ORNL/TM--11454

DE91 005642

Engineering Physics and Mathematics Division

A DATA ACQUISITION WORK STATION FOR ORELA

B. D. Rooney
J. H Todd
R. R. Spencer
L. W. Weston

Date Published - September 1990

Prepared for
Energy Programs Division

OAK RIDGE NATIONAL LABORATORY
Oak Ridge, Tennessee 37831
operated by
Martin Marietta Energy Systems, Inc.
for the
U.S. DEPARTMENT OF ENERGY
under Contract No. DE-AC05-84OR21400

TABLE OF CONTENTS

ABSTRACT

A new multiparameter data acquisition system has been developed and fabricated at the Oak Ridge Electron Linear Accelerator (ORELA) which utilizes an IBM PS/2 model 80 personal computer and data handler with a 2048 word buffer. The acquisition system can simultaneously acquire data from one, two, or three digitizers, multiplex up to four detectors, read and control up to 16 scalers, and output 32 D.C. logic signals which can be used to control external instrumentation. Software has been developed for the OS/2 operating system, supporting multiparameter data storage for up to three million channels with the capability of collecting data in a background mode, to make the computer available for other tasks while collecting data. The system also supports multiparameter biasing and can collect, crunch, and store data at rates as high as 30,000 events per second.

CHAPTER 1
INTRODUCTION

Nuclear spectrometry frequently involves complex analyzer systems and computers to rapidly analyze, sort, and store applicable data. Many systems can be extremely elaborate and costly, depending on the experiment and the type of information desired. For single parameter events, such as pulse height analysis, there are an abundance of available analyzers on the market, many at a very reasonable price. However, multiparameter data acquisition systems involving multiple detectors and digitizers usually require a costly and elaborate computer system having extensive memory requirements. The availability of analyzer systems that have storage capacity above one million channels is extremely limited with the cost of available systems being very expensive.

These requirements at the Oak Ridge Electron Linear Accelerator (ORELA), along with the need to replace aging analyzers and computers within current budgets, led to the design and fabrication of a data acquisition system consisting of a hardware interface and data acquisition software which utilizes an IBM PS/2 model 80 personal computer. Developed as the primary replacement and upgrade of older computer equipment, the new system is capable of analyzing 64 bits of information per event into four parameter storage, using non-linear binning, and can employ multiparameter discrimination. Having a maximum capacity of three million channels, the system's low cost makes it possible to provide each experimenter at ORELA with an independent system.

An IBM PS/2 personal computer was chosen over its contemporaries because of its architecture, compatibility, and multi-tasking capability. Collection of data can be performed with top priority, while the computer is being used for other tasks such as data analysis, making real time analysis of data possible in some situations. Software, in the form of a device driver, allows users to easily write specialized programs that have access to data during acquisition. All software in this manual has been designed to be run under the OS/2 operating system.

This manual serves as a user's guide for the IBM PS/2 data acquisition system providing the reader with detailed information on setting up and using the system for a variety of applications, including multiparameter data storage, multiparameter biasing, time of flight energy display, and software development. This manual also serves as a guide for the computer code ANALYZER, a general purpose program which provides real time display of data, backup file support, timer support, time of flight energy calculation, and other functions. Other programs can easily be written in either the protected mode or the DOS compatible mode to interface with the acquisition driver by following the format specified in Chapter 6.

CHAPTER 2
DESCRIPTION

2.1 GENERAL

A block diagram of the data acquisition system is shown in Figure 2.1. The system consists of an IBM PS/2 model 80 personal computer attached to several external devices through a data handler which includes a 2048 x 16 bit FIFO buffer. External instrumentation for the initial implementation includes one ORTEC time digitizer clock, two Nuclear Data ADCs, eight JORWAY scalers, and an inverter for the data lines coming from the ADCs. The experimenter has the option to configure the system for specific and fewer digitizers through switches on the front of the data handler. There are also 32 D.C. level output lines that can be used to control external instrumentation.

Control of the data handler is accomplished by a computer through a commercially available I/O board, which allows the computer to communicate with the data handler and transfer information. The data handler has been designed to accept and store data from each digitizer until the computer is ready to analyze it. The scalers are controlled likewise with all data being transferred via the same interface. Transfer and processing of data is performed by the following procedure.

1) The data handler receives a data ready signal from each applicable digitizer, informing the data handler that there is converted data ready to be transferred.

2) The data handler stores the data from each digitizer into a buffer (FIFO) and then simultaneously resets each digitizer. This is performed in less than two microseconds, after which each digitizer is ready to acquire new data.

3) Every 31 milliseconds the computer halts the current program and jumps to an interrupt routine where each 16 bit word in the data handler is transferred directly to the CPU. Transfer of data to the computer does not prevent the data handler from accepting new events at any time.

4) The CPU checks the most significant bit on each word using a 1,0,0,0 sequence to ensure the correct number of 16 bit words have been transferred for every event.

5) After the correct number of words have been transferred, as determined by the word select switch, the event is analyzed and stored in its applicable channel(s).

6) After the FIFO buffer in the data handler is emptied, the computer returns to the current program or process that was halted.

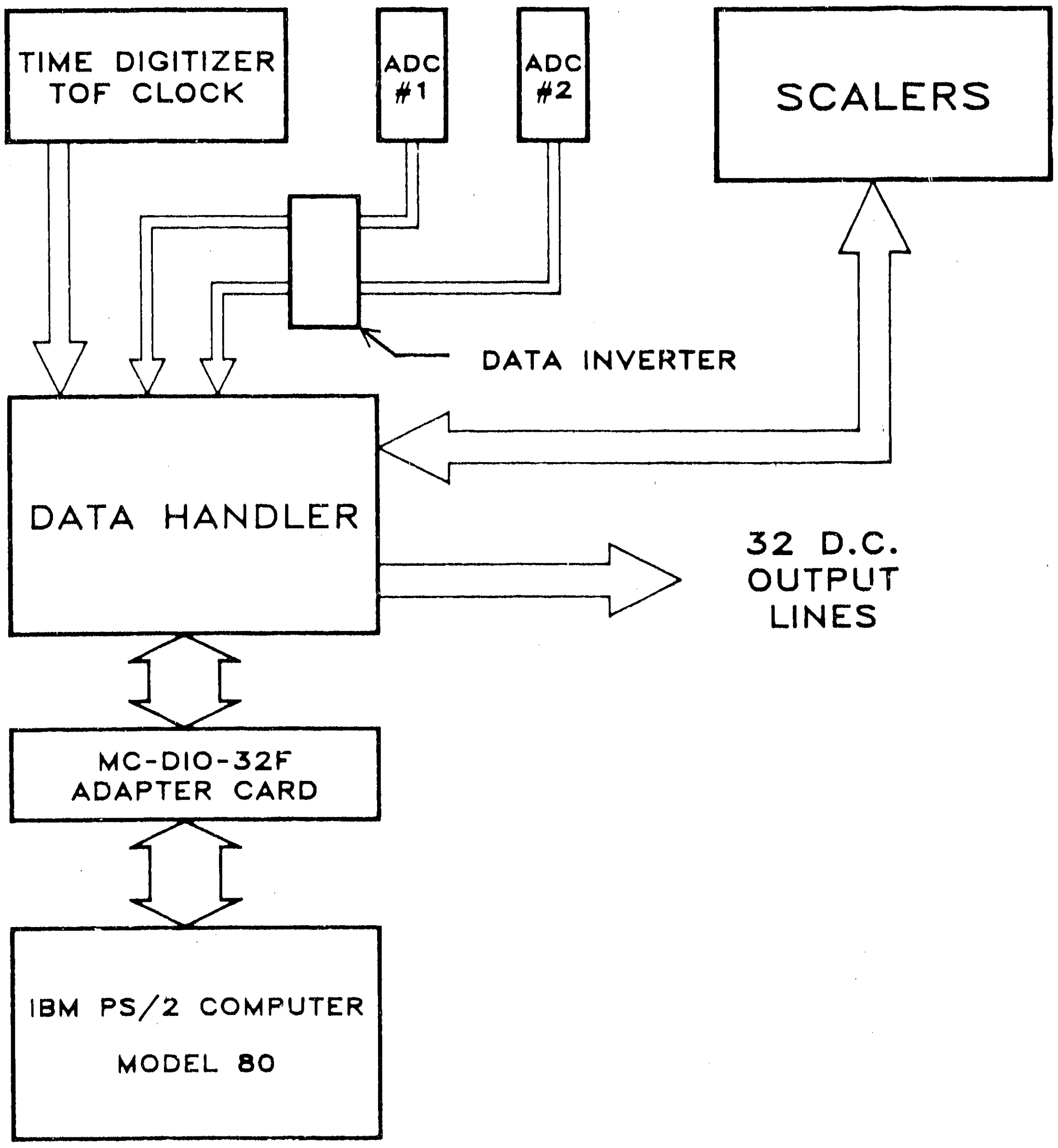


Figure 2.1. Block diagram of data acquisition system.

2.2 COMPUTER

This data acquisition system utilizes an IBM PS/2 model 80-111 personal computer to control the data handler and analyze each event. The computer contains an 80386 microprocessor with a 20 MHz clock and has a 80387 math co-processor. Additional adapters installed into the computer consist of an I/O parallel port card and extended memory. Memory may be extended up to 16 megabytes, giving the system approximately a three million channel capacity, with each channel consisting of 32 bits (four billion counts per channel). At least 2 megabytes of memory are reserved for the OS/2 operating system.

2.3 MC-DIO-32F I/O PORT ADAPTER

The computer requires a 32 bit parallel port I/O adapter to enable the computer to communicate with the data handler. The acquisition system has been developed to use a commercially available interface card, MC-DIO-32F, from National Instruments Corporation. This I/O card is installed into one of the computer expansion slots and connected to the data handler using a 50 line ribbon cable.

The MC-DIO-32F adapter must be installed and configured to a base port address of D000 hex before operation. To accomplish this, the user is referred to the instructions that come with the board. The interrupt level and DMA channel are currently not used in this system, th .s these parameters may be disabled or set to whatever the user desires.

All communication with the data handler, including the transfer of data, is accomplished through this adapter using four parallel I/O ports (A,B,C, & D). Ports A and B are used for transferring data, while ports C and D are used for interface control (start, stop, etc.). Port C also controls which data is currently on ports A and B. Thus, ports A and B can be used for transferring digitizer data, scaler data, or input of test data into the data handler FIFO memory.

2.4 SOFTWARE

The methodology of software development has been to provide the user with a versatile data acquisition system that can easily interface with any user program in the OS/2 operating system. This provides the user with the ability to expand and enhance data analysis and display routines at his or her leisure and to support possible upgrades in computer hardware.

Three main programs have been written to assist the user in controlling and displaying data; LOADCRUN.EXE, ANALYZER.EXE, and DEVICE2.SYS. Each of these programs are described in detail in Chapters 4, 5, and 6, respectively. DEVICE2.SYS is a device driver that controls the data handler, performing all necessary tasks to start, stop, and transfer data. Program LOADCRUN is used to load a crunch file into the device driver. A crunch file is an ASCII data file containing parameters supplied by the user to determine how data is to be stored. ANALYZER provides the user with a general purpose program which starts, stops, and displays data by accessing the device driver. Other data acquisition programs can easily be written to replace ANALYZER using the format in Chapter 6 and the example in Appendix C.

2.5 DATA HANDLER

The data handler provides the necessary hardware for receiving data from each digitizer and storing it in a 2048 x 16 bit word first in first out (FIFO) buffer until the computer is ready to receive it. The FIFO buffer enables the computer to transfer data from the buffer while the data handler is accepting data from the digitizers. The data handler also provides the essential signals required to start, stop, and read up to 16 scalers; however, current software supports only eight scalers. The device also has an output port with 32 DC logic lines which can be used to control external instrumentation. Although control is primarily through software, some configuration must be performed using the switches on the front panel of the data handler. Detailed information on the data handler is provided in Chapter 3.

2.6 DIGITIZERS

As many as four digitizers may be used in this data acquisition system; however, current software supports only three at this time, one clock and two ADCs. Which digitizers are employed can be controlled from switches on the front face of the data handler. The system is currently set up to accept up to 13 bits of data (8192 channels) from two pulse height ADCs and 26 bits of data and four tag bits from a single time digitizer (clock). Additional bits may be used if required; however, this will require some modification in the device driver software. The data line configuration from each digitizer to the data handler is described in Appendix D. Other digitizers may be used in place of those described in this report; however, each data line signal must correspond to the same lines defined in Appendix D. Some ADC's may require a data line inverter.

2.7 DATA INVERTER

The system shown in Figure 2.1 includes a data inverter between two of the ADCs and the data handler. This is needed when using Nuclear Data ADCs, since all logic on the data bus is inverted from that for which the system has been designed for. The data inverter illustrated in Figure 2.1 supports up to three Nuclear Data ADCs.

2.8 SCALERS

Up to eight JORWAY scalers are supported by software on this data acquisition system. Each scaler connects to the data handler by way of a cable assembly which has nine connectors, one for each scaler and one to the data handler. Other scalers may be used if they follow the same control logic as JORWAY scalers.

In addition, there are three BNC connectors on the back side of the data handler that are used to provide start, stop, and reset signals to all scalers. On the front face, there are stop, start, and reset buttons that can be used to control the scalers manually.

2.9 INPUT RATES

The dead time of the data handler is no greater than 2 microseconds. This permits an input rate in excess of 500,000 events per second for bursts of data not exceeding the buffer capacity of 2048 words. The MC-DIO-32F interface performs all the necessary handshaking requirements with the data handler to place any data in the FIFO buffer directly onto the designated port address in the computer. This transfer to the computer is accomplished in less than 200 nanoseconds, giving programs almost immediate access to data.

The maximum average input rate of this data acquisition system over an extended period of time is highly dependent on the crunch table loaded into memory. For example, the acquisition driver is capable of taking one event and crunching and storing it in up to nine different locations. The driver can also perform multiparameter discrimination on each event for multiple detectors. All this takes time. Measurements, using software referenced in this manual, have resulted in input rates as fast as 30,000 events per second when using a simple crunch routine with one digitizer. This was performed while the computer was running entirely in the protected mode. Acquiring data in the DOS compatibility mode results in a 10 percent reduction in the input rate due to the operating system switching in and out of the protected mode during data storage.

If desired, additional performance and speed may be obtained by modifying the data acquisition driver software. Simplifying the crunch and interrupt routines in the device driver can substantially affect the maximum average input rate. Measurements have resulted in data acquisition rates in excess of 100,000 events per second for simple storage routines; however, modifying the data acquisition driver is only recommended for very specific applications where flexibility is not a requirement.

CHAPTER 3
DATA HANDLER

3.1 DESCRIPTION

The data handler is an external device that provides an interface between each digitizer and the PS/2 computer. It provides the necessary hardware and controls to accept data from one to four digitizers, read 16 scalers or digital registers, and outputs 32 D.C. logic signals which can be used to control instrumentation. It contains a 2048-word FIFO buffer allowing non-synchronous input and output of data. This enables the data handler to accept data independently from the computer, resulting in very short deadtimes, less than two microseconds, for data storage. The total deadtime can be shortened to approximately one microsecond through modifications in the data handler, depending upon the requirements of the external digitizers. The data handler is completely software controlled, except for some switches on the front panel which select the applicable digitizers and control the number of sixteen bit words to be included in each event.

3.2 FUNCTIONAL LOGIC

Figure 3.1 is an outline of the data handler integrated into a system. Figure 3.2 is a functional outline of the data handler. There are four ports in the system. In addition, there is one control line from the system. Ports 1 and 2 are data input ports with their attendant control lines. Port 3 is an output port only. Port 4 is an I/O port to the PS/2 computer via the MC-DIO-32F adapter card.

Data is presented to port 1 and consists of up to 64 bits per event. The data handler accepts the data as one, two, three, or four words, each word consisting of 16 bits and stores the event into temporary memory (FIFO). The number of words per event that are accepted and stored is controlled by the word select switch on the front panel.

