PIO-16/16L(PC)V

For the IBM PC/AT

User's Guide

Copyright

Product Configuration

PIO-16/16L(PC)V Board ... 1
User's Guide (this booklet) ... 1

Unpacking:
This board is specially packed to prevent damage in shipping. It is wrapped in an anti-static bag.

Note!

Do not remove the board from its protective packaging until the computer case is open and ready for installation. Electrical static can cause damage to electronic components.

Note!

Check the contents to make sure that you have everything listed above. If you don't have all the items, please contact CONTEC.

User feedback

We have tried to consider all possible issues in the preparation of this User's Guide. Should you notice any omission, mistake, or questionable item in the document, please free to contact:

CONTEC MICROELECTRONICS U.S.A., INC.

Phone: (800)888-8884 (Call Toll Free)
(408)434-6767
Fax: (408)434-6884

Table of Contents

List of Figures

List of Tables

INTRODUCTION

About the PIO-16/16L(PC)V Board

The PIO-16/16L(PC)V is a 16 channel digital input and output interface board for the IBM PC/AT and compatible computers. It can also be installed into a CONTEC I/O expansion unit. It inputs 16 current-source signals and outputs 16 current-sink signals.

Features

- Photo-Insulated input/outputs providing improved noise resistance
- Up to 16 (8 signals x 2 groups) current-source type input signals
- Up to 16 (8 signals x 2 groups) current-sink type output signals
- Two input signals can also generate interrupt requests
- Up to 35 DVC, 200mA per signal, max.(2A per common, max.) output ability

Limited Three Year Warranty

CONTEC Interface boards are warranted by CONTEC MICRO-ELECTRONICS, U.S.A., Inc. to be free from defects in material and workmanship for up to three year from the date of purchase by the original purchaser.

Replacement or repair will be free of charge only when this device is returned to CONTEC U.S.A., freight prepaid with the original invoice.

This warranty is not applicable for scratches or normal wear, but only for the electronic circuitry and original boards. It is also not applicable if the device has been tampered with or damaged through abuse, mistreatment, neglect or unreasonable use, or if the original invoice is not included, in which case repairs will be considered beyond the warranty policy. If a replacement with a new device is needed, regular factory prices will be charged, and the product will be returned to you COD, and no other written warranty will apply.

The obligation of the warrantor is solely to repair or replace the product. In no event will the warrantor be liable for any incidental or consequential damages due to such defect or consequences that arise from inexperienced usage, misuse, or malfunction of this device.

How to Obtain Service

For replacement or repair, return the device freight prepaid, with a copy of the original invoice. Please obtain a Return Merchandise Authorization Number (RMA) from our Sales Administration Department before returning any product. No product will be accepted without an RMA number.

CONTEC MICROELECTRONICS U.S.A., INC.
2188 Bering Drive
San Jose, CA 95131

Phone: (800) 888-8884 (Call Toll Free)
(408) 434-6767
Fax: (408) 434-6884

Functions

• Input

This board installed on a personal computer (PC) inputs up to 16 digital signals in groups each consisting of eight signals from an external device and passes them to the PC. The PC accesses this board for input through the four arbitrarily configurable input ports. When the IN instruction is executed to read data through any of these input ports, the buffer gate corresponding to that input port opens to receive the group of digital signals from the external device. The signals sent to the PC at this time have negative logic. Since the two signals among the 16 input signals are user-assignable as interrupt inputs of the PC, the user can use them as interrupt request signals.

• Output

This board writes up to 16 digital signals in groups each consisting of eight signals to the external device. The PC accesses the board for output through the four arbitrarily configurable output ports. When the OUT instruction is executed to write data to any of these output ports, the latch circuit corresponding to that output port holds the data. The digital signals are then electrically insulated by the photocoupler and output to the connected external device as a group of signals via the transistor. The signals output to the external device at this time have negative logic. The data in the latch circuit remains intact until the OUT instruction is executed again.

Organization of This Guide

This guide consists of the following chapters and appendix:

Chapter 1. Setup
This chapter explains how to set switches on this board.

Chapter 2. External Connection
This chapter describes the interface connector and external I/O circuits on the board.

Chapter 3. I/O Port Bit Assignment
This chapter describes the assignments and definitions of the individual bits in the I/O ports on the board.

Chapter 4. Board Specifications
This chapter contains the specifications and block diagram of the board.

Appendix
This appendix provides useful information on using the board for your reference.

Chapter 1. SETUP

Component Locations

Figure 1. identifies the major parts on the board.

Note that the switch and jumper settings in the illustration below indicate their factory defaults.

