
       case  6:
       case  7:
       case  8:
       case  9:
       case 10:
       case 11:
       case 12:
       case 13:
             for(i = 1; i < NQ + 1; i++)
                fprintf(INfilePTR,"\n %ld ",XQREAL[i]) ;
             for(i = 1; i < NQ + 1; i++)
                fprintf(INfilePTR,"\n %ld ",XQIMAG[i]) ;
             break ;
       default:
             printf("\n  ERROR IN TYPE OF OPERATION \n");
             exit(0) ;
    }
    fclose(INfilePTR) ;
    return     ;
} /*************** END OF SAVEIN  **********/

/*********************************************
 *         SAVEOUT(OP)                       *
 * SAVES OUTPUT DATA IN FILE. THE DATA FORMAT *
 * DEPENDS ON THE OPERATION BEING PERFORMED.  *
 *********************************************/
void SAVEOUT(OP)
int  OP;
{
   int i ;
   char *BUFFER;
```

```
    _setvideomode(_TEXTC80 ) ;
  /* _setbkcolor(_LIGHTCYAN);   */
    _clearscreen(_GCLEARSCREEN);
    printf("\n  ENTER NAME OF OUTPUT FILE : ") ;
    gets(OUTfileNAME);

    if((OUTfilePTR =fopen(OUTfileNAME,"w")) ==0)
    {
       printf("\n  CANNOT OPEN FILE %s FOR WRITING \n",OUTfileNAME);
       exit(0);
    } ;
    printf("\n    SAVING DATA,  WAIT...");
    switch(OP)
    {
       case  1:
       case  2:
       case  5:
       case  6:
       case  7:
       case  8:
       case  9:
       case 10:
       case 11:
       case 12:
       case 13:
          for(i = 1; i < NQ + 1; i++)
           fprintf(OUTfilePTR,"\n X %d  %ld  \t j  %ld  ", i - 1, XQREAL[i],XQIMAG[i]);
            break       ;
       case  3:
       case  4:
            for(i = 1; i < NQ + 1; i++)
              fprintf(OUTfilePTR,"\n X  %d   %ld ", i - 1, XQREAL[i]) ;
              break  ;
       default:
              printf("\n  ERROR IN TYPE OF OPERATION \n");
              exit(0) ;
    }
   fclose(OUTfilePTR) ;
   return    ;
} /********** END  OF SAVEOUT **********/

/*********************************************
 *       ENTER_DATA(OP)              *
 * ENABLES DATA TO BE ENTERED FROM THE KEY-  *
 * BOARD.                          *
 *********************************************/
void ENTER_DATA(OP)
int OP;
{
  int i ;
  char *BUFFER , DUMB[80] ;
```

```
   _setvideomode(_TEXTC80 ) ;
/* _setbkcolor(_LIGHTBLUE) ;        */
   _clearscreen(_GCLEARSCREEN) ;
   printf("\n  ENTER THE NUMBER OF DATA POINTS :  ") ;
   NQ = atoi(gets(BUFFER)) ;

   if(NQ >1024)
   {
      printf(" \n value of N too big ") ;
      exit(0) ;
   }

   switch(OP)
   {
      case  1:
            for(i = 1; i < NQ + 1; i++)
            {
               printf("\n  X %d =  ", i - 1) ;
               XQREAL[i] = atol(gets(BUFFER)) ;
               XQIMAG[i] = 0   ;
            }
            break       ;
      case  2:
            for(i = 1; i < NQ + 1; i++)
            {
               printf("\n RealX %d =  ", i - 1) ;
               XQREAL[i] = atol(gets(BUFFER)) ;
               printf("\n  ImagX %d =  ", i - 1) ;
               XQIMAG[i] =  atol(gets(BUFFER)) ;
            }
            break       ;

      case  3:
      case  4:
            for(i = 1; i < NQ + 1; i++)
            {
               printf("\n  X %d =  ", i - 1) ;
               XQREAL[i] = atol(gets(BUFFER)) ;
               XQIMAG[i] = XQREAL[i]  ;
            }
            break    ;
      case  5:
      case  6:
      case  7:
      case  8:
      case  9:
      case 10:
      case 11:
      case 12:
      case 13:
            for(i = 1; i < NQ + 1; i++)
            {
```

```
            printf("\n  X %d =  ", i - 1) ;
            XQREAL[i] =  atol(gets(BUFFER)) ;
         }
         for(i = 1; i < NQ + 1; i++)
         {
            printf("\n  Y %d =  ", i - 1);
            XQIMAG[i] =  atol(gets(BUFFER)) ;
         }
         break ;
      default:
         printf("\n  ERROR IN TYPE OF OPERATION \n");
         exit(0) ;
   }
   return  ;
} /*************** END  OF  ENTER_DATA ********/

/***********************************************
 *       MODIFY_DATA(OP)                       *
 * ENABLES INPUT DATA TO BE MODIFIED           *
 * FROM THE KEYBOARD.                          *
 ***********************************************/
void MODIFY_DATA(OP)
int OP;
{
   int i , KEYIN ;
   char *BUFFER;

   _setvideomode(_TEXTC80 ) ;
/* _setbkcolor(_LIGHTRED );     */
   _clearscreen(_GCLEARSCREEN);
   printf("\n  PRESS  < ENTER >  TO CONTINUE  ") ;
   gets(BUFFER) ;
   while(1)
   {
      printf("\n  ENTER C FOR CONTINUE AND Q FOR QUIT  ");
      KEYIN =  _bios_keybrd(_KEYBRD_READ) & 0x00FF ;
      if( KEYIN == 'C' || KEYIN == 'c' )
      {
         gets(BUFFER);
         printf("\n  Index      : ") ;
         i = atoi(gets(BUFFER)) ;
         if( (i < 0)  ||  ( i  > NQ) )
         {
            printf("\n   index out of range \n");
            break ;
         }
         gets(BUFFER) ;

         switch(OP)
         {
            case  1:
               printf("\n  X %d =  ",i) ;
```

```
                scanf("%ld",&XQREAL[i + 1]) ;
                XQIMAG[i +  1] = 0  ;
                break       ;

          case 2:
                printf("\n  RealX %d =  ",i) ;
                scanf("%ld ",&XQREAL[i + 1]) ;
                printf("\n  ImagX %d =  ",i) ;
                scanf("%ld ",&XQIMAG[i + 1]) ;
                break       ;

          case 3:
          case 4:
                printf("\n  X %d =  ",i) ;
                scanf("%ld",&XQREAL[i + 1]) ;
                XQIMAG[i + 1] = XQREAL[i + 1]  ;
                break   ;
          case 5:
          case 6:
          case 7:
          case 8:
          case 9:
          case 10:
          case 11:
          case 12:
          case 13:
                printf("\n  X %d =  ",i) ;
                scanf("%ld",&XQREAL[i + 1]) ;
                printf("\n  Y %d =  ",i) ;
                scanf("%ld",&XQIMAG[i + 1]) ;
                break ;
          default:
                printf("\n  ERROR IN TYPE OF OPERATION \n");
                exit(0) ;
        }
      }
      else if( KEYIN == 'Q' || KEYIN == 'q' )
         return ;
    }
   return ;
} /*************** END OD MODIFY_DATA *********/

/***********************************************
*       DISPLAYIN(OP)                 *
* DISPLAYS INPUT DATA ON THE SCREEN       *
***********************************************/
void DISPLAYIN(OP)
int  OP;
{
   int i  ;
   char *BUFFER ;
```

```
_setvideomode(_TEXTC80 ) ;
/* _setbkcolor(_LIGHTMAGENTA);   */
_clearscreen(_GCLEARSCREEN);
switch(OP)
{
   case 1:
   case 3:
   case 4:
        for(i = 1; i < NQ + 1; i++)
        {
           if( !(i%20) )
           {
              printf("\n  PRESS < ENTER > TO CONTINUE");
              getchar() ;
           }

           printf("\n  X %d =  %ld ", i - 1, XQREAL[i]) ;
        }
        break        ;
  case 2:
        for(i = 1; i < NQ + 1; i++)
        {
           if( !(i%20) )
           {
              printf("\n  PRESS < ENTER > TO CONTINUE");
              getchar();
           }
         printf("\nX %d =  %ld j  %ld ", i - 1, XQREAL[i],XQ
        }
        break        ;
  case 5:
  case 6:
  case 7:
  case 8:
  case 9:
  case 10:
  case 11:
  case 12:
  case 13:
      for(i = 1; i < NQ + 1; i++)
      {
         if( !(i%20) )
         {
            printf("\n  PRESS < ENTER > TO CONTINUE");
            getchar() ;
         }
         printf("\n X %d =  %ld ", i - 1, XQREAL[i]) ;
      }
     printf("\n PRESS   < ENTER > TO CONTINUE \n");
     gets(BUFFER) ;
     for(i = 1; i < NQ + 1; i++)
     {
```

```
            if( !(i%20) )
            {
               printf("\n  PRESS ANY KEY TO CONTINUE");
               getchar() ;
            }
            printf("\n Y %d =  %ld ", i - 1, XQIMAG[i]) ;
         }
         break ;
      default:
         printf("\n  ERROR IN TYPE OF OPERATION \n");
         exit(0) ;
   }
   printf("\n PRESS   < ENTER > TO CONTINUE \n");
   gets(BUFFER) ;
   return    ;
} /*************** END  OF  DISPLAYIN *****/

/***********************************************
 *       DISPLAYOUT(OP)                        *
 * DISPLAYS OUTPUT DATA ON THE SCREEN          *
 ***********************************************/
void DISPLAYOUT(OP)
int  OP;
{
   int i ;
   char *BUFFER ;