The temporary memory has a capacity of 2048 words containing 18 bits. Two of these bits are not used as data bits but can be used for other purposes such as a flag to indicate special situations. An example of the use of the bits would be to maintain correlation when an event consists of many words. A front panel switch permits the use of all 16 bits in the four data words for data or in another position will encode the most significant bit of the four words in a sequence of 1,0,0,0. This encoding will permit software checks to ensure that

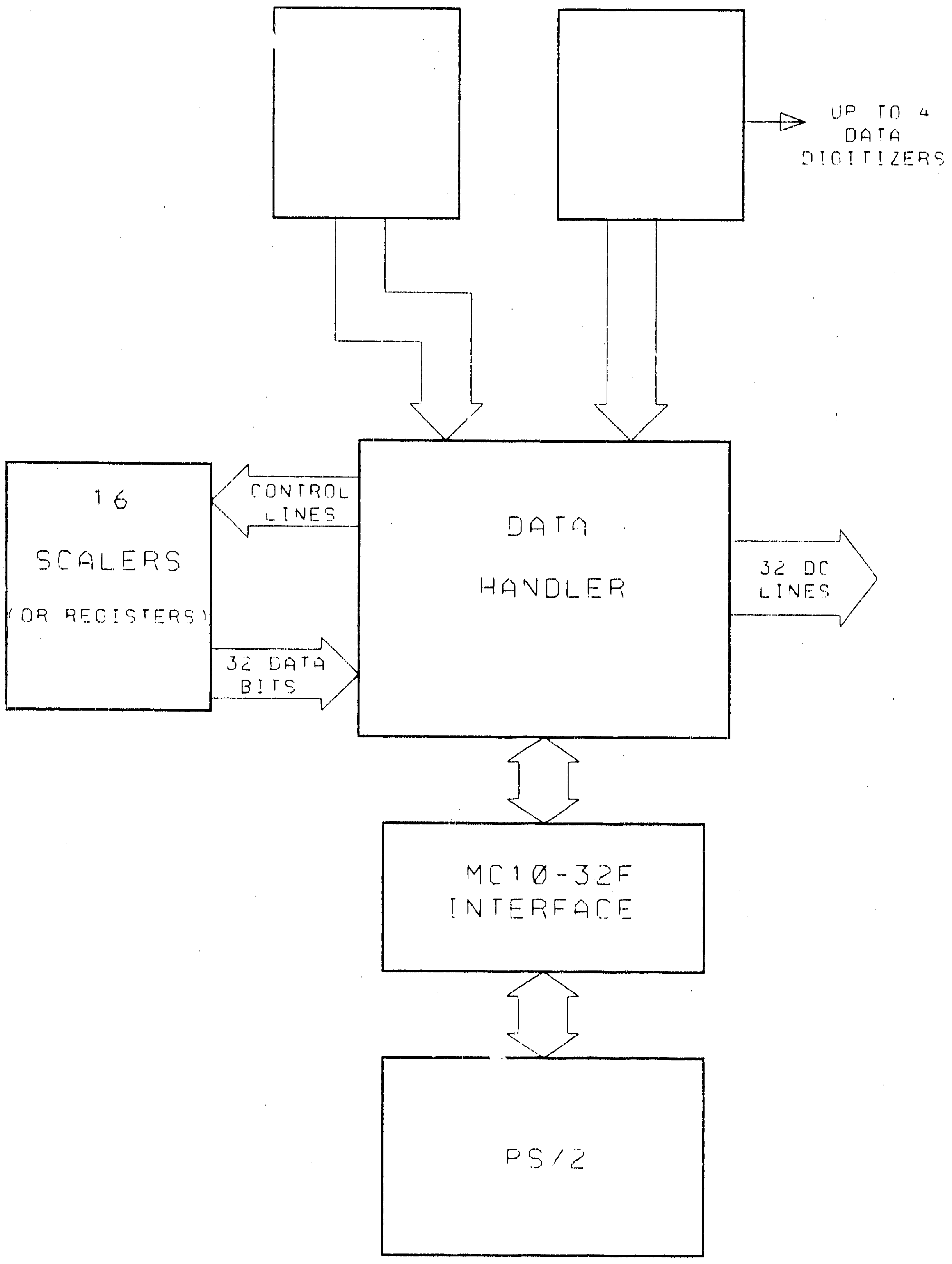


Figure 3.1. Data handler system.

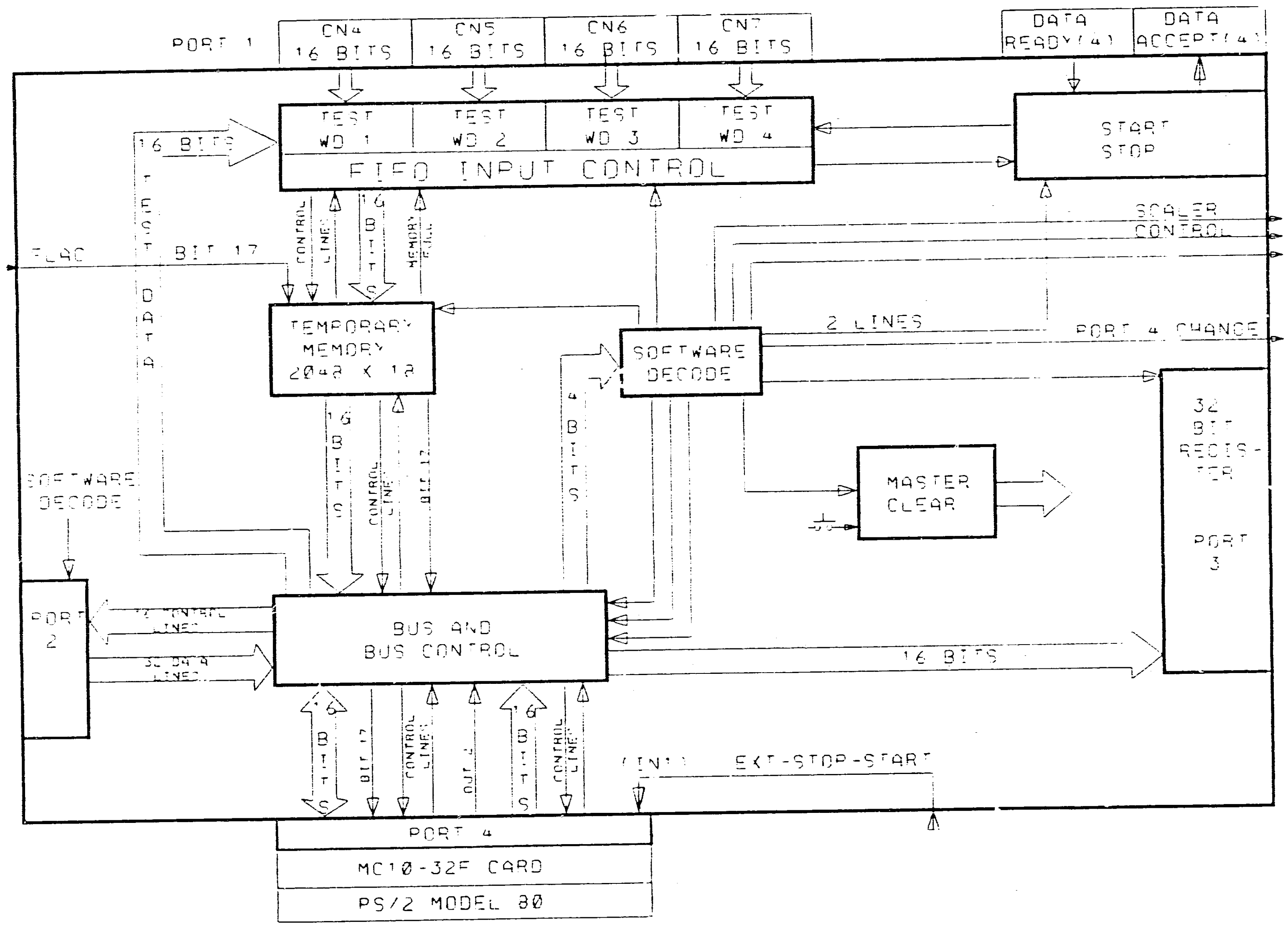


Figure 3.2. Logic diagram.

correlation of the four words are maintained. If the words are detected out of sequence action can be initiated by the program. In the software outlined in this manual the detection of an out-of-sequence series of words causes a Master Clear signal to be generated. This signal clears the temporary memory and generates a data accepted signal to clear the external equipment.

Time required to accept an event of four words is less than two microseconds. This time can be reduced to less than one microsecond by reducing the width of the data accepted pulses to the external equipment. The action of the input can be considered as a hardware DMA with a transfer rate of 1 million words per second and a word length of 64 bits. The size of the temporary memory is 512 words of 64 bit length or 2048 words of 16 bits length.

The input and output of the temporary memory are independent processes. The memory control processes the data ready signals from the selected external digitizers. When all of the data ready signals from the selected equipment are present, the data is transferred to the memory and a data accepted signal is generated and sent to all external equipment. The output of data from the temporary memory to the PS/2 is controlled by the PS/2 through the bus control system. The speed with which data can be removed from the temporary memory via the MC-DIO-32F adapter card varies with the program being used. Measurements have resulted in a maximum transfer rate of approximately 400,000 words of 16 bit length per second. However, software in this manual has been written to crunch each event before storage. To crunch and store a single event of four 16 bit words takes approximately 30 microseconds (see Section 2.9). This permits a maximum average input rate of approximately 30,000 events of 64 bits length per second.

Port 2 is an input port. This port will accept 32 bit data from one to sixteen digital registers. Sixteen control lines are available, under software control, to read up to the selected number of units.

Port 3 is an output port. This port will output D.C. levels on 32 lines. These 32 lines can be used directly to control external equipment or can be decoded to generate up to 2^{32} lines. These lines are under software control. Port 4 is attached to the MC-DIO-32F interface card installed into one of the expansion slots in the PS/2. This port contains 32 bidirectional lines, four lines used for handshaking with the data handler, two input control lines, and two output control lines.

The two additional lines, one control into and one indicating line out of the system, are used to perform the following functions. The control line into the system writes a bit into an internal register. The program monitors this bit and causes the system to stop taking data as long as this bit is low and restarts the system when the bit returns high. The indicator line is used by the software to output a pulse that indicates that the levels on the 32 D.C. lines have been changed.

All input and output lines, with the exception of the scaler control lines (the 16 lines in port 2) and the start, stop, and reset lines to the scalers should be considered to be standard TTL drive and input. The 16 scaler control lines will sink 20 milliamperes. The start, stop, and reset lines will drive 50 ohms with a +12 volt pulse.

3.2.1 Software Control

Port 4 contains two bi-directional 16-bit buses, bus 1 and bus 2, two pairs of data handshaking lines, and two pairs of flag lines. The two pairs of flag lines go to registers in the MC-DIO-32F card. One line in each pair reflects the condition of a bit that is set by the software. The other line in each pair can be used to set a bit in an internal register that can be monitored by the software. Line IN1 is accessed through the temporary memory and can be used for data correlation if desired (software in this manual does not support this). Line IN2 is set by a D.C. level generated externally. The software recognizes a low on this bit and generates a signal that stops the data acquisition as long as the bit is low and restarts when the bit goes high. The other two flag lines, OUT1 and OUT2, are used in the data handler with their respective buses, bus 1 and bus 2, to generate data and control functions. For example, bus 2 with OUT2 low will generate the software control pulses that control the system. Bus 2 with OUT2 high will cause the output of the 32 D.C. lines.

Table 1 is a listing of the codes and their functions.

3.3 FRONT PANEL SWITCHES AND CONNECTORS (see Figure 3.3)

POWER (ON/OFF): This switch provides power to the data handler. The power should be turned off prior to connecting or disconnecting any of the rear panel cable assemblies.

DATA READY SWITCHES (ON/OFF): These four switches enable or disable the data ready signals and determine which digitizers must supply a data ready signal before the data handler recognizes a valid event. Table 2 illustrates the proper setting for these switches for various combinations of digitizers when using software referenced in this manual.

Table 1. Software Codes and Functions

Code	Function
XXXX XXXX XXXX 0000	Not Used
XXXX XXXX XXXX 0001	Bus to Scalers
XXXX XXXX XXXX 0010	Stop Scalers
XXXX XXXX XXXX 0011	Reset Scalers
XXXX XXXX XXXX 0100	Start Scalers
XXXX XXXX XXXX 0101	Bus to Data Input
XXXX XXXX XXXX 0110	Test Data Out
XXXX XXXX XXXX 0111	Master Clear
XXXX XXXX XXXX 1000	Not Used
XXXX XXXX XXXX 1001	Step Through Scaler Reads
XXXX XXXX XXXX 1010	Start System Acquire
XXXX XXXX XXXX 1011	Stop System Acquire
XXXX XXXX XXXX 1100	Not Used
XXXX XXXX XXXX 1101	Output Test Word
XXXX XXXX XXXX 1110	Pulse Indicates D.C. Line Change
XXXX XXXX XXXX 1111	Removes Bus From All Ports

Table 2. Front Panel Switch Settings

	Data Ready				
Digitizer	SW1	SW2	SW3	SW4	Word Select
TOF[1] Only	ON	OFF	OFF	OFF	2
PH1[2] Only	OFF	OFF	ON	OFF	3
PH2[3] Only	OFF	OFF	OFF	ON	4
TOF & PH1	ON	OFF	ON	OFF	3
PH1 & PH2	OFF	OFF	ON	ON	4
TOF, PH1, & PH2	ON	OFF	ON	ON	4

[1]TOF refers to the clock digitizer
[2]PH1 refers to pulse height analyzer #1
[3]PH2 refers to pulse height analyzer #2

ORNL-PHOTO 2333-90

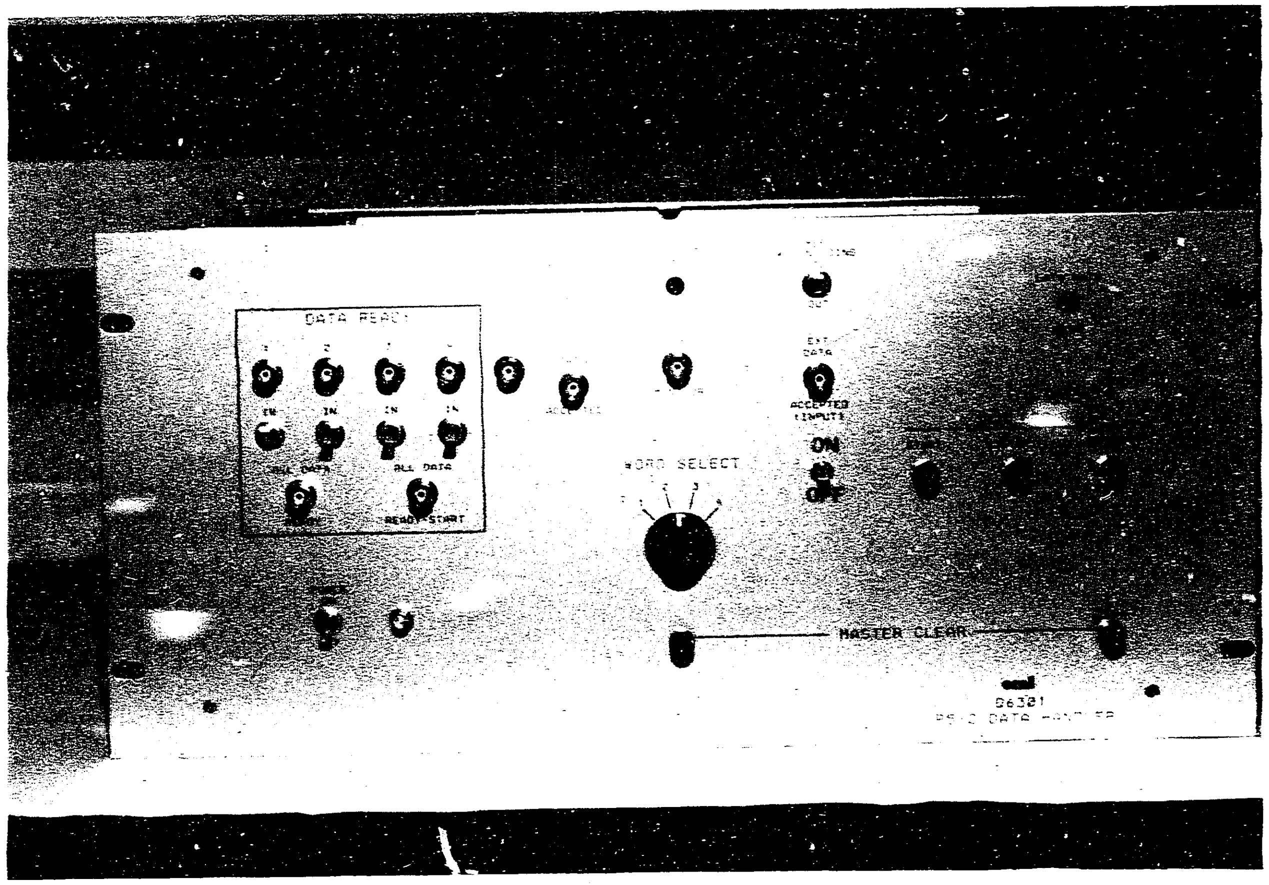


Figure 3.3. Front panel for PS/2 Data Handler.

DATA READY (BNC): These four BNC connectors are attached to the data ready lines of each corresponding connector on the rear panel (CN4 - CN7), respectively. These connectors therefore provide monitoring points for these signals.

ALL DATA READY (BNC): This connector provides a monitoring point for the signal that indicates that all of the selected data ready signals are present.

ALL DATA READY START (BNC): This connector monitors the data ready signal as seen by the FIFO input (indicates that all of the selected data ready signals are present and that the system has been started).

MSB WD ENCODING (IN/OUT): When set to the IN position, this switch sets the most significant bit of each word transferred to memory using a 1,0,0,0 pattern for up to four words. Software in this manual requires that this switch be set to the IN position at all times.

MASTER RESET: The two pushbuttons with this label between them must be depressed at the same time. This action clears all data from the temporary memory, generates a data accepted signal to the external digitizers, and removes the bus from the temporary memory. The system must be restarted after this action. This action can also be generated by software. The software can, of course, restart the system after a program generated master clear.

