KEITHLEY

PIO-INT and PCF-PIO-INT

User Guide

A GREATER MEASURE OF CONFIDENCE

WARRANTY

Hardware

Keithley Instruments, Inc. warrants that, for a period of one (1) year from the date of shipment (3 years for Models 2000, 2001, 2002, 2010 and 2700), the Keithley Hardware product will be free from defects in materials or workmanship. This warranty will be honored provided the defect has not been caused by use of the Keithley Hardware not in accordance with the instructions for the product. This warranty shall be null and void upon: (1) any modification of Keithley Hardware that is made by other than Keithley and not approved in writing by Keithley or (2) operation of the Keithley Hardware outside of the environmental specifications therefore.

Upon receiving notification of a defect in the Keithley Hardware during the warranty period, Keithley will, at its option, either repair or replace such Keithley Hardware. During the first ninety days of the warranty period, Keithley will, at its option, supply the necessary on site labor to return the product to the condition prior to the notification of a defect. Failure to notify Keithley of a defect during the warranty shall relieve Keithley of its obligations and liabilities under this warranty.

Other Hardware

The portion of the product that is not manufactured by Keithley (Other Hardware) shall not be covered by this warranty, and Keithley shall have no duty of obligation to enforce any manufacturers' warranties on behalf of the customer. On those other manufacturers' products that Keithley purchases for resale, Keithley shall have no duty of obligation to enforce any manufacturers' warranties on behalf of the customer.

Software

Keithley warrants that for a period of one (1) year from date of shipment, the Keithley produced portion of the software or firmware (Keithley Software) will conform in all material respects with the published specifications provided such Keithley Software is used on the product for which it is intended and otherwise in accordance with the instructions therefore. Keithley does not warrant that operation of the Keithley Software will be uninterrupted or error-free and/or that the Keithley Software will be adequate for the customer's intended application and/or use. This warranty shall be null and void upon any modification of the Keithley Software that is made by other than Keithley and not approved in writing by Keithley.

If Keithley receives notification of a Keithley Software nonconformity that is covered by this warranty during the warranty period, Keithley will review the conditions described in such notice. Such notice must state the published specification(s) to which the Keithley Software fails to conform and the manner in which the Keithley Software fails to conform to such published specification(s) with sufficient specificity to permit Keithley to correct such nonconformity. If Keithley determines that the Keithley Software does not conform with the published specifications, Keithley will, at its option, provide either the programming services necessary to correct such nonconformity or develop a program change to bypass such nonconformity in the Keithley Software. Failure to notify Keithley of a nonconformity during the warranty shall relieve Keithley of its obligations and liabilities under this warranty.

Other Software

OEM software that is not produced by Keithley (Other Software) shall not be covered by this warranty, and Keithley shall have no duty or obligation to enforce any OEM's warranties on behalf of the customer.

Other Items

Keithley warrants the following items for 90 days from the date of shipment: probes, cables, rechargeable batteries, diskettes, and documentation.

Items not Covered under Warranty

This warranty does not apply to fuses, non-rechargeable batteries, damage from battery leakage, or problems arising from normal wear or failure to follow instructions.

Limitation of Warranty

This warranty does not apply to defects resulting from product modification made by Purchaser without Keithley's express written consent, or by misuse of any product or part.

Disclaimer of Warranties

EXCEPT FOR THE EXPRESS WARRANTIES ABOVE KEITHLEY DISCLAIMS ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION, ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. KEITHLEY DISCLAIMS ALL WARRANTIES WITH RESPECT TO THE OTHER HARDWARE AND OTHER SOFTWARE.

Limitation of Liability

KEITHLEY INSTRUMENTS SHALL IN NO EVENT, REGARDLESS OF CAUSE, ASSUME RESPONSIBILITY FOR OR BE LIABLE FOR: (1) ECONOMICAL, INCIDENTAL, CONSEQUENTIAL, INDIRECT, SPECIAL, PUNITIVE OR EXEMPLARY DAMAGES, WHETHER CLAIMED UNDER CONTRACT, TORT OR ANY OTHER LEGAL THEORY, (2) LOSS OF OR DAMAGE TO THE CUSTOMER'S DATA OR PROGRAMMING, OR (3) PENALTIES OR PENALTY CLAUSES OF ANY DESCRIPTION OR INDEMNIFICATION OF THE CUSTOMER OR OTHERS FOR COSTS, DAMAGES, OR EXPENSES RELATED TO THE GOODS OR SERVICES PROVIDED UNDER THIS WARRANTY.

KEITHLEY

Keithley Instruments, Inc. 28775 Aurora Road • Cleveland, Ohio 44139 • 440-248-0400 • Fax: 440-248-6168
1-888-KEITHLEY (534-8453) • www.keithley.com

Sales Offices:	BELGIUM:	Bergensesteenweg 709 • B-1600 Sint-Pieters-Leeuw • 02-363 00 40 • Fax: 02/363 00 64
	CHINA:	Yuan Chen Xin Building, Room 705 • 12 Yumin Road, Dewai, Madian • Beijing 100029 • 8610-6202-2886 • Fax: 8610-6202-2892
	FINLAND:	Tietäjäntie 2 • 02130 Espoo • Phone: 09-54 75 08 10 • Fax: 09-25 10 51 00
	FRANCE:	3, allée des Garays • 91127 Palaiseau Cédex • 01-64 53 20 20 • Fax: 01-60 11 77 26
	GERMANY:	Landsberger Strasse 65 • 82110 Germering • 089/84 93 07-40 • Fax: 089/84 93 07-34
	GREAT BRITAIN:	Unit 2 Commerce Park, Brunel Road • Theale • Berkshire RG7 4AB • 0118 929 7500 • Fax: 0118 929 7519
	INDIA:	Flat 2B, Willocrissa • 14, Rest House Crescent • Bangalore 560 001 • 91-80-509-1320/21 • Fax: 91-80-509-1322
	ITALY:	Viale San Gimignano, 38 • 20146 Milano • 02-48 39 16 01 • Fax: 02-48 30 22 74
	JAPAN:	New Pier Takeshiba North Tower 13F • 11-1, Kaigan 1-chome • Minato-ku, Tokyo 105-0022 • 81-3-5733-7555 • Fax: 81-3-5733-7556
	KOREA:	2FL., URI Building • 2-14 Yangjae-Dong • Seocho-Gu, Seoul 137-888 • 82-2-574-7778 • Fax: 82-2-574-7838
	NETHERLANDS:	Postbus 559 • 4200 AN Gorinchem • 0183-635333 • Fax: 0183-630821
	SWEDEN:	c/o Regus Business Centre • Frosundaviks Allé 15, 4tr • 169 70 Solna • 08-509 04 679 • Fax: 08-655 26 10
	SWITZERLAND:	Kriesbachstrasse 4 • 8600 Dübendorf • 01-821 94 44 • Fax: 01-820 30 81
	TAIWAN:	1FL., 85 Po Ai Street • Hsinchu, Taiwan, R.O.C. • 886-3-572-9077• Fax: 886-3-572-9031

KEITHLEY

Safety Precautions

The following safety precautions should be observed before using this product and any associated instrumentation. Although some instruments and accessories would normally be used with non-hazardous voltages, there are situations where hazardous conditions may be present.

This product is intended for use by qualified personnel who recognize shock hazards and are familiar with the safety precautions required to avoid possible injury. Read and follow all installation, operation, and maintenance information carefully before using the product. Refer to the manual for complete product specifications.

If the product is used in a manner not specified, the protection provided by the product may be impaired.

The types of product users are:

Responsible body is the individual or group responsible for the use and maintenance of equipment, for ensuring that the equipment is operated within its specifications and operating limits, and for ensuring that operators are adequately trained.

Operators use the product for its intended function. They must be trained in electrical safety procedures and proper use of the instrument. They must be protected from electric shock and contact with hazardous live circuits.

Maintenance personnel perform routine procedures on the product to keep it operating properly, for example, setting the line voltage or replacing consumable materials. Maintenance procedures are described in the manual. The procedures explicitly state if the operator may perform them. Otherwise, they should be performed only by service personnel.

Service personnel are trained to work on live circuits, and perform safe installations and repairs of products. Only properly trained service personnel may perform installation and service procedures.

Keithley products are designed for use with electrical signals that are rated Installation Category I and Installation Category II, as described in the International Electrotechnical Commission (IEC) Standard IEC 60664. Most measurement, control, and data I/O signals are Installation Category I and must not be directly connected to mains voltage or to voltage sources with high transient over-voltages. Installation Category II connections require protection for high transient over-voltages often associated with local AC mains connections. Assume all measurement, control, and data I/O connections are for connection to Category I sources unless otherwise marked or described in the Manual.

Exercise extreme caution when a shock hazard is present. Lethal voltage may be present on cable connector jacks or test fixtures. The American National Standards Institute (ANSI) states that a shock hazard exists when voltage levels greater than 30V RMS, 42.4V peak, or 60VDC are present. **A good safety practice is to expect that hazardous voltage is present in any unknown circuit before measuring.**

Operators of this product must be protected from electric shock at all times. The responsible body must ensure that operators are prevented access and/or insulated from every connection point. In some cases, connections must be exposed to potential human contact. Product operators in these circumstances must be trained to protect themselves from the risk of electric shock. If the circuit is capable of operating at or above 1000 volts, **no conductive part of the circuit may be exposed.**

Do not connect switching cards directly to unlimited power circuits. They are intended to be used with impedance limited sources. NEVER connect switching cards directly to AC mains. When connecting sources to switching cards, install protective devices to limit fault current and voltage to the card.

Before operating an instrument, make sure the line cord is connected to a properly grounded power receptacle. Inspect the connecting cables, test leads, and jumpers for possible wear, cracks, or breaks before each use.

When installing equipment where access to the main power cord is restricted, such as rack mounting, a separate main input power disconnect device must be provided, in close proximity to the equipment and within easy reach of the operator.

For maximum safety, do not touch the product, test cables, or any other instruments while power is applied to the circuit under test. ALWAYS remove power from the entire test system and discharge any capacitors before: connecting or disconnecting cables or jumpers, installing or removing switching cards, or making internal changes, such as installing or removing jumpers.

Do not touch any object that could provide a current path to the common side of the circuit under test or power line (earth) ground. Always make measurements with dry hands while standing on a dry, insulated surface capable of withstanding the voltage being measured.

The instrument and accessories must be used in accordance with its specifications and operating instructions or the safety of the equipment may be impaired.

Do not exceed the maximum signal levels of the instruments and accessories, as defined in the specifications and operating information, and as shown on the instrument or test fixture panels, or switching card.

When fuses are used in a product, replace with same type and rating for continued protection against fire hazard.

Chassis connections must only be used as shield connections for measuring circuits, NOT as safety earth ground connections.

If you are using a test fixture, keep the lid closed while power is applied to the device under test. Safe operation requires the use of a lid interlock.

If ⏚ or ⏚ is present, connect it to safety earth ground using the wire recommended in the user documentation.

The ⚠ symbol on an instrument indicates that the user should refer to the operating instructions located in the manual.

The ⚡ symbol on an instrument shows that it can source or measure 1000 volts or more, including the combined effect of normal and common mode voltages. Use standard safety precautions to avoid personal contact with these voltages.

The **WARNING** heading in a manual explains dangers that might result in personal injury or death. Always read the associated information very carefully before performing the indicated procedure.

The **CAUTION** heading in a manual explains hazards that could damage the instrument. Such damage may invalidate the warranty.

Instrumentation and accessories shall not be connected to humans.

Before performing any maintenance, disconnect the line cord and all test cables.

To maintain protection from electric shock and fire, replacement components in mains circuits, including the power transformer, test leads, and input jacks, must be purchased from Keithley Instruments. Standard fuses, with applicable national safety approvals, may be used if the rating and type are the same. Other components that are not safety related may be purchased from other suppliers as long as they are equivalent to the original component. (Note that selected parts should be purchased only through Keithley Instruments to maintain accuracy and functionality of the product.) If you are unsure about the applicability of a replacement component, call a Keithley Instruments office for information.