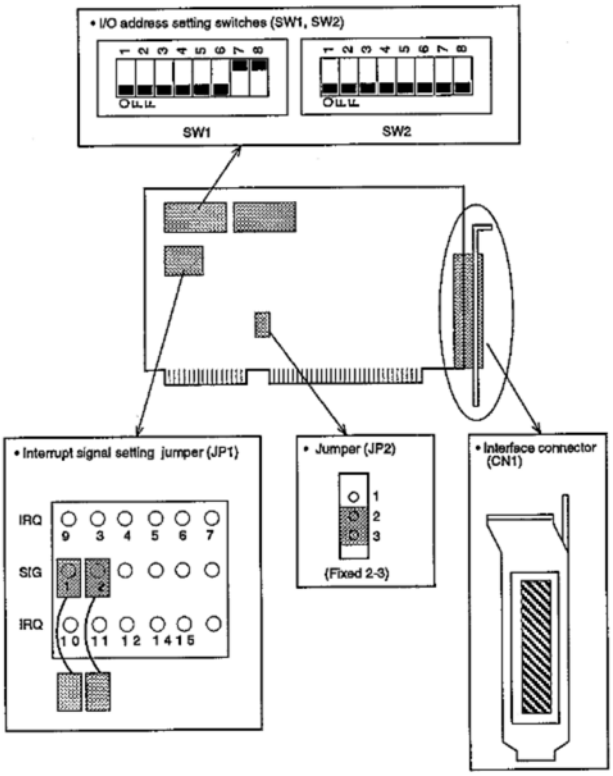


Figure 1. Names and Factory Defaults of Parts

Setting I/O Addresses

This board is an I/O device controlled by input/output instructions from by the personal computer. I/O devices include those built-in the personal computer and expansion boards. I/O addresses are numbers for distinguishing individual I/O devices. The I/O address assigned to each I/O device is a four-digit hexadecimal number, such as 0300H, used to identify that I/O device.

In general, each expansion board is controlled by using a range of consecutive I/O addresses. Of these consecutive I/O addresses, the first value is the I/O base address of the expansion board.
This board uses consecutive I/O addresses for the two ports.

Notes

The PC/AT and compatibles operate hardware devices by executing I/O instructions on I/O address in a range of [0000H to FFFFH]. On these PCs, however, specific I/O addresses are used or reserved by the system for the CRT, keyboard, and other controls as shown in the address map found in their technical manual. That is, the user cannot uses these system-assigned I/O addresses.
Although I/O addresses available to the user are limited, CONTEC recommends the ranges of I/O addresses listed in Figure 2. for use by this board.

Recommended I/O addresses

*300H to *31FH
*700H to *71FH
*B00H to *B1FH
*F00H to *F1FH (* : Any value from 0 to F)

Figure 2. Recommended I/O Addresses

Although these recommendations specify the three low-order digits of each I/O address (in hexadecimal), you can select the high-order digit freely from among 0 to F.

If your PC uses more than one expansion board, the I/O address range occupied by each board must not overlap that for another.

If a LAN board has been installed on your PC, 300H to 31FH have already been used for the board. Be careful not to assign an I/O address range to this board, which conflicts with the I/O address range for the LAN board.

Setting Method

Use the on-board DIP switches (SW1 and SW2) to set the I/O base address of this board. Individual bits in the SW1 and SW2 correspond to the 15 high-order bits (A15 to A1) in the I/O base address. Set A0 always to "0" (OFF).

The ON and OFF states of bits in the SW1 and SW2 correspond to the binary values of "1" and "0" in the I/O base address, respectively.

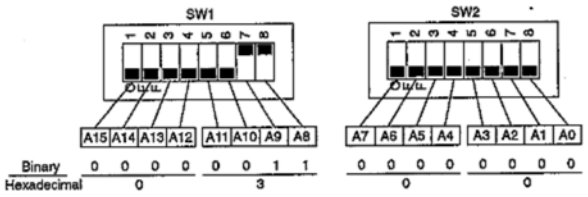


Figure 3. I/O Address Setting

Figure 3 shows an I/O base address setting of 0300H, assigning the I/O address range specified in Table 1. to this board.

Table 1. Example of I/O address range

Functions to be used	General-purpose I/O function
I/O addresses to be occupied	0300H to 0301H (2 ports)

Setting Interrupt Levels

This board can use signals, such as two digital signals among 16 input signals, as interrupt request signals. These signals are used to issue interrupt requests to the PC, making the interrupt functions of the PC available. Use the on-board jumper (JP1) to set interrupt levels.

To disable interrupts, use lead strapping connectors to prevent input signals from being connected to specific levels.

To enable interrupts, use the on-board jumper (JP1) to set interrupt levels. The interrupt levels set for this board are IRQs 3 to 7, 9 to 12, 14, and 15. Set those not used for the PC and for any other board. Up to four levels of interrupt request signals can be assigned, corresponding to input signals on a one-to-one basis.