WORD SELECT (1-4): This switch determines the number of 16 bit words stored for each valid event. Software in this manual assumes the first word contains the most significant bits coming from a time digitizer and the second contains the least significant bits. The software also assumes that the other two 16-bit words contain data from two other digitizers. Software herein requires that this switch must be set to at least two words.

DATA ACCEPTED (BNC): This BNC is a monitor for the data accepted signal that is sent to all external equipment after each valid event.

START MONITOR (BNC): This BNC connector supplies a TTL high when the system is started (is in the acquire mode).

EXT DATA ACCEPT INPUT (BNC): A logic pulse (TTL high) input to this connector will produce a data accepted output to all external digitizers. This input can be used to ensure correlation of data contained in external digitizers.

EXT DATA ACCEPT INPUT ON/OFF (SWITCH): This switch enables/disables the above input. This switch must be in the off position if the input is not being used.

SCALER START SWITCH: This pushbutton generates a pulse to the rear panel BNC connector. This signal can be generated by software. This signal will drive 50 ohms with a 12 volt pulse.

SCALER STOP (SWITCH): This pushbutton generates a pulse to the rear panel BNC connector. This signal can also be generated by software and will drive 50 ohms with a 12 volt pulse.

SCALER RESET (SWITCH): This pushbutton generates to a rear panel BNC connector. As above, this pulse can be generated by software and has the same drive capability.

ACCEPT DATA (LED): This LED is lighted when the system is in the acquire mode.

DATA RATE HIGH (LED): This LED is lighted when the temporary memory (FIFO) is full. This is an indication that the data rate is high enough to fill the FIFO and that data is probably being lost.

3.4 REAR PANEL CONNECTORS (see Figure 3.4)

CN4 (WORD 1 INPUT): This connector corresponds to the first 16 bit word stored for every valid event. It is normally attached to the tags and most significant bits of the time digitizer. The pin connections are illustrated in Appendix D.

CN5 (WORD 2 INPUT): This input connector corresponds to the second 16 bit word stored for every valid event. It is normally attached to the less significant bits of the time digitizers. The pin connections are illustrated in Appendix D.

CN6 (WORD 3 INPUT): This input connector corresponds to the third word stored for every valid event. This connector is normally attached to a pulse height analyzer (denoted by PH1). The pin connections are illustrated in Appendix D.

CN7 (WORD 4 INPUT): This input corresponds to the fourth word stored for each valid event. This connector is normally attached to a second pulse height analyzer (denoted as PH2). The pin connections are illustrated in Appendix D.

ORNL-PHOTO 2335-90

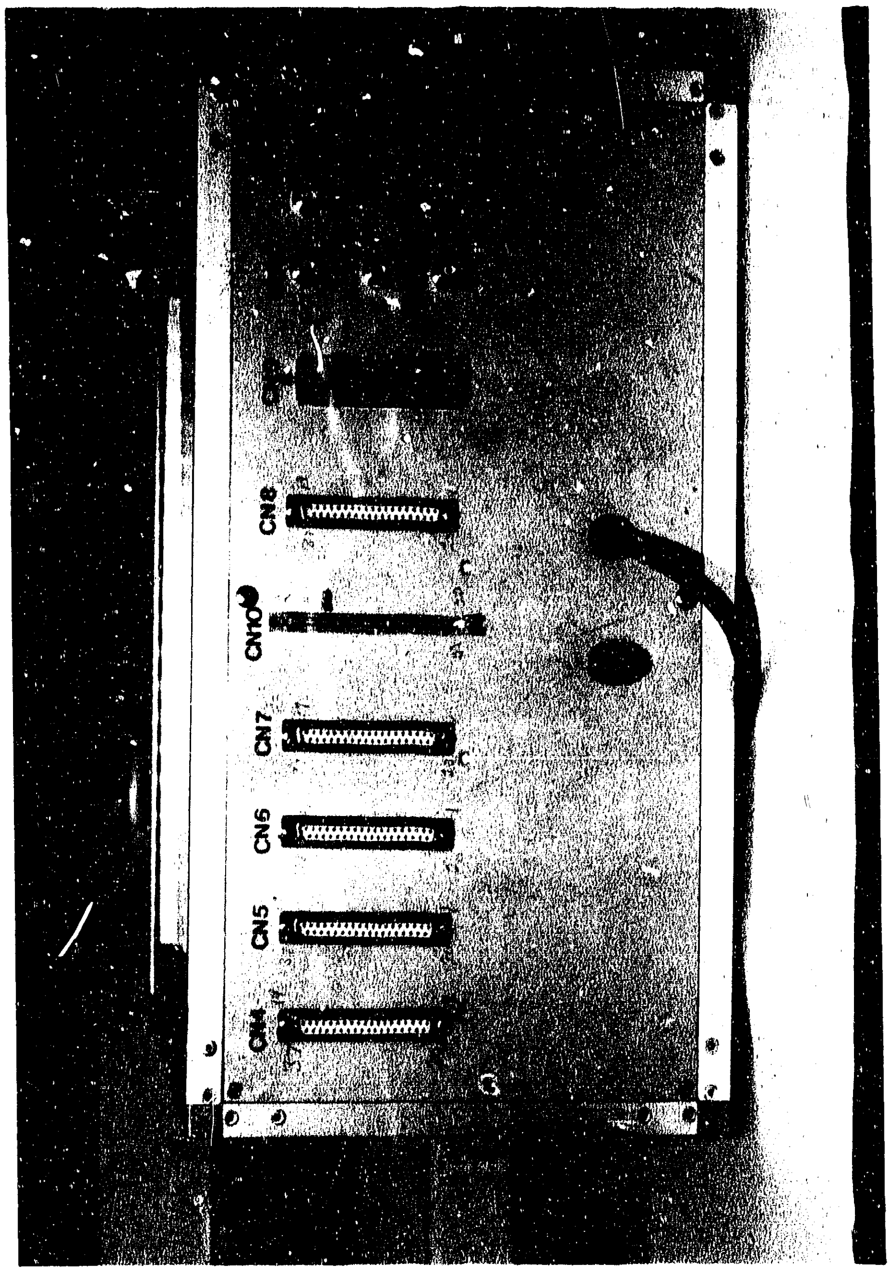


Figure 3.4. Rear panel connectors for PS/2 Data Handler.

CN8 (D.C. OUTPUT): This is a 37 pin connector which supplies the 32 D.C. output lines that can be used to control external instrumentation. A high signal on each line is approximately +5 volts. The pin connections are illustrated in Appendix D.

CN9 (SCALER INPUT): This connector contains the 32 input lines and 16 output control lines that are used to read external instrumentation, such as scalers. The pin connections are illustrated in Appendix D.

CN10 (COMPUTER): This is a 50 pin connector which attaches the data handler to the MC-DIO-32F interface card that is installed in one of the computer expansion slots. All data and control signals to and from the computer are routed through this connector. The pin connections are the same as for the MC-DIO-32F card.

SCALER START: This is a BNC connector that can be used to generate a start pulse for scaler control.

SCALER STOP: This is a BNC connector that can be used to generate a stop pulse for scaler control.

SCALER RESET: This is a BNC connector that can be used to generate a reset pulse for scaler control.

D.C. CHANGE: This is a BNC connector that will generate a pulse whenever a command 14 is written to the data handler. Software presented in this manual automatically generates a pulse at this connector every time the D.C. output lines are modified.

3.5 SOFTWARE CONTROL

The data handler is controlled by the four least significant bits written to port C of the MC-DIO-32F interface card. Ports C and D on the interface card must be configured as output ports with the handshaking mode enabled. This is performed by the device driver referenced in this manual. Writing a number from 1 to 15 to port C will control the data handler. Again, the device driver given in this manual performs all these functions for the user, along with the other required operations. Each command is briefly described in Appendix E for the user's reference.

3.6 OUTPUT D.C. LOGIC

To modify the D.C. output lines on the rear panel connector, the OUT2 bit on the MC-DIO-32F interface card must be set high by the program. Once this bit is set high the program must write two 16-bit words to ports C and D (bus 2 in port 4 of the data handler) to output the desired 32 lines. Once the output lines are changed, the OUT2 bit must be brought low again. The device driver referenced in this manual will perform all these steps along with pulsing the D.C. CHANGE connector on the rear of the data handler.

3.7 MC-DIO-32F INTERFACE CARD

The MC-DIO-32F interface card is manufactured by National Instruments. References on this card can be found in the manual on this card.

CHAPTER 4
CRUNCH FILES

4.1 GENERAL

Since as many as 64 bits of information may be contained in every event, a method to pull out applicable data and crunch it into available computer memory is required. A crunch table supplies the parameters for this process. Each event is analyzed using a crunch table that has been loaded into the data acquisition driver. The crunch table also establishes the parameters that are needed to enable tag inputs, setup multiparameter biasing (PSD), and determine how each event is binned and stored (i.e. one, two, or three dimensional storage).

Either of two procedures can be used to load crunch tables: 1) program LOADCRUN.EXE, run from the protected mode of OS/2; or 2) the load command directly from program ANALYZER. This chapter describes both procedures and also provides a description of the format used to create and edit crunch files. Appendix B illustrates several examples of crunch files.

4.2 PROGRAM LOADCRUN

A crunch file may be loaded into computer memory using program LOADCRUN.EXE. This program must be run from the OS/2 protected mode with the device driver, DEVICE2.SYS, installed. The following example illustrates the command line format.

```
LOADCRUN D:\CALIB.TBL /p
```

The above command loads the crunch parameters listed in file CALIB.TBL, found in the root directory on drive D. The optional parameter, /p, is used to print out the crunch table after loading it into memory. If a format error is found in the crunch file, the program will display an error message and then terminate loading. If computer memory does not allow memory allocation for the number of storage channels needed for the crunch table, the program will terminate loading without allocating any memory. Memory allocated by LOADCRUN will be de-allocated whenever a new crunch file is loaded into memory.

Successful loading of the crunch table will be confirmed with a message displaying the crunch file name and the number of channels allocated. One channel is equal to four bytes (32 bits) of computer memory; thus, to allocate one million channels, the computer must have at least four megabytes of consecutive free memory. Starting LOADCRUN without including a file name on the command line causes the program to prompt the user for a path and file name.

4.3 LOADING THE CRUNCH FILE FROM ANALYZER

The crunch parameters needed for the data acquisition driver may also be loaded from program ANALYZER. Chapter 5 describes ANALYZER in more detail; however, the load command is discussed briefly here. The load command in ANALYZER performs the same operation as program LOADCRUN. It reads a crunch file and transfers the parameters into the data acquisition driver.

The load command is entered by typing the letter "L", followed by the name of the crunch file. The following example illustrates the format used to reload a crunch file using ANALYZER.

```
L D:\CALIB.TBL
```

The above example loads the crunch file CALIB.TBL, found in the root directory on drive D. An error in the crunch file results in ANALYZER informing the user and terminating execution.

4.4 CRUNCH FILE FORMAT

All crunch files contain only ASCII text characters, yet may be comprised of several crunch sections. Comments may be inserted at the beginning of each crunch file; however, no remark may contain the key words TAG#1, PSD MODE, or SECTION, since these words mark the beginning of a new crunch section. The rest of this chapter describes the format used in each crunch section and provides examples to aid the user in setting up and editing his own crunch files. Figure 4.1 provides a listing of crunch file CRUNCH.TBL, an example of a typical crunch file that uses tags, PSD mode, and several crunch sections for multiple storage.

Any crunch file may be edited by a line or full screen editor, from either the OS/2 protected mode or the DOS compatibility mode. Remember, once a crunch table is edited,

```
A>TYPE CRUNCH.TBL

TAG#1: YES
TAG#2: YES
TAG#3: YES
TAG#4: NO

PSD MODE ON
PH2
PH1
128
4
100
1, 2, 4
64, 64, 128, 256, 512, 1024, 1024, 1024, 1024, 1024, 2048

SECTION 1
PARAMETERS  2
PH1
1024, 8
TOF
1, 1000
8, 32
10, 100
20, 200
TAGS:  1, 2, 4, 101, 102, 104

SECTION 2
PARAMETERS 1
PH1
512,16
TAGS:  1, 2, 4, 101, 102, 104

SECTION 3
PARAMETERS 1
TOF
1, 1000
8, 32
10, 100
20, 200
TAGS:  1, 2, 4, 101, 102, 104

SECTION 4
PARAMETERS 1
PH1
1,8192
TAGS:  1, 2, 3, 4, 5, 6, 7, 101, 102, 104


A>
```

Figure 4.1. Listing of example crunch file.

it must be reloaded into the data acquisition driver to activate any changes. The following rules apply to all crunch files.

1. Any character may be lower or upper case.
2. TOF is used to refer to the time digitizer clock.
3. PH1 is used to refer to the first pulse height ADC.
4. PH2 is used to refer to the second pulse height ADC.
5. No more than 64 windows may be used in PSD mode.
6. No more than nine separate crunch sections may be included in any one crunch file.
7. Each channel coming from the time digitizer clock is assumed to be one nanosecond (i.e. the clock "tic" is one nanosecond).

4.4.1 TAG SECTION

An optional data section in the crunch file, referred to as the tag section, may be used to define which tag inputs on the time digitizer are enabled. If used, it must be the first data section found in the crunch file and must be comprised of four lines. Each tag must be listed with a "YES" or "NO" following the tag number. A "YES" indicates that the tag input is enabled while a "NO" indicates that the tag input is disabled. If enabled, tags one, two, three, and four are worth a value of 1, 2, 4, and 8 respectively. This permits identifying any combination of tags per event. The following example,

```
TAG#1: YES      /* value = 1 */
TAG#2: YES      /* value = 2 */
TAG#3: YES      /* value = 4 */
TAG#4: NO       /* value = 8 */
```

illustrates a tag section that can be used to enable tags one, two, and three. The comments to the right of each line are only a reminder of what each tag is worth, and are not required. An event which includes a high signal at tag input number one will add a 1 to the tag data register. A high signal at tag input number two will add a 2 to the tag register. A high signal at tags one and two will result in a tag register value of 3. A high signal at tags one and three will result in a tag value of 5. A high signal at tag four will have no effect when using the above example. If the tag section is completely left out of the crunch file, all tag inputs are disabled and the tag value for each event is zero.

4.4.2 PSD MODE SECTION

Another optional data section, referred to as the PSD section, may be used to set up multiparameter discrimination, also referred to as pulse shape discrimination (PSD). The PSD section must follow the tag section, if used, and come before any crunch sections. Comprised of eight lines, it uses the format listed below. The reader may see other examples in Appendix B containing remarks and descriptions inserted on each of these lines. Remarks are allowed since LOADCRUN and ANALYZER recognizes only numbers and certain key words (PSD MODE, PH1, PH2, and TOF).

```
PSD MODE ON
PH2
PH1
128
4
100
1, 2, 4
64, 64, 128, 256, 512, 1024, 1024, 1024, 1024, 1024, 2048
```

Line 1: This line indicates that the next seven lines of the crunch file are PSD parameter data. A "YES" or "ON" found on the first line of the PSD section will enable the PSD mode. One may keep this section in the crunch file and disable the PSD mode by replacing the "ON" key word with the word "OFF".

Line 2: This line determines the parameter used to set the bias channel for PSD. It is this parameter's spectrum that is displayed when program ANALYZER is in the PSD mode. Normally this parameter corresponds to the pulse shape ADC.

Line 3: This line determines the parameter used in establishing the window bins for the PSD decisions. Normally this parameter corresponds to the pulse height ADC. The window bin width parameters are given in line 8.

Line 4: Number of channels into which the PSD analyzer data will be crunched. This is the number of channels that will be allocated by the computer for each window and must range from 32 to 512 and be factorable by 2^n.

Line 5: This is the crunch factor for the PSD parameter. For the above example, the PSD digitizer gain must be set on 512. Thus, each event from the PSD analyzer will be crunched by a factor of 4 into 128 channels. The crunch factor here must be a number from 1 to 128, and be factorable by 2^n.

Line 6: This is the value added to the tag register if the event falls on or above the PSD bias channel. All bias channels are input separately using program ANALYZER.

Line 7: These are the event tags applicable for PSD analysis. If the tag register consists of one of these tag values, PSD analysis will be performed; otherwise, PSD analysis will be discarded for that event and crunching will continue (i.e., a tag of 3 will not be analyzed for PSD).

Line 8: These numbers establish the window bins for each tag listed in line 7. Each number represents the number of consecutive channels from the window parameter (normally the pulse height analyzer) that will be used for each PSD window. All numbers here must be on the same line and separated by commas and should sum to the ADC conversion gain. This line may extend out to 256 characters.

4.4.3 THE CRUNCH SECTION

The crunch file may contain as many as nine crunch sections, not including the tag or PSD sections. Each crunch section establishes the parameters that will be used to analyze and store each event. Thus, every event may be crunched several times, each with different crunch parameters. All crunch sections follow both the tag and PSD mode sections. They may use one, two, or three parameters, allowing up to three dimensional storage capability. An example illustrating two parameter crunching follows with a brief explanation of each line.

```
SECTION
PARAMETERS 2
PH1
1024, 8
TOF
1, 1000
8, 32
10, 100
20, 200
TAGS: 1, 2, 4, 101, 102, 104
```

Line 1: The key word "SECTION" identifies the start of a new crunch section. Every crunch section must begin with this key word.