   _setvideomode(_TEXTC80 ) ;
/*  _setbkcolor(_LIGHTMAGENTA);      */
   _clearscreen(_GCLEARSCREEN);

   switch(OP)
   {
      case  1:
      case  2:
      case  5:
      case  6:
      case  7:
      case  8:
      case  9:
      case 10:
      case 11:
      case 12:
      case 13:
         for(i = 1; i < NQ + 1; i++)
         {
            if( !(i%20) )
            {
               printf("\n  PRESS  < ENTER > TO CONTINUE");
               getchar() ;
            }
            printf("\n X %d  %ld  j  %ld ", i - 1, XQREAL[i],XQIMAG[i]) ;
```

```
            }
            break       ;
        case 3:
        case 4:
            for(i = 1; i < NQ + 1; i++)
            {
               if( !(i%20) )
               {
                  printf("\n  PRESS  < ENTER > TO CONTINUE");
                  getchar() ;
               }
              printf("\n X  %d   %ld ", i - 1, XQREAL[i]) ;
            }
            break ;
        default:
            printf("\n  ERROR IN TYPE OF OPERATION \n");
            exit(0) ;

    }
    printf("\n PRESS   < ENTER > TO CONTINUE \n");
    gets(BUFFER) ;
    return    ;
} /**************** END OF DISPLAYOUT **********/

/************************************************
 *       GETabsMAX(long *DATA)          *
 * COMPUTES THE ABSOLUTE MAXIMUN OF AN      *
 * ARRAY OF LONG INTEGERS                   *
 ************************************************/
long GETabsMAX(long *data)
{
   register int  i ;
   long dataABSmax , dataMAX, dataMIN ;

   for( dataMAX = *(data+1), dataMIN = dataMAX, i = 2; i < NQ + 1; i++)
   {
      if( *(data + i) > dataMAX )  dataMAX =  *(data + i)  ;
      if( *(data + i) <  dataMIN )  dataMIN = *(data + i)  ;
   }
   dataMAX = labs(dataMAX) ;
   dataMIN = labs(dataMIN) ;
   dataABSmax = ( dataMAX >= dataMIN ) ?  dataMAX :  dataMIN  ;
   return dataABSmax ;
} /*************** END OF GETabsMAX **********/


/************************************************
 *   PROCESS_DATA(choice)                *
 *   PERFORMS ACTUAL COMPUTATION         *
 ************************************************/
void PROCESS_DATA(OPTIONS1 choice)
{
```

```
        frexp( (double) NQ, &MQ ) ;
        MQ -= 1 ;
        QOPSTATUS = 0 ;

        VBANKNO = QBANKNO ;
        NMIRQST() ;

} /********** END OF PROCESS_DATA **********/

unsigned cursor(value)
unsigned value;
{
      union REGS inregs, outregs;
      int ret;

      inregs.h.ah = 3;       /* Get old cursor */
      inregs.h.bh = 0;
      int86(0x10,&inregs,&outregs);
      ret = outregs.x.cx;
      inregs.h.ah = 1;       /* Set new cursor */
      inregs.x.cx = value;
      int86(0x10,&inregs,&outregs);
      return(ret);
}
```

C. IBM SOFTWARE PART III

```
/*******************************
 *     SIG2.C                  *
 *******************************/

#include "hsig.h"

extern long XDATAMAX, XDATAMIN, XDATAabsMAX ;
extern int OPERATION , pointMAX ;
extern unsigned char CHANNEL ;
extern double Fs ;
extern int FIRST ;
extern char INfileNAME[], OUTfileNAME[] ;
extern unsigned int COUNTVAL ;
extern tecdata TECDATA ;
extern teccntl TECCNTL ;
extern tecstatus TECSTATUS ;
extern int DATACOUNT ;
int XOFFSET ;
char *XLABEL[]=
      {
         "0000", "0025", "0050", "0075",
         "0100", "0125", "0150", "0175",
         "0200", "0225", "0250", "0275",
         "0300", "0325", "0350", "0375",
         "0400", "0425", "0450", "0475",
         "0500", "0525", "0550", "0575",
         "0600", "0625", "0650", "0675",
         "0700", "0725", "0750", "0775",
         "0800", "0825", "0850", "0875",
         "0900", "0925", "0950", "0975",
        "1000", "1025", NULL
      } ;
/***********************************************
 * GRAPH(OP,OUTDATA,GTITLE,XAXISLABEL)         *
 *                                             *
 * OP     --> TYPE OF OPERATION                *
 * OUTDATA   = 1 FOR OUTPUT DATA               *
 *           = 0 FOR INPUT DATA                *
 * GTITLE  -->  TITLE OF GRAPH                 *
 * XAXISLABEL--> LABEL ON THE X - AXIS         *
 ***********************************************/
void GRAPH(int OP, int OUTDATA, char GTITLE[], char XAXISLABEL[])
{
  char BBBB[10], CCCC[10], DDDD[10], EEEE[10] ;
                    /* NUMERICAL LABELS ON THE X-AXIS */
  int i, CHOICE, pointMAX, IMAGINARY ;
  int XSPACING ;
  char buffer[80] ;
  long imMAX, reMAX ;
  short COLOR1 = 12, COLOR2 = 14 ;
```

```
/*  _remappalette(0,11L);       */
    _setvideomode( _ERESCOLOR );
    _setbkcolor(_LIGHTBLUE) ;
    _clearscreen(_GCLEARSCREEN);
    _setcolor(13);
    _rectangle(_GFILLINTERIOR, 0,0, 639, 349);
    _setcolor(11) ;
    _rectangle(_GFILLINTERIOR, 5, 5, 635, 345);
    _setcolor(10) ;
    _rectangle(_GFILLINTERIOR, 70, 25, 637, 347);
    _setcolor(9) ;
    _rectangle(_GFILLINTERIOR, 75, 30, 632, 342);
    _setlogorg(115,190);
    _setcolor(10);
    for(i= -20; i < 512 ; i +=20)
    {
        _moveto(i,-159);
        _lineto(i,150);
        _moveto(i-1,-159);
        _lineto(i-1,150);
    }
    for(i= -140; i < 151 ; i +=20)
    {
        _moveto(-39, i);
        _lineto(520, i);
        _moveto(-39, i-1);
        _lineto(517, i-1);
    }
    _setcolor(12);
    _moveto(0,0);
    _lineto(515,0);
    _moveto(0,-1);
    _lineto(515,-1);
    _moveto(505,-10);
    _lineto(515,-1);
    _lineto(505,10);
    _moveto(504,-11);
    _lineto(515,0);
    _lineto(504,11);
    _moveto(0,-152);
    _lineto(0,152);
    _moveto(-1,-152);
    _lineto(-1,152);
    _moveto(1,-152);
    _lineto(1,152);
    _moveto(-10,-142);
    _lineto(0,-152);
    _lineto(10,-142);
    _moveto(-11,-141);
    _lineto(-1,-152);
    _lineto(11,-141);
```

```
_settextposition(5,30);
_settextcolor(13);
_outtext(GTITLE);
_settextposition(3,25);
_settextcolor(14) ;
_outtext("<-- SCROLL LEFT  --> SCROLL RIGHT ");
_settextposition(4,30);
_outtext("      < ENTER > END SCROLLING");
_settextcolor(15) ;
_setcolor(15) ;
_rectangle(_GFILLINTERIOR, 104, 104, 157,117) ;
_setcolor(COLOR1) ;
_moveto(107,109) ;
_lineto(150,109) ;
_moveto(107,110) ;
_lineto(150,110) ;
_settextcolor(14) ;
_settextposition(22,36) ;
_outtext("REAL") ;
_setcolor(15) ;
_rectangle(_GFILLINTERIOR, 304, 104, 357,117) ;
_setcolor(COLOR2) ;
_moveto(307,109) ;
_lineto(350,109) ;
_moveto(307,110) ;
_lineto(350,110) ;
_settextposition(22,61) ;
_outtext("IMAGINARY") ;
_settextcolor(15) ;
_settextposition(4,11);
_outtext("MAGN");
_settextposition(7,12);
_outtext("100");
_settextposition(11,13);

_outtext("40");
_settextposition(14,13);     /*  ORIGIN  */
_outtext("00");
_settextposition(17,12);
_outtext("-40");
_settextposition(20,12);
_outtext("-80");
DRAW1_INIT(GTITLE, COLOR1, COLOR2) ;  /*  DRAWS GRAPHIC SCREEN  */
switch(NQ)
{
   case  2:
         strcpy(BBBB, "0000") ;
         strcpy(CCCC, "0001") ;
         strcpy(DDDD, "0002") ;
         strcpy(EEEE, "0003") ;
         XSPACING  = 100 ;
         break  ;
```

```
    case 4:
         strcpy(BBBB, "0000") ;
         strcpy(CCCC, "0002") ;
         strcpy(DDDD, "0004") ;
         strcpy(EEEE, "0006") ;
         XSPACING = 50 ;
         break ;
    case 8:
         strcpy(BBBB, "0000") ;
         strcpy(CCCC, "0004") ;
         strcpy(DDDD, "0008") ;
         strcpy(EEEE, "0012") ;
         XSPACING = 25 ;
         break ;
    case 16:
         strcpy(BBBB, "0000") ;
         strcpy(CCCC, "0010") ;
         strcpy(DDDD, "0020") ;
         strcpy(EEEE, "0030") ;
         XSPACING = 10 ;
         break ;
    case 32:
    case 64:
         strcpy(BBBB, "0000") ;
         strcpy(CCCC, "0025") ;
         strcpy(DDDD, "0050") ;
         strcpy(EEEE, "0075") ;
         XSPACING = 4 ;
         break ;
    case 128:
    case 256:
    case 512:
    case 1024:
         strcpy(BBBB, "0025") ;
         strcpy(CCCC, "0050") ;
         strcpy(DDDD, "0075") ;
         strcpy(EEEE, "0100") ;
         XSPACING = 4 ;
         break ;
    default:
         printf("\n FATAL ERROR \n");
         exit(1) ;
}

if( NQ > 100 )
   XOFFSET = 0 ;
else
   XOFFSET = 100 ;
                    /* PRINT LABELS ON GRAPHIC SCREEN */
DRAW2_INIT(BBBB, CCCC, DDDD, EEEE, XAXISLABEL ) ;
imMAX = 0L ;
if(OUTDATA)
```