To clean an instrument, use a damp cloth or mild, water based cleaner. Clean the exterior of the instrument only. Do not apply cleaner directly to the instrument or allow liquids to enter or spill on the instrument. Products that consist of a circuit board with no case or chassis (e.g., data acquisition board for installation into a computer) should never require cleaning if handled according to instructions. If the board becomes contaminated and operation is affected, the board should be returned to the factory for proper cleaning/servicing.

User Guide

for the

PIO-INT Pattern Recognition Board & PCF_PIO-INT Language Drivers

New Contact Information

Keithley Instruments, Inc.
28775 Aurora Road, Cleveland, OH 44139

Technical Support: 1-888-KEITHLEY
Monday – Friday 8:00 a.m. to 5:00 p.m. (EST)
Fax: (440) 248-6168
http://www.keithley.com

Contents

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

The PIO-INT Pattern Recognition board is a high-current, 24-line, parallel, digital I/O interface board with extended interrupt capabilities for IBM PC-XT/AT types of computers that are ISA- and EISA-bus compatible.

1.2 FEATURES

- 24 TTL/DTL Digital I/O Lines.
- 16 lines with extensive interrupt capabilities.
- Bit change interrupt generation.
- Pattern match interrupt generation.
- Flexible digital I/O based on popular 8255 P.P.I.
- PC/XT/AT ISA and EISA bus compatible.
- Can use extended interrupt levels of PC/AT.
- Compatible with MetraByte's PIO-12 with extended interrupt capabilities.
- ±12V and ±5V Power available from PC bus.
- Programmable interrupt level selection.
- Programmable interrupt delay.

1.3 APPLICATIONS

- General purpose & Interrupt driven digital I/O.
- Contact or switch change monitoring.
- Bit pattern matching or comparison.
- Process activation on defined bit states or changes.

1.4 FUNCTIONAL DESCRIPTION

The PIO-INT consists of three separate byte-wide data ports, PA, PB, and PC provided directly from a standard 8255 P.P.I. The PA and PB ports are monitored by additional circuitry capable of generating interrupts either on the change of any bit in the port(s) or when a specific combination (pattern) of bits appears at the port(s). The PC port can be divided into two nybble wide ports (PC upper and PC lower) and is useful as an auxiliary I/O port although it is not monitored and has no interrupt generating capabilities.

The PA, PB, PC upper and PC lower ports may be individually configured as input or output in any combination by the 8255 control register and in this respect THE PIO-INT is identically functional in digital I/O and programming to MetraByte's PIO-12, but has considerably enhanced interrupt capabilities. All ports can always be read and written to as normal 8 bit I/O ports.

There are 2 types of interrupt operations for the PA and PB ports, as follows:

1. Bit interrupts. A change of any un-masked bit either from 0 to 1 or 1 to 0 will generate an interrupt. The bit(s) that changed can be read from the Interrupt Status register, reading this register also clears the interrupt. Only bits that are activated by the interrupt Mask Registers (one for each port) will generate an interrupt.
2. Pattern interrupts. An interrupt is generated on a given pattern of bits in a port. Only un-masked bits (set by the mask register) can participate in a pattern match interrupt and they are compared with a stored pattern of bits in the PIO-INT's Pattern Match Registers (one for each port).

Two mode bits in the interrupt control register determine PIO-INT interrupt generation, as follows:

M1	M0	INTERRUPT STATUS
0	0	Interrupts disabled
0	1	PA only
1	0	PA OR PB
1	1	PA AND PB

This gives a variety of conditions under which the PIO-INT will generate an interrupt, including bit changes or pattern matches in only PA or either or both PA and PB ports or'ed or and'ed together. In addition, by masking any port off (set all mask register bits to zero), interrupts can be suppressed from that port allowing interrupts from single or multiple bits in the other port. This gives a large variety of possible operating conditions such as:

- No interrupts at all (disabled) - simple I/O board.
- Interrupt on selected bit change(s) in one or both ports.
- Interrupt on selected bit change(s) in one port and/or a pattern match in the other port.
- Interrupt on pattern matches in either or both ports (independent 1-8 bit or full 1-16 bit matches)

In order to prevent spurious interrupts (for example, from transition states of bits in a pattern match or noise glitches on an individual bit change), generation of an interrupt is delayed by a 2-stage digital filter. The filter is clocked by a programmable discrete decade frequency ranging from 1Hz to 10MHz. The effect of the filter is to allow actuation delays from 100-200ns to 1-2 seconds before an interrupt is generated (that is, the interrupting condition must be maintained for at least the filter delay time). If the interrupting condition (pattern or bit change) is not sustained for this programmable time interval, no interrupt will be produced by the PIO-INT. Filter delays are independently selectable for the PA and PB ports.

The connector pin out is identical to the PIO-12, except that no external interrupt input and enable are provided as they would be redundant (hence Pins 1 & 2 of the rear connector have no connection). On power up, all the mask and interrupt control registers are cleared, so until initialized, the PIO-INT cannot generate interrupts.

1.5 SPECIFICATIONS

Logic Inputs and Outputs

	MIN.	MAX.
Input logic low voltage	-0.5V	+0.8V
Input logic high voltage	+2.0V	+5.0V
Input current logic low: PA & PB ports	0	-0.4mA
PC port	-10uA	+10uA
Input current logic high: PA & PB ports	0	+40uA
PC port	-10uA	+10uA
Output low voltage: Isink = 1.4mA		
all ports	0	+0.45V
Output high voltage: Isource = 200uA		
all ports	+2.4V	+5.5V

Interrupt levels supported

PC & PC/XT (5 levels)	2, 3, 4, 5, 7
PC/AT (10 levels)	2/9, 3, 4, 5, 7, 10, 11, 12, 14, 15

Programmable interrupt delay time

100ns, 1uS, 10uS, 100uS, 1mS,

10mS

100mS, 1 sec.

Power Consumption

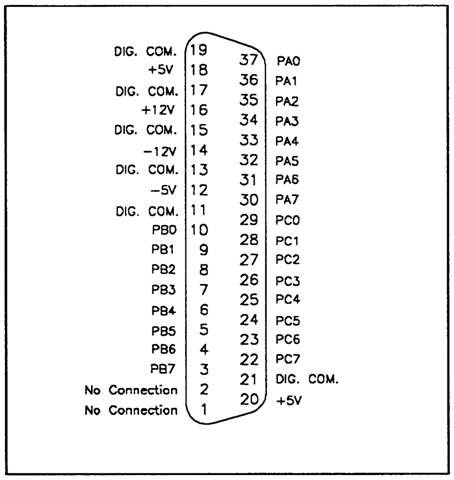
800mA typ. @ +5v

Environmental Specifications

Operating temperature range	0 to +50 deg. C.
Storage temperature range	-40 to +100 deg. C.
Humidity	0 to 90% non-condensing
Size	2/3 full slot length

1.6 CONNECTOR PIN ASSIGNMENTS

A rear view of the PIO-INT 37-pin, D connector is as shown in the diagram.



1.7 LIST OF DISTRIBUTION FILES

Driver Source Files

FILE NAME	DESCRIPTION
PIOINT.BIN	Driver for BASIC(A) programs.
PIOINT.QLB	Driver for QUICK BASIC programs.
PIOINT.LIB	Driver for 'C', PASCAL AND FORTRAN.
PIOINT.OBJ	Driver routine which is linkable to compiled BASIC.
TURBOPAS.OBJ	PIO-INT Driver for TURBO PASCAL.
PIOINT.ASM	Assembly source of Driver routine.
PIOIPCF.ASM	Source code PIO-INT Language Interface for Pascal, C and Fortran (PCF).
PIOINT.ADR	ASCII file, containing BASE ADDRESS, readable by BASIC, 'C', PASCAL and FORTRAN example programs.

DOcumentation, Utility, and Executable Programs

FILE NAME	DESCRIPTION
FILES.DOC	This file.
PIOIPCF.DOC	Info. on Multi-Language PCF Call Structures.
HOWTOBIN.DOC	Information on how to create a .BIN file.
README.DOC	Information about current driver software supplied (updated with every revision).
DIPSW.EXE	PIO-INT hardware installation aid.
MAKEBIN.EXE	Utility to convert a .COM file to a .BIN file for use with BASIC programs.

Example Programs

FILE NAME	**DESCRIPTION**
MSCSEX1.C	Microsoft 'C' example program (Small Model).
TCSEX1.C	TURBO 'C' example program (Small Model).
TP_EX1.PAS	TURBO PASCAL example program.
MSPEX1.PAS	Microsoft PASCAL example program.
MSFEX1.FOR	Microsoft FORTRAN example program.
EXPIOINT.BAS	Basic example program.
QBPIOINT.BAS	Quick Basic example program.

CHAPTER 2

HARDWARE INSTALLATION & SETUP

2.1 GENERAL

This chapter provides instructions for the installing the PIO-INT in an IBM PC-XT or AT and compatible models. The chapter begins with procedures for unpacking and inspection, which are followed by settings for the Base Address switch and instructions for board installation. There are also instructions for making working copies of your distribution diskettes.

2.2 UNPACKING AND INSPECTING

After you remove the wrapped board from its outer shipping carton, proceed as follows:

1. Place one hand firmly on a metal portion of the computer chassis (the computer must be turned Off and grounded). You place your hand on the chassis to drain off static electricity from the package and your body, thereby preventing damage to board components.
2. Allow a moment for static electricity discharge; carefully unwrap the board from its anti-static wrapping material.
3. Inspect the board for signs of damage. If any damage is apparent, return the board to the factory.
4. Check the contents of your package against its packing list to be sure the order is complete. Report any missing items to MetraByte immediately.

You may find it advisable to retain the packing material in case the board must be returned to the factory.

2.3 BASE ADDRESS SWITCH SETTINGS

Before installing the board, check its Base Address setting on the *ADDRESS SELECT* switch. This switch is preset at the factory for an address of 300 Hex, as shown in the Base Address Switch diagram.

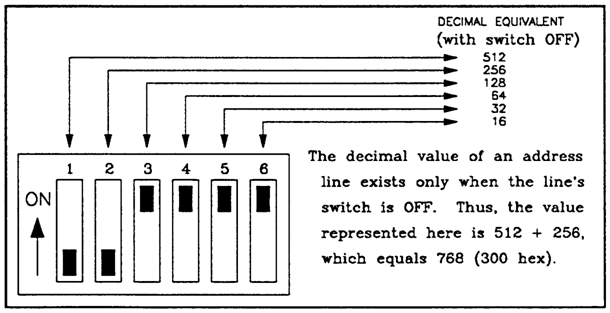

DECIMAL EQUIVALENT
(with switch OFF)
512
256
128
64
32
16
1
2
3
4
5
6
ON
The decimal value of an address line exists only when the line's switch is OFF. Thus, the value represented here is 512 + 256, which equals 768 (300 hex).

Diagram of the Base Address switch.

To communicate with a specific device such as a disk drive, a monitor, another PIO-INT board, etc., the computer must refer its communication to the device's Base Address. A Base Address is typically expressed as a 3-digit Hex number.

A PIO-INT board is preset for a Base Address of 300 Hex, which is within the address range used for a Prototype Card (300 to 31F, as shown in the table below). This default value will function in most computers without conflict, thereby eliminating any need for address selection and configuration.

However, if you have a need to change the Base Address from its preset value, you must select an address within a range of 000 to 3FF (0 to 1023 Decimal). In addition, the address must be on an 8-byte boundary, and it must not conflict with addresses the computer is already using for other devices. As an aid to selecting a usable 3-digit Hex number, the following table is an industry-standard I/O address map for the full 000 to 3FF range.