Line 2: Number of parameters used for this section. This must be a number from 1 to 3. The word PARAMETERS is optional on this line.

Line 3: This is the first crunch parameter. It must be either a PH1, PH2, or a TOF.

Line 4: This line represents the crunch factor for the first parameter (PH1). This will crunch the PH1 digitizer data by a factor of 8, into no more than 1024 channels. For this example, the gain of the PH1 digitizer should be set on 8192. More lines may be inserted here to divide this parameter into different crunch factors.

Line 5: This identifies the second parameter used for this crunch section. This must be a PH1, PH2, or a TOF.

Lines 6-9: These lines are similar to line 4, except they represent the crunch factors for the second parameter (TOF). In the above example, any event occurring in the first 1000 nanoseconds will be stored in the first TOF channel. The next 8 x 32 nanoseconds will be crunched down into eight channels using a crunch factor of 32 and so forth.

Line 10: This line is always the last line in each crunch section. It determines which tags are applicable to this section and under which tag base the event will be stored. A tag base is the starting channel for the applicable tag section. For this section, data would be stored as a function of three parameters (PH1 x TOF x TAG). If all tag inputs are disabled, this line should contain a zero.

CHAPTER 5
PROGRAM ANALYZER

5.1 GENERAL

ANALYZER, Version II, is a general purpose program used for controlling the data acquisition driver and providing real time display of data. The program has been designed to provide several useful functions which include time of flight energy calculation, multiparameter biasing (PSD), and backup file support. ANALYZER communicates with the data acquisition device driver using the same methods as described in Chapter 6.

Program ANALYZER functions only in the DOS compatibility mode of OS/2 and must be run from a hard disk environment. This chapter will describe the installation of ANALYZER and provide detailed information on using available functions and commands.

5.2 ANALYZER INSTALLATION

To install ANALYZER onto the hard disk, run the program INSTALL.EXE located on the installation disk. This installation program will prompt the user for the drive and directory where the ANALYZER files are to be installed and then copies the following files into that designated directory.

ANALYZER.EXE	@KEY.WIN
LOADCRUN.EXE	@PSD.WIN
CRUNCH.TBL	@SCALER.WIN
@ANAL.WIN	@SCRNTOP.WIN
@CALIB.WIN	README

The file CRUNCH.TBL contains a crunch table which can be modified by the user to obtain the appropriate crunch parameters desired or the user may create a separate crunch file under a different file name. Other files that begin with the @ character are data files used by ANALYZER for graphic display. The README document contains a summary of ANALYZER commands and provides any information that may not have been included in this manual.

After the above files have been copied, the installation program copies DEVICE2.SYS into the root directory of drive C and modifies the file CONFIG.SYS to include the following device command.

```
DEVICE = DEVICE2.SYS
```

This command loads the data acquisition driver into computer memory whenever the computer is started (booted up). After the installation program finishes, the computer must be rebooted before program ANALYZER can be started.

5.3 STARTING ANALYZER

Program ANALYZER may be run only in the DOS compatibility mode. Every time ANALYZER is run, it reads the last crunch file that was loaded into the data acquisition driver. This obtains the same crunch parameters used by the driver so ANALYZER may display tag and calibration information. If the number of channels calculated by ANALYZER differs from that of the device driver, the crunch table will be reloaded and all channels zeroed. This avoids incorrect display of tag and calibration information if the crunch file has been changed and not reloaded into computer memory. Thus, care must be taken not to change crunch files during data acquisition or loss of data may result when re-starting the ANALYZER program. If no crunch file has been loaded into the driver, ANALYZER will reload the last crunch file automatically. If starting ANALYZER for the first time and no crunch file has been loaded, the user will be automatically prompted for a crunch file name.

As long as the crunch file that ANALYZER reads matches the crunch table loaded into the data acquisition driver, ANALYZER will not change anything in the system. If the system is in the acquire mode when ANALYZER is started, it will continue to accumulate data. If the acquisition driver is not acquiring data when ANALYZER is started, it will still display whatever data is in computer memory. Thus, care should be taken to zero all memory prior to acquiring new data.

5.4 ANALYZER DISPLAY

ANALYZER has the capability of real time two-dimensional display (Channel versus Counts) and can scale the display from 32 to 2048 channels horizontally and up to 67 million counts vertically. It can also overlap and display different sections of memory while in the static mode.

The following keys are used to adjust the display; however, these keys apply only to the non-PSD mode since the PSD mode uses some of these keys differently. Additional control of the display is available through the commands listed in appendix A.

PgUp: Hitting the page up key will shift the display up by one screen. If this key is used in the PSD mode, it will shift the display to the next higher window.

<u>PgDn:</u> Hitting the page down key will shift the display down by one screen. If this key is used in the PSD mode, it will shift the display to the next lower window.

<u>LEFT ARROW:</u> The left arrow key will shift the display by one channel in the positive direction. In the cursor mode, this key will shift the cursor down by one channel.

<u>RIGHT ARROW:</u> The right arrow key will shift the display by one channel in the negative direction. In the cursor mode, this key will shift the cursor up by one channel.

<u>CTRL LEFT ARROW:</u> Pressing the left arrow key while holding down the control key will shift the display approximately 5 percent of the horizontal width in the positive direction.

<u>CTRL RIGHT ARROW:</u> Pressing the left arrow key while holding down the control key will shift the display approximately 5 percent of the horizontal width in the negative direction.

<u>UP ARROW:</u> The up arrow key will decrease the vertical scale of the display by a factor of two. The minimum vertical scale available is 32 counts. This key has no effect in the logarithmic mode.

<u>DOWN ARROW:</u> The down arrow key will increase the vertical scale of the display by a factor of two. The maximum vertical scale available is over 67 million counts. This key has no effect in the logarithmic mode.

5.5 FUNCTION KEYS

<u>START (F1):</u> This function key will start and stop data acquisition. This key will be highlighted when the system is acquiring data.

<u>CLEAR (F2):</u> This function key will zero all scalers and channels as defined by the crunch table.

<u>CURSR (F3):</u> This function key activates a cursor on the current display. When active, it displays the cursor channel and number of counts in that channel. It will also display the neutron energy for TOF data if calibration parameters have been entered using F4. The cursor is not available in the PSD mode and it will be

deactivated if a new crunch file is reloaded into computer memory.

<u>CALIB (F4):</u> This function key prompts the user for time of flight energy calculation parameters. NOTE: The gamma flash channel must be entered in units of uncrunched channels and each channel is assumed to be one nanosecond. The flight path must be entered in units of meters. Incorrect input may disable any energy calculation. An error message may indicate an impossible energy calculation.

<u><< >> (F5):</u> This function key expands the horizontal display by a factor of two. Minimum horizontal display is 32 channels.

<u>>> << (F6):</u> This function key increases the number of channels horizontally displayed by a factor of two. The maximum number of channels that may be displayed is 2048.

<u>LOG (F7):</u> This function key will toggle the vertical display between a logarithmic and linear scale.

<u>PSD (F8):</u> This function key places ANALYZER into the PSD display mode, giving the user the capability to easily adjust the bias channel for each PSD window by using the right and left arrow keys. The PgUp and PgDn keys will shift the display through different windows and tags, where each window has one bias marker. This marker represents the bias channel where any event occurring on or to the right of the marker is tagged with the value given in the PSD section of the crunch file. This tag value is added to the current tag register before any crunching of data is performed. If the event falls to the left of the bias marker, the tag register is not affected. Function key F8 has no effect if the PSD mode is not enabled in the crunch file.

<u>SAVE (F9):</u> This function key is used to save all channel data and scaler counts into a data file. If pressed the user will be prompted for an output file name and whether it is to be saved in text (ASCII) or binary format. Binary format consists of unsigned long integer format (4 bytes per channel). Scaler and run time data are also saved at the end of each data file.

<u>LOAD (F10):</u> This function key is used to load an ANALYZER data file into memory. When this key is pressed the user will be prompted for a file name. The program will automatically determine if the file contains text (ASCII) or binary data

and will then load the file into computer memory. If the scalers are currently being displayed when a file is loaded into memory, the scalers will also be loaded into the computer; however, the scaler data will not be physically loaded into each scaler.

PLOT (F11): This function key will dump the screen contents to a HP Laserjet Printer.

EXIT (F12): This function key will cause the computer to exit program ANALYZER and return to the DOS operating system. If this key is pressed during data acquisition the computer will continue to acquire data. Exiting will not effect any data in the computer memory, unless the crunch file currently being used is changed.

5.6 BOTTOM LINE COMMANDS

Special instructions may be entered into ANALYZER by typing them out on the keyboard; these are echoed to the bottom line of the display. Hitting the enter key only will always execute the command line that was last entered. For example, entering "A512" will shift the display by 512 channels. Every time the return key is hit thereafter, the display will be shifted 512 channels until a new command is entered.

These commands are designed to perform specific tasks not covered by the function keys and gives the user additional versatility in displaying and analyzing data. A summary of all ANALYZER commands are given in Appendix A with a brief definition. The rest of this section contains additional details on several of these commands for the reader's information.

5.6.1 LOAD CRUNCH FILE COMMAND

The load crunch file command may be used to load a new crunch table into the data acquisition driver from program ANALYZER. This command may be employed by entering an L and the crunch file name. This performs the exact same function as program LOADCRUN.EXE, except that it may be executed while running ANALYZER. If the system is in the acquire mode when this command is used, the system will be stopped and all channels zeroed. An example of this command would be "L D:\CRUNCH.TBL".

5.6.2 DISPLAY OVERFLOWS

The overflow command displays overflows (events that are not stored) that have been detected during crunching of data. Overflows are displayed by typing the letter "O" and then hitting the enter (return) key. Several parameters will be displayed on the screen, halting the real time display if the system is running; however, the system will continue to acquire data. Hitting any key thereafter will clear the screen and return the display to its normal mode. An explanation of each overflow parameter follows.

PSD =	Number of events that were outside the range of the PSD parameter as listed in the PSD section of the crunch file. Any counts here indicate that the PSD section or ADC gain should be modified. Each event listed here is completely discarded with no additional crunching.
TOF =	Number of events from the time digitizer clock (TOF) that were outside TOF crunch section. Since several TOF crunch sections may be used in a crunch file, a single event may result in more than one TOF overflow.
PH1 =	Number of events from Pulse Height Analyzer #1 (PH1) that were outside PH1 crunch section. Since several PH1 crunch sections may be used in a crunch file, a single event may result in more than one PH1 overflow.
PH2 =	Number of events from Pulse Height Analyzer #2 (PH2) that were outside PH2 crunch section. Since several PH2 crunch sections may be used in a crunch file, a single event may result in more than one PH2 overflow.
PSD UNDER =	Number of PSD events that fall within the first two channels of any PSD window. Each event recorded here is discarded completely with no additional crunching.
MAX CHAN =	Number of crunches that fall above the maximum allocated channel. This indicates a severe software or operating system problem and should be corrected.
PSD TAGS =	Number of events with a tag value not applicable to the PSD section of the crunch file. For example, if PSD is to be performed on events with tags 1, 2, or 4 only, and an event occurs with a tag 5, no PSD will

be performed and the event will be counted here. Crunching would still continue.

PSD WINDOW = Number of PSD events that fall into a window not covered by the PSD section in the crunch file. This indicates that either more windows are needed or larger channel widths for some windows are needed. The crunch file should be modified if any counts are recorded here.

5.6.3 DISPLAYING TOTAL EVENTS AND REJECTS

This command displays total events, rejects, and the average event count rate for the current run. It is employed by typing the letter "R" and hitting the enter (return) key. The number of rejects displayed corresponds to the uncorrelated events. Since each event may consist of as many as four 16 bit words, each word has its most significant bit set or cleared to provide a method for the data acquisition driver to detect missing or extraneous data in the buffer. If the acquisition driver detects a bad event it discards it and clears the interface buffer, recording it as a reject. A short beep will sound for each reject.

5.6.4 SETTING THE PRESET TIMER

This command allows the user to set the run time for ANALYZER. It is employed by entering the letter "T" and the desired run time in seconds (i.e. T100). Setting the timer to zero equals infinity. This command is used with the ANALYZER program only. Data acquisition will not stop if user is acquiring data in the background mode.

A variation of this command can be used to automatically save all data when time out occurs, then clear and restart the system. An example would be, T1000+FILE.000. If the extension is left off of this command, the program will automatically start from 000. In this example, program ANALYZER would perform the following steps:

1) Acquire data for 1000 seconds then stop.
2) Save all data under the file name of FILE.000. If FILE.000 exists, the data would be saved under the file name of FILE.001, and so forth.
3) Zero all channels and scalers.
4) Start acquiring new data.
5) Go back to step one.

5.7 THE AUTO BACKUP FILE

Every 5000 seconds during data acquisition, program ANALYZER automatically backups all data to the hard disk using file name @BACKUP.DAT. This insures that malfunctions in the computer or power outages result in no more than the last 5000 seconds of data being lost. Reloading the backup file into memory is accomplished the same way as loading in any other ANALYZER data file, using the LOAD function key (F10).

Program ANALYZER performs the backup procedure by first stopping data acquisition and saving the current run into file @BACKUP.DAT using binary format. Once all data is saved, ANALYZER continues data acquisition and displays the time at the bottom of the screen when backup was performed. Program ANALYZER uses this backup procedure only when ANALYZER is running. No backup procedure is employed while running the data acquisition in the background mode. Program ANALYZER never deletes the data file @BACKUP.DAT, except during the next backup when it overwrites @BACKUP.DAT with new data.

CHAPTER 6
DATA ACQUISITION DRIVER

6.1 GENERAL

Data acquisition and control of the data handler is accomplished by software in the form of a device driver. This methodology is required to utilize hardware interrupts under the OS/2 operating system and also to permit other high level programs to control and access data, either from the OS/2 protected mode or the DOS compatibility mode. In the OS/2 protected mode, the device driver allows several programs to access the data concurrently during data acquisition.

The device driver, also referred to as the data acquisition driver, performs the actual transfer of data from the interface buffer to computer memory and also performs required crunching and binning of data. Other programs wanting to start, stop, and access data must go through the device driver labeled DEVICE2.SYS described here

DEVICE2.SYS is installed into memory during computer startup (boot up), allowing other programs to access the driver by writing to the device file name "DEVICE_2". An example program written in Microsoft C is included in Appendix C.

This chapter describes the format and procedures which allow other programs to communicate with the data acquisition driver. The following will be of specific interest for readers who want to write programs to display and analyze data. For those who do not need a specialized program, a general purpose program, ANALYZER, runs in the DOS compatibility mode. Chapter 5 contains more information on program ANALYZER.

6.2 DRIVER INSTALLATION

The data acquisition driver is installed into computer memory using the DEVICE command from the CONFIG.SYS file. The CONFIG.SYS file is found in the root directory of the boot up drive (normally drive C) and must be edited to include the following line.

```
DEVICE = C:\DEVICE2.SYS
```

This example assumes that the file, DEVICE2.SYS, is located in the root directory on drive C; however, any path may be specified. Remember, once the CONFIG.SYS file is edited, the computer must be rebooted to install the device driver.

Installation may also be performed by running INSTALL.EXE, located on the ANALYZER installation disk. This installation program automatically copies DEVICE2.SYS to the root directory of drive C and includes the above command in CONFIG.SYS.

6.3 LOADING IN A CRUNCH TABLE

A crunch table provides the parameters needed by the data acquisition driver to sort out applicable data in each event and store it into an appropriate channel. The crunch table, or crunch file as it may be referred too, is provided by the user and must be loaded into the driver before acquiring data.

A crunch table may be loaded into the data acquisition driver by running the program LOADCRUN.EXE. The LOADCRUN program is run from the protected mode which reads an ASCII file containing parameters needed by the acquisition driver to analyze and store data. These crunch parameters are installed into the device driver by LOADCRUN and remain in memory until a new crunch table is reloaded or a specific command is sent to the driver to deallocate all crunch memory. Chapter 4 contains additional information on LOADCRUN.EXE and on crunch table format.

Until a crunch table is loaded into the data acquisition driver, most commands to the driver are disregarded. The exception to this rule is the "DRIVER STATUS" command. This command may be used at anytime and can determine if a crunch table has been loaded into memory by examining the number of channels allocated by the driver. If the number of channels allocated is zero, no crunch table has been loaded.

6.4 DATA ACQUISITION DRIVER CONTROL

Control of the data acquisition driver and transfer of data is performed by writing the address of a long integer array to the device called "DEVICE_2". The first integer of the array should contain a value from 1 to 15, which will instruct the data acquisition driver which specific command is to be executed. This section explains each of these commands and provides an example that can be used by the reader in a C language program.

Before any of these command statements can be executed, the program must first open a path to the device driver using an unbuffered format. The following example, written in Microsoft C, illustrates how a program may open a path to the data acquisition driver.

```
int device;
device = open("DEVICE_2",0x0002);
```

The preceding statements allow a C language program to write commands to the driver using the format described below and are intended to be used with all of the following examples. All of the following commands can be used while the system is acquiring data. The reader is also referred to Appendix C for an example of a complete C language program which implements several commands together.

(1) DRIVER STATUS: This command returns several parameters from the data acquisition driver that can be used for checking the status of the current run.