```
    {
      switch(OP)
      {
         case 1:
         case 2:
         case 5:
              imMAX = GETabsMAX(XQIMAG) ;
         case 3:
         case 4:
         case 6:
              reMAX = GETabsMAX(XQREAL) ;
              break ;
        default:
              printf("\n  FATAL ERROR \n");
              exit(1) ;
      }
    }
    else
    {
      switch(OP)
      {
         case 2:
         case 5:
         case 6:
              imMAX = GETabsMAX(XQIMAG) ;
         case 1:
         case 3:
         case 4:
              reMAX = GETabsMAX(XQREAL) ;
              break ;
        default:
              printf("\n  FATAL ERROR \n");
              exit(1) ;
      }
    }
    if( reMAX > imMAX )
       XDATAabsMAX  =  reMAX  ;
    else
       XDATAabsMAX  =  imMAX  ;
    _settextposition(6,25);
    _outtext("MAXIMUN ABSOLUTE  VALUE FOR MAGN =  ");
    ltoa(XDATAabsMAX, buffer, 10);
    _outtext(buffer);
    if( XDATAabsMAX >= 0  && XDATAabsMAX  < XRANGE1 ) CHOICE = 1     ;
    if( XDATAabsMAX >= XRANGE1  &&  XDATAabsMAX  < XRANGE2) CHOICE =
        2 ;
    if( XDATAabsMAX >= XRANGE2 )  CHOICE = 3                ;
    IMAGINARY = 0 ;
   if(OUTDATA)
   {
      switch(OP)
      {
```

```
        case 1:
        case 2:
        case 5:
             IMAGINARY = 1;

        case 3:
        case 4:
        case 6:
             SCALE_DATA(CHOICE, IMAGINARY) ;
             PLOThiN(GTITLE,XAXISLABEL,IMAGINARY, XSPACING, COLOR1,
COLOR2) ;
             break ;
        default:
             printf("\n FATAL ERROR \n");
             exit(1) ;
     }
   }
   else
   {
      switch(OP)
      {
        case 2:
        case 5:
        case 6:
             IMAGINARY = 1 ;
        case 1:
        case 3:
        case 4:
             SCALE_DATA(CHOICE, IMAGINARY) ;
             PLOThiN(GTITLE,XAXISLABEL,IMAGINARY, XSPACING, COLOR1,
COLOR2) ;
             break ;
        default:
             printf("\n FATAL ERROR \n");
             exit(1) ;
     }
   }
   _setvideomode(_DEFAULTMODE ) ;
   _clearscreen(_GCLEARSCREEN ) ;
} /*************** END OF GRAPH ***********/

 ***********************************************
 *      SCALE_DATA(CHOICE, IMAGINARY)          *
 * SCALE ARRAY OF LONG INTEGERS SO THAT        *
 * VALUES LIE WITHIN A DEFINED RANGE           *
 ***********************************************/
void SCALE_DATA( int CHOICE, int IMAGINARY)
{
  register int i ;
  long ptMAX, TEMP ;
  float ptRE, ptIM, ptM ;
  TEMP = XDATAabsMAX ;
```

```
    for(i = 1 ; i < NQ + 1 ; i++)
    {
       pointRE[i] = XQREAL[i] ;
       pointIM[i] = XQIMAG[i] ;
    }
    if( CHOICE == 3)
    {
       XDATAabsMAX = XDATAabsMAX / 10;
       for(i = 1; i < NQ + 1; i++)
          pointRE[i] = pointRE[i] / 10 ;
    }
    if( CHOICE == 3 || CHOICE == 2 )
    {
       XDATAabsMAX = XDATAabsMAX / 10000 ;
       for(i = 1; i < NQ + 1; i++)
          pointRE[i] = pointRE[i] / 10000 ;
    }

    ptM   = XDATAabsMAX ;
    ptM  /= 100.0       ;
    for(i = 1; i < NQ + 1; i++)
    {
       ptRE     = ( float ) pointRE[i] ;
       ptRE    /= ptM                  ;
       pointRE[i] = ( long ) ptRE      ;
    }
    if(IMAGINARY)
    {
       if( CHOICE == 3)
       {
          for(i = 1; i < NQ + 1; i++)
             pointIM[i] = pointIM[i] / 10 ;
       }
       if( CHOICE == 3 || CHOICE == 2 )
       {
          for(i = 1; i < NQ + 1; i++)
             pointIM[i] = pointIM[i] / 10000 ;
       }
       for(i = 1; i < NQ + 1; i++)
       {
          ptIM     = ( float ) pointIM[i] ;
          ptIM    /= ptM                  ;
          pointIM[i] = ( long ) ptIM      ;
       }
    }
    return ;
} /********** END OF SCALE_DATA **********/

/*****************************************************
* PLOTh(XQREAL,XQIMAGINARY,XSPACING,COLOR1,COLOR2)   *
*                                                    *
```

```
* IMAGINARY = 1 FOR AN ORDERED PAIR                    *
*                   OF DATA ELEMENTS                    *
*             = 0 FOR SINGLE INTEGER DATA               *
*                   ELEMENTS                            *
* COLOR1 AND COLOR2 ARE THE COLORS USED                 *
* FOR PLOTTING                                          *
*********************************************************/
void PLOThiN(char *GTITLE, char *XAXISLABEL, int IMAGINARY, int
XSPACING, short COLOR1, short COLOR2)
{
  int X, Y,  i                    ;
  int XSTART, XEND                ;
  int KEYIN, XDIR                 ;
  int START_TEMP, DIR_TEMP, UPDATE_FLAG ;
  int OLDSTART, OLDEND            ;
  char BBBB[10], CCCC[10], DDDD[10], EEEE[10]  ;

  XSTART  = 1                     ;
  if( NQ < 100 ) XEND  = NQ       ;
  else       XEND  = 125          ;
  X = XOFFSET                     ;
  _setcolor(COLOR1)               ;
  for(i = XSTART ; i < XEND + 1 ; i++)
  {
     X  +=  XSPACING              ;
     Y  = ( int ) pointRE[i]      ;
     Y  = - Y                     ;
     _moveto( X, 0 )              ;
     _lineto( X, Y )              ;
  }

  if(IMAGINARY)
  {
     X = XOFFSET                  ;
     _setcolor(COLOR2)            ;
     for(i =XSTART ; i < XEND + 1 ; i++)
     {
        X  +=  XSPACING           ;
        Y  = ( int ) pointIM[i]   ;
        Y  = - Y                  ;
        _moveto( X + 1, 0 )       ;
        _lineto( X + 1, Y )       ;
     }
  }
  OLDSTART = XSTART ;
  OLDEND  = XEND  ;
  if( NQ > 100 )
  {
     XDIR  = 1 ;
     for(;;)
     {
        KEYIN = (_bios_keybrd(_KEYBRD_READ) & 0xFF00) >> 8 ;
```

```
if(KEYIN == RIGHT )
{
   if( ( XEND + 25 ) < NQ )
   {
      UPDATE_FLAG  = 2     ;
      XEND       += 25     ;
      XSTART     += 25     ;
      START_TEMP  = XSTART ;
      XDIR       += 1      ;
      DIR_TEMP    = XDIR   ;
   }
   else
   {
      UPDATE_FLAG -= 1          ;
      XEND        = NQ          ;
      XSTART      = START_TEMP + 25 ;
      XDIR        = DIR_TEMP   + 1  ;
   }
} /* END IF KEYIN == RIGHT */
else if(KEYIN == LEFT )
{
   if( ( XSTART - 25 ) > 0 )
   {
      UPDATE_FLAG  = 2     ;
      XEND       -= 25     ;
      XSTART     -= 25     ;
      XDIR       -= 1      ;
   }
   else
   {
      UPDATE_FLAG -= 1     ;
      XEND        = 125    ;
      XSTART      = 1      ;
      XDIR        = 1      ;
   }
} /* END IF KEYIN == LEFT */
else if( KEYIN == ENTER )
   break ;
if( (KEYIN == RIGHT || KEYIN == LEFT) && UPDATE_FLAG > 0 )
{
   X = XOFFSET     ;
   _setcolor(9) ;

   for(i = OLDSTART ; i < OLDEND + 1 ; i++)
   {
      X += XSPACING          ;
      Y  = ( int ) pointRE[i]  ;
      Y  = - Y               ;
      _moveto(X,0)           ;
      _lineto(X,Y)           ;
   }
   if(IMAGINARY)
```

```
        {
          X = XOFFSET      ;
          for(i = OLDSTART ; i < OLDEND + 1 ; i++)
          {
            X  +=  XSPACING          ;
            Y  = ( int ) pointIM[i]  ;
            Y  = - Y                 ;
            _moveto(X + 1, 0)        ;
            _lineto(X + 1, Y)        ;
          }
        }
        OLDSTART = XSTART ;
        OLDEND  = XEND  ;
        DRAW1_INIT(GTITLE, COLOR1, COLOR2)                    ;
        strcpy(BBBB, XLABEL[XDIR] )                           ;
        strcpy(CCCC, XLABEL[XDIR + 1] )                       ;
        strcpy(DDDD, XLABEL[XDIR + 2] )                       ;
        strcpy(EEEE, XLABEL[XDIR + 3] )                       ;
        DRAW2_INIT(BBBB, CCCC, DDDD, EEEE, XAXISLABEL)   ;
        X = XOFFSET      ;
        _setcolor(COLOR1) ;
        for(i = XSTART  ; i < XEND + 1 ; i++)
        {
          X  +=  XSPACING          ;
          Y  = ( int ) pointRE[i]  ;
          Y  = - Y                 ;
          _moveto(X,0)             ;
          _lineto(X,Y)             ;
        }
        if(IMAGINARY)
        {
          X = XOFFSET      ;
          _setcolor(COLOR2) ;
          for(i = XSTART  ; i < XEND + 1 ; i++)
          {
            X  +=  XSPACING          ;
            Y  = ( int ) pointIM[i]  ;
            Y  = - Y                 ;
            _moveto(X + 1, 0)        ;
            _lineto(X + 1, Y)        ;
          }
        } /* END OF IF IMAGINARY */
      } /* END OF IF LEFT OR RIGHT */
    } /* END FOREVER */
  } /* END OF IF N > 100 */
  _settextposition(8,3);
  _settextcolor(14);
  _outtext("MENU");
  _settextposition(10,3);
  _settextcolor(12);
  _outtext("Q");
  _settextcolor(10);
```