Table of industry-standard I/O addresses for peripheral devices.

HEX RANGE	USAGE	HEX RANGE	USAGE
000 to 1FF	Internal System	387 to 37F	LPT1:
200 to 20F	Game	380 to 38C	SDLC comm.
210 to 217	Expansion unit	380 to 389	Binary comm. 2
220 to 24F	Reserved	3A0 to 3A9	Binary comm. 1
278 to 27F	Reserved	3B0 to 3BF	Mono dsp/LPT1:
2F0 to 2F7	LPT2:	3C0 to 3CF	Reserved
2F8 to 2FF	COM2:	3D0 to 3DF	Color graphics
300 to 31F	Prototype card	3E0 to 3E7	Reserved
320 to 32F	Hard disk	3F0 to 3F7	Floppy disk
		3F8 to 3FF	COM1:

After you select a Base Address, run the DIPSW.EXE program (contained in your PIO-INT software) to determine the proper setting for the driver board's Base Address switch. To run this program, log to the directory that contains DIPSW.EXE and type DIPSW <Enter> The program will respond with the question DESIRED BASE ADDRESS ----> ?

Your response should be to type a decimal number (or a Hex number preceded by &H, such as &H300) representing the selected address. The computer will display the corresponding Base Address switch settings in diagram form, as shown in the Base Address switch diagram. If the entry is unacceptable, the computer will display an explanatory statement and a request for another entry. As the Base Address switch diagram indicates, the Base Address is set in binary code; also, the switches have value only in the OFF position.

2.4 HARDWARE INSTALLATION

To install the PIO-INT in a PC, proceed as follows.

WARNING: ANY ATTEMPT TO INSERT OR REMOVE ANY ADAPTER BOARD WITH THE COMPUTER POWER ON COULD DAMAGE YOUR COMPUTER!

1. Turn Off power to the PC and all attached equipment.
2. Remove the cover of the PC as follows: First remove the cover-mounting screws from the rear panel of the computer. Then, slide the cover of the computer about 3/4 of the way forward. Finally, tilt the cover upwards and remove.
3. Choose an available option slot. Loosen and remove the screw at the top of the blank adapter plate. Then slide the plate up and out to remove.
4. Hold the PIO-INT board in one hand placing your other hand on any metallic part of the PC/AT chassis (but not on any components). This will safely discharge any static electricity from your body.
5. Make sure the board switches have been properly set (refer to the preceding section).
6. Align the board connector with the desired accessory slot and with the corresponding rear-panel slot. Gently press the board downward into the socket. Secure the board in place by inserting the rear-panel adapter-plate screw.

7. Replace the computer's cover. Tilt the cover up and slide it onto the system's base, making sure the front of the cover is under the rail along the front of the frame. Replace the mounting screws.
8. Plug in all cords and cables. Turn the power to the computer back on.

You are now ready to make any necessary system connections, install the PIO-INT software, and perform calibration and perform checks on calibration and adjustment, as described in Section 8.3 of Chapter 8.

MetraByte recommends that you retain the static-shield packaging for possible future removal and handling of the PIO-INT board.

2.5 UTILITY SOFTWARE BACK UP

Distribution software is furnished on 5.25", 360K floppy diskettes. To accommodate users with 3.5" floppy drives, the Software is also available on a 720K diskette.

As soon as possible, make a working copy of your PIO-INT software using the procedures that follow. Store your original software copy in a safe place as a backup.

The following back-up procedures cover the more common computer configurations: a single floppy-disk drive (with hard disk), dual-floppy disk drives, and a hard disk.

2.5.1 Single Floppy-Disk Drive

To copy to another diskette in a single floppy-disk machine (with hard disk), proceed as follows:

1. Turn on power to your computer and display.
2. After system boot-up, the DOS prompt should be **C:>**
3. Be sure the DOS file *DISKCOPY.EXE* is in the root (C:\) directory. Then, type **DISKCOPY A: A:**
4. Insert the *source* diskette (your PIO-INT Utility diskette) into Drive A. The system will prompt you through the disk copying process. When the source diskette has been copied into memory, the System will ask you to insert the *target* diskette into Drive A. The *target* diskette is a blank disk that is to be your back-up disk.
5. When completed, the computer will ask COPY ANOTHER (Y/N)?. Respond by typing **N**.
6. Put the original PIO-INT diskette in a safe place. Label the back-up disk *PIO-INT Working Disk*. Use this disk to run your PIO-INT programs.

2.5.2 Dual Floppy-Disk Machines

If your dual floppy-drive PC has a hard disk, follow the procedure in Section 2.1.1, substituting the command **DISKCOPY A: B:** in Step 3. Then insert the PIO-INT diskette in Drive A and the Target diskette in Drive B.

If your machine has no hard disk, use the following procedure.

1. Turn on power to your computer and display, and place your DOS diskette in Drive A.
2. The DOS prompt should be **A:>** If not, type **A:** followed by **< Enter >** Be sure the diskette in Drive A contains the DISKCOPY.EXE file.

3. Then, type **DISKCOPY A: B:**
4. Insert the *source* diskette (your PIO-INT diskette) into Drive A. The system will prompt you through the disk copying process. It will ask you to insert the *target* diskette into Drive B. The *target* diskette is a blank disk that is to be your back-up disk.
5. When completed, the computer will ask COPY ANOTHER (Y/N)?. Respond by typing **N** .
6. When copying is complete, put the original PIO-INT diskette in a safe place. Label the back-up disk *PIO-INT Working Disk.* Use this disk to run the software.

2.5.3 Hard Disk Machines

To copy your PIO-INT files to a hard disk:

1. Start your computer. You should see a prompt, which indicates you are at the DOS level (for example, if your hard drive is designated as C , you should see the prompt **C:\>**).
2. The following instructions create a special directory for the PIO-INT Utility disk files. At the DOS prompt, type: **mkdir PIO** followed by < **Enter** >. Change to the AWFG directory by typing: **CD \PIO** followed by < **Enter** >
3. Place the Distribution diskette into Floppy Drive A and type **A:** . When the prompt changes from **C:\>** to **A:\>** , type **copy *.* C:** followed by < **Enter** >.
4. You have now copied the contents of the Utility diskette to your hard disk. Store the Utility diskette in a safe place.

CHAPTER 3

CONFIGURATION & PROGRAMMING INFORMATION

3.1 BASE ADDRESS, I/O MAP, & INTERRUPT LEVEL SELECTION

The PIO-INT uses a contiguous block of 16 I/O addresses. The Base Address of this block is switch-selectable and can be placed on any 16-bit I/O address boundary in the range 000H to 3F0H. Because of system limitations, addresses from 200H to 3FOH are available on the PC and PC-XT and addresses from 100H to 3F0H are available on the PC-AT. Allowing for other plug-in adapter boards, there will still be a wide choice of unused I/O address options available in any computer. The I/O address map of the PIO-INT is as follows:

Base Address	+0	8255 PA data port	(R/W)
	+1	8255 PB data port	(R/W)
	+2	8255 PC data port	(R/W)
	+3	8255 control register	(W only)
	+4	Interrupt control	(R/W) *
	+5	Global interrupt status	(R only)
	+6	PA Mask Register	(R/W) *
	+7	PB Mask Register	(R/W) *
	+8	Filter Control Register	(R/W) *
	+9	PA Pattern Match Register	(R/W)
	+10	PB Pattern Match Register	(R/W)
	+11	PA bit interrupt status	(R only)
	+12	PB bit interrupt status	(R only)
	+13 - 15	Not used	

* These registers are cleared on power up or hardware reset

Note that the first four I/O addresses correspond directly with the 8255 PPI so that the PIO-INT programs identically to the PIO-12 for pure digital I/O (see PIO-12 data sheet and programming examples). The other registers in the I/O map serve only to control the interrupt functions which are automatically disabled on power up (or hardware reset).

On a PC/AT or EISA/ISA bus computer, the PIO-INT can be programmed to operate on any of the following 10 bus interrupt levels 2/9, 3, 4, 5, 7, 10, 11, 12, 14 or 15. When used in a PC/XT based machine, only 5 levels 2, 3, 4, 5 or 7 may be used due to the XT bus structure. Level 6 is not provided due to its universal use by the floppy disk controller in all machines, and level 13 is also omitted due to its use by the numeric coprocessor on PC/AT machines.

3.2 REGISTERS

3.2.1 8255 Registers

The 8255 registers consist of the three read/write data ports PA, PB, and PC and the standard write-only control register. For information, see the 8255 data sheet or PIO-12 programming information.

3.2.2 Interrupt Control Register

This is an 8-bit read/write register cleared on power up. It sets the basic operating conditions of the board such as choice of interrupt level, source of interrupt, and whether the PA and PB ports are operating in the bit or pattern match mode of generating interrupts.

D7	D6	D5	D4	D3	D2	D1	D0
LEV8	LEV4	LEV2	LEV1	M1	M0	PTB	PTA
						[bit/pat]	

The PIO-INT is built with a small auxiliary connector so that when used in PC-AT (ISA or EISA) bus machines, it can utilize 10 of the 11 available interrupt levels. When used in a plain PC bus (62-pin connector) machine, the auxiliary connector remains unconnected and the PIO-INT can use only five of the six available interrupts. The interrupt level is selected by the LEV1 - 8 bits, as follows:

INTERRUPT_LEVEL	LEV8	LEV4	LEV2	LEV1	NOTES
2	0	0	1	0	PC/XT only
3	0	0	1	1	
4	0	1	0	0	
5	0	1	0	1	
7	0	1	1	1	
9	1	0	0	1	PC/AT only
10	1	0	1	0	PC/AT only
11	1	0	1	1	PC/AT only
12	1	1	0	0	PC/AT only
14	1	1	1	0	PC/AT only
15	1	1	1	1	PC/AT only

Note that Interrupt Level 6 is not available as it is generally used by the floppy-disk drives on PC-XT and AT machines and is an inconvenient level to use. Likewise, Level 13 is used by the numeric coprocessor on PC-AT bus machines. Setting the LEV1 - 8 bits to binary values of 0, 1, 6, or 13 (that is, any illegal level) will disable interrupts from the PIO-INT.

Mode bits M1 and M0 select the source of the interrupts and the logical combination of the interrupts from the PA and PB ports, as follows:

M1	M0	FUNCTION
0	0	Interrupts disabled
0	1	PA OR'ed with PB
1	0	PA AND'ed with PB
1	1	PA AND'ed with PB (same as 1 0)

Interrupts are disabled when both mode bits are 0. This enables/disables generation of interrupts from the PIO-INT. If the PA and PB ports are OR'd together (mode 0 1), then an interrupt condition originating from either port will generate a bus interrupt. If the PA and PB ports are AND'd together, then interrupts must be generated by both ports before a bus interrupt is produced.

The PTA/PTB pattern/bit control bits control how each port generates an interrupt. In the bit mode (PTA or PTB = 0), an interrupt is generated by a change of state of any un-masked bit in the port. The bit that changed state can be determined by reading the PA or PB port bit interrupt status registers. In the Pattern Match Mode (PTA or PTB = 1), an interrupt is generated only when the combination of unmasked bits corresponding to the pattern loaded in the corresponding port pattern registers is matched. Note that the mask registers are active in both types of comparison. In either case, the interrupting port can be determined by reading the global interrupt status register. It is possible to operate one port in bit mode, the other in pattern mode or AND both ports together to do full 16-bit pattern matches.