C program example:

```
long a[52];
a[0] = 1;                              /* Get acquisition driver status */
write(device,(char*)&a[0],1);
```

Returned parameters:

a[0] = 1, device driver error
10, device driver not running.
11, device driver running (collecting data).

a[1] = Runtime in seconds.

a[2] = Number of channels allocated by the crunch table. (A zero returned here indicates that a crunch file has not been loaded.)

a[3] = Total number of events for the current run.

a[4] = Number of rejects (uncorrelated events). Each event labeled as a reject is discarded and forces the interface buffer to be cleared. A non-zero number here may indicate a hardware interface problem.

a[5] = Number of overflows found during PSD analysis. If an event is above the range of the PSD parameter, the event is discarded and recorded here.

a[6] = Number of overflows found in crunching the PH1 analyzer data.

a[7] = Number of overflows found in crunching the PH2 analyzer data.

a[8] = Number of overflows found in crunching the TOF analyzer data.

a[9] = Number of underflows found during PSD discrimination. Any PSD event that is found in the first two channels of any PSD window is discarded and recorded here.

a[10]=	Number of events that have been calculated to fall above the maximum channel number. This would indicate a severe problem with the computer or software.
a[11]=	Number of non-applicable tags found during PSD analysis. If a tag is recorded that is not applicable to any PSD tags, no PSD analysis is performed and the event is recorded here. Crunching would continue.
a[12]=	Number of window overflows found during PSD analysis. If an event occurs above the given window range, no PSD analysis is performed and the event is recorded here. Crunching would continue.

<u>**(2) START DATA ACQUISITION:**</u> This command instructs the data acquisition driver to start or continue data acquisition. If a crunch file has not been loaded into computer memory or the system is already acquiring data, this command will be disregarded.

C program example:

```
long a[4];
a[0] = 2;                                    /* Start data acquisition */
write(device,(char*)&a[0],1);
```

Returned parameters:

a[0] = 11, if successful (collecting data).

<u>**(3) STOP DATA ACQUISITION:**</u> This command will stop data acquisition. If the system is already stopped, this command will be disregarded.

C program example:

```
long a[1];
a[0] = 3;                          /* Stop acquisition command */
write(device,(char*)&a[0],1);
```

Returned parameters:

a[0] = 10, if successful (system stopped)

<u>(4) ZERO ALL CHANNELS:</u> This command zeros all channels used for data storage. It also clears all scalers and zeros all overflow counters and event counters. This command will be ignored if a crunch table has not been loaded into computer memory.

C program example:

```
long a[1];
a[0] = 4;                              /* zero memory command  */
write(device,(char*)&a[0],1);
```

Returned parameters:

```
a[0] = 10, if successful
```

<u>(5) DEALLOCATE MEMORY:</u> This command down loads any crunch table that had been previously loaded into the device driver and frees all memory that was used for data storage.

C program example:

```
long a[1];
a[0] = 5;                              /* Deallocate memory command */
write(device,(char*)&a[0],1);
```

Returned parameters:

```
a[0] = 10, if successful
```

<u>(6) READ SCALERS:</u> This command is used to read the eight scalers connected to the interface. The data currently displayed on each scaler will be transferred into an array passed by the requesting program.

C program example:

```
long a[9];
a[0] = 6;                              /* Read scalers command  */
write(device,(char*)&a[0],1);
```

Returned parameters:

```
a[0] = 10, if successful
a[1] = scaler 1 counts
a[2] = scaler 2 counts
```

```
a[3] = scaler 3 counts
a[4] = scaler 4 counts
a[5] = scaler 5 counts
a[6] = scaler 6 counts
a[7] = scaler 7 counts
a[8] = scaler 8 counts
```

(7) LOAD IN CHANNEL DATA: This command will load data into the acquisition driver from an array passed by a program. The user must specify the starting channel and number of channels that will be transferred. The maximum number of channels that may be transferred at one time is 16382 channels (64k bytes). Thus, to transfer more channels will require that this command be used more than once. If the last channel extends beyond the maximum channel, no channels will be transferred.

C program example:

```
long a[1006];
a[0] = 7;                  /* Load channel data command      */
a[1] = 0                   /* Start with channel zero        */
a[2] = 1000;               /* Transfer 1000 channels         */
a[3] = total events;
a[4] = rejects;
a[5] = run time;
a[6] = channel 0 data;     /* start of data to be loaded     */
a[7] = channel 1 data;
a[8] = channel 2 data;
a[9] = channel 3 data;
a[10] = channel 4 data;
a[11] = channel 5 data;
a[12] = channel 6 data;
      .
      .
a[1005] = channel 999 data;
write(device,(char*)&a[0],1);
```

Returned parameters:

```
a[0] = number of channels transferred
```

<u>(8) GET CHANNEL DATA:</u> This command will transfer data from the acquisition driver to an array passed by the requesting program. The user must specify the starting channel and number of channels that will be transferred. The maximum number of channels that may be transferred at one time is 16382 channels (64k bytes). Thus, to transfer more channels will require that this command be used more than once. This command may be used while the acquisition driver is in the acquire mode without affecting data accumulation. If the last channel extends beyond the maximum channel, no channels will be transferred.

C program example:

```
long a[1003];
a[0] = 8;                 /* Get channel data command */
a[1] = 0;                 /* Start with channel zero    */
a[2] = 1000;              /* Transfer 1000 channels     */
write(device,(char*)&a[0],1);
```

Returned parameters:

```
a[0] = number of channels transferred
a[1] = not used
a[2] = not used
a[3] = channel 0 data
a[4] = channel 1 data
a[5] = channel 2 data
a[6] = channel 3 data
a[7] = channel 4 data
a[8] = channel 5 data
a[9] = channel 6 data
    .
    .
a[1001] = channel 998 data
a[1002] = channel 999 data
```

(9) LOAD IN PSD BIAS MARKERS: This command allows the user to set the PSD bias marker for each window when using the PSD discrimination mode. For each window there is one PSD bias marker which represents a channel in that window. If any event occurs on or above the bias marker for that specific window, the PSD tag value will be added to the tag register and all crunching for that event will then use the summed tag value. If PSD mode is disabled or a crunch table has not been loaded into computer memory, this command will still set the PSD bias markers; however, they will not be used until a new crunch table is reloaded.

C program example:

```
long a[100];
a[0] = 9;                    /* Load bias markers command */
a[1] = PSD bias channel for window 1
a[2] = PSD bias channel for window 2
a[3] = PSD bias channel for window 3
a[4] = PSD bias channel for window 4
a[5] = PSD bias channel for window 5
a[6] = PSD bias channel for window 6
a[7] = PSD bias channel for window 7
          .

          .
write(device,(char*)&a[0],1);
```

Returned parameters:

a[0] = 10, if successful

(10) SET OUTPUT LOGIC SIGNALS: This command allows the user to set 32 DC logic signals to the output port located on the rear of the interface buffer. Each logic signal is approximately +5 volts when high and grounded when low.

C program example:

```
long a[2];
a[0] = 15;                   /* Set output logic signals      */
a[1] = 1 + 2 + 4 + 256;      /* Set lines 1,2,3, & 9 high     */
write(device,(char*)&a[0],1);
```

Returned parameters:

a[0] = 10 if successful

APPENDIX A
ANALYZER COMMANDS

Axxx	Add **xxx** channels to the base to shift the current display.
Bxxx	Display channels starting at Base channel **xxx**.
Cxxx	Compare by overlapping channels starting at channel **xxx**.
Ixxx yyy	Integrate / Sum the channels starting at **xxx** with **yyy** being the number of channels integrated. A period may be used in place of **xxx** to represent the cursor channel (i.e. I.4096).
L FILE.TBL	Load a new crunch file with the name of FILE.TBL.
O	Display overflows.
PC	Print out the crunch table.
PS	Print out scalers, overflows, and runtime.
Pxxx yyy	Print out the counts in each channel starting at **xxx** with **yyy** being the number of channels printed. A period may be used in place of **xxx** to represent the cursor channel (i.e. P.1024).
R	Display total events, rejects, and count rate.
Sxxx	Subtract **xxx** channels to the base to shift the current display.
Txxx	Set the preset timer to **xxx** seconds (0 = infinity).
Txxx+FILE	Set the preset timer to **xxx** seconds. After time out occurs, the data will be saved under the name FILE.000, FILE.001, etc. and then cleared and restarted (i.e. T1800+FUSION).
W	Print out the PSD bias marker positions.

APPENDIX B
EXAMPLES OF CRUNCH FILES

EXAMPLE 1: This crunch file will only store the first 2048 channels coming from the first pulse height ADC (PH1). No crunching of channels is performed. No tags are used. PSD mode is disabled.

```
SECTION 1
PARAMETERS 1
PH1
2048,1
TAGS: 0
```

EXAMPLE 2: This crunch file will store 8192 channels coming from the second pulse height ADC (PH2) and crunch them into 512 channels, using a 16 channel crunch.

```
SECTION 1
PARAMETERS 1
PH2
512,16
TAGS: 0
```

EXAMPLE 3: This crunch file will look at data coming from the time digitizer only. It crunches the first 1000 channels into one channel. Then the next 2048 channels are crunched into 512 channels using a 4 channel crunch. It then crunches the next 4096 channels into 512 channels using a 8 channel crunch and so forth. This crunch table will allocate 4097 channels of computer memory (16388 bytes). Each channel coming from the time digitizer clock is assumed to be one nanosecond in width. No tags are used here.

```
SECTION 1
PARAMETERS 1
TOF
1,1000
512,4
512,8
512,16
512,32
1024,64
1024,128
TAGS: 0
```

EXAMPLE 4: This crunch file performs two dimensional (2 parameter) storage using data from the one pulse height ADC as one of the parameters and data from the time digitizer as the other parameter. This file will allocate 2048 x 37 (75776) channels or 303104 bytes of computer memory. No tags are used here. The PSD mode is disabled.

```
SECTION 1
PARAMETERS 2
PH1
2048,4
TOF
1,5000
4,128
8,256
8,512
16,1024
TAGS: 0
```

EXAMPLE 5: This file enables tag inputs 1, 2, and 3 on the time digitizer. It will store data from the one pulse height ADC and the time digitizer as a function of the tag register. If two tags are recorded for one event their value will be summed into the tag register. For example, if TAG#1 (value 1) and TAG#3 (value 4) are recorded during the same event, the tag for that event will be 5. A tag value of 5 will not be recorded anywhere using this crunch table.

```
TAG#1 YES
TAG#2 YES
TAG#3 YES
TAG#4 NO

SECTION 1
PARAMETERS 1
PH1
2048,1
TAGS: 1,2,4

SECTION 2
PARAMETERS 1
TOF
1,5000
4,128
8,256
8,512
16,1024
TAGS: 1,2,4
```

EXAMPLE 6: This crunch file enables tag inputs 1, 2, and 3. It uses the pulse shape discrimination mode (PSD mode). Each PSD window uses 512 channels from PH2 crunched into 128 channels. The crunch factor in the PSD section may only be factors of 2 (ie. 2,4,8,16,32,64). The PSD mode uses window parameter PH1 divided into windows of 64,64,128,... channels. Note that section 4 is used to record the total number of events as a function of tag only.

```
TAG#1:  YES
TAG#2:  YES
TAG#3:  YES
TAG#4:  NO

PSD MODE .............. ON
PSD PARAMETER ......... PH2
WINDOW PARAMETER ...... PH1
NUMBER OF CHANNELS .... 128
CRUNCH FACTOR ......... 4
VALUE ADDED TO TAG .... 100
APPLICABLE TAGS ....... 1,2,4
WINDOWS (channel width)  64,64,128,256,512,1024,1024,1024,
  2048,2048

SECTION 1
PARAMETERS 1
PH1
2048,4
TAGS:  1,2,4,101,102,104

SECTION 2
PARAMETERS 1
TOF
2048,4
TAGS:  1,2,4,101,102,104

SECTION 3
PARAMETERS 2
PH1
1024,8
TOF
1,950
8,32
8,48
8,64
8,96
8,160
8,256
TAGS:  1,2,4,101,102,104

SECTION 4
PARAMETERS 1
PH1
1,8192
TAGS: 0,1,2,3,4,5,6,7,100,101,102,103,104,105,106,107
```

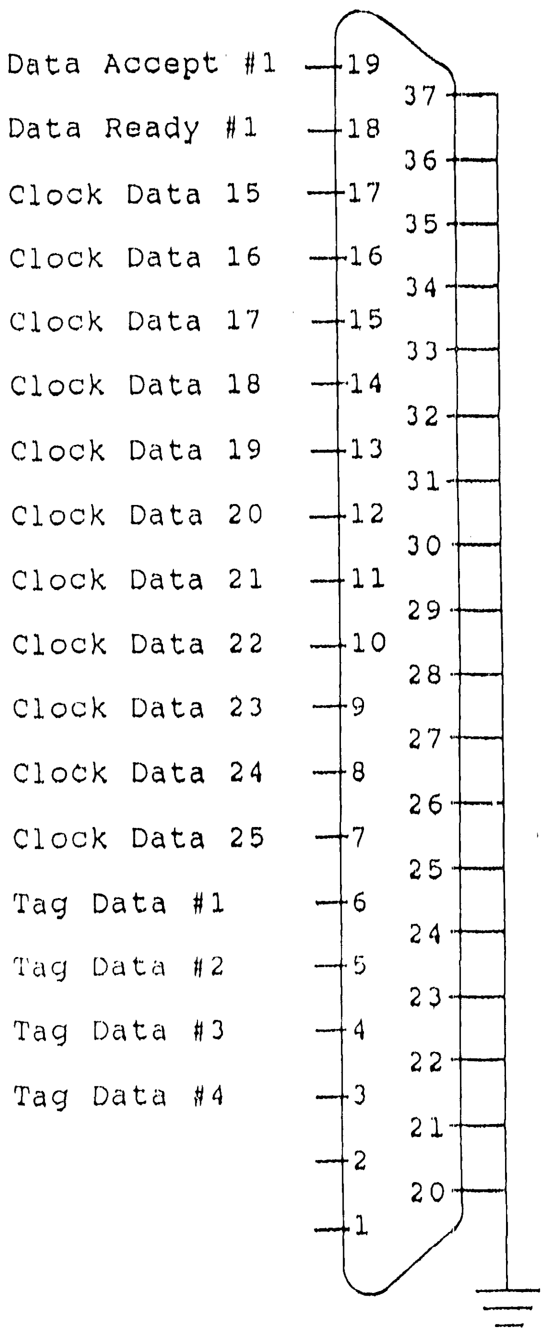
APPENDIX C
EXAMPLE PROGRAM

```
/*==============================================================================
=                                                                              =
=   This is an example of a protected mode program written in Microsoft C to   =
=                        illustrate the following steps:                       =
=                                                                              =
=                        1) Load a crunch file                                 =
=                        2) Zero all channels and scalers                      =
=                        2) Start acquiring data                               =
=                        3) Stop acquiring data                                =
=                        4) Print out channels 100 thru 119                    =
=                                                                              =
=   Compile instruction:  CL /AL /Lp /FPi87 EXAMPLE.C /link /NOD LLIBC7P+DOSCALLS  =
=                                                                              =
==============================================================================*/

   #include <io.h>
   #include <stdio.h>
   #include <process.h>
   int device, i;
   unsigned long a[200];

main ()
{
   /*  load crunch file D:\CRUNCH.TBL  */
   spawnlp(P_WAIT,"C:\\LOADCRUN","C:\\LOADCRUN","D:\\CRUNCH.TBL",NULL);
   printf("\n");

   device = open("device_1",0x0002);                        /* open device driver */

   a[0] = 4;                                        /* zero all channels  */
   write(device,(char*)&a[0],1);
   printf("System zeroed:\n");

   a[0] = 2;                                        /* start acquiring data */
   write(device,(char*)&a[0],1);
   printf("System started:\n");

   printf("Hit any key to stop and print out channels 100 thru 119 ...\n\n");
   while (!kbhit());

   a[0] = 3;                                        /* stop acquiring data */
   write(device,(char*)&a[0],1);
   printf("System stopped:\n");

   a[0] = 8;                                        /* get channel data          */
   a[1] = 100;                              /* starting with channel 100 */
   a[2] = 20;                               /* transfer 20 channels      */
   write(device,(char*)&a[0],1);
   for (i=3; i<23; i++) printf("Chan %d = %ld\n",i+97,a[i]);

   exit(0);
}
```