Notes

(1) When using interrupts, set interrupt levels which are not used for any other resource.
(2) Do not plug or unplug any strapping connector on the JP1 when power has been supplied to the PC (or I/O expansion unit) on which this board has been installed.

Setting Method

Use the on-board jumper (JP1) to set interrupt levels.

Disabling Interrupts

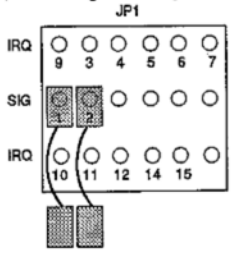


Figure 4. Disabling Interrupts

Enabling Interrupts

Use strapping connectors to connect input signals to the interrupt levels you want to assign. The assignable interrupt levels are IRQs 3 to 7, 9 to 12, 14, and 15. Note, however, that IRQs 10 to 15 cannot be used on PCs with XT (8-bit) buses.

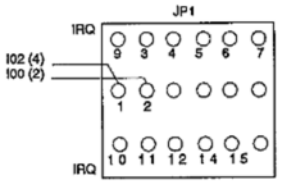
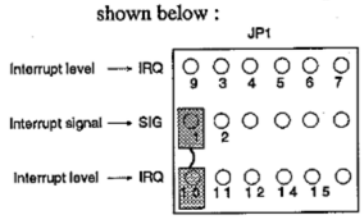


Figure 5. Interrupt Settings

Example : To connect SIG1(I02) from the interface board to IRQ10 on the PC/AT as an interrupt request signal, set the JP1 as shown below :



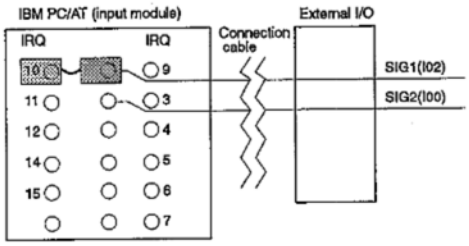


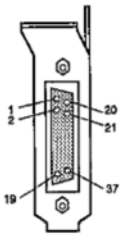
Figure 6. Sample Interrupt Settings

Chapter 2. EXTERNAL CONNECTION

Interface Connector

Connecting the Interface Connector

To connect an external device to this board, plug the cable from the device into the interface connector (CN1).



- Connector used
  37-pin D-sub connector (Female)
  DCLC-J37SAF-20L9 (mfd. by JAE) equivalent
  Screw nut : UNC #4-40 (inch screw)
- Applicable connector
  17JE-23370-02(D8C) (mfd. by DDK, Male)
  FDCD-37P (mfd. by HIROSE, Male)
  DC-37P-N (mfd. by JAE, Male)

Figure 7. Interface Connector

Optional Cables

Flat cable with 37-pin D-sub connectors at either end :

- PCB37P-1.5 (1.5m)
- PCB37P-3 (3m)
- PCB37P-5 (5m)

Shielded cable with 37-pin D-sub connectors at either end :

- PCB37PS-0.5 (0.5m)
- PCB37PS-1.5 (1.5m)
- PCB37PS-3 (3m)
- PCB37PS-5 (5m)

Flat cable with a 37-pin D-sub connector at one end :

- PCA37P-1.5 (1.5m)
- PCA37P-3 (3m)
- PCA37P-5 (5m)

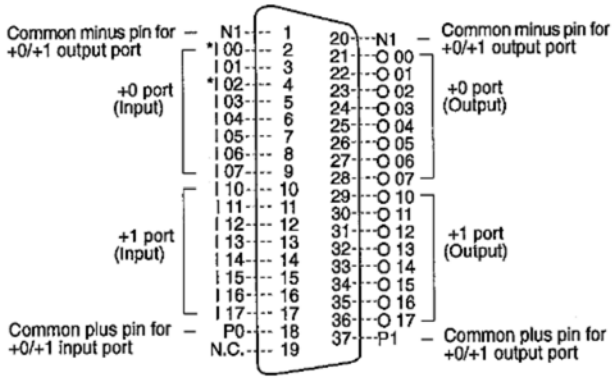
Shielded cable with a 37-pin D-sub connector at one end :

- PCA37PS-1.5 (1.5m)
- PCA37PS-3 (3m)
- PCA37PS-5 (5m)

Optional Accessories
Screw Terminal : EPD-37
Termination Panel : DTP-3(PC)
Termination Panel : DTP-4(PC)
Signal Monitor for Digital I/O : CM-32(PC)E

Interface Connector Pin Assignment

To connect an external device to this interface board, plug it into the on-board 37-pin connector.



*I00 and *I02 are also used as interrupt signals

Figure 8. Interface Connector Pin Assignments

Input Circuit and Output Circuit

Input Circuit

The input circuit of this board is illustrated in following Figure. The on-board photocouplers isolate internal input circuits from outside devices. The input channels are current source type signals. You need an additional power supply that is isolated from the PC system to drive these insulation circuits. When you use a 12VDC power, each input channel will consumes about 4mA current; when 24VDC external power supply is selected, each input channel will consumes about 8mA current.