3.2.3 Global Interrupt Status Register

This is an 8-bit read-only register that provides information on the source of an interrupt and also clears the interrupt from the PIO-INT. When operating in the bit change mode, the individual bit(s) within a port that generated an interrupt can be determined from the interrupt status registers for each port, and these should be read first as they will also be cleared by reading the global interrupt status register. The bit assignment is as follows:

D7	D6	D5	D4	D3	D2	D1	D0
IRQ	INTE	IRQB	IRQA	LEV8	LEV4	LEV2	LEV1

Bits D0 - D3 return the active interrupt level (identical to that set in the Interrupt Control Register, it can also be read from the Interrupt Control Register). Bit D4 (IRQA) is set if the PA port generated the interrupt. Bit D5 (IRQB) is set if the PB port generated the interrupt. Bit D6 (INTE) corresponds to interrupts being enabled (that is, both M0 and M1 bits of the Interrupt Control register are non-zero). Bit D7 (the IRQ bit) corresponds to the bus-interrupt signal and is set according to the AND/OR combinations of the port interrupts IRQA and IRQB selected by the mode bits (M0 and M1) of the interrupt control register. **Reading the the Global Interrupt Status Register clears interrupts from the PIO-INT.**

3.2.4 Mask Registers

These are 8-bit read/write registers, one for Port A and one for Port B, that control which bits in the ports can generate interrupts from bit changes or participate in a bit comparison with the pattern registers. A '0' in the mask register disables the corresponding bit in the data port; a '1' enables it. The mask registers are cleared on power up.

3.2.5 Filter Register

This is an 8-bit read/write register that selects the filter delays for each port. It is cleared on power up, selecting the minimum filter delay (100ns).

D7	D6	D5	D4	D3	D2	D1	D0
NOT USED	RB4	RB2	RB1	NOT USED	RA4	RA2	RA1
	PB port filter			PA port filter			

R-4, R-2, & R-1: Select internal decade clock source for filter as follows:

R-4	R-2	R-1	FREQUENCY
0	0	0	10 MHz
0	0	1	1 MHz
0	1	0	100 KHz
0	1	1	10 KHz
1	0	0	1 KHz
1	0	1	100 Hz
1	1	0	10 Hz
1	1	1	1 Hz

On power up, the filter register is cleared, selecting the shortest delay (100-200nS). The user can then easily select from a broad range of decade scaled frequencies giving delays extending to 1-2 seconds.

3.2.6 Pattern Match Registers

When operating in the pattern comparison mode, these 8-bit read/write registers are loaded with the bit pattern required to generate an interrupt. Depending on the individual mask register bit settings, not all bits of a port may be involved in a pattern match. The pattern match registers are **not** cleared on power up or hardware reset and should be initialized to the desired pattern before enabling pattern match interrupt operation.

3.2.7 Interrupt Status Registers

These 8-bit read-only registers provide information on which bit(s) in a port generated a bit change interrupt. They are relevant only when operating a port in the Bit Change Interrupt Mode (that is, when the PTA or PTB bits of the interrupt control register are zero). In the Pattern Match Mode, the interrupt bit status registers will return zero. To clear either bit interrupt status register, read the global interrupt status register.

3.3 PROGRAMMING

Programming of the 8255 PPI for digital I/O is standard and is explained in Chapter 5. Since the 8255 is located at the first four I/O addresses in the PIO-INT's address map, the PIO-12 programming examples are equally valid for the PIO-INT.

Programming of the interrupt functions requires use of the additional PIO-INT registers and the installation of an interrupt handler. A sample assembly language driver callable from BASIC together with its source listing is provided on a utility program disk with the PIO-INT. The hardware has been designed to accommodate the following possible operations:

1. Software selectable interrupt level (no jumpers). You can install your interrupt handler for a particular level and select operation of the PIO-INT on that level in one initializing operation.
2. Flexible generation of interrupts either by change of a bit in a port or a defined pattern of data being established on one or both ports.
3. Delaying of the interrupt by a decade programmable delay in the range 100ns to 1-2 seconds. This amounts to requiring an interrupting condition to be maintained for the programmable time before a bus interrupt is generated. It is especially useful in noisy environments or where false transitional states may momentarily exist.
4. An interrupting condition from Port A may be OR'd or AND'd with an interrupting condition from Port B.
5. The source of an interrupt, Port A or Port B, or in the case of bit change interrupts, the actual bit(s) that changed may be determined from the PIO-INT's status registers.
6. Mask registers allow for disabling of any bits involved in either bit change or pattern match operations.
7. Pattern Match Registers store the desired bit comparison pattern for generating pattern match interrupts.

The PIO-INT register structure and hardware capabilities are designed to make the generation of software for interrupt routines for real-world interrupt handling as simple as possible, for example the hardware eliminates the need for operations like software debouncing of switch closures etc. Not all high level languages support the writing of interrupt handlers, C and assembly language are excellent, whereas BASIC and FORTRAN have no capabilities and require callable extensions.

CHAPTER 4

MODE CALL PROGRAMMING

4.1 CALL SYNTAX

CALL PIOINT (MD% , D%(0), FLAG%)

4.2 MODE 0: INITIALIZE

Data on entry to the call:

```
MD% = 0

D%(0) = Base I/O address (range 100H - 3F0H PC/AT, 200H - 3F0H PC/XT)
D%(1) = PA port direction (1 = input, 0 = output)
D%(2) = PB port direction (1 = input, 0 = output)
D%(3) = PC0-3 port direction (1 = input, 0 = output)
D%(4) = PC4-7 port direction (1 = input, 0 = output)
D%(5) = Interrupt level (2,3,4,5,7,10,11,12,14,15 - range 0 - 15)
D%(6) = Type of interrupt on PA port (0 = bit, 1 = pattern)
D%(7) = Type of interrupt on PB port (0 = bit, 1 = pattern)
D%(8) = Interrupt configuration:- (range 0 - 3)
                        0 - all disabled
                        1 - PA port only
                        2 - PA OR'ed with PB port
                        3 - PA AND'ed with PB port
D%(9) - D%(16) = x (value irrelevant)

FLAG% = x (value irrelevant)
```

Data on exit from the call:

```
MD% = 0 (unchanged)

D%(0) - D%(16) = value unchanged
```

Possible errors returned:

FLAG%	0 - No error, call executed O.K.
	1 - Wrong base address (range 100H to 3F0H/AT or 200H to 3FFH/XT)
	2 - No hardware present at base address
	4 - Illegal mode number (MD% < 0 or MD% > 12)
	11 thru 18 - Illegal value of D%(Error # - 10)

Check for hardware at Base Address.

Check if in AT (presence of 2nd. 8259 - block levels 10-15 if not).

4.3 MODE 1: READ DATA FROM PORTS

Data on entry to the call:

```
MD% = 1

D%(0) - D%(16) = x (value irrelevant)

FLAG% = x (value irrelevant)
```

Data on exit from the call:

```
MD% = 1 (unchanged)

D%(0) = Port A data
D%(1) = Port B data
D%(2) = Port C, PC0-3 data
D%(3) = Port C, PC4-7 data
D%(4) - D%(16) = value unchanged
```

Possible errors returned:

FLAG%	0 - No error, call executed O.K.
	3 - Not initialized, run mode 0 before selecting this mode
	4 - Illegal mode number (MD% < 0 or MD% > 12)

4.4 MODE 2: WRITE DATA TO ALL PORTS

Data on entry to the call:

```
MD% = 2

D%(0) = Port A data (range 0 - 255)
D%(1) = Port B data (range 0 - 255)
D%(2) = Port C, PC0-3 data (range 0 -15)
D%(3) = Port C, PC4-7 data (range 0 - 15)
D%(4) - D%(16) = x (value irrelevant)
```

Data on exit from the call:

```
MD% = 2 (unchanged)

D%(0) - D%(16) = value unchanged

FLAG% = x (value irrelevant)
```

Possible errors returned:

FLAG%	0 - No error, call executed O.K.
	3 - Not initialized, run mode 0 before selecting this mode
	4 - Illegal mode number (MD% < 0 or MD% > 12)
	10 thru 13 - Illegal value of D%(Error # - 10)

4.5 MODE 3: WRITE DATA TO ONE PORT

Data on entry to the call:

```
MD% = 3

D%(0) = Port select:- (range 0 - 3)
                         0 = PA port
                         1 = PB port
                         2 = PC lower port (PC0-3)
                         3 = PC upper port (PC4-7)
D%(1) = data (range 0 -255)
D%(2) - D%(16) = x (value irrelevant

FLAG% = x (value irrelevant)
```

Data on exit from the call:

```
MD% = 3 (unchanged)

D%(0) - D%(16) = value unchanged
```

Possible errors returned:

FLAG%	0 - No error, call executed O.K.
	3 - Not initialized, run mode 0 before selecting this mode
	4 - Illegal mode number (MD% < 0 or MD% > 12)
	10 thru 11 - Illegal value of D%(Error # - 10)

4.6 MODE 4: READ MASK REGISTERS

Data on entry to the call:

```
MD% = 4

D%(0) - D%(16) = x (value irrelevant)

FLAG% = x (value irrelevant)
```

Data on exit from the call:

```
MD% = 4 (unchanged)
```

```
D%(0) = PA port mask register data (range 0 - 255)
D%(1) = PB port mask register data (range 0 - 255)
D%(2) - D%(16) = value unchanged
```

Possible errors returned:

FLAG%	0 - No error, call executed O.K.
	3 - Not initialized, run mode 0 before selecting this mode
	4 - Illegal mode number (MD% < 0 or MD% > 12)

4.7 MODE 5: WRITE MASK REGISTERS

Data on entry to the call:

```
MD% = 5

D%(0) = PA port mask register data (range 0 -255)
D%(1) = PB port mask register data (range 0 - 255)
D%(2) - D%(16) = x (value irrelevant)

FLAG% = x (value irrelevant)
```

Data on exit from the call:

```
MD% = 5 (unchanged)

D%(0) - D%(16) = value unchanged
```

Possible errors returned:

FLAG%	0 - No error, call executed O.K.
	3 - Not initialized, run mode 0 before selecting this mode
	4 - Illegal mode number (MD% < 0 or MD% > 12)
	10 thru 11 - Illegal data value

4.8 MODE 6: READ PATTERN MATCH REGISTERS

Data on entry to the call:

```
MD% = 6

D%(0) - D%(16) = x (value irrelevant)

FLAG% = x (value irrelevant)
```

Data on exit from the call:

```
MD% = 6 (unchanged)

D%(0) = PA port pattern match register data
D%(1) = PB port pattern match register data
D%(2) - D%(16) = value unchanged
```

Possible errors returned:

FLAG%	0 - No error, call executed O.K.
	3 - Not initialized, run mode 0 before selecting this mode
	4 - Illegal mode number (MD% < 0 or MD% > 12)

4.9 MODE 7: WRITE PATTERN MATCH REGISTERS

Data on entry to the call:

```
MD% = 7

D%(0) = PA port pattern match register data (range 0 - 255)
D%(1) = PB port pattern match register data (range 0 - 255)
D%(2) - D%(16) = x (value irrelevant)

FLAG% = x (value irrelevant)
```

Data on exit from the call:

```
MD% = 7 (unchanged)

D%(0) - D%(16) = value unchanged
```

Possible errors returned:

FLAG%	0 - No error, call executed O.K.
	3 - Not initialized, run mode 0 before selecting this mode
	4 - Illegal mode number (MD% < 0 or MD% > 12)
	10 thru 11 - Illegal data value

4.10 MODE 8: SET FILTER RESPONSE

Data on entry to the call:

```
MD% = 8

D%(0) = PA port filter rate (range 0 - 7)
D%(1) = PB port filter rate (range 0 - 7)
D%(2) - D%(16) = x (value irrelevant)
```

```
FLAG% = x (value irrelevant)
```

Data on exit from the call:

```
MD% = 8 (unchanged)

D%(0) - D%(16) = value unchanged
```

Possible errors returned:

FLAG%	0 - No error, call executed O.K.
	3 - Not initialized, run mode 0 before selecting this mode
	4 - Illegal mode number (MD% < 0 or MD% > 12)
	10 thru 11 - Illegal data value