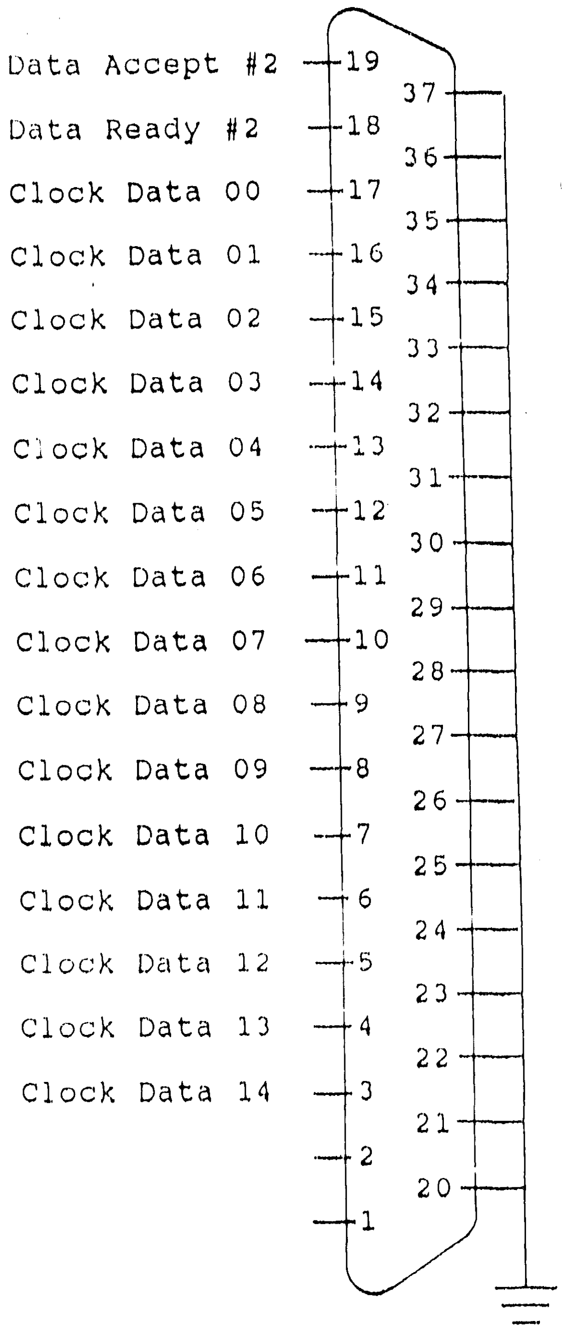
APPENDIX D

REAR PANEL PIN CONNECTIONS

CN4 CONNECTOR (WORD #1):



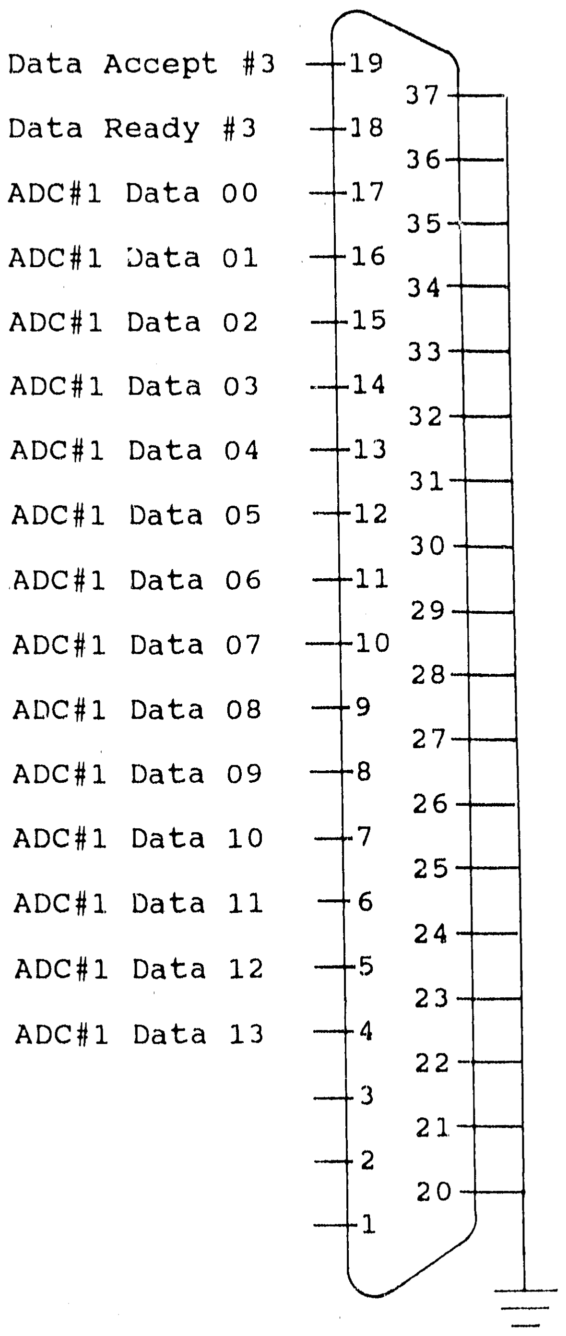
CN5 CONNECTOR (WORD #2):



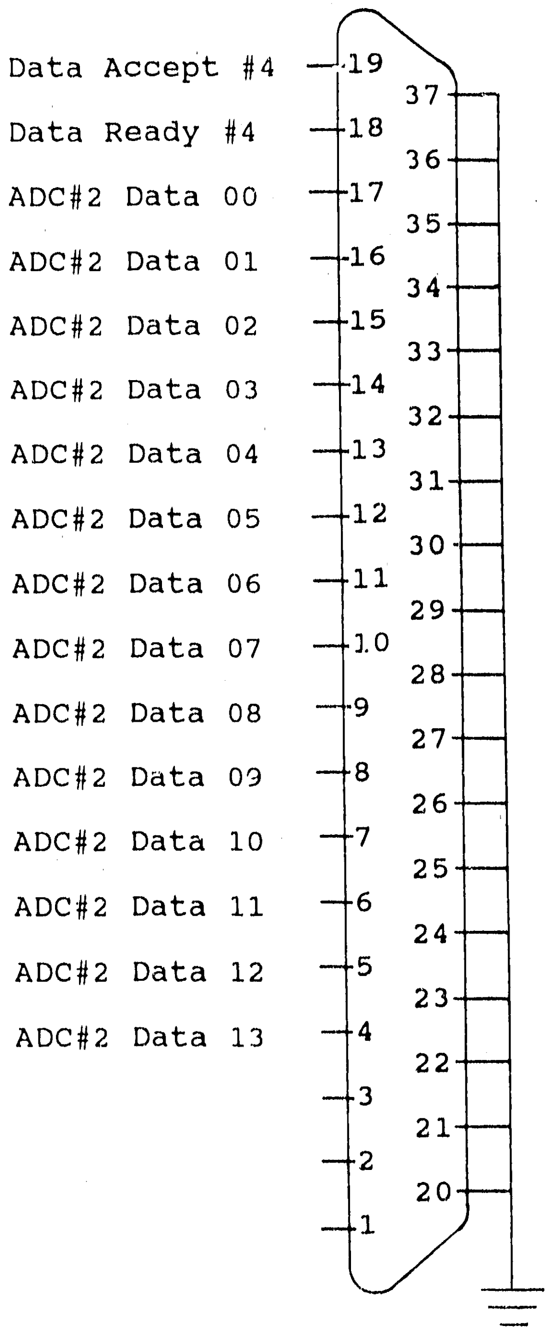
APPENDIX D - Continued

REAR PANEL PIN CONNECTIONS

CN6 CONNECTOR (WORD #3):



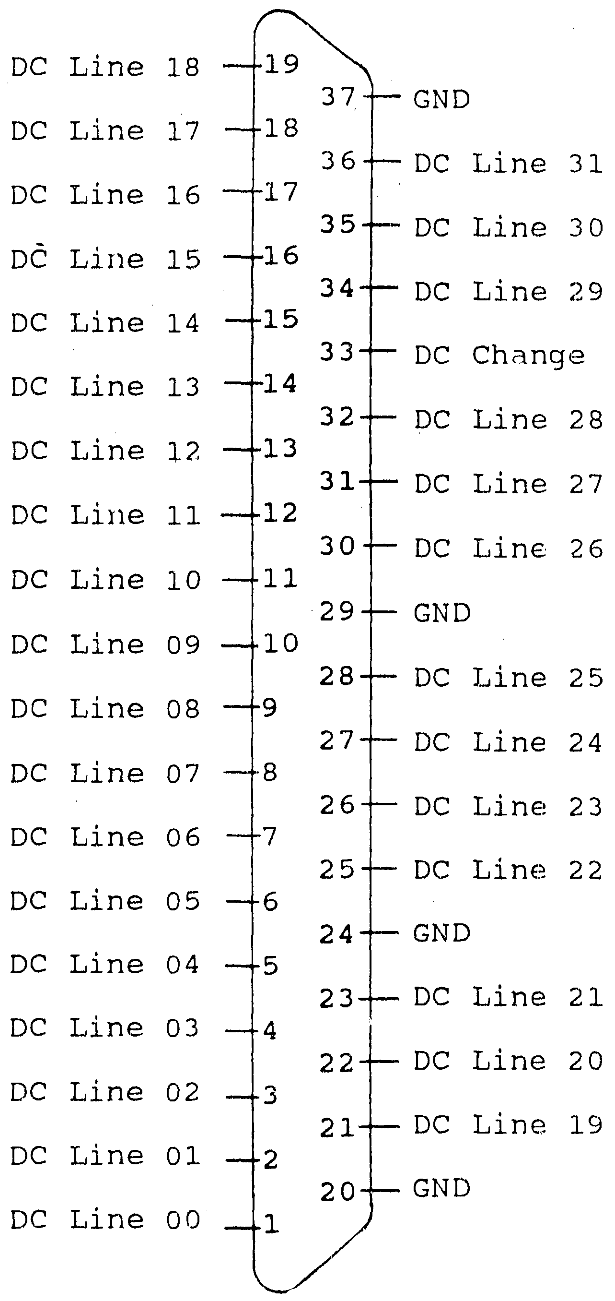
CN7 CONNECTOR (WORD #4):



APPENDIX D - Continued

REAR PANEL PIN CONNECTIONS

CN8 PIN CONNECTIONS (D.C. OUTPUT)



APPENDIX D - Continued

REAR PANEL PIN CONNECTIONS

CN9 PIN CONNECTIONS (SCALER INPUT)

Data Input 00	-	A	Data Input 25	-	d
Data Input 01	-	B	Data Input 26	-	e
Data Input 02	-	C	Data Input 27	-	f
Data Input 03	-	D	Data Input 28	-	h
Data Input 04	-	E	Data Input 29	-	j
Data Input 05	-	F	Data Input 30	-	k
Data Input 06	-	H	Data Input 31	-	m
Data Input 07	-	J	OVERLW	-	n
Data Input 08	-	K	Cntrl Line 00	-	p
Data Input 09	-	L	Cntrl Line 01	-	r
Data Input 10	-	M	Cntrl Line 02	-	s
Data Input 11	-	N	Cntrl Line 03	-	t
Data Input 12	-	P	Cntrl Line 04	-	u
Data Input 13	-	R	Cntrl Line 05	-	v
Data Input 14	-	S	Cntrl Line 06	-	w
Data Input 15	-	T	Cntrl Line 07	-	x
Data Input 16	-	U	Cntrl Line 08	-	y
Data Input 17	-	V	Cntrl Line 09	-	z
Data Input 18	-	W	Cntrl Line 10	-	AA
Data Input 19	-	X	Cntrl Line 11	-	BB
Data Input 20	-	Y	Cntrl Line 12	-	CC
Data Input 21	-	Z	Cntrl Line 13	-	DD
Data Input 22	-	a	Cntrl Line 14	-	EE
Data Input 23	-	b	Cntrl Line 15	-	FF
Data Input 24	-	c	GND	-	HH

APPENDIX E
DATA HANDLER LOW LEVEL I/O COMMANDS

Port C Output	Description
1	Scaler Data: Places data bus (ports A & B of the interface card) on port 2 (Scalers) of the data handler. This removes the data bus from the FIFO memory and from the test data input.
2	Scaler Stop: This generates a +12 volt pulse at the rear panel BNC connector labeled Scaler Stop.
3	Scaler Reset: This generates a +12 volt pulse at the rear panel BNC connector labeled Scaler Reset.
4	Scaler Start: This generates a +12 volt pulse at the rear panel BNC connector labeled Scaler Start.
5	FIFO Memory: Places data bus (ports A & B of the interface card) on port 1 (FIFO Memory) of the data handler. This removes the data bus from the Scalers and from the test data input.
6	Test Data Input: Places data bus (ports A & B of the interface card) onto the test data input circuit of the data handler. This requires re-configuring ports A & B as write ports. This command also removes the data bus from the Scalers and turns off data input from external world.
7	Master Reset: Performs a master reset and clears all data from FIFO memory. This also removes the data bus (ports A & B) from any input port.
9	Scaler Step: This steps through the control lines going to port 2 of the data handler. If attached to the scalers, it advances the scaler bus by one half scaler.
10	Accept Data: This enables the data handler to start accepting and storing data from each applicable digitizer.
11	Block Data: This disables the data handler from accepting and storing data from any digitizer.
13	False Data Ready: This command provides a false data ready signal used to input test data into the FIFO memory.
14	D.C. Line Monitor: This generates a pulse at the rear panel BNC connector labeled D.C. Line Monitor.
15	Disable Data Bus: This removes the data bus (ports A & B of the interface card) from everything.