```
        _outtext("UIT");
        _settextcolor(15) ;
        _settextposition(12,3);
        _settextcolor(12);
        _outtext("P");
        _settextcolor(10);
        _outtext("RINT");
        _settextcolor(15) ;
        for(;;)
        {
           KEYIN = _bios_keybrd(_KEYBRD_READ) & 0x00FF     ;
            if( KEYIN == 'Q' || KEYIN == 'q' )
              return                    ;
        } /* END OF SECOND FOREVER  */
    } /*************** END OF PLOThiN **********/

    void  DRAW1_INIT(char *GTITLE, short COLOR1, short COLOR2)
    {
      int i ;

      _setlogorg(115,190);
      _setcolor(10);
      for(i= 20 ; i < 512 ; i +=20)
      {
         _moveto(i,-100);
         _lineto(i,100);
         _moveto(i-1,-100);
         _lineto(i-1,100);
      }
      for(i= -100; i < 101 ; i +=20)
      {
         _moveto(1, i);
         _lineto(520, i);
         _moveto(1, i-1);
         _lineto(517, i-1);
      }
    }

    void DRAW2_INIT(char *BBBB, char *CCCC, char *DDDD, char *EEEE, char
    *XAXISLABEL)
    {
      _settextcolor(15) ;
      _settextposition(15,26);
      _outtext(BBBB);
      _settextposition(15,39);
      _outtext(CCCC);
      _settextposition(15,51);
      _outtext(DDDD);
      _settextposition(15,63);
      _outtext(EEEE);
      _settextposition(15,75);
      _outtext(XAXISLABEL);
```

```
    return ;
}


void SETFREQ(void)
{
    if (Fs > 20.0 )
    {
        if (Fs > 10000.0)
            Fs = 10000.0 ;
        outp(TIMERCNTL, 0x01);
        outp(TIMERDATA, 0x25);
        outp(TIMERDATA, 0x0B);
        COUNTVAL = (unsigned short)( (1000000.0 / Fs) - 1.0);
    }
    else if( Fs > 1.9999 )
    {
        outp(TIMERCNTL, 0x01);
        outp(TIMERDATA, 0x25);
        outp(TIMERDATA, 0x0C);
        COUNTVAL = (unsigned short)( (100000.0 / Fs) - 1.0);
    }
    else if( Fs > 0.19999)
    {
        outp(TIMERCNTL, 0x01);
        outp(TIMERDATA, 0x25);
        outp(TIMERDATA, 0x0D);
        COUNTVAL = (unsigned short)( (10000.0 / Fs) - 1.0);
    }
    else if( Fs > 0.019999)
    {
        outp(TIMERCNTL, 0x01);
        outp(TIMERDATA, 0x25);
        outp(TIMERDATA, 0x0E);
        COUNTVAL = (unsigned short)((1000.0 / Fs) - 1.0);
    }
    else if( Fs > 0.0019999)
    {
        outp(TIMERCNTL, 0x01);
        outp(TIMERDATA, 0x25);
        outp(TIMERDATA, 0x0F);
        COUNTVAL = (unsigned short)( (100.0 / Fs) - 1.0);
    }

    else if( Fs < 0.0019999)
    {
        outp(TIMERCNTL, 0x01);
        outp(TIMERDATA, 0x25);
        outp(TIMERDATA, 0x0F);
        COUNTVAL = 0xFFFF ;
    }
```

```
    return ;
}

void START_TIMER(void)
{

    TECDATA.WORD = COUNTVAL ;

    outp(TIMERINTACK, 0)          ; /*  CLEAR TIMER INTERRUPT */
    outp(TIMERCNTL, 0x09)         ; /* SELECT LOAD REGISTER 1 */
    outp(TIMERDATA, TECDATA.BYTE.LO)  ; /* PUT COUNTVAL VALUE IN */
    outp(TIMERDATA, TECDATA.BYTE.HI)  ; /*  LOAD REGISTER 1      */
    outp(TIMERCNTL, 0x61)         ; /* LOAD AND ARM COUNTER 1      */
    outp(32,0xE1)                 ; /* Set IRQ2 to highest priority  */
    outp(33,0xF9)                 ; /* Mask off other interrupts     */
    outp(32,98)                   ; /* Clear IBM interrupt mechanism */

    DATACOUNT = 1 ;

    return ;

}

void STOP_TIMER(void)
{
    outp(TIMERCNTL, 0xC1) ;   /* DISARM COUNTER 1 */
    outp(32,0xE7)         ;   /* Resume normal priority setting */
    outp(33,0)            ;   /* Allow all types of interrupt */
    outp(TIMERINTACK, 0) ;    /* Clear TECMAR interrupt */

    return ;
}

void  READADC(void)
{

    outp(TIMERCNTL, 0x09)          ;
    outp(TIMERDATA, TECDATA.BYTE.LO) ;
    outp(TIMERDATA, TECDATA.BYTE.HI) ;
    outp(TIMERCNTL, 0x61)          ;
    outp(TIMERINTACK , 0)          ;
    outp(MUXADDR, CHANNEL)         ;
    outp(SOCADDR, SOC)             ;

    for(;;)
    {
        TECSTATUS.SWHOLE = (unsigned char ) inp(STATUSADDR) ;
        if(TECSTATUS.SFIELD.EOC = 1)
            break ;
    }

    TECDATA.BYTE.LO =  inp(ADCLSB)    ;
```

```
    TECDATA.BYTE.HI = inp(ADCMSB)    ;

    if(DATACOUNT < NQ + 1)
    {
       XQREAL[ DATACOUNT ] = ( long ) TECDATA.WORD ;
       XQIMAG[ DATACOUNT ] = 0L  ;
       DATACOUNT += 1 ;
    }

}

void INIT_TECMAR(void)
{

   outp(32,98);
   outp(TIMERCNTL,      193);   /* Disarm counter #1 of 9513 timer */
   outp(TIMERINTACK,      0);   /* Clear TECMAR interrupt          */
   outp(CNTLADDR,       144);   /* TECMAR control byte */
   outp(MUXADDR,    CHANNEL);   /* A/D channel #0 */
   outp(TIMERCNTL,     0xFF);   /* master reset 9513 */
   outp(TIMERCNTL,     0xE8);   /* disable data pointer auto sequencing */
   outp(TIMERCNTL,     0x17);   /* select master mode register */
   outp(TIMERDATA,     0x10);   /* low byte of master mode register */
   outp(TIMERDATA,     0xd1);   /* high byte of master mode register */
   outp(TIMERCNTL,     0x09);   /* select counter 1's load register */
   outp(32,0x20)            ;

}

void  NEWFREQ(void)
{
   char sbuf[80] ;

   _setvideomode( _TEXTC80 ) ;
   system("cls") ;

   printf("\n    SAMPLING FREQUENCY ( HERTZ ) :  ") ;

   Fs = (double) atof(gets(sbuf)) ;

   SETFREQ() ;

   return ;

}

void  NEWADCCHNL(void)
{

   char sbuf[80] ;

   _setvideomode( _TEXTC80 ) ;
```

```
    system("cls") ;

    printf("\n    ADC CHANNEL ADRESS  : ") ;

    CHANNEL = (unsigned char) ( (atoi(gets(sbuf))) & 0x00FF ) ;

    return ;
}

void  ACQUIRE(void)
{

   char sbuf[80] ;

   _setvideomode( _TEXTC80 ) ;
   system("cls") ;

   printf("\n    N =  : ") ;

   NQ = atoi(gets(sbuf))       ;

   INIT_TECMAR() ;
   START_TIMER() ;
   while(DATACOUNT < NQ + 1) ;
   STOP_TIMER() ;

   return ;
}
```

D. COPROCESSOR SOFTWARE

```
  PAGE 60,130
  TITLE  ZFT
CD_TEXT SEGMENT WORD 'CODE'
CD_TEXT ENDS
ZFT_TEXT  SEGMENT  PAGE 'CODE'
ZFT_TEXT  ENDS
_DATA  SEGMENT  WORD  PUBLIC  'DATA'
_DATA  ENDS
CONST  SEGMENT  WORD  PUBLIC  'CONST'
CONST  ENDS
_BSS  SEGMENT  WORD  PUBLIC  'BSS'
_BSS  ENDS

DGROUP GROUP  CONST,  _BSS,  _DATA

     PUBLIC  _N, _NQ
     PUBLIC  _M, _MQ
     PUBLIC  _SIGN
     PUBLIC  _XREAL, _XQREAL
     PUBLIC  _XIMAG, _XQIMAG
     PUBLIC  _QTABLE, _VTABLE
     PUBLIC  _XN, _XQN
     PUBLIC  _YN, _YQN
     PUBLIC  _NTOTAL, _NQTOTAL
     PUBLIC  _NTIMES, _NQTIMES
     PUBLIC _pointRE
     PUBLIC _pointIM
     PUBLIC  _VEC15SAV, _INPORT, _OUTPORT
     PUBLIC _INTVEC02, _NMIFLAG, _NMICNT, _BANKID
     PUBLIC _INTVEC00, _INTVEC04, _INTVEC01, _INTVEC03
     PUBLIC _QIRQ7FLAG , _VIRQ7FLAG
     PUBLIC _QIRQ7CNT  , _VIRQ7CNT
     PUBLIC _QOPCODE   , _VOPCODE
     PUBLIC _QOPSTATUS , _VOPSTATUS
     PUBLIC _QBANKNO   , _VBANKNO
     PUBLIC _IRQ7FLAG
     PUBLIC _IRQ7CNT
     PUBLIC _OPCODE
     PUBLIC _OPSTATUS
     PUBLIC _BANKNO