4.11 MODE 9: ENABLE INTERRUPT

Data on entry to the call:

```
MD% = 9

D%(0) - D%(16) = x (value irrelevant)

FLAG% = x (value irrelevant)
```

Data on exit from the call:

```
MD% = 9 (unchanged)

D%(0) - D%(16) = value unchanged
```

Possible errors returned:

FLAG%	0 - No error, call executed O.K.
	3 - Not initialized, run mode 0 before selecting this mode
	4 - Illegal mode number (MD% < 0 or MD% > 12)
	5 - Interrupt already enabled

4.12 MODE 10: DISABLE INTERRUPT

Data on entry to the call:

```
MD% = 10

D%(0) - D%(16) = x (value irrelevant)

FLAG% = x (value irrelevant)
```

Data on exit from the call:

```
MD% = 10 (unchanged)

D%(0) - D%(16) = value unchanged
```

Possible errors returned:

FLAG%	0 - No error, call executed O.K.
	3 - Not initialized, run mode 0 before selecting this mode
	4 - Illegal mode number (MD% < 0 or MD% > 12)
	6 - Interrupt already disabled

4.13 MODE 11: READ INTERRUPT STATUS & CLEAR INTERRUPT

Data on entry to the call:

```
MD% = 11

D%(0) - D%(16) = x (value irrelevant)

FLAG% = x (value irrelevant)
```

Data on exit from the call:

```
MD% = 11 (unchanged)

D%(0) = PA interrupt      1 = PA Port Generate Interrupt;
                                0 = If Not.
D%(1) = PB interrupt      1 = PB Port Generate Interrupt;
                                0 = If Not.
D%(2) = Board interrupt   (see Notes 4 & 5 of Sect 3.3)
D%(3) = PA interrupt status register
D%(4) = PB interrupt status register
D%(5) - D%(16) = value unchanged
```

Possible errors returned:

FLAG%	0 - No error, call executed O.K.
	3 - Not initialized, run mode 0 before selecting this mode
	4 - Illegal mode number (MD% < 0 or MD% > 12)

4.14 MODE 12: RETURN CONFIGURATION

Data on entry to the call:

```
MD% = 12

D%(0) - D%(16) = x (value irrelevant)
```

```
FLAG% = x (value irrelevant)
```

Data on exit from the call:

```
MD% = 12 (unchanged)

D%(0) = Interrupt status 1 = Enable; 0 = Disable.
D%(1) = PA port direction1 = Input; 0 = Output.
D%(2) = PB port direction1 = Input; 0 = Output.
D%(3) = PC lower port direction1 = Input; 0 = Output.
D%(4) = PC upper port direction1 = Input; 0 = Output.
D%(5) = Interrupt level
D%(6) = Type of interrupt on PA port (0 = bit, 1 = pattern)
D%(7) = Type of interrupt on PB port (0 = bit, 1 = pattern)
D%(8) = Interrupt configuration:-
                         0 - all disabled
                         1 - PA port only
                         2 - PA OR'ed with PB port
                         3 - PA AND'ed with PB port
D%(9) = PA mask register
D%(10) = PB mask register
D%(11) = PA interrupt status register
D%(12) = PB interrupt status register
D%(13) = PA filter rate
D%(14) = PB filter rate
D%(15) = PA pattern register
D%(16) = PB pattern register
```

Possible errors returned:

FLAG%	0 - No error, call executed O.K.
	3 - Not initialized, run mode 0 before selecting this mode
	4 - Illegal mode number (MD% < 0 or MD% > 12)

* * * * *

CHAPTER 5

INSTRUCTIONS FOR RETURNS

Before returning any equipment for repair, please call   to notify MetraByte's technical service personnel. If possible, a technical representative will diagnose and resolve your problem by telephone. If a telephone resolution is not possible, the technical representative will issue you a Return Material Authorization (RMA) number and ask you to return the equipment. Please reference the RMA number in any documentation regarding the equipment and on the outside of the shipping container.

Note that if you are submitting your equipment for repair under warranty, you must furnish the invoice number and date of purchase.

When returning equipment for repair, please include the following information:

1. Your name, address, and telephone number.
2. The invoice number and date of equipment purchase.
3. A description of the problem or its symptoms.

Repackage the equipment. Handle it with ground protection; use its original anti-static wrapping, if possible.

Be sure to reference the RMA number on the outside of the package!

APPENDIX **A**

PCF_PIO-INT LANGUAGE DRIVERS

Contents

* * * * *

SECTION **A1**

INTRODUCTION

A1.1 OVERVIEW

MetraByte's PCF_PIO-INT is for Pascal, C, and Fortran programmers writing data acquisition and control routines for the PIO-INT Pattern Recognition Board. The PCF_PIO-INT supports all memory models for the following languages;

- Microsoft C (V4.0 - 6.0)
- Microsoft QuickC (V1.0 - 2.0)
- Borland Turbo C (V1.0 - 2.0)
- Microsoft PASCAL (V3.0 - 4.0)
- Borland Turbo PASCAL (V3.0 - 5.0)
- Microsoft FORTRAN (V4.0 - 4.1)
- QuickBASIC (V4.0 & higher)
- GW, COMPAQ, and IBM BASIC (V2.0 & higher)

The PCF_PIO-INT consists of several assembly language drivers for the various supported languages along with example programs for each language. This manual is structured to illustrate memory model usage for each of the above languages and to include a brief example program at the end of each language section. Full source listings are included on the supplied disk.

This manual is not an introduction or operating guide to the supported PIO-INT boards. You should be familiar with the boards' various operating MODES, PARAMETERS, and ERROR codes before attempting PCF_PIO-INT implementation. Refer to the main sections of this manual supplied with your PIO-INT MetraByte board for a complete discussion of hardware and related functionality.

PCF_PIO-INT Distribution Software is furnished on a 5.25" floppy diskette. A 3.5" diskette version is available as an option.

A1.2 IMPLEMENTATION

Before working with this interface package, you are urged to become familiar with PIO-INT board functions and specifications. Example programs herein do not assume any knowledge of these boards since the programs are general in nature and do not actually implement features of any specific board. They are limited to the actual language interface for the various languages supported.

In the following chapter, each interface driver (implemented via a CALL statement) consists of three position-dependent parameters. These are MODE, ARGUMENT (or D), and FLAG, as follows:

MODE	Type of function to be executed by the PIO-INT.
D	Function dependent arguments required for execution

FLAG	Error number, if any, corresponding to selected MODE

* * * * *

SECTION **A2**

INTERFACE DRIVERS

A2.1 MICROSOFT C (V4.0 - 6.0) & QUICKC (V1.0 - 2.0) Small Model

Model:	Small ("/AS") switch on command line
Passes:	word size pointers (offset, no DS register)
Sequence:	Arguments Passed Right to Left
Default Calling Convention:	Arguments Passed by Value (Passing pointers to a subroutine is considered pass-by-value convention)

Example

'C' Call:

'C' Declaration:

.ASM Subroutine:

```
mscs_pioint (&Mode, D, &Flag);
extern void mscs_pioint(int*,int*,int*);
```

The following assembly code shows how the driver handles user arguments:

```
_mscs_pioint proc near
               push bp         ; save base pointer
               mov bp,sp       ; save stack pointer
               .               ; [bp+4] holds offset of Mode
               .               ; [bp+6] holds offset of D
               .               ; [bp+8] holds offset of Flag
               .               ; Program execution here
               .               ;
               .               ;
               pop bp          ;restore bp & sp prior to exit
               ret             ;return
_mscs_pioint endp
```

Other:

This information is provided for those wishing to create their own drivers:

- _mscs_pioint is declared "PUBLIC" in the .ASM file
- mscs_pioint is declared "extern" in the "C" file

- The .ASM file contains the ".model small" directive (MASM & TASM only)
- Add leading underscore "_" to all mscs_pioint occurrences in .ASM file
- mscs_pioint is a near call
- mscs_pioint must be in a segment fname_TEXT (where fname is the name of the file where mscs_pioint resides) if .ASM file contains mixed model procedures.

Medium Model

Model:	Medium ("/AM") switch on command line
Passes:	Word-size pointers (offset, no DS register)
Sequence:	Arguments Passed Right to Left
Default Calling Convention:	Arguments Passed by Value

Example

'C' Call:
'C' Declaration:
.ASM Subroutine:

```
mscm_pioint (&Mode, D, &Flag);
extern void mscm_pioint(int*,int*,int*);
```

The following assembly code shows how the driver handles user arguments:

```
_mscm_pioint proc far          ; far CALL (dword return address)
                push bp         ; save base pointer
                mov bp,sp       ; save stack pointer
                .               ; [bp+6] holds offset of Mode
                .               ; [bp+8] holds offset of D
                .               ; [bp+10] holds offset of Flag
                .               ; Program execution here
                .               ;
                .               ;
                pop bp          ;restore bp & sp prior to exit
                ret             ;return
_mscm_pioint endp
```

Other:

This information is provided for those wishing to create their own drivers:

- _mscm_pioint is declared "PUBLIC" in the .ASM file
- mscm_pioint is declared "extern" in the "C" file
- The .ASM file contains the ".model medium" directive (MASM & TASM only)
- Add leading underscore "_" to all mscm_pioint occurrences in .ASM file
- mscm_pioint is a far call

- mscm_pioint must be in a segment fname_TEXT (where fname is the name of the file where mscm_pioint resides), else Linker returns an error.

Large Model

Model:	Large ("/AL") switch on command line
Passes:	dword size pointers (offset and DS register)
Sequence:	Arguments Passed Right to Left
Default Calling Convention:	Arguments Passed by Value

Example

'C' Call:
'C' Declaration:
.ASM Subroutine:

```
mscl_pioint (&Mode, D, &Flag);
extern void mscl_pioint(int*,int*,int*);
```

The following assembly code shows how the driver handles user arguments:

```
_mscl_pioint proc far          ; far CALL (dword return address)
               push bp         ; save base pointer
               mov bp,sp       ; save stack pointer
               .               ; [bp+6] holds offset of Mode
               .               ; [bp+10] holds offset of D
               .               ; [bp+14] holds offset of Flag
               .               ; Program execution here
               .               ;
               .               ;
               pop bp          ;restore bp & sp prior to exit
               ret             ;return
_mscl_pioint endp
```

Other:

This information is provided for those wishing to create their own drivers:

- _mscl_pioint is declared "PUBLIC" in the .ASM file
- mscl_pioint is declared "extern" in the "C" file
- The .ASM file contains the ".model large" directive (MASM & TASM only)
- Add leading underscore "_" to all mscl_pioint occurrences in .ASM file
- Both code and data use dword (segment/offset) pointers
- mscl_pioint must be in a segment fname_TEXT (where fname is the name of the file where mscl_pioint resides), else Linker returns an error.