APPENDIX F
SOURCE CODE FOR DATA ACQUISITION DRIVER

```
Name    DEVICE2
Title  'DATA ACQUISITION DEVICE DRIVER - VERSION II'

;-----------------------------------------------------------------------------
;Compile example:
;       MASM DEVICE2.ASM;
;       LINK DEVICE2.OBJ,C:\DEVICE2.SYS,,DOSCALLS.LIB,DEVICE2.DEF;
;
;DEVICE2.DEF listing:
;       LIBRARY DEVICE2
;       PROTMODE
;       CODE PRELOAD
;       DATA PRELOAD
;-----------------------------------------------------------------------------

PhysToVirt      equu    15h
UnPhysToVirt    equ     32h
AllocPhys       equ     18h
VirtToPhys      equ     16h
AllocGDT        equ     2Dh
PhysToGDT       equ     2Eh
FreePhys        equ     19h
SetTimer        equ     1Dh
TickCount       equ     33h
ResetTimer      equ     1Eh

CFG1            equ     0D000h            ; MC-DIO-32F port address
CFG2            equ     0D002h
STAT            equ     0D004h
PORT1           equ     0D006h
PORT2           equ     0D008h

extrn   DOSWRITE:far

;------------------------- DEVICE DATA SEGMENT -----------------------

DGROUP  group   _DATA
_DATA   segment word public 'DATA'

                                        ; device drive header...
header  dd      -1                      ; link to next device driver
        dw      8880h                   ; device attribute word
        dw      Strat                   ; "Strategy" routine entry point
        dw      0                       ; (reserved)
        db      'DEVICE_2'              ; logical device name
        db      8 dup (0)               ; (reserved)

var1    dw      4 dup (0)
devhlp  dd      ?                       ; DevHlp entry point
tem_ax  dw      0                       ; data storage
tem_bx  dw      0
running dw      0
runtime dd      0
starttm dd      0
working dw      0
liveoff dw      0
numseg  dw      0
numword dw      0
numcrun dw      0
mem_add dd      0
word1   dw      0
itag    dw      0
win     dw      0
tand    dw      0
```

```
section dw      0
chan    dd      0
maxchn  dd      0
kmax    dw      0
anal    dw      0
nh      dw      0
tevent  dd      0
reject  dd      0
overf0  dd      0
overf1  dd      0
overf2  dd      0
overf3  dd      0
overf4  dd      0
overf5  dd      0
overf6  dd      0
overf7  dd      0
psdmem  dd      0
saveit  dw      0
qwcrt   dw      0
vedi    dd      0
vesi    dd      0
veax    dd      0
vebx    dd      0
vecx    dd      0
vedx    dd      0
parm    dd      5 dup (0)
psd     dd      200 dup (0)
psdm    dd      100 dup (512)
crun    dd      1000  dup (0)
GDT     dw      0

wlen    dw      ?                       ; receives DOSWRITE length

ident   db      13,10
        db      'Device driver for ANALYZER II installed.'
        db      13,10

ident_len equ $-ident

END_DS  EQU     $
_DATA   ends

;-------------------------- CODE SEGMENT -------------------------------

_TEXT   segment word public 'CODE'
        assume cs:_TEXT,ds:DGROUP,es:NOTHING
        .386P

Strat   proc    far                     ; device driver Strategy routine,
        push    es                      ; called by OS/2 kernel with
        push    ebx                     ; ES:BX = address of request packet
        push    eax
        push    ecx
        push    edx

        mov     di,es:[bx+2]            ; get command code from packet
        and     di,0FFh

S1:     cmp     di,8
        jne     S2
        cli
        call    Write                   ; write statement executed
        sti
        jmp     Exit

S2:     cmp     di,13
        jne     S3
        cli
        call    Open                    ; open device driver
        sti
        jmp     Exit

S3:     cmp     di,0
        jne     Exit
        call    Install                 ; initialize device driver

Exit:   pop     edx
        pop     ecx
        pop     eax
        pop     ebx
        pop     es
        mov     es:[bx+3],WORD PTR 0100h  ; return with no problem
        ret

Strat   endp

;----------------------- Initialize Interface Card ---------------------

Open    proc    near
        cmp     running,1                  ; return if acquiring data
        je      Opend

        mov     dx,CFG1                    ; set ports A&B for handshaking
        mov     ax,0100h                   ;  in read and pulse mode
        out     dx,ax
        call    delay
        mov     ax,0000h
        out     dx,ax
        call    delay
        mov     ax,0610h
        out     dx,ax
        call    delay

        mov     dx,CFG2                    ; set ports C&D for handshaking
        mov     ax,0100h                   ;  in write mode
        out     dx,ax
        call    delay
        mov     ax,0000h
        out     dx,ax
        call    delay
        mov     ax,0600h
        out     dx,ax
        call    delay
        mov     ax,0620h
        out     dx,ax
        call    delay

        mov     dx,PORT2                   ; bus disabled
        mov     ax,15
        out     dx,ax
        call    delay

        mov     ax,7                       ; reset FIFO
        out     dx,ax
        call    Delay

        mov     dx,PORT1                   ; remove anything from PORT1
        in      ax,dx
Opend:  ret

Open    endp

;----------------------- DECODE WRITE COMMAND -------------------------

Write   proc    near

        mov     ax,es:[bx+16]           ; put address of string in es:bx
        mov     bx,es:[bx+14]
        mov     tem_ax, ax
        mov     tem_bx, bx
        mov     cx,0
        mov     dh,1
        mov     dl,PhysToVirt
        call    devhlp                  ; virt address now in es:di
        mov     bx,di

        mov     al,es:[di]

Write0: cmp     al,217                  ; stop and reload crunch
        jne     Write1
        call    Init
        jmp     Wexit

Write1: cmp     al,1                    ; check status
        jne     Write2
        call    Check
        jmp     Wexit

Write2: cmp     al,2                    ; start
        jne     Write3
        call    Start
        jmp     Wexit

Write3: cmp     al,3                    ; stop
        jne     Write4
        call    Stop
        jmp     Wexit

Write4: cmp     al,4                    ; zero everything
        jne     Write5
        call    Reset
        jmp     Wexit

Write5: cmp     al,5                    ; deallocate all memory
        jne     Write6
        call    Devcls
        jmp     Wexit

Write6: cmp     al,6                    ; read scalers
        jne     Write7
        call    Scaler
        jmp     Wexit

Write7: cmp     al,7                    ; load in data
        jne     Write8
        call    Load
        jmp     Wexit

Write8: cmp     al,8                    ; get data
        jne     Write9
        call    Get
        jmp     Wexit

Write9: cmp     al,9                    ; load in psd bias markers
        jne     Write15
        call    Marker
        jmp     Wexit

Write15: cmp    al,15                   ; output to dc connector
        jne     Wexit
        call    OutDC
        jmp     Wexit

Wexit:  mov     dl,UnPhysToVirt
        call    devhlp
        ret

Write   endp

;------------------------ CHECK DRIVER STATUS --------------------------
```

```
Check   proc    near
        mov     es:[di],dword ptr 10    ; device not running
        cmp     running,1
        jne     Check2
        mov     es:[di],dword ptr 11    ; device running

        cmp     liveoff,1
        je      Check2
        call    Gtime                   ; update runtime
        mov     ebx,starttm
        cmp     eax,ebx
        jge     Check1
        add     eax,604800
Check1: sub     eax,ebx
        mov     runtime,eax

Check2: mov     eax,runtime
        mov     es:[di+4],eax           ; return runtime A(1)

        mov     eax,maxchn
        mov     es:[di+8],eax           ; return crunch chan A(2)

        mov     eax,tevent
        mov     es:[di+12],eax          ; return total events A(3)

        mov     eax,reject
        mov     es:[di+16],eax          ; return rejects A(4)

        mov     eax,overf0
        mov     es:[di+20],eax          ; return PSD overflows A(5)

        mov     eax,overf1
        mov     es:[di+24],eax          ; return PH1 overflows A(6)

        mov     eax,overf2
        mov     es:[di+28],eax          ; return PH2 overflows A(7)

        mov     eax,overf3
        mov     es:[di+32],eax          ; return TOF overflows A(8)

        mov     eax,overf4
        mov     es:[di+36],eax          ; return PSD underflows A(9)

        mov     eax,overf5
        mov     es:[di+40],eax          ; return maxchn overflows A(10)

        mov     eax,overf6
        mov     es:[di+44],eax          ; return tag overflows A(11)

        mov     eax,overf7
        mov     es:[di+48],eax          ; return window overflows A(12)

        ret
Check   endp

;------------------------ START DATA ACQUISTION ------------------------

Start   proc    near
        cmp     maxchn,0
        je      Startd
        cmp     running,1
        je      Startd

        mov     es:[di], dword ptr 11   ; return 11 (running)
        mov     liveoff,0

        call    open                    ; initialize I/O board

        mov     dx,CFG1                 ; test for number of words per event
        mov     ax,0100h
```

```
          out     dx,ax
          call    delay
          mov     ax,0000h
          out     dx,ax
          call    delay
          mov     ax,0021h
          out     dx,ax
          call    delay
          mov     dx,PORT2
          mov     ax,7
          out     dx,ax
          call    delay
          mov     ax,6
          out     dx,ax
          call    delay
          mov     ax,13
          out     dx,ax
          call    delay
          mov     ax,0
          out     dx,ax
          call    delay
          mov     ax,7
          out     dx,ax
          call    delay
          mov     ax,6
          out     dx,ax
          call    delay
          mov     cx,1
Loop1:    mov     dx,PORT1
          mov     ax,cx
          out     dx,ax
          call    delay
          mov     dx,CFG1
          mov     ax,0023h
          out     dx,ax
          call    delay
          mov     ax,0021h
          out     dx,ax
          call    delay
          add     cx,1
          cmp     cx,5
          jl      Loop1
          mov     dx,PORT2
          mov     ax,13
          out     dx,ax
          call    delay
          mov     ax,0
          out     dx,ax
          call    delay
          mov     dx,CFG1
          mov     ax,0100h
          out     dx,ax
          call    delay
          mov     ax,0000h
          out     dx,ax
          call    delay
          mov     ax,0610h
          out     dx,ax
          call    delay
          mov     dx,PORT2
          mov     ax,5
          out     dx,ax
          call    delay
          mov     dx,PORT1
          in      ax,dx
          call    delay
          in      ax,dx
          call    delay
          in      ax,dx
          call    delay
```

```
        in      ax,dx
        call    delay
        and     ax,15
        mov     numword,ax

        mov     dx,PORT2
        mov     ax,5
        out     dx,ax
        call    delay

        mov     dx,PORT2
        mov     ax,7
        out     dx,ax
        call    delay

        mov     dx,PORT1
        in      ax,dx
        call    delay

        mov     dx,PORT2
        mov     ax,10                   ; enable data in
        out     dx,ax
        call    Delay

        mov     dx,PORT2
        mov     ax,4                    ; start scalers
        out     dx,ax
        call    Delay

        call    Gtime                   ; get start time
        mov     ebx,runtime
        cmp     eax,ebx
        jge     Start1
        add     eax,604800
Start1: sub     eax,ebx
        mov     starttm,eax

        mov     ax,offset cs:intr       ; pointer to timer handler
        mov     dl,SetTimer
        call    devhlp

        mov     running,1
Startd: ret

Start   endp

;------------------------- STOP DATA ACQUISTION ------------------------

Stop    proc    near
        mov     es:[di], word ptr 10            ; return 10 (stopped)
        cmp     running,0
        je      Stopd

        cmp     liveoff,1                       ; was livetime off?
        jne     Stop0
        mov     liveoff,0
        call    Gtime                           ; get start time
        mov     ebx,runtime
        cmp     eax,ebx
        jge     tart1
        add     eax,604800
tart1:  sub     eax,ebx
        mov     starttm,eax

Stop0:  mov     ax,11                           ; disable data in
        mov     dx,PORT2
        out     dx,ax
        call    Delay

        mov     ax,2                            ; stop scalers
        out     dx,ax
        call    Delay

        mov     ax,offset cs:intr               ; remove timer handler
        mov     dl,ResetTimer
        call    devhlp

        mov     ax,7                            ; reset FIFO
        out     dx,ax
        call    Delay

        mov     running,0

        call    Gtime                           ; update runtime
        mov     ebx,starttm
        cmp     eax,ebx
        jge     Stop1
        add     eax,604800
Stop1:  sub     eax,ebx
        mov     runtime,eax

Stopd:  ret

Stop    endp

;------------------------ ZERO CHANNELS & SCALERS -----------------------

Reset   proc    near
        cmp     maxchn,0
        je      Resetd

        mov     es:[di], word ptr 10            ; return 10 (reset O.K.)
        call    Clrmem

Resetd: ret

Reset   endp

;------------------------- FREE ALLOCATED MEMORY ------------------------

Devcls  proc    near

        mov     es:[di], dword ptr 10           ; return 10 for success
        call    stop                            ; stop everything

        cmp     maxchn,0
        je      Devd

        mov     dl,UnPhysToVirt
        call    devhlp

        mov     bx,word ptr mem_add             ; free all memory
        mov     ax,word ptr mem_add+2
        mov     dl,FreePhys
        call    devhlp
        mov     maxchn,0
Devd:   ret

Devcls  endp

;------------------------- CLEAR EXTENDED MEMORY ------------------------

Clrmem  proc    near

        mov     tevent,0                        ; clear counters
        mov     reject,0
        mov     overf0,0
        mov     overf1,0
        mov     overf2,0
        mov     overf3,0
        mov     overf4,0
        mov     overf5,0
        mov     overf6,0
        mov     overf7,0
        mov     runtime,0

        mov     dx,PORT2                        ; clear scalers
        mov     ax,3
        out     dx,ax

        cmp     maxchn,0                        ; return if no memory
        jg      Clr1
        ret

Clr1:   mov     ax,numseg
        mov     saveit,ax               ; save number of segments to clear

zero1:  mov     ax,word ptr mem_add+2           ; ax:bx 32 physical address
        mov     bx,word ptr mem_add
        dec     saveit
        add     ax,saveit
        mov     cx,0
        mov     dh,1
        mov     dl,PhysToVirt
        call    devhlp                          ; virt address now in es:di

        mov     cx,04000h
zero2:  mov     es:[di],dword ptr 0
        add     di,4
        loop    zero2
        cmp     saveit,0
        jg      zero1

        mov     dl,UnPhysToVirt
        call    devhlp
        ret

Clrmem  endp

;----------------------- READ SCALERS ---------------------------------

Scaler  proc    near
        mov     es:[di], dword ptr 10

        cmp     running,0                       ; empty FIFO if running
        je      SCA1

        mov     dx,PORT2                        ; disable FIFO bus
        mov     ax,15
        out     dx,ax
        call    delay

        mov     dx,STAT                         ; check REG1
        in      ax,dx
        and     ax,32
        jz      SCA1
        mov     dx,PORT1                        ; save word1
        in      ax,dx
        mov     word1,ax

SCA1:   mov     dx,PORT2                        ; enable scaler bus
        mov     ax,1
        out     dx,ax
        call    Delay
        call    Delay

        mov     cx,8
SCA3:   add     di,4
        mov     dword ptr es:[di],0
        mov     dx,PORT1
        in      ax,dx
        mov     dx,ax

        mov     bx,dx
        shr     bx,12
        and     ebx,01111b
        mov     eax,10000000
        imul    eax,ebx
        add     dword ptr es:[di],eax

        mov     bx,dx
        shr     bx,8
        and     ebx,01111b
        mov     eax,1000000
        imul    eax,ebx
        add     dword ptr es:[di],eax

        mov     bx,dx
        shr     bx,4
        and     ebx,01111b
        mov     eax,100000
        imul    eax,ebx
        add     dword ptr es:[di],eax

        mov     bx,dx
        and     ebx,01111b
        mov     eax,10000
        imul    eax,ebx
        add     dword ptr es:[di],eax

        mov     dx,PORT2
        mov     ax,9
        out     dx,ax
        call    Delay
        mov     dx,PORT1
        in      ax,dx
        mov     dx,ax

        mov     bx,dx
        shr     bx,12
        and     ebx,01111b
        mov     eax,1000
        imul    eax,ebx
        add     dword ptr es:[di],eax

        mov     bx,dx
        shr     bx,8
        and     ebx,01111b
        mov     eax,100
        imul    eax,ebx
        add     dword ptr es:[di],eax

        mov     bx,dx
        shr     bx,4
        and     ebx,01111b
        mov     eax,10
        imul    eax,ebx
        add     dword ptr es:[di],eax

        mov     bx,dx
        and     ebx,01111b
        add     dword ptr es:[di],ebx
        cmp     dword ptr es:[di],99999999
        jle     SCA4
        mov     dword ptr es:[di],0

SCA4:   mov     dx,PORT2
        mov     ax,9
        out     dx,ax
        call    Delay
        dec     cx
        cmp     cx,0
        jg      SCA3

        mov     dx,PORT1                        ; clear PORT1
        in      ax,dx
        call    delay

        mov     dx,PORT2
        mov     ax,5                            ; enable FIFO bus
        out     dx,ax

        ret
Scaler  endp

;------------------------ LOAD DATA INTO MEMORY ------------------------

Load    proc    near
        mov     es:[di], dword ptr 0
        cmp     maxchn,0
        jg      Load0
Lex:    ret

Load0:  mov     eax,es:[di+12]          ; tevents
        mov     tevent,eax
        mov     eax,es:[di+16]          ; rejects
        mov     reject,eax
        mov     eax,es:[di+20]          ; runtime
        mov     runtime,eax

        mov     eax,es:[di+4]
        add     eax,es:[di+8]
        mov     ebx,maxchn
        add     ebx,3000
        cmp     eax,ebx
        jg      Lex

        mov     ecx,0
        mov     cx,es:[di+8]            ; cx = number of channels to trans
        mov     es:[di], ecx            ; return number of chan transfered
        cmp     cx,0
        je      Lex
        shl     cx,2
        mov     eax,es:[di+4]           ; starting channel
        shl     eax,2                   ; 4 bytes per channel
        add     eax,mem_add             ; add memory address
        mov     bx,ax                   ; bx = low
        shr     eax,16                  ; ax = high
        mov     dh,1                    ; put in es:di
        mov     dl,PhysToVirt
        call    devhlp

        clc
        push    ds
        mov     ax,tem_ax
        mov     bx,tem_bx
        add     bx,24
        adc     ax,0
        mov     dh,0
        mov     dl,PhysToVirt
        call    devhlp                  ; ds:si = address of source

        mov     bx,0
Load1:  mov     eax,ds:[si+bx]          ; source (exended memory)
        mov     es:[di+bx],eax          ; target memory
        add     bx,4
        cmp     bx,cx
        jl      Load1

        pop     ds
        ret
Load    endp

;------------------------  FETCH DATA FROM MEMORY  ------------------------

Get     proc    near
        mov     es:[di], dword ptr 0
        cmp     maxchn,0
        jg      Get0
Gex:    ret

Get0:   mov     eax,es:[di+4]
        add     eax,es:[di+8]
        mov     ebx,maxchn
        add     ebx,3000
        cmp     eax,ebx
        jg      Gex

        mov     ecx,0
        mov     cx,es:[di+8]            ; cx = number of channels to trans
        mov     es:[di],cx              ; return number of chan transfered
        cmp     cx,0