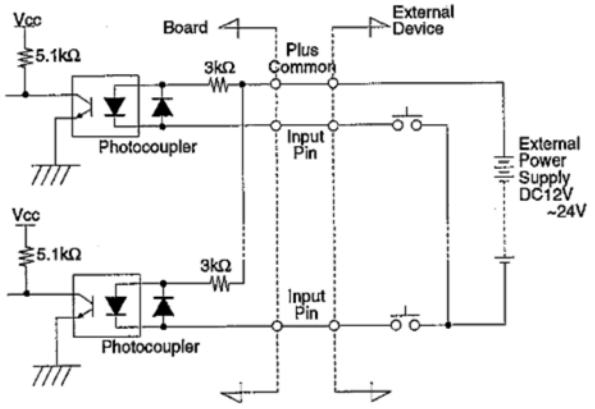


Figure 9. Input Circuit

Output Circuit

The output circuit of this board is illustrated in following Figure. The output channel is a photocoupler-insulated open-collector type (sink type). You need an additional power supply that is isolated from the PC system to drive these insulation circuits. The maximum output current rating is 200 mA per channel or 2 A per common (16 output channels share a common power supply).

Note!

There are not surge voltage protection circuits on board for protecting output transistors. To drive inductive loads such as relays and lamps by this board, therefore, a measures against surge voltage must be taken on the load side.

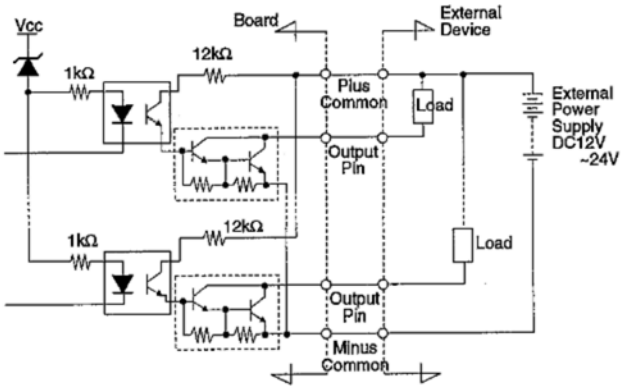


Figure 10. Output Circuit

Chapter 3. I/O PORT BIT ASSIGNMENT

I/O Port Bit Assignment

This board is accessed by the PC through two I/O ports. A group of eight external digital signals (input and output signals) connected to the on-board connector is assigned to each port.

Input Port Bit Assignment

The bits in input ports are assigned as shown below:

I/O base address		D7	D6	D5	D4	D3	D2	D1	D0
	+0H	Input group 0 (+0 port)							
		I07 [9]	I06 [8]	I05 [7]	I04 [6]	I03 [5]	I02 [4]	I01 [3]	I00 [2]
	+1H	Input group 1 (+1 port)							
		I17 [17]	I16 [16]	I15 [15]	I14 [14]	I13 [13]	I12 [12]	I11 [11]	I10 [10]

Ixx is an input signal name; numbers in brackets [ ] are connector pin numbers.
I00 and I02 can also serve as interrupt signals.

Figure 11. Input Port Bit Assignment

When input data is "ON," the corresponding bit contains "1." If the data is "OFF," the bit contains "0."

Example : Check whether I07 is "ON" when the I/O base address is 300H.

(1) BASIC (MS-DOS version)

```
DATA%=INPUT(&H300)
IF(DATA% AND &H80)=&H80 THEN •••
```

(2) Microsoft C or C++ (MS-DOS version)

```
data_in=inp(0x300);
if(data_in & 0x80) •••
```

Output Port Bit Assignment

The bits in output ports are assigned as shown below:

I/O base address	D7	D6	D5	D4	D3	D2	D1	D0
+0H	Output group 0 (+0 port)							
	O07 [28]	O06 [27]	O05 [26]	O04 [25]	O03 [24]	O02 [23]	O01 [22]	O00 [21]
+1H	Output group 1 (+1 port)							
	O17 [36]	O16 [35]	O15 [34]	O14 [33]	O13 [32]	O12 [31]	O11 [30]	O10 [29]

Oxx is an output signal name; numbers in brackets [ ] are connector pin numbers.

Figure 12. Output Port Bit Assignment

When "1" is output to a bit, the corresponding output data is set to "ON." If "0" is output to the bit, the data is set to "OFF."

Example : Set only O07 to "ON" when the I/O base address is 300H.

(1) BASIC (MS-DOS version)

```
OUT &H300,&H80
```

(2) Microsoft C or C++ (MS-DOS version)

```
outp(0x300,0x80);
```

Chapter 4. BOARD SPECIFICATIONS

Block Diagram