Microsoft 'C' Example

```
/*                              MSCSEX1.C                              */
/*                                                                     */
/*                       PIO-INT C Example Program                     */
/*                                                                     */
/*                     Keithley Metrabyte Corporation                  */
/*                                                                     */
/*              This Program Uses the Small Model Function Call        */
/*                                                                     */

#include <stdio.H>
#include <time.h>
#include <conio.h>

/***************************************************************************/
/*                                                                         */
/*     The Following are the function Calls for different models:         */
/*                                                                         */
/*     mscs_pioint(mode,param,flag) : Call from Microsoft C small Model.   */
/*     mscm_pioint(mode,param,flag) : Call from Microsoft C medium Model.  */
/*     mscl_pioint(mode,param,flag) : Call from Microsoft C large Model.   */
/*                                                                         */
/***************************************************************************/

extern  mscs_pioint(int *, int *, int *);  /* DECLARE CALL structure */
       .
       .

/***************************************************************************/
/*                                                                         */
/*                              Main program                               */
/*                                                                         */
/***************************************************************************/

main()
{
       .
       .

};

       .
       .
       .

/***************************************************************************/
/*                                                                         */
/*                              Mode 0                                     */
/*                                                                         */
/***************************************************************************/
```

```
void mode0()
{
FILE *infile;

                                   /* Initialize PIO-INT using Mode 0 */
Mode=0;                            /* Setup for Mode 0 */
Flag=0;                            /* flag or error variable
                                   /* D[0]=0x300; Alternative to set the
                                      base address, */
                                   /* if PIOINT.ADR file is not used  */
D[1]=0;                            /* PA port = output */
D[2]=0;                            /* PB port = outut */
D[3]=0;                            /* PC0-3 port = output */
D[4]=0;                            /* PC4-7 port = output */
D[5]=7;                            /* Interrupt Level */
D[6]=0;                            /* "bit" type interrupt at PA port */
D[7]=0;                            /* "bit" type interrupt at PB port */
D[8]=2;                            /* Interrupt at PA port only  */

if ( (infile = fopen("pioint.adr","r")) == NULL)
{
    system("cls");
    printf("Cannot open PIOINT.ADR file !\n");
    exit(0);
}

if ( fscanf(infile,"%d",&D[0]) == EOF)
{
    printf("PIOINT.ADR file is empty !\n");
    exit(0);
}

mscs_pioint(&Mode, D, &Flag);      /* Execute Mode 0 */

if (Flag < 10 )
  printf("\n\n%s",error[Flag]);
else
  printf("\n\n%s%d",error[10],Flag-10);
      .
      .
      .
}
      .
      .
      .
```

A2.2 BORLAND TURBO 'C' (V1.0 - 2.0)

Small Model

Model:	Small ("-ms") switch on command line
Passes:	word size pointers (offset, no DS register)
Sequence:	Arguments Passed Right to Left

Default Calling Convention:	Arguments Passed by Value

Example

'C' Call:
'C' Declaration:
.ASM Subroutine:

```
tcs_pioint (&Mode, D, &Flag);
extern void tcs_pioint(int*,int*,int*);
```

The following assembly code shows how the driver handles user arguments:

```
_tcs_pioint     proc near
                push bp          ; save base pointer
                mov bp,sp        ; save stack pointer
                .                ; [bp+4] holds offset of Mode
                .                ; [bp+6] holds offset of D
                .                ; [bp+8] holds offset of Flag
                .                ; Program execution here
                .                ;
                .                ;
                pop bp           ;restore bp & sp prior to exit
                ret              ;return
_tcs_pioint endp
```

Other:

This information is provided for those wishing to create their own drivers:

- _tcs_pioint is declared "PUBLIC" in the .ASM file
- tcs_pioint is declared "extern" in the "C" file
- The .ASM file contains the ".model small" directive (MASM & TASM only)
- Add leading underscore "_" to all tcs_pioint occurrences in .ASM file
- tcs_pioint is a near call
- tcs_pioint must be in a segment fname_TEXT (where fname is the name of the file where tcs_pioint resides), else Linker returns an error.

Medium Model

Model:	Medium ("-mm") switch on command line
Passes:	word size pointers (offset, no DS register)
Sequence:	Arguments Passed Right to Left
Default Calling Convention:	Arguments Passed by Value

Example

'C' Call:
'C' Declaration:
.ASM Subroutine:

```
tcm_pioint (&Mode, D, &Flag);
extern void tcm_pioint(int*,int*,int*);
```

The following assembly code shows how the driver handles user arguments:

```
_tcm_pioint proc far          ; dword pointer return address
            push bp           ; save base pointer
            mov bp,sp         ; save stack pointer
            .                 ; [bp+6] holds offset of Mode
            .                 ; [bp+8] holds offset of D
            .                 ; [bp+10] holds offset of Flag
            .                 ; Program execution here
            .                 ;
            .                 ;
            pop bp            ;restore bp & sp prior to exit
            ret               ;return
_tcm_pioint endp
```

Other:

This information is provided for those wishing to create their own drivers:

- _tcm_pioint is declared "PUBLIC" in the .ASM file
- tcm_pioint is declared "extern" in the "C" file
- The .ASM file contains the ".model medium" directive (MASM & TASM only)
- Add leading underscore "_" to all tcm_pioint occurrences in .ASM file
- tcm_pioint must be in a segment fname_TEXT (where fname is the name of the file where tcm_pioint resides), else Linker returns an error.

Large Model

Model:	Large ("-ml") switch on command line
Passes:	dword size pointers (offset and DS register)
Sequence:	Arguments Passed Right to Left
Default Calling Convention:	Arguments Passed by Value

Example

'C' Call:
'C' Declaration:
.ASM Subroutine:

```
tcl_pioint (&Mode, D, &Flag);
```

```
extern void tcl_pioint(int*,int*,int*);
```

The following assembly code shows how the driver handles user arguments:

```
_tcl_pioint proc far           ; dword pointer return address
                push bp        ; save base pointer
                mov bp,sp      ; save stack pointer
                .              ; [bp+6] holds offset of Mode
                .              ; [bp+10] holds offset of D
                .              ; [bp+14] holds offset of Flag
                .              ; Program execution here
                .              ;
                .              ;
                pop bp         ;restore bp & sp prior to exit
                ret            ;return
_tcl_pioint endp
```

Other:

This information is provided for those wishing to create their own drivers:

- _tcl_pioint is declared "PUBLIC" in the .ASM file
- tcl_pioint is declared "extern" in the "C" file
- The .ASM file contains the ".model large" directive (MASM & TASM only)
- Add leading underscore "_" to all tcl_pioint occurrences in .ASM file
- Both code & data use dword (segment/offset) pointers
- tcl_pioint must be in a segment fname_TEXT (where fname is the name of the file where tcl_pioint resides), else Linker returns an error.

Turbo 'C' Example

```
/*                             TCSEX1.C                                */
/*                                                                     */
/*                     PIO-INT TURBO C Example Program                 */
/*                                                                     */
/*                      Keithley Metrabyte Corporation                 */
/*                                                                     */
/*               This Program Uses the Small Model Function Call       */
/*                                                                     */

#include <stdio.H>
#include <time.h>
#include <conio.h>
```

```
/*************************************************************************/
/*                                                                       */
/*     The Following are the function Calls for different models:       */
/*                                                                       */
/*     tcs_pioint(mode,param,flag) : Call from Turbo C small Model.      */
/*     tcm_pioint(mode,param,flag) : Call from Turbo C medium Model.     */
/*    tcl_pioint(mode,param,flag) : Call from Turbo C large Model.       */
/*                                                                       */
/*************************************************************************/

extern  tcs_pioint(int *, int *, int *);  /* DECLARE CALL structure */

        .
        .
        .

/*************************************************************************/
/*                                                                       */
/*              Main program                                             */
/*                                                                       */
/*************************************************************************/

main()
{

        .
        .
        .

};

        .
        .
        .

/*************************************************************************/
/*                                                                       */
/*                                          Mode 0                       */
/*                                                                       */
/*************************************************************************/

void mode0()
{
FILE *infile;

                      /* Initialize PIO-INT using Mode 0 */
Mode=0;               /* Setup for Mode 0 */
Flag=0;               /* flag or error variable */
                      /* D[0]=0x300; Alternative to set the base address, */
                      /* if PIOINT.ADR file is not used       */
D[1]=0;               /* PA port = output */
D[2]=0;               /* PB port = outut */
D[3]=0;               /* PC0-3 port = output */
D[4]=0;               /* PC4-7 port = output */
D[5]=7;               /* Interrupt Level */
D[6]=0;               /* "bit" type interrupt at PA port */
D[7]=0;               /* "bit" type interrupt at PB port */
D[8]=2;               /* Interrupt at PA port only  */

if ( (infile = fopen("pioint.adr","r")) == NULL)
{
     system("cls");
     printf("Cannot open PIOINT.ADR file !\n");
     exit(0);
}
if ( fscanf(infile,"%d",&D[0]) == EOF)
{
     printf("PIOINT.ADR file is empty !\n");
     exit(0);
}
tcs_pioint(&Mode, D, &Flag);      /* Execute Mode 0 */

if (Flag < 10 )
  printf("\n\n%s",error[Flag]);
else
  printf("\n\n%s%d",error[10],Flag-10);

        .
        .
        .

}

        .
        .
        .
```

A2.3 MICROSOFT PASCAL (V3.0 - 4.0)

Medium Model

Model:	Medium
Passes:	word size pointers (offset address only)
Sequence:	Arguments Passed Left to Right
Default Calling Convention:	Arguments Passed by Value

Example

PASCAL Call: Result: =	msp_pioint (Var1, Var2, Var3);
PASCAL Declaration:	PROCEDURE msp_pioint(VAR Var1:integer;VAR Var2;VAR Var3: integer):external;

.ASM Subroutine:

The following assembly code shows how the driver handles user arguments:

```
msp_pioint proc far            ; far call (dword return address)
```

```
                push bp         ; save base pointer
                mov bp,sp       ; save stack pointer
                .               ; [bp+6] holds offset of Flag
                .               ; [bp+8] holds offset of Params
                .               ; [bp+10] holds offset of Mode
                .               ; Program execution here
                .               ;
                .               ;
                mov ax,n        ; Return Value for Function In ax register
                pop bp          ;
                ret 6           ; return and pop bp & sp values prior to
exit
msp_pioint endp
```

Other:

This information is provided for those wishing to create their own drivers:

- msp_pioint is declared "PUBLIC" in the .ASM file
- msp_pioint is declared external in the calling program
- msp_pioint resides in segment_TEXT (default of the .model command)

Microsoft PASCAL Example

```
PROGRAM MSPEX1(Input,Output);
(*********************************************************************)
(* Keithley Metrabyte Corporation                                    *)
(* File: MSPEX1.PAS                                                  *)
(*                                                                   *)
(* Demonstration program for the PIO-INT using Microsoft Pascal.     *)
(* To Compile:                                                       *)
(*              Type:      PL MSPEX1.PAS ;                           *)
(*                                                                   *)
(*********************************************************************)
TYPE DARRAY = ARRAY[0..16] of INTEGER;

FUNCTION TICS:WORD;EXTERN;
PROCEDURE MSP_PIOINT(VAR Mode:INTEGER; VAR D:DARRAY; VAR
Flag:INTEGER);EXTERN;

       .
       .
       .

(*********************************************************************)
(*                                                                   *)
(*     Mode 0                                                      *)
(*                                                                   *)
(*********************************************************************)

PROCEDURE  Mode0;
BEGIN
  Mode := 0;
  Flag := 0;
```

```
(*D[0]  := 768;              Alternative to set the base address, *)
(*                              if PIOINT.ADR file is not used       *)
  D[1]  := 0;
  D[2]  := 0;
  D[3]  := 0;
  D[4]  := 0;
  D[5]  := 7;
  D[6]  := 0;
  D[7]  := 0;
  D[8]  := 2;

  ASSIGN(FileIn,'PIOINT.ADR');
  RESET(FileIn);
  READLN(FileIn,D[0]);
  CLOSE(FileIn);

        .
        .
        .

  MSP_PIOINT(Mode,D,Flag);

        .
        .
        .
END;

        .
        .
        .

(**********************************************************************)
(*                                                                    *)
(*              Main                                                  *)
(*                                                                    *)
(**********************************************************************)
BEGIN

        .
        .
        .

END.
```

A2.4 BORLAND TURBO PASCAL (VER 3.0 - 4.0)

Borland's Turbo PASCAL supports a compact and a large memory model. The compact model supports one code segment and multiple data segments. In this model, the code segment is limited to 64K with assembly routine calls being near calls. The data segment is unlimited. The large model permits unlimited code and data segments with assembly calls and data access being far calls.

The program (TINST.EXE) shipped with TURBO PASCAL can change the calling convention so that the user may not know which convention they are using. The default state is "OFF" or compact mode. In order to ascertain which mode you are using, run the "TINST.EXE" program.