DATAPORT        EQU  340H
CLRINTPORT      EQU  341H
RQINTPORT       EQU  342H
RESETPORT       EQU  343H
INTA00          EQU  020H
EOI             EQU  020H
INTPRESENT      EQU  05A5AH
```

```
STATUSOK        EQU  0A5A5H
A_BANK          EQU  0000H
B_BANK          EQU  0FFFFH
ZERODIVERR      EQU  4444H
OVFLERR         EQU  6666H
FATERR          EQU  3333H

S0000 SEGMENT AT 0000H
      ORG 0000H
_INTVEC00       DW    0          ; INTVEC00 IP DIV BY 0
                DW    0          ; INTVEC00 CS
_INTVEC01       DW    0          ; INTVEC IP
                DW    0          ; INTVEC CS
_INTVEC02       DW    0          ; INTVEC02 IP NMI VECTOR
                DW    0          ; INTVEC02 CS
_INTVEC03       DW    0          ; INTVEC IP
                DW    0          ; INTVEC CS
_INTVEC04       DW    0          ; INTVEC04 IP OVERFLOW
                                   VECTOR
                DW    0          ; INTVEC04 CS
                ORG   0400H
_ARITHERR       DW    0            ; ARITHMETIC ERROR FLAG
_NMIFLAG        DW    0            ; NMI FLAG
_NMICNT         DW    0            ; NIM COUNT
_BANKID         DW    0            ; BANK ID NUMBER
S0000 ENDS
S9000 SEGMENT  AT 9000H
_QTABLE         DW  1025 DUP(0)    ; SINE TABLE
_XXQ            DW  2  DUP(0)
_YYQ            DW  2  DUP(0)
_ZZQ            DW  4  DUP(0)
FLAGQ           DW    0
_NQ             DW    0
_MQ             DW    0            ; NUMBER OF DATA POINTS
_SIGNQ          DW    0
UREALQ          DW    0
UIMAGQ          DW    0
TREALQ          DW  2  DUP(0)
TIMAGQ          DW  2  DUP(0)
IQ              DW    0
JQ              DW    0
KQ              DW    0
QLQ             DW    0
LLEQ            DW    0
LE1Q            DW    0
IPQ             DW    0
STEPQ           DW    0
_XQREAL         DW  2056 DUP(0)    ; REAL PART OF DATA
_XQIMAG         DW  2056 DUP(0)    ; IMAGINARY PART OF DATA
NQPLUS2         DW    0
NQOVER2         DW    0
_AAXQ           DW    0
```

```
_BBXQ           DW    0
_CCYQ           DW    0
_DDYQ           DW    0
                DD    258 DUP(0)
_XQN            DD    1026 DUP(0)       ; FILTER INPUT DATA
                DD    34 DUP(0)
_YQN            DD    1026 DUP(0)       ; FILTER OUTPUT DATA
_NQTIMES        DW    0
_NQTOTAL        DW    0
_NQCOUNT        DW    0
_NQLOOP         DW    0
_NQSAVE         DW    0
_FQIRADDR       DW    2 DUP(0)
_QIRQ7FLAG      DW    0
_QIRQ7CNT       DW    0
_QOPCODE        DW    0
_QOPSTATUS      DW    0
_QBANKNO        DW    0
S9000   ENDS
SF800   SEGMENT   AT   0F800H
TABLE           DW   1025 DUP(0)
_XX             DW   2   DUP(0)
_YY             DW   2   DUP(0)
_ZZ             DW   4   DUP(0)
FLAG            DW    0
_N              DW    0
_M              DW    0
_SIGN           DW    0
UREAL           DW    0
UIMAG           DW    0
TREAL           DW   2   DUP(0)
TIMAG           DW   2   DUP(0)
I               DW    0
J               DW    0
K               DW    0
QL              DW    0
LLE             DW    0
LE1             DW    0
IP              DW    0
STEP            DW    0
_XREAL          DW   2056 DUP(0)
_XIMAG          DW   2056 DUP(0)
NPLUS2          DW    0
NOVER2          DW    0
_AAX            DW    0
_BBX            DW    0
_CCY            DW    0
_DDY            DW    0
                DD    258 DUP(0)
_XN             DD    1026 DUP(0)
                DD    34 DUP(0)
_YN             DD    1026 DUP(0)
```

```
_NTIMES         DW    0
_NTOTAL         DW    0
_NCOUNT         DW    0
_NLOOP          DW    0
_NSAVE          DW    0
_FFIRADDR       DW  2 DUP(0)
_IRQ7FLAG       DW    0
_IRQ7CNT        DW    0
_OPCODE         DW    0
_OPSTATUS       DW    0
_BANKNO         DW    0
SF800   ENDS
CD_DATA SEGMENT WORD   'FARDATA'

.XLIST
_VTABLE     DW        0          ;     0.000000
;  SINE TABLE GOES HERE
 .LIST

_pointRE        DD  1026 DUP(0)    ; REAL PART OF GRAPH DATA
_pointIM        DD  1026 DUP(0)    ; IMAGINARY PART OF GRAPH DATA
_VEC15SAV       DW    0
                DW    0
_INPORT         DW    0
_OUTPORT        DW    0
_VIRQ7FLAG      DW     0
_VIRQ7CNT       DW     0
_VOPCODE        DW     0
_VOPSTATUS      DW     0
_VBANKNO        DW     0
CD_DATA ENDS

    ASSUME  CS: CD_TEXT, DS: DGROUP, SS: DGROUP

CD_TEXT    SEGMENT
    ASSUME  CS: CD_TEXT
;  _ COP LOADS COPROCESSOR CODE INTO SHARED MEMORY

    PUBLIC _COP
_COP    PROC FAR
     push   bp
     mov    bp,sp

     push    ds
     push    es
     push    di
     push    si
     push    cx

    mov  ax, SEG _START
    mov  ds,ax
    mov  si,OFFSET _START
```

```
        mov  ax,9000h
        mov  es,ax
        mov  di,0700h
        mov  cx,_BEGINA - _START + 6
        cld
        cli
 rep    movsb
        sti

        mov  ax, SEG _BEGINA
        mov  ds,ax
        mov  si,OFFSET _BEGINA
        mov  ax,9000h
        mov  es,ax
        mov  di,7F00h
        mov  cx, _RESET - _BEGINA + 4
        cld
        cli
 rep    movsb
        sti

        mov  ax, SEG _RESET
        mov  ds,ax
        mov  si,OFFSET _RESET
        mov  ax,9000H
        mov  es,ax
        mov  di,7FF0H
        mov  cx,17
        cld
        cli
 rep    movsb
        sti

        pop     cx
        pop     si
        pop     di
        pop     es
        pop     ds

       mov     sp,bp
       pop     bp
       ret
_COP   ENDP

; _RES PULLS COPROCESSOR OUT OF RESET STATE
       PUBLIC _RES
_RES   PROC FAR
       push    bp
       mov     bp,sp

       push    dx
       push    cx
```

```
        mov   dx,RESETPORT
        mov   al,1
        out   dx,al
        mov   cx,0FFFFH
DLY1:
        or    ax,ax
        loop  DLY1
        pop   cx
        pop   dx

        mov   sp,bp
        pop   bp
        ret
_RES    ENDP

; _NMI SEND AN NMI SIGNAL TO THE COPROCESSOR

        PUBLIC _NMIRQST
_NMIRQST PROC FAR
        push  bp
        mov   bp,sp

        push  dx
        push  cx
        mov   dx,RQINTPORT
        mov   al,1
        out   dx,al
        pop   cx
        pop   dx

        mov   sp,bp
        pop   bp
        ret
_NMIRQST  ENDP

; _IRQ2SER IS THE INTERRUPT SERVICE ROUTINE
; FOR THE TIMER ON THE TECMAR BOARD
        PUBLIC _IRQ2SER
_IRQ2SER    PROC FAR
        push  bp
        mov   bp,sp

        push  dx
        mov   dx,CLRINTPORT
        mov   al,0
        out   dx,al
        sti

        push  es
        push  bx
        mov   ax,CD_DATA
        mov   es,ax
```

```
        mov  WORD PTR es:_VIRQ7FLAG, INTPRESENT
        inc  WORD PTR es:_VIRQ7CNT
        cli

        mov   al,EOI
        out   INTA00,al
        pop   bx
        pop   es
        pop   dx

        mov   sp,bp
        pop   bp
        iret

IRQ7SER     ENDP

; _PUTIRQ2VEC INSTALLS THE INTERRUPT SERVICE ROUTINE
;  FOR THE TIMER ON THE TECMAR BOARD
        PUBLIC _PUTIRQ2VEC
        ASSUME  DS:CD_DATA
PUTIRQ2VEC    PROC FAR
        push  bp
        mov   bp,sp

        push  ds
        push  es
        push  bx

        mov   dx,OFFSET _IRQ2SER
        mov   ax,SEG _IRQ2SER
        mov   ds,ax
        mov   ah,25H    ; SET INTERRUPT FUNCTION CODE
        mov   al,10     ; IRQ7 MAPS ONTO INT TYPE 10
        int   21h
        pop   bx
        pop   es
        pop   ds

        mov   sp,bp
        pop   bp
        ret
        nop
_PUTIRQ2VEC    ENDP

CD_TEXT ENDS
        ASSUME  CS:ZFT_TEXT, DS:SF800,  SS:S0000 , ES:S0000
ZFT_TEXT   SEGMENT
        PUBLIC _START, _BEGINA, _RESET
        PUBLIC _CRD
_CRD   PROC FAR
; _CRD IS THE MAIN PROGRAM FOR THE COPROCESSOR BOARD
        ORG 0700H
```