        je      Gex
        shl     cx,2
        mov     eax,es:[di+4]           ; starting channel
        shl     eax,2                   ; 4 bytes per channel
        add     eax,mem_add             ; add memory address
        mov     bx,ax                   ; bx = low
        shr     eax,16                  ; ax = high
        mov     dh,1                    ; put in es:di
        mov     dl,PhysToVirt
        call    devhlp

        clc
        push     ds
        mov     ax,tem_ax
        mov     bx,tem_bx
        add     bx,12
        adc     ax,0
        mov     dh,0
        mov     dl,PhysToVirt
        call    devhlp                  ; ds:si = address of target memory

        mov     bx,0
Get1:   mov     eax,es:[di+bx]          ; source (exended memory)
        mov     ds:[si+bx],eax          ; target memory
        add     bx,4
        cmp     bx,cx
        jl      Get1

        pop      ds
        ret
Get     endp

;-----------------------  LOAD PSD BIAS MARKERS  -------------------------

Marker  proc    near
        mov     es:[di], word ptr 10

        mov     bx,0                            ; transfer psd markers
        mov     si,offset ds:psdm
lpsdm:  mov     eax,es:[di+bx]
        mov     ds:[si+bx],eax
        add     bx,4
        cmp     bx,400
        jl      lpsdm

        ret
Marker  endp
```

```
;------------------------ LOAD CRUNCH TABLE ---------------------------

Init    proc    near

        call    Stop                            ; stop
        call    Devcls                          ; free all memory

        mov     ax,tem_ax                       ; reload crun[0] address
        mov     bx,tem_bx
        mov     cx,04000h
        mov     dh,1
        mov     dl,PhysToVirt
        call    devhlp

        mov     ax,es:[di+40]                   ; AND tag value - crun(10)
        mov     tand,ax

        mov     bx,0                            ; transfer psd array
        mov     si,offset ds:psd
lpsd:   mov     eax,es:[di+bx+2000]
        mov     ds:[si+bx],eax
        add     bx,4
        cmp     bx,1000
        jl      lpsd
        mov     eax,psd[0]                      ; start of psd memory
        mov     psdmem,eax

        mov     bx,0                            ; transfer crun array
        mov     si,offset ds:crun
lcrun:  mov     eax,es:[di+bx+4000]
        mov     ds:[si+bx],eax
        add     bx,4
        cmp     bx,4000
        jl      lcrun

        mov     eax,es:[di+4]                   ; eax = num of chan required
        mov     maxchn,eax
        add     eax,3000
        shr     eax,14
        add     ax,1                            ; ax = number of 64k seg
        mov     numseg,ax

        clc
        mov     bx,0                            ; allocate extended memory
        mov     ax,numseg
        mov     dh,0                            ; memory above 1 meg
        mov     dl,AllocPhys
        call    devhlp
        jnc     Init2                           ; jmp if allocated
        mov     maxchn,0

Init2:  mov     word ptr mem_add,bx             ; save starting mem physmem
        mov     word ptr mem_add+2,ax
        mov     bx,tem_bx
        mov     ax,tem_ax
        mov     cx,4
        mov     dh,1
        mov     dl,PhysToVirt
        call    devhlp
        mov     eax,maxchn
        mov     es:[di], eax                    ; return number allocated
        cmp      eax,0
        je       Initd
        call    Clrmem

        mov     ax,word ptr mem_add+2           ; put CHAN[0] in es:di
        mov     bx,word ptr mem_add
        mov     cx,0
        mov     dh,1
        mov     dl,PhysToVirt
```

```
call    devhlp

push    ds                              ; put address of es in ds:si
sgdt    var1
mov     ax,word ptr var1+4
mov     bx,es
add     bx,word ptr var1+2
mov     cx,0
mov     dh,0
mov     dl,PhysToVirt
call    devhlp

push    word ptr ds:[si]                ; save CHAN[0] descriptor
push    word ptr ds:[si+2]
push    word ptr ds:[si+4]
push    word ptr ds:[si+6]
pop     word ptr es:[di]
pop     word ptr es:[di+2]
pop     word ptr es:[di+4]
pop     word ptr es:[di+6]
pop     ds

mov     ax,es:[di]                      ; descriptor in overf
mov     word ptr overf0,ax
mov     ax,es:[di+2]
mov     word ptr overf1,ax
mov     ax,es:[di+4]
mov     word ptr overf2,ax
mov     ax,es:[di+6]
mov     word ptr overf3,ax

mov     dword ptr es:[di],0
mov     dword ptr es:[di+4],0

mov     ax,word ptr var1+4              ; put GDT descriptor in es:di
mov     bx,GDT
add     bx,word ptr var1+2
mov     cx,0
mov     dh,1
mov     dl,PhysToVirt
call    devhlp

mov     ax,word ptr overf3              ; modifiy decriptor and save
mov     es:[di],ax
mov     ax,word ptr overf2
mov     es:[di+2],ax
mov     ax,word ptr overf1
mov     es:[di+4],ax
mov     ax,word ptr overf0
or      ax,128
mov     es:[di+6],ax

mov     overf0,0
mov     overf1,0
mov     overf2,0
mov     overf3,0

mov     dx,PORT2                        ; disable data in
mov     ax,11
out     dx,ax
call    Delay

mov     ax,3                            ; zero scalers
out     dx,ax
call    Delay

mov     ax,15                           ; disable FIFO bus
out     dx,ax
call    Delay
```

```
        mov     ax,7                    ; reset FIFO
        out     dx,ax
        call    Delay

Initd:  ret
Init    endp

;------------------------ OUTPUT DC LOGIC -------------------------------

OutDC   proc    near
        mov     es:[di],dword ptr 10            ; return 10 to a[0]

        mov     dx,CFG2                         ; set OUT2 high
        mov     ax,0621h
        out     dx,ax
        call    delay

        mov     dx, PORT2                       ; output first 16 bits
        mov     ax,es:[di+4]
        out     dx,ax
        call    delay

        mov     ax,es:[di+6]                    ; output next 16 bits
        out     dx,ax
        call    delay

        mov     dx,CFG2                         ; set OUT2 low
        mov     ax,0620h
        out     dx,ax
        call    delay

        mov     dx, PORT2                       ; pulse P14 output
        mov     ax,14
        out     dx,ax

        ret
OutDC   endp

;----------------------- CRUNCH DATA & STORE IT --------------------------

Crunch  proc    near
        mov     numcrun,0
        jmp     short   Crun0

rejd:   inc     dword ptr reject            ; reject found in data
        mov     dx,PORT2
        mov     ax,11                       ; disable data in
        out     dx,ax
        call    Delay
        mov     ax,7                        ; clear FIFO
        out     dx,ax
        call    delay
        mov     dx,PORT1                    ; clear PORT1
        in      ax,dx
        call    delay
        mov     dx,PORT2
        mov     ax,10                       ; enable data in
        out     dx,ax
        call    Delay
        mov     ax,5                        ; enable FIFO bus
        out     dx,ax
Edn:    call    beep
Edone:  ret

Crun0:  inc     numcrun                        ; return if CR too fast
        cmp     numcrun,7000
        jg      Edn

        sti                                    ; enable interupts breifly
        nop
        cli

        mov     cx,numword
        mov     dx,STAT                         ; check DRDY1
        in      ax,dx
        and     ax,64
        jz      Edone                           ; ret if DRDY1 not set

;------  word 1  (most significant 16 bits of TOF clock & tags) ----------

        mov     dx,PORT1
        mov     ax,word1
        mov     word1,0
        cmp     ax,0
        jne     Crun1
        in      ax,dx                           ; read ports A & B
Crun1:  bt      ax,15                           ; copy bit 15 to carry flag
        jnc     rejd                            ; rej if carry flag not Set
        mov     bx,ax
        and     ax,0000011111111111b
        shl     eax,16
        shr     bx,11
        and     bx,tand
        mov     itag,bx                         ; word 5 (tags)

;------  word 2  (least significant 16 bits of TOF clock) ----------

        in      ax,dx                           ; read ports A & B
        shl     ax,1                            ; copy bit 15 to carry flag
        jc      rejd                            ; reject if carry bit set
        shr     eax,1
        mov     dword ptr parm[4],eax           ; words 1 & 2 (tof)
        cmp     cx,2
        je      stor

;------  word 3  (pulse height analyzer #1 PH1)  ----------

        in      ax,dx                           ; read ports A & B
        bt      ax,15                           ; copy bit 15 to carry flag
        jc      rejd                            ; reject if carry bit set
        and     ax,0001111111111111b
        mov     word ptr parm[8],ax
        cmp     cx,3
        je      stor

;------  word 4  (pulse height analyzer #2 PH2)  ----------

        in      ax,dx                           ; read ports A & B
        bt      ax,15                           ; copy bit 15 to carry flag
        jc      rejd                            ; reject if carry bit set
        and     ax,0001111111111111b
        mov     word ptr parm[12],ax

stor:   inc     dword ptr tevent                ; add 1 to total events

;--------------  CHECK FOR PSD TAG  ------------

        mov     si,offset ds:psd                ; ds:si = psd(0)
        cmp     dword ptr ds:[si],0
        je      CONT2                           ; if no psd

        mov     di,0                            ; DI = PCHAN channel
        mov     bx,40                           ; PSD(10)
        mov     WIN,1                           ; applicable window
        mov     ax,itag
        mov     cx,ds:[si+24]                   ; number of det/tags
TTAG:   mov     dx,ds:[si+bx]                   ; applicable tag?
        cmp     dx,ax
        je      DTAG
        add     bx,8
        add     di,word ptr ds:[si+32]          ; add channels per det/tag
        mov     dx,word ptr ds:[si+28]
        add     WIN,dx                          ; add number of windows/tag
        loop    TTAG
        inc     overf6                          ; no tag - continue crunch
        jmp     CONT2

DTAG:   mov     bx,ds:[si+8]
        shl     bx,2
        mov     ax,word ptr parm[BX]            ; window parameter channel

        mov     bx,120                          ; PSD(30)
        mov     cx,ds:[si+28]                   ; calculate which window
PTAG:   mov     dx,ds:[si+BX]                   ; window channel cutoff
        cmp     ax,dx
        jl      KTAG
        add     di,ds:[si+12]
        add     bx,4
        add     WIN,1
        loop    PTAG

        inc     overf7                          ; window overflow
        jmp     crun0
overfp: inc     overf0                          ; psd overflow
        jmp     crun0
under:  inc     overf4                          ; psd underflow
        jmp     crun0
under1: inc     overf5                          ; maxchn overflow
        call    beep
        jmp     crun0

KTAG:   mov     bx,ds:[si+4]                    ; PSD parameter
        shl     bx,2
        mov     ax,word ptr parm[bx]            ; PSD channel
        mov     cx,ds:[si+16]                   ; crunch factor
        cmp     cl,0
        je      KTAG1
        shr     ax,cl
KTAG1:  mov     bx,ds:[si+12]
        cmp     ax,bx
        jge     overfp
        cmp     ax,2                            ; psd underflow - reject
        jl      under
        jmp     YTAG

YTAG:   mov     bx,WIN                          ; add tag if right of marker
        shl     bx,2
        cmp     ax,word ptr psdm[bx]
        jl      short   STORE
        mov     cx,word ptr psd[20]             ; add if right of marker
        add     itag,cx

STORE:  add     di,ax                           ; di = PSD chan+prev windows
        mov     eax,0
        mov     ax,di                           ; PSDMEM start of PSD memory
        add     eax,psdmem                      ; eax = PSD memory channel

        cmp     eax,maxchn
        jg      bigerr
        cmp     eax,2
        jl      under1
        shl     eax,2                           ; increment channel eax
        inc     dword ptr es:[eax]

;------------ continue with crunch -----------

CONT2:  mov     si,offset ds:crun               ; ds:si = crun[1000]
        add     si,8
        mov     ax,ds:[si]                      ; number of crunch segments
        mov     section,ax
```

```
L2701:  add     si,4                         ; find start of next section
        bt      word ptr ds:[si+2],15
        jnc     L2701
        mov     chan,0

        add     si,4                         ; N = N + 1

        mov     di,0                         ; K loop
        mov     ax,ds:[si]
        mov     kmax,ax
L2702:  inc     di

        add     si,8                         ; EBX = IP
        mov     bx,ds:[si-4]
        shl     bx,2
        mov     anal,bx
        mov     ebx,dword ptr parm[bx]

        mov     ax,ds:[si]                   ; NH
        mov     cx,ax
        mov     dl,24
        mul     dl
        add     ax,si
        mov     nh,ax

L2703:  add     si,24
        mov     eax,ds:[si-16]
        cmp     ebx,eax
        jl      L3000
L2704:  loop    L2703

        mov     ax,ds:[si+4]
        shl     ax,3
        add     ax,4
        add     si,ax
        push    si                           ; increment overflow
        mov     si,offset ds:overf0
        add     si,anal
        inc     dword ptr ds:[si]
        pop     si
        jmp     L3100

L3000:  sub     ebx,ds:[si-20]
        mov     eax,ebx
        mov     ebx,ds:[si-4]
        cdq
        div     ebx

        add     eax,ds:[si-12]

        mov     edx,ds:[si]
        mul     edx

        add     chan,eax

        mov     si,nh

        mov     ax,kmax
        cmp     di,ax
        jl      L2702
        mov     di,itag

L2705:  add     si,4
        mov     cx,ds:[si]

TAGIT:  add     si,8
        mov     dx,ds:[si-4]         ; IG = DX
        cmp     dx,di
        jne     NEXTT
        mov     eax,chan                ; EAX = CHAN
        add     eax,dword ptr ds:[si]   ; add tag base
        cmp     eax,maxchn
        jg      bigerr

        shl     eax,2                   ; increment channel eax
        inc     dword ptr es:[eax]

L3100:  dec     section
        cmp     section,0               ; is this the last section?
        jg      L2701                   ; go do next section
        jmp     CRUNO                   ; go check STAT

NEXTT:  loop    TAGIT                   ; if more tags goto tagit
        jmp     L3100

bigerr: inc     overf5
        call    beep
        jmp     Crun0

Crunch  endp

;----------------------- INTERRUPT PROCEEDURE ---------------------------

Intr    proc    far
        cli
        cmp     working, 1        ; return if working
        je      Intrd

        mov     working,1
        pushad

        cmp     liveoff,1               ; was livetime off?
        jne     Intr0
        mov     dx,STAT                 ; check IN2
        in      ax,dx
        and     ax,8
        jz      Intr2                   ; jmp if IN2 is still low
        mov     liveoff,0
        mov     dx,PORT2
        mov     ax,10                   ; enable data in
        out     dx,ax
        call    Delay
        mov     dx,PORT2
        mov     ax,4                    ; start scalers
        out     dx,ax
        call    Delay
        call    Gtime                   ; get start time
        mov     ebx,runtime
        cmp     eax,ebx
        jge     Itart1
        add     eax,604800
Itart1: sub     eax,ebx
        mov     starttm,eax
        jmp     Intr1

Intr0:  mov     dx,STAT                 ; check livetime IN2
        in      ax,dx
        and     ax,8
        jnz     Intr1
        mov     liveoff,1
        mov     ax,11                   ; disable data in
        mov     dx,PORT2
        out     dx,ax
        call    Delay
        mov     ax,2                    ; stop scalers
        out     dx,ax
        call    Gtime                   ; update runtime
        mov     ebx,starttm
        cmp     eax,ebx
        jge     Ck1
        add     eax,604800
Ck1:    sub     eax,ebx
        mov     runtime,eax
        jmp     Intr2

Intr1:  mov     bx,word ptr mem_add     ; switch to protected mode
        mov     ax,word ptr mem_add+2
        mov     cx,0
        mov     dh,1
        mov     dl,PhysToVirt
        call    devhlp

        push    es
        mov     es,GDT
        call    crunch                  ; empty FIFO and Crunch
        pop     es

        mov     dl,UnPhysToVirt           ; return to original mode
        call    devhlp

Intr2:  popad
        sti
        nop
        mov     working,0
Intrd:  sti
        ret

Intr    endp

;------------------ DELAY PROCEEDURE FOR OUT COMMAND --------------------

Delay   proc    near
        nop
        nop
        nop
        ret
Delay   endp

;------------------ GET CLOCK TIME IN SECONDS --------------------------

Gtime   proc    near

G1:     mov     dx,70h                  ; wait for permision
        mov     ax,0ah
        out     dx,al
        inc     dx
        in      al,dx
        and     al,128
        jnz     G1

        mov     dx,70h                  ; get seconds
        mov     ecx,0
        mov     eax,0
        out     dx,al
        inc     dx
        in      al,dx
        mov     dl,al
        and     dl,15
        mov     cl,dl
        shr     al,4
        mov     bx,10
        imul    ax,bx
        add     ecx,eax

        mov     dx,70h                  ; add minutes
        mov     eax,2
        out     dx,al
        inc     dx
        in      al,dx
        mov     dl,al
        and     eax,15
        mov     ebx,60
        imul    eax,ebx
        add     ecx,eax
        mov     al,dl
        shr     al,4
        and      eax,15
        mov     ebx,600
        imul    eax,ebx
        add     ecx,eax

        mov     dx,70h                  ; add hours
        mov     eax,4
        out     dx,al
        inc     dx
        in      al,dx
        mov     dl,al
        and     eax,15
        mov     ebx,3600
        imul    eax,ebx
        add     ecx,eax
        mov     al,dl
        shr     al,4
        and      eax,15
        mov     ebx,36000
        imul    eax,ebx
        add     ecx,eax

        mov     dx,70h                  ; add days
        mov     eax,6
        out     dx,al
        inc     dx
        in      al,dx
        and     eax,15
        dec      eax
        mov     ebx,86400
        imul    eax,ebx
        add     ecx,eax

        mov     dx,70h
        mov     eax,0Dh
        out     dx,al
        mov     eax,ecx

        ret
Gtime   endp


;------------------ THIS ROUTINE SOUNDS A SHORT BEEP --------------------

Beep    proc     near
        push     cx
        push     ax
        in       al,61h         ; read 8255 on system board
        mov      ah,al
        or       al,3
        out      61h,al
        mov      cx,8000h       ; delay count for beep
        jmp      Beep1
Beep1:  loop     Beep1
        mov      al,ah
        out      61h,al         ; turn off speaker
        pop      ax
        pop      cx
        ret
Beep    endp

;------------------------- INITILIZATION PROCEDURE ---------------------
```

```
Install proc    near

        mov     ax,es:[bx+14]           ; save devhlp address
        mov     word ptr devhlp,ax
        mov     ax,es:[bx+16]
        mov     word ptr devhlp+2,ax

        mov     word ptr es:[bx+14],offset _TEXT:Install
        mov     word ptr es:[bx+16],offset DGROUP:END_DS

        mov     ax,ds                   ; allocate GDT selector
        mov     es,ax
        mov     di,offset ds:GDT
        mov     cx,1
        mov     dl,AllocGDT
        call    devhlp

        call    open                    ; initialize I/O board

        mov     dx,PORT2
        mov     ax,11                   ; data in disabled
        out     dx,ax
        call    delay

        mov     ax,7                    ; reset FIFO
        out     dx,ax
        call    delay

        push    1                       ; message that device was loaded
        push    ds
        push    offset DGROUP:ident
        push    ident_len
        push    ds
        push    offset DGROUP:wlen
        call    DOSWRITE
        ret

Install endp
_TEXT   ends
        end
```