Compact Model

Model:	Compact (Forces far call "OFF" in TINST.EXE)
Passes:	dword size pointers (offset and segment)
Sequence:	Arguments Passed Left to Right
Default Calling Convention:	Arguments Passed by Value

Example

PASCAL Call: Result: =	tp_pioint (Var1, Var2, Var3);
PASCAL Declaration:	PROCEDURE tp_pioint(VAR Var1:integer;VAR Var2;VAR Var3: integer):external;

.ASM Subroutine:

(Either Model)

The following assembly code shows how the driver handles user arguments:

```
tp_pioint       proc near       ; near call (single word return address)
                push bp         ; save base pointer
                mov bp,sp       ; save stack pointer
                .               ; [bp+4] holds offset of VAR3
                .               ; [bp+8] holds offset of VAR2
                .               ; [bp+12] holds offset of VAR1
                .               ; Program execution here
                .               ;
                .               ;
                mov ax,n        ; return Value for Function In ax register
                pop bp
                ret 12          ; return & pop values prior to exit
tp_pioint       endp
```

Other:

This information is provided for those wishing to create their own drivers:

- Use the $L 'Metacommand' to link the object file containing external function tp_pioint, i.e. {$l turbopas} (Link to file turbopas.obj).
- The VAR declarative forces pass by reference (address of variable) in the function declaration. Default is pass by value (pushing the actual integer value onto the stack).
- tp_pioint is declared external in the calling program . Remember that in PASCAL, functions return a value whereas procedures never do.
- The .ASM file contains an explicit declaration of the code segment containing tp_pioint. Turbo PASCAL handles segments in a primitive manner which is not compatible with the '.model' statements available in MASM or TASM. The function tp_pioint must reside in a segment called 'CODE'! Turbo PASCAL will not accept any other segment name. If tp_pioint is not in segment "CODE", the linker returns an "unresolved external" error. The Segment Declaration for "CODE" in the .ASM file must appear as:

```
CODE SEGMENT WORD PUBLIC
ASSUME CS:CODE
.
.    ; CODE GOES HERE
.
CODE  ENDS
```

Large Model

Model:	Large (Forces far call "ON" in TINST.EXE)
Passes:	dword size pointers (offset and segment)
Sequence:	Arguments Passed Left to Right
Default Calling Convention:	Arguments Passed by Value

Example

PASCAL Call: Result: =	tp_pioint (Var1, Var2, Var3);
PASCAL Declaration:	PROCEDURE tp_pioint(VAR Var1:integer;VAR Var2;VAR Var3: integer):external;

.ASM Subroutine:

(Either Model)

The following assembly code shows how the driver handles user arguments:

```
tp_pioint      proc far        ; far call (dword return address)
               push bp         ; save base pointer
               mov bp,sp       ; save stack pointer
               .               ; [bp+6] holds dword of VAR3
               .               ; [bp+10] holds dword of VAR2
               .               ; [bp+14] holds dword of VAR1
               .               ; Program execution here
               .               ;
               .               ;
               mov ax,n        ; return Value for Function In ax register
               pop bp
               ret 12          ; return & pop values prior to exit
tp_pioint      endp
```

Other:

This information is provided for those wishing to create their own drivers:

- Use the $L 'Metacommand' to link the object file containing external function tp_pioint. For example; {$l turbopas} (Link file turbopas.obj).
- The VAR declarative forces pass by reference (address of variable) in the function declaration. Default is pass by value (pushing the actual integer value onto the stack).
- tp_pioint is declared external in the calling program along with the type of return value (integer). Remember, in PASCAL, functions return a value procedures don't.

- The .ASM file contains an explicit declaration of the code segment containing tp_pioint.

Turbo PASCAL Example

```
PROGRAM TP_EX1(Input,Output);
{$L TURBOPAS}
{$I-}
USES CRT,Dos;
(***********************************************************************)
(* Keithley Metrabyte Corporation                                      *)
(* File: TP_EX1.PAS                                                    *)
(*                                                                     *)
(* Demonstration program for the PIO-INT using Turbo Pascal.           *)
(* To Compile:                                                         *)
(*                 Type:      PL TP_EX1.PAS ;                          *)
(*                                                                     *)
(***********************************************************************)
TYPE DARRAY   = ARRAY[0..16] of INTEGER;
TYPE StrArray = ARRAY[0..3] of STRING[18];
PROCEDURE TP_PIOINT(VAR Mode:INTEGER; VAR D:DARRAY; VAR
Flag:INTEGER);EXTERNAL;


       .
       .
       .

(***********************************************************************)
(*                                                                     *)
(*     Mode 0                                                        *)
(*                                                                     *)
(***********************************************************************)
PROCEDURE  Mode0;
BEGIN
  Mode := 0;
  Flag := 0;
(*D[0] := 768;       Alternative to set the base address, if *)
(*     PIOINT.ADR file is not used              *)
  D[1] := 0;
  D[2] := 0;
  D[3] := 0;
  D[4] := 0;
  D[5] := 7;
  D[6] := 0;
  D[7] := 0;
  D[8] := 2;

  ASSIGN(FileIn,'PIOINT.ADR');
  RESET(FileIn);
  READLN(FileIn,D[0]);
  IF ( IOResult <> 0 ) THEN
  BEGIN
    ClrScr;
    WRITELN('PIOINT.ADR file not found !! ');
    HALT;
  END;
  CLOSE(FileIn);
```

```
            .
            .
            .

    TP_PIOINT(Mode,D,Flag);

            .
            .
            .

  END;

            .
            .
            .

  (*********************************************************************)
  (*                                                                   *)
  (*              Main                                                 *)
  (*                                                                   *)
  (*********************************************************************)
  BEGIN

            .
            .
            .
  END.
```

A2.5 MICROSOFT FORTRAN (V4.0 AND UP)

Large Model

Model:	Large
Passes:	dword size pointers (offset and DS register)
Sequence:	Arguments Passed Left to Right
Default Calling Convention:	Arguments Passed by Reference

Example

FORTRAN Call:	call fpioint(Var1, Var2, Var3);
FORTRAN Declaration:	None necessary in FORTRAN source file (Fortran assumes that undeclared subroutines or functions are external. It is left to the linking process to provide the required .LIB or .OBJ files. However, the function name should conform to ANSI FORTRAN rules for integer functions.

.ASM Subroutines:

NOTE: FORTRAN integer functions (beginning with letters i, j, or k) return results in the ax register whereas non-integer functions reserve 4 bytes on the calling stack for a far pointer to the result. Non-integer functions pass their arguments starting at location bp+14 after the "push bp" and "mov bp,sp" instructions have been executed. Keithley

MetraByte's FORTRAN <--> Assembly routines predominantly use type integer to avoid the non-integer problem. Using non-integer functions may be a problem when returning pointers, floating point results, long integers, etc. The user should use the IMPLICIT INTEGER (A-Z) declaration causing all Functions and Variables to be implicitly type integer unless declared otherwise. Also note that FORTRAN calls by Reference. This method places the address of the passed parameters (rather than the parameters themselves) onto the stack at the time of the call to any function or subroutine. As a convenience, PCF_PIO-INT provides two functions (INBYT and OUTBYT) for directly addressing the registers.

Integer (Default) Function or Subroutine

The following assembly code shows how the driver handles user arguments:

```
fpioint         proc far        ; dword pointer return address
                push bp         ; save base pointer
                mov bp,sp       ; save stack pointer
                .               ; [bp+6] holds offset of VAR3
                .               ; [bp+10] holds offset of VAR2
                .               ; [bp+14] holds offset of VAR1
                .               ; Program execution here
                .               ;
                .               ;
                mov ax,n        ; return Value for Function In ax register
                pop bp
                ret             ;
fpioint         endp
```

NOTES:

1. VAR3 = Return Value of Function
2. Function fpioint must be declared as an integer * 2 fucntion.

Microsoft FORTRAN Example

```
C*****************************************************************
C
C       MSFEX1.FOR
C
C       Example program for the PIO-INT using Microsoft Fortran
C
C       Keithley Metrabyte Corporation
C
C*****************************************************************
        program msfex1

        character NULL,ESC,PROMPT(3)
        integer*2 key,mode,flag,d(17)
        COMMON    /ASCII/NULL,ESC
        COMMON    /SIGN/PROMPT
        COMMON    mode,flag,d
```

```
                .
                .
                .

        end

                .
                .
                .

C***********************************************************************
C
C       mode 0
C
C***********************************************************************
        subroutine mode0

        integer*2 flag,d(17),mode
        COMMON    mode,flag,d

        open (unit = 9,err=999,status = 'old', file ='pioint.adr')

        flag = 0
        mode = 0
C
C       d(1) = 768                     Alternative to set the base address,
C              if PIOINT.ADR file is not used
C
        d(2) = 0
        d(3) = 0
        d(4) = 0
        d(5) = 0
        d(6) = 7
        d(7) = 0
        d(8) = 0
        d(9) = 2

        rewind (9)
        read(9,10) d(1)
10      format(i3)

        call fpioint(mode,d(1),flag)

                .
                .
                .

        end


                .
                .
                .
```

A2.6 INTERPRETED BASIC (GW, COMPAQ, IBM, ETC.)

Medium Model (Only Model Available)

Model:	Medium (Far Calls, Single Data)
Passes:	word size pointers (offset and no DS Register)
Sequence:	Arguments Passed Left to Right
Default Calling Convention:	Arguments Passed by Reference

Example

BASIC Call:	12500 CALL pioint(MODE%, D%(0), FLAG%)
BASIC Declaration:	NONE NECESSARY IN BASIC SOURCE CODE. However, a "BLOAD" (Binary load of .BIN file) of the binary file containing the external subroutine must be done prior to calling that subroutine.

.ASM Subroutine:

The following assembly code shows how the driver handles user arguments:

```
Location 0 (Beginning of Code Segment)

                jmp pioint

                .
                .
pioint          proc far        ; far call (dword return address)
                push bp         ; save base pointer
                mov bp,sp       ; save stack pointer
                .               ; [bp+6] holds offset of Flag
                .               ; [bp+8] holds offset of Params
                .               ; [bp+10] holds offset of Mode
                .
                .               ; Program execution here
                .
                .
                pop bp          ; restore bp & sp prior to exit
                ret
pioint          endp
```

NOTE BASIC requires that the .BIN file containing the callable subroutine "pioint(Mode%, D%(0), Flag%)" reside at location 0 in the .ASM segment or to "jmp" (unconditional jump) to the .BIN file. A BASIC "jmp " will always jump to location 0 in the .ASM code segment.