```
_START:
        mov   ax,0F800h
        mov   ds,ax
        mov   ax,0
        mov   es,ax
        mov   ss,ax
        mov   sp,1FFEh
        mov   WORD PTR  es:_NMIFLAG,0
        mov   WORD PTR  es:_NMICNT,0
        mov   WORD PTR  es:_BANKID, A_BANK
        mov   WORD PTR  es:_ARITHERR, 0
        mov   WORD PTR  _IRQ7FLAG,0
        mov   WORD PTR  _IRQ7CNT,0
        mov   WORD PTR  _OPCODE,0
        mov   WORD PTR  _OPSTATUS,0
        mov   WORD PTR  _BANKNO, A_BANK
        mov   WORD PTR  es:_INTVEC00, OFFSET _DIVBY0ERR
        mov   WORD PTR  es:_INTVEC00 + 2 , 0
        mov   WORD PTR  es:_INTVEC02, OFFSET _NMISER
        mov   WORD PTR  es:_INTVEC02 + 2 , 0
        mov   WORD PTR  es:_INTVEC04, OFFSET _OVERFLOW
        mov   WORD PTR  es:_INTVEC04 + 2 , 0
NOACTION:
        mov   ax, WORD PTR es:_NMIFLAG
        cmp   ax, INTPRESENT
        je    ACTION
        jmp   NOACTION
ACTION:
        mov   WORD PTR es:_NMIFLAG,0
        call NEAR PTR _BANKA
        mov   ax, WORD PTR es:_NMICNT
        mov   WORD PTR es:_NMICNT, 0
        cmp   ax, 1
        jne   FATALERR
        mov   bx, _OPCODE
        call  WORD PTR cs:[bx + OFFSET _FUNCTION]
        cmp   WORD PTR es:_ARITHERR, 0
        jne   JOIN2
        mov   WORD PTR _OPSTATUS, STATUSOK
        jmp   SHORT JOIN3
JOIN2:
        mov   ax, WORD PTR es:_ARITHERR
        mov   WORD PTR _OPSTATUS, ax
        mov   WORD PTR es:_ARITHERR, 0
        jmp   SHORT JOIN3
FATALERR:
        mov   WORD PTR _OPSTATUS, FATERR
        mov   WORD PTR es:_ARITHERR, 0
JOIN3:
NOACTN:
        mov   ax, WORD PTR es:_NMIFLAG
        cmp   ax, INTPRESENT
```

```
    je   ACTN
    jmp  NOACTN
ACTN:
    mov  WORD PTR es:_NMIFLAG,0
    call NEAR PTR _BANKB
    mov  ax, WORD PTR es:_NMICNT
    mov  WORD PTR es:_NMICNT, 0
    cmp  ax, 1
    jne  FTLERR
    mov  bx, _OPCODE
    call WORD PTR cs:[bx + OFFSET _FUNCTION]
    cmp  WORD PTR es:_ARITHERR, 0
    jne  JOIN4
    mov  WORD PTR _OPSTATUS, STATUSOK
    jmp  SHORT JOIN5
JOIN4:
    mov  ax, WORD PTR es:_ARITHERR
    mov  WORD PTR _OPSTATUS, ax
    mov  WORD PTR es:_ARITHERR, 0
    jmp  SHORT JOIN5
FTLERR:
    mov  WORD PTR _OPSTATUS, FATERR
    mov  WORD PTR es:_ARITHERR, 0
JOIN5:
    jmp NOACTION
_FUNCTION   DW  OFFSET  DONOTHING
        DW  OFFSET  _DFFT
        DW  OFFSET  _INVFFT
        DW  OFFSET  _PGRAM
        DW  OFFSET  _AUTOCORRL
        DW  OFFSET  _CROSSCORRL
        DW  OFFSET  _LCONVL
        DW  OFFSET  _PCONVL
        DW  OFFSET  _FFIRLP
        DW  OFFSET  _FFIRBP
        DW  OFFSET  _FFIRHP
        DW  OFFSET  _IIRLPF
        DW  OFFSET  _IIRBPF
        DW  OFFSET  _IIRHPF
_CRD   ENDP

; _ BANKA PUTS BANK A IN THE MEMORY ADDRESS SPACE OF THE IBM PC
    PUBLIC _BANKA
_BANKA  PROC  NEAR
    push  es
    push  bx
    mov  bx,4000h
    mov  es,bx
    mov  bx,0
    mov  BYTE PTR es:[bx],0
    pop  bx
    pop  es
```

```
        ret
_BANKA  ENDP

; _ BANKB PUTS BANK B IN THE MEMORY ADDRESS SPACE OF THE IBM PC
        PUBLIC _BANKB
_BANKB  PROC  NEAR
        push  es
        push  bx
        mov   bx,8000h
        mov   es,bx
        mov   bx,0
        mov   BYTE PTR es:[bx],0
        pop   bx
        pop   es
        ret
_BANKB  ENDP

; _ NMISER IS THE NMI HANDLER FOR THE COPROCESSOR
        PUBLIC _NMISER
_NMISER PROC NEAR
        mov   al,0
        out   01h,al
        mov  WORD PTR es:_NMIFLAG, INTPRESENT
        inc  WORD PTR es:_NMICNT
        iret
_NMISER ENDP
; DONOTHING IS A NO OPERATION FUNCTION FOR THE COPROCESSOR
DONOTHING  PROC  NEAR
        nop
        nop
        nop
        nop
        ret
DONOTHING  ENDP
; _ DIVBY0ERR IS THE HANDLER FOR THE DIVIDE BY ZERO EXCEPTION
        PUBLIC  _DIVBY0ERR
_DIVBY0ERR    PROC  NEAR
            mov  WORD PTR es:_ARITHERR, ZERODIVERR
            iret
_DIVBY0ERR    ENDP
; _ OVERFLOW IS THE OVERFLOW EXCEPTION HANDLER
;  FOR THE COPROCESSOR
        PUBLIC _OVERFLOW
_OVERFLOW   PROC  NEAR
        mov  WORD PTR es:_ARITHERR, OVFLERR
        iret
_OVERFLOW  ENDP
; DIRECT FOPURIER TRANSFORM ROUTINE
        PUBLIC   _DFFT
_DFFT       PROC    NEAR
        mov     _SIGN,-1
        call    NEAR PTR _FFT
```

```
        ret
_DFFT   ENDP
; INVERSE FOURIER TRANSFORM ROUTINE
        PUBLIC  _INVFFT
_INVFFT PROC    NEAR
        mov     _SIGN,1
        call    NEAR PTR _ZFFT
        call    NEAR PTR NDIV
        ret
_INVFFT ENDP
; CORRELATION FUNCTION ROUTINE
        PUBLIC  _CORR
_CORR   PROC    NEAR
        call NEAR PTR PAD0
        mov     _SIGN,-1
        call    NEAR PTR _ZFFT
        mov     FLAG,0
        call    NEAR PTR SORT
        call    NEAR PTR NDIV
        mov     _SIGN,1
        call    NEAR PTR _ZFFT
        ret
_CORR   ENDP
; POWER SPECTRUM ROUTINE
        PUBLIC  _PGRAM
_PGRAM  PROC    NEAR
        mov     _SIGN,-1
        call    NEAR PTR _ZFFT
        mov     FLAG,0
        call    NEAR PTR SORT
        ret
_PGRAM  ENDP
; LINEAR CONVOLUTION ROUTINE
        PUBLIC  _LCONVL
_LCONVL PROC    NEAR
        call NEAR PTR PAD0
        mov     _SIGN,-1
        call    NEAR PTR _ZFFT
        mov     FLAG,-1
        call NEAR PTR SORT
        call NEAR PTR NDIV
        mov     _SIGN,1
        call    NEAR PTR _ZFFT
        ret
_LCONVL ENDP
; CYCLIC CONVOLUTION ROUTINE
        PUBLIC  _PCONVL
_PCONVL PROC    NEAR
        mov     _SIGN,-1
        call    NEAR PTR _ZFFT
        mov     FLAG,-1
        call NEAR PTR SORT
```

```
        call  NEAR PTR NDIV
        mov     _SIGN,1
        call    NEAR PTR _ZFFT
        ret
_PCONVL   ENDP
; AUTOCORRELATION ROUTINE
        PUBLIC _AUTOCORRL
_AUTOCORRL  PROC NEAR
          call  NEAR PTR _CORR
          ret
_AUTOCORRL  ENDP
; CROSSCORRELATION ROUTINE
        PUBLIC _CROSSCORRL
_CROSSCORRL  PROC NEAR
          call  NEAR PTR _CORR
          ret
_CROSSCORRL  ENDP


        PUBLIC _FFIRLP
_FFIRLP     PROC NEAR
          nop
          nop
          nop
          nop
          nop
          nop
          nop
          nop
          nop
          nop
          nop
          nop
          nop
          nop
          nop
          ret
_FFIRLP     ENDP


        PUBLIC _FFIRBP
_FFIRBP     PROC NEAR
          nop
          nop
          nop
          nop
          nop
          nop
          nop
          nop
          nop
          nop
```

```
        nop
        nop
        nop
        nop
        nop
        ret
_FFIRBP     ENDP

        PUBLIC _FFIRHP
_FFIRHP     PROC NEAR
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        ret
_FFIRHP     ENDP

        PUBLIC _IIRLPF
_IIRLPF     PROC NEAR
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        nop
        ret
_IIRLPF     ENDP

        PUBLIC _IIRBPF
```

```
_IIRBPF     PROC NEAR
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            ret
_IIRBPF     ENDP

            PUBLIC _IIRHPF
_IIRHPF     PROC NEAR
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            nop
            ret
_IIRHPF     ENDP

FAST FOURIER TRANSFORM ROUTINE
_ZFFT   PROC    NEAR
        push    bp
        mov     bp,sp
        push    bx
        push    cx
        push    si
        push    di
        mov     J,1
        mov     I,1
; DATA RE-ORDERING
FOR1:
```