Creation of a .BIN file is accomplished as follows:

1. Create the .ASM Source Code File 'EXAMPLE.ASM'
2. Assemble 'EXAMPLE.ASM' thus creating 'EXAMPLE.OBJ'
3. Link 'EXAMPLE.OBJ' to create 'EXAMPLE.EXE'

4. Run EXE2BIN on 'EXAMPLE.EXE' (DOS Utility) to create 'EXAMPLE.COM'
5. Run MAKEBIN.EXE (Keithley MetraByte Utility) on 'EXAMPLE.COM' to create

   'EXAMPLE.BIN'

```
MASM EXAMPLE ;
LINK EXAMPLE ;
EXE2BIN EXAMPLE.EXE EXAMPLE.COM
MAKEBIN EXAMPLE.COM
```

The Following Example Program Illustrates a BASIC CALL:

```
10 '**************************************************************
20 '*                                                            *
30 '*  EXPIOINT.BAS - Example showing use of PIOINT.BIN driver   *
40 '*                        for Basic                           *
50 '*  Keithley MetraByte Corporation                            *
60 '**************************************************************
70 SCREEN 0 : WIDTH 80      'select 80 character wide text display
80 CLS                      'clear the display
90 KEY OFF                  'turn off function key display on line 25
100 '
110 'Load the PIOINT.BIN driver into an area of unused memory:-
120 CLEAR, 49152!
130 DEF SEG = 0
140 SG = 256 * PEEK(&H511) + PEEK(&H510)
150 SG = SG + 49152!/16
160 DEF SEG = SG
170 BLOAD "PIOINT.BIN",0  'load from disk with zero offset (,0) in this
                          'segment

       .
       .
       .

320 DIM D%(16)
330 MD% = 0          'mode variable - integer type
340 FLAG% = 0        'flag or error variable - integer type

       .
       .
       .

450 'Call MODE 0 to initialize the PIO-INT hardware and set its
       'configuration:-
460 '(Note: You must execute mode 0 before selecting any other mode)
470 MD% = 0           'select mode 0
480 OPEN "PIOINT.ADR" FOR INPUT AS #1
490 INPUT #1, D%(0)   'read in base address from PIOINT.ADR file
500 CLOSE #1
510 'D%(0) = &H300    specify PIO-INT base I/O address,
520 '                 if PIOINT.ADR file is not used.
530 D%(1) = 0        'set PA port direction ( 0 = output, 1 = input )
540 D%(2) = 0        'set PB port direction ( 0 = output, 1 = input )
550 D%(3) = 0        'set PC0-3 port direction ( 0 = output, 1 = input )
560 D%(4) = 0        'set PC4-7 port direction ( 0 = output, 1 = input )
```

```
570 D%(5) = 7         'select operation on interrupt level 7
580 D%(6) = 0         'select bit change type interrupts on PA port
590 D%(7) = 0         'select pattern match interrupts on PB port
600 D%(8) = 2         'OR the port interrupts (see manual)
610 'Mode 0 only uses the first 9 elements of D%(16) to pass data
620 CALL PIOINT (MD%, D%(0), FLAG%)   'execute the call to initialize
630 PRINT
640 PRINT"Mode 0 initialization:-"
650 PRINT
660 IF FLAG% < 10 THEN PRINT"Errors:     ";EC$(FLAG%)
670 IF FLAG% >=10 THEN PRINT"Errors:     ";EC$(10);FLAG%-10;:PRINT")"

        .
        .
        .
```

QUICKBasic

Medium Model (Only Model Available)

Model:	Medium (Far Calls, Single Data)
Passes:	word size pointers (offset and no DS Register)
Sequence:	Arguments Passed Left to Right
Default Calling Convention:	Arguments Passed by Reference

Example

BASIC Call:	CALL QBPIOINT(MODE%, VARPTR(D%(0)), FLAG%)
BASIC Declaration:	DECLARE SUB QBPIOINT (MD%,BYVAL DUMMY%, FLAG%) The Declaration tells QuickBASIC that the subroutine expects three arguments and that the middle argument is to be passed by value. Remember that BASIC normally passes all arguments by reference (address). This is the only method for passing an array to a subroutine in BASIC: passing the value of the address of the array in effect passes the array by reference. To make use of the callable assembly routine, a ".QLB" (Quick Library) file is created out of the original .ASM source file. Although the format of the subroutine is identical to those used by interpreted BASIC packages, both the Quick BASIC integrated development environment (QB.EXE) and the command line complier (BC.EXE) expect the subroutine to be in a specially formatted .QLB library file. Unlike interpreted BASIC packages, Quick BASIC actually links to the assembly .QLB library file so it is not necessary to include the "jmp QBPIOINT" instruction at location 0 (of the source file) as in interpreted BASIC.

.ASM Subroutine:

The following assembly code shows how the driver handles user arguments:

```
QBPIOINT        proc far        ; far call (dword return address)
                push bp         ; save base pointer
                mov bp,sp       ; save stack pointer
                .               ; [bp+6] holds offset of Flag
                .               ; [bp+8] holds offset of D
                .               ; [bp+10] holds offset of Mode
                .
                .               ; Program execution here
                .
                .
                pop bp          ; restore bp & sp prior to exit
                ret
QBPIOINT        endp
```

NOTE When creating a .QLB file, it is good practice to make a .LIB of the same version as a backup file.

Creation of a .QLB file is accomplished as follows:

1. Create the .ASM Source Code File 'EXAMPLE.ASM'

2. Assemble 'EXAMPLE.ASM' thus creating 'EXAMPLE.OBJ'

3. Link 'EXAMPLE.OBJ' with the "/q" option to create 'EXAMPLE.QLB'

```
MASM EXAMPLE ;
LINK /q  EXAMPLE ;
```

A .LIB file is created by:

1. Create the .ASM Source Code File 'EXAMPLE.ASM'

2. Assemble 'EXAMPLE.ASM' thus creating 'EXAMPLE.OBJ'

3. Use Utility LIB.EXE to add EXAMPLE.OBJ to 'EXAMPLE.LIB'

(Remove old EXAMPLE.OBJ from Library)

```
LIB EXAMPLE.LIB -EXAMPLE
```

(Create New .OBJ) **MASM EXAMPLE ;**
(Add New .OBJ to Library) **LIB EXAMPLE,LIB +EXAMPLE ;**

4. To use the .QLB file in the QB integrated environment/editor, invoke QB.EXE with the /l option (QB /l qlbname.qlb,) where qlbname.qlb is the file containing BASICsub.

5. To use the .LIB file with the command line complier (BC.EXE), simply specify "EXAMPLE.LIB" in the link process.

The Following Example Program Illustrates a QuickBASIC CALL:

```
10 '*****************************************************************
20 '*                                                               *
30 '*     QBPIOINT.BAS - Example showing use of PIOINT.QLB driver   *
40 '*                    for Quick Basic                            *
50 '*    Keithley MetraByte Corporation                            *
60 '*****************************************************************
70 '
80  DIM D%(16)
90  COMMON SHARED D%()
100 DECLARE SUB QBPIOINT (MD%, BYVAL DUMMY%, FLAG%)

        .
        .
        .

280 MD% = 0          'mode variable - integer type
290 FLAG% = 0        'flag or error variable - integer type

        .
        .
        .

330 'Call MODE 0 to initialize the PIO-INT hardware and set its
        'configuration:-
340 '(Note: You must execute mode 0 before selecting any other mode)
350 MD% = 0           'select mode 0
360 OPEN "PIOINT.ADR" FOR INPUT AS #1
370 INPUT #1, D%(0)   'read in base address from PIOINT.ADR file
380 CLOSE #1
390 'D%(0) = &H300    specify PIO-INT base I/O address,
400 '                 if PIOINT.ADR file is not used.
410 D%(1) = 0         'set PA port direction ( 0 = output, 1 = input )
420 D%(2) = 0         'set PB port direction ( 0 = output, 1 = input )
430 D%(3) = 0         'set PC0-3 port direction ( 0 = output, 1 = input )
440 D%(4) = 0         'set PC4-7 port direction ( 0 = output, 1 = input )
450 D%(5) = 7         'select operation on interrupt level 7
460 D%(6) = 0         'select bit change type interrupts on PA port
470 D%(7) = 0         'select pattern match interrupts on PB port
480 D%(8) = 2         'OR the port interrupts (see manual)
490 'Mode 0 only uses the first 9 elements of D%(16) to pass data
500 CALL QBPIOINT (MD%, VARPTR(D%(0)), FLAG%) 'execute the call to
        'initialize
510 PRINT
520 PRINT"Mode 0 initialization:-"
530 PRINT
540 IF FLAG% < 10 THEN PRINT"Errors:     ";EC$(FLAG%)
550 IF FLAG% >=10 THEN PRINT"Errors:     ";EC$(10);FLAG%-10;:PRINT")"

        .
        .
        .
```

A2.7 PIOINT.LIB GENERAL PURPOSE LIBRARY

pioint.LIB	This is a general purpose library file which provides control of the PIO-INT and related boards. This file can be linked with programs written in C, PASCAL, FORTRAN, or QuickBASIC to provide access to the PIO-INT operating modes.

NOTE: This library cannot be used with TurboPASCAL. However, TurboPASCAL may be used with Turbops.obj (see below).

The following is a brief description of the available call routines:

mscs_pioint(mode,param,flag)	:	Call from Microsoft C Small Model
mscm_pioint(mode,param,flag)	:	Call from Microsoft C Medium Model
mscl_pioint(mode,param,flag)	:	Call from Microsoft C Large Model
tcs_pioint(mode,param,flag)	:	Call from Turbo C Small Model
tcm_pioint(mode,param,flag)	:	Call from Turbo C Medium Model
tcl_pioint(mode,param,flag)	:	Call from Turbo C Large Model
msp_pioint(mode,param,flag)	:	Call from Microsoft PASCAL
QBpioint(mode,param,flag)	:	Call from Microsoft QuickBASIC
fpioint(mode,param,flag)	:	Call from Microsoft FORTRAN

Linking the Library "pioint.lib" to the user program is accomplished after program compilation by including it in the link line as follows:

```
link userprog.obj,userprog,,user.lib_pioint.LIB;
```

userprog.obj is an object module produced by compilation of the user program.
userprog should be used for the resultant executable .EXE file.
user.lib is any other user library, if applicable.

For TurboPASCAL, the entry point is:

```
tp_pioint(mode,D,flag)     :        Call from TurboPASCAL program
```

The user program should have the directive

```
{$L turbopas}
```

at the beginning of the user program. This directive will ensure that the TPC compiler/linker will include the proper interface object.

TURBOPAS.OBJ	(from the PCF-PIOINT disk)

* * * * *

KEITHLEY

Keithley Instruments, Inc.	28775 Aurora Road • Cleveland, Ohio 44139 • 440-248-0400 • Fax: 440-248-6168 **1-888-KEITHLEY (534-8453) • www.keithley.com**
Sales Offices: BELGIUM:	Bergensesteenweg 709 • B-1600 Sint-Pieters-Leeuw • 02-363 00 40 • Fax: 02/363 00 64
CHINA:	Yuan Chen Xin Building, Room 705 • 12 Yumin Road, Dewai, Madian • Beijing 100029 • 8610-6202-2886 • Fax: 8610-6202-2892
FINLAND:	Tietäjäntie 2 • 02130 Espoo • Phone: 09-54 75 08 10 • Fax: 09-25 10 51 00
FRANCE:	3, allée des Garays • 91127 Palaiseau Cédex • 01-64 53 20 20 • Fax: 01-60 11 77 26
GERMANY:	Landsberger Strasse 65 • 82110 Germering • 089/84 93 07-40 • Fax: 089/84 93 07-34
GREAT BRITAIN:	Unit 2 Commerce Park, Brunel Road • Theale • Berkshire RG7 4AB • 0118 929 7500 • Fax: 0118 929 7519
INDIA:	Flat 2B, Willocrissa • 14, Rest House Crescent • Bangalore 560 001 • 91-80-509-1320/21 • Fax: 91-80-509-1322
ITALY:	Viale San Gimignano, 38 • 20146 Milano • 02-48 39 16 01 • Fax: 02-48 30 22 74
JAPAN:	New Pier Takeshiba North Tower 13F • 11-1, Kaigan 1-chome • Minato-ku, Tokyo 105-0022 • 81-3-5733-7555 • Fax: 81-3-5733-7556
KOREA:	2FL., URI Building • 2-14 Yangjae-Dong • Seocho-Gu, Seoul 137-888 • 82-2-574-7778 • Fax: 82-2-574-7838
NETHERLANDS:	Postbus 559 • 4200 AN Gorinchem • 0183-635333 • Fax: 0183-630821
SWEDEN:	c/o Regus Business Centre • Frosundaviks Allé 15, 4tr • 169 70 Solna • 08-509 04 679 • Fax: 08-655 26 10
SWITZERLAND:	Kriesbachstrasse 4 • 8600 Dübendorf • 01-821 94 44 • Fax: 01-820 30 81
TAIWAN:	1FL., 85 Po Ai Street • Hsinchu, Taiwan, R.O.C. • 886-3-572-9077• Fax: 886-3-572-9031