```
        mov     ax,I
        cmp     ax,_N
        jl      IFF1
        jmp     ENDFOR1
IFF1:
        cmp     ax,J
        jl      CONT1
        jmp     ENDIF1
CONT1:
        mov     si,J
        shl     si,1
        shl     si,1
        mov     di,I
        shl     di,1
        shl     di,1
        mov     ax,_XREAL[si]
        mov     TREAL,ax
        mov     ax,_XREAL[si+2]
        mov     TREAL+2,ax
        mov     ax,_XIMAG[si]
        mov     TIMAG,ax
        mov     ax,_XIMAG[si+2]
        mov     TIMAG+2,ax
        mov     ax,_XREAL[di]
        mov     _XREAL[si],ax
        mov     ax,_XREAL[di+2]
        mov     _XREAL[si+2],ax
        mov     ax,_XIMAG[di]
        mov     _XIMAG[si],ax
        mov     ax,_XIMAG[di+2]
        mov     _XIMAG[si+2],ax
        mov     ax,TREAL
        mov     _XREAL[di],ax
        mov     ax,TREAL+2
        mov     _XREAL[di+2],ax
        mov     ax,TIMAG
        mov     _XIMAG[di],ax
        mov     ax,TIMAG+2
        mov     _XIMAG[di+2],ax
ENDIF1:
        mov     ax,_N
        cwd
        mov     cx,2
        idiv    cx
        mov     K,ax
        mov     bx,J
WHILE1:
        cmp     ax,bx
        jge     ENDWHILE1
        sub     bx,ax
        cwd
        idiv    cx
```

```
        jmp     WHILE1
ENDWHILE1:
        add     bx,ax
        mov     J,bx
        mov     K,ax
        inc     I
        jmp     FOR1
ENDFOR1:
; COMPUTATIONS
        mov     QL,1
        mov     LLE,1
FOR2:
        mov     ax,LLE
        mov     LE1,ax
        shl     LLE,1
        mov     UREAL,16384
        mov     UIMAG,0
        mov     ax,1024
        cwd
        idiv    LLE
        mov     STEP,ax
        mov     J,1
FOR3:
        mov     ax,J
        mov     I,ax
FOR4:
        add     ax,LE1
        mov     IP,ax
        mov     si,ax
        shl     si,1
        shl     si,1
        mov     di,I
        shl     di,1
        shl     di,1
        mov     ax,_XIMAG[si]
        mov     _XX,ax
        mov     ax,_XIMAG[si+2]
        mov     _XX+2,ax
        mov     ax,UIMAG
        mov     _YY,ax
        call    NEAR PTR _MFT
        mov     ax,_ZZ
        mov     TREAL,ax
        mov     ax,_ZZ+2
        mov     TREAL+2,ax
        mov     ax,_XIMAG[si]
        mov     _XX,ax
        mov     ax,_XIMAG[si+2]
        mov     _XX+2,ax
        mov     ax,UREAL
        mov     _YY,ax
        call    NEAR PTR _MFT
```

```
mov     ax,_ZZ
mov     TIMAG,ax
mov     ax,_ZZ+2
mov     TIMAG+2,ax
mov     ax,_XREAL[si]
mov     _XX,ax
mov     dx,_XREAL[si+2]
mov     _XX+2,ax
mov     ax,UREAL
mov     _YY,ax
call    NEAR PTR _MFT
mov     ax,_ZZ
mov     dx,_ZZ+2
sub     ax,TREAL
sbb     dx,TREAL+2
mov     TREAL,ax
mov     TREAL+2,dx
mov     ax,_XREAL[si]
mov     _XX,ax
mov     ax,_XREAL[si+2]
mov     _XX+2,ax
mov     ax,UIMAG
mov     _YY,ax
call    NEAR PTR _MFT
mov     ax,_ZZ
mov     dx,_ZZ+2
add     ax,TIMAG
adc     dx,TIMAG+2
mov     TIMAG,ax
mov     TIMAG+2,dx
mov     ax,_XREAL[di]
mov     dx,_XREAL[di+2]
sub     ax,TREAL
sbb     dx,TREAL+2
mov     _XREAL[si],ax
mov     _XREAL[si+2],dx
mov     ax,_XIMAG[di]
mov     dx,_XIMAG[di+2]
sub     ax,TIMAG
sbb     dx,TIMAG+2
mov     _XIMAG[si],ax
mov     _XIMAG[si+2],dx
mov     ax,_XREAL[di]
mov     dx,_XREAL[di+2]
add     ax,TREAL
adc     dx,TREAL+2
mov     _XREAL[di],ax
mov     _XREAL[di+2],dx
mov     ax,_XIMAG[di]
mov     dx,_XIMAG[di+2]
add     ax,TIMAG
adc     dx,TIMAG+2
```

```
        mov     _XIMAG[di],ax
        mov     _XIMAG[di+2],dx
        mov     ax,I
        add     ax,LLE
        mov     I,ax
        cmp     ax,_N
        jg      ENDFOR4
        jmp     FOR4
ENDFOR4:
        mov     ax,STEP
        cwd
        imul    J
        mov     si,ax           ;  sine index
        add     ax,256
        mov     di,ax           ;  cosine index
        shl     di,1
        shl     si,1
        mov     ax,TABLE[di]
        mov     UREAL,ax
        mov     ax,TABLE[si]
        mov     UIMAG,ax
        cmp     _SIGN,0
        jg      NOCHGE
        neg     UIMAG
NOCHGE:
        mov     ax,J
        add     ax,1
        mov     J,ax
        cmp     ax,LE1
        jg      ENDFOR3
        jmp     FOR3
ENDFOR3:
        mov     ax,QL
        add     ax,1
        mov     QL,ax
        cmp     ax,_M
        jg      ENDFOR2
        jmp     FOR2
ENDFOR2:
        mov     ax,1
        pop     di
        pop     si
        pop     dx
        pop     cx
        pop     bx
        mov     sp,bp
        pop     bp
        ret
_ZFFT   ENDP

_MFT    PROC    NEAR
        push    bp
```

```
        mov     bp,sp
        push    si
        push    di
        push    dx
        push    cx
        push    bx
        cmp     _XX+2,0
        jnz     B00
        cmp     _XX,0
        jnz     B00
        jmp     ZERO
B00:
        cmp     _YY,0
        jnz     NZERO
ZERO:
        mov     _ZZ,0
        mov     _ZZ+2,0
        jmp     FIN
NZERO:
        mov     ax,_XX+2
        or      ax,_XX+2
        js      C1X
;                       XX  POSITIVE , YY DON'T CARE
        mov     ax,_YY
        or      ax,_YY
        js      C01
;                       XX POSITIVE , YY POSITIVE
C00:
        call    NEAR PTR M32
        jmp     FIN
;                       XX POSITIVE, YY NEGATIVE
C01:
        neg     _YY
        call    NEAR PTR M32
        call    NEAR PTR NZZ
        jmp     FIN
C1X:
        call    NEAR PTR NXX
        mov     ax,_YY
        or      ax,_YY
        js      C11
;                       XX NEGATIVE , YY POSITIVE
C10:
        call    NEAR PTR M32
        call    NEAR PTR NZZ
        jmp     FIN
;                       XX NEGATIVE, YY NEGATIVE
C11:
        neg     _YY
        call    NEAR PTR M32
FIN:
        pop     bx
```

```
        pop     cx
        pop     dx
        pop     di
        pop     si
; Line 3
        mov     sp,bp
        pop     bp
        ret
_MFT    ENDP

NXX     PROC    NEAR
        push    bp
        mov     bp,sp
        push    dx
        mov     ax,_XX
        mov     dx,_XX+2
        not     ax
        not     dx
        add     ax,1
        adc     dx,0
        mov     _XX,ax
        mov     _XX+2,dx
        pop     dx
        mov     sp,bp
        pop     bp
        ret
NXX     ENDP

NZZ     PROC    NEAR
        push    bp
        mov     bp,sp
        push    dx
        mov     ax,_ZZ
        mov     dx,_ZZ+2
        not     ax
        not     dx
        add     ax,1
        adc     dx,0
        mov     _ZZ,ax
        mov     _ZZ+2,dx
        pop     dx
        mov     sp,bp
        pop     bp
        ret
NZZ     ENDP

M32     PROC    NEAR
        push    bp
        mov     bp,sp
        mov     ax,_XX
        mul     _YY             ; XX(LSW) * YY
        mov     bx,ax           ; ALFA 0 IN BX
```

```
        mov     cx,dx
        mov     ax,_XX+2
        mul     _YY             ; XX(MSW) * YY
        add     ax,cx           ; ALFA 1 IN AX
        adc     dx,0            ; ALFA 2 IN DX
        mov     _ZZ,bx
        mov     _ZZ+2,ax
        mov     _ZZ+4,dx
        call    NEAR PTR AJST
        mov     sp,bp
        pop     bp
        ret
M32     ENDP

AJST    PROC    NEAR
        push    bp
        mov     bp,sp
        push    bx
        mov     bx,16384
        mov     ax,_ZZ          ; ALFA 0
        mov     dx,0
        div     bx              ; ALFA 0 / 16384
        mov     _ZZ,ax
        mov     ax,_ZZ+2
        mov     bx,4
        mul     bx              ; 4 x ALFA 1
        add     ax,_ZZ
        adc     dx,0
        mov     _ZZ,ax          ; L SIG WORD FINAL ANS.
        mov     _ZZ+2,dx
        mov     ax,_ZZ+4
        mul     bx              ; 4 x ALFA 2
        add     ax,_ZZ+2
        mov     _ZZ+2,ax        ; M SIG WORD FINAL ANS.
        pop     bx
        mov     sp,bp
        pop     bp
        ret
AJST    ENDP

SORT    PROC    NEAR
        push    bp
        mov     bp,sp
        push    si
        push    di
        push    dx
        push    cx
        push    bx
        mov     si,4
        mov     cx,2
HERE1:
        mov     ax, XIMAG[si]
```

```
        mov     dx,_XIMAG[si+2]
        mov     bx,_N
        idiv    bx
        mov     _CCY,ax
        mov     ax,_XREAL[si]
        mov     dx,_XREAL[si+2]
        mov     bx,_N
        idiv    bx
        mov     _AAX,ax
        imul    _CCY
        mov     _XREAL[si],ax
        mov     _XREAL[si+2],dx
        mov     _XIMAG[si],0
        mov     _XIMAG[si+2],0
        mov     si,_N
        sar     si,1
        add     si,1            ; N / 2  + 1
        shl     si,1
        shl     si,1
        loop    HERE1
        mov     ax,_N
        add     ax,2
        mov     NPLUS2,ax
        mov     ax,_N
        sar     ax,1
        mov     NOVER2,ax
        mov     I,2
        mov     ax,2
FOR9:
        mov     si,I
        shl     si,1
        shl     si,1
        mov     di,NPLUS2
        sub     di,I
        shl     di,1
        shl     di,1
        mov     ax,_XREAL[di]
        mov     dx,_XREAL[di+2]
        sub     ax,_XREAL[si]
        sbb     dx,_XREAL[si+2]
        mov     bx,_N
        shl     bx,1
        idiv    bx
        mov     _DDY,ax
        mov     ax,_XREAL[di]
        mov     dx,_XREAL[di+2]
        add     ax,_XREAL[si]
        adc     dx,_XREAL[si+2]
        mov     bx,_N
        shl     bx,1
        idiv    bx
        mov     _AAX,ax
```

```
        mov     ax,_XIMAG[si]
        mov     dx,_XIMAG[si+2]
        add     ax,_XIMAG[di]
        adc     dx,_XIMAG[di+2]
        mov     bx,_N
        shl     bx,1
        idiv    bx
        mov     _CCY,ax
        mov     ax,_XIMAG[si]
        mov     dx,_XIMAG[si+2]
        sub     ax,_XIMAG[di]
        sbb     dx,_XIMAG[di+2]
        mov     bx,_N
        shl     bx,1
        idiv    bx
        mov     _BBX,ax
        mov     cx,FLAG
        or      cx,cx
        js      OTHER1
        mov     ax,_AAX
        imul    _CCY
        mov     _XREAL[si],ax
        mov     _XREAL[si+2],dx
        mov     ax,_BBX
        imul    _DDY
        add     ax,_XREAL[si]
        adc     dx,_XREAL[si+2]
        mov     _XREAL[si],ax
        mov     _XREAL[si+2],dx
        mov     _XREAL[di],ax
        mov     _XREAL[di+2],dx
        mov     ax,_AAX
        imul    _DDY
        mov     _XIMAG[si],ax
        mov     _XIMAG[si+2],dx
        mov     ax,_BBX
        imul    _CCY
        sub     ax,_XIMAG[si]
        sbb     dx,_XIMAG[si+2]
        jmp     DOWN1
OTHER1:
        mov     ax,_BBX
        imul    _DDY
        mov     _XREAL[si],ax
        mov     _XREAL[si+2],dx
        mov     ax,_AAX
        imul    _CCY
        sub     ax,_XREAL[si]
        sbb     dx,_XREAL[si+2]
        mov     _XREAL[si],ax
        mov     _XREAL[si+2],dx
        mov     _XREAL[di],ax
```

```
        mov     _XREAL[di+2],dx
        mov     ax,_AAX
        imul    _DDY
        mov     _XIMAG[si],ax
        mov     _XIMAG[si+2],dx
        mov     ax,_BBX
        imul    _CCY
        add     ax,_XIMAG[si]
        adc     dx,_XIMAG[si+2]
DOWN1:
        mov     _XIMAG[si],ax
        mov     _XIMAG[si+2],dx
        not     ax
        not     dx
        add     ax,1
        adc     dx,0
        mov     _XIMAG[di],ax
        mov     _XIMAG[di+2],dx
        inc     I
        mov     ax,I
        cmp     ax,NOVER2
        jg      ENDFOR9
        jmp     FOR9
ENDFOR9:
        pop     bx
        pop     cx
        pop     dx
        pop     di
        pop     si
        mov     sp,bp
        pop     bp
        ret
SORT    ENDP

NDIV    PROC    NEAR
        push    bp
        mov     bp,sp
        push    si
        push    bx
        mov     ax,1
        mov     I,ax
FOR5:
        mov     si,I
        shl     si,1
        shl     si,1
        mov     ax,_XREAL[si]
        mov     dx,_XREAL[si+2]
        mov     bx,_N
        call    NEAR PTR LDIV
        mov     _XREAL[si],ax
        mov     _XREAL[si+2],dx
        mov     ax,_XIMAG[si]
```

```
        mov     dx,_XIMAG[si+2]
        mov     bx,_N
        call NEAR PTR LDIV
        mov     _XIMAG[si],ax
        mov     _XIMAG[si+2],dx
        inc     I
        mov     ax,I
        cmp     ax,_N
        jg      ENDFOR5
        jmp     FOR5
ENDFOR5:
        pop     si
        mov     sp,bp
        pop     bp
        ret
NDIV    ENDP

LDIV    PROC    NEAR
        push    bp
        mov     bp,sp
        push    cx
        mov     cx,dx
        or      cx,cx
        jns     POS100
        not     ax
        not     dx
        add     ax,1
        adc     dx,0
POS100:
        mov     _XX,ax
        mov     ax,dx
        mov     dx,0
        div     bx
        mov     _XX+2,ax
        mov     ax,_XX
        div     bx
        mov     dx,_XX+2
        or      cx,cx
        jns     POS110
        not     ax
        not     dx
        add     ax,1
        adc     dx,0
POS110:
        pop     cx
        mov     sp,bp
        pop     bp
        ret
LDIV    ENDP

PAD0    PROC    NEAR
        push    bp
```

```
        mov     bp,sp
        push    si
        mov     ax,_N
        add     ax,1
        mov     I,ax            ; old  N + 1
        inc  WORD PTR  _M       ; new  M = old M + 1
        shl  WORD PTR  _N,1     ; new  N = twice  old N
FOR500:
        mov     si,I
        shl     si,1
        shl     si,1
        mov     _XREAL[si],0
        mov     _XREAL[si+2],0
        mov     _XIMAG[si],0
        mov     _XIMAG[si+2],0
        inc     I
        mov     ax,I
        cmp     ax,_N
        jg      EDFR500
        jmp     FOR500
EDFR500:
        pop     si
        mov     sp,bp
        pop     bp
        ret
PAD0    ENDP

        PUBLIC _SELECT_SAVE
_SELECT_SAVE    PROC  NEAR
        push    dx
        push    cx
        push    si
        push    di
        mov     I,-514
FOR510:
        mov     si,I
        shl     si,1
        shl     si,1
        mov   WORD PTR     _XN[si],0
        mov   WORD PTR     _XN[si + 2],0
        inc     I
        cmp     I,0
        jg      EDFR510
        jmp     FOR510
EDFR510:
        mov     ax,_NTOTAL
        mov     dx,0
        mov     cx,512
        div     cx
        mov     _NTIMES,ax
        mov     _NSAVE,-512
        mov     _NLOOP,1
```

```
        mov     bx,_FFIRADDR
FOR520:
        mov     _NCOUNT,1
FOR530:
        mov     si,_NCOUNT
        mov     di,si
        add     di,_NSAVE
        shl     si,1
        shl     si,1
        shl     di,1
        shl     di,1
        mov     ax,[bx][si]
        cwd
        mov  WORD PTR _XIMAG[si],ax
        mov  WORD PTR _XIMAG[si + 2],dx
        mov  WORD PTR _XIMAG[si + 2048],0
        mov  WORD PTR _XIMAG[si + 2050],0
        mov    ax, WORD PTR   _XN[di]
        mov  WORD PTR _XREAL[si],ax
        mov    ax, WORD PTR   _XN[di + 2]
        mov  WORD PTR _XREAL[si + 2],ax
        mov    ax, WORD PTR   _XN[si]
        mov  WORD PTR _XREAL[si + 2048],ax
        mov    ax, WORD PTR   _XN[si + 2]
        mov  WORD PTR _XREAL[si + 2050],ax
        inc    _NCOUNT
        cmp    _NCOUNT,512
        jg     EDFR530
        jmp    FOR530
EDFR530:
        push   bx
        call   NEAR PTR  CALCSS
        pop    bx
        mov    _NCOUNT,1
FOR540:
        mov    si,_NCOUNT
        add    si,512
        mov    di,si
        shl    si,1
        shl    si,1
        add    di,_NSAVE
        shl    di,1
        shl    di,1
        mov    ax, WORD PTR _XREAL[si]
        mov    WORD PTR _YN[di],ax
        mov    ax, WORD PTR _XREAL[si + 2]
        mov    WORD PTR _YN[di + 2],ax
        inc    _NCOUNT
        cmp    _NCOUNT,512
        jg     EDFR540
        jmp    FOR540
EDFR540:
```

```
        add     _NSAVE,512
        inc     _NLOOP
        mov     ax,_NLOOP
        cmp     ax,_NTIMES
        jg      EDFR520
        jmp     FOR520
EDFR520:
        pop     di
        pop     si
        pop     cx
        pop     dx
        ret
_SELECT_SAVE ENDP

CALCSS    PROC    NEAR
        push    bp
        mov     bp,sp
        mov     _SIGN,-1
        call    NEAR PTR _ZFFT
        mov     FLAG, -1
        call    NEAR PTR SORT
        call    NEAR PTR NDIV
        mov     _SIGN,1
        call    NEAR PTR _ZFFT
        mov     sp,bp
        pop     bp
        ret
CALCSS    ENDP


    PUBLIC _LOCATE
_LOCATE PROC

_BEGINA:
    mov  ax,0F800h
    mov  ds,ax
    mov  bx,0700h
    mov  ax,0
    mov  es,ax
    mov  di,0700h
    mov  cx, _BEGINA  - _START + 6
NEXTB:
    mov   ax, WORD PTR [bx]
    mov  WORD PTR es:[di],ax
    add   bx,2
    add   di,2
    loop  NEXTB

    jmp  FAR PTR _HOME

_LOCATE ENDP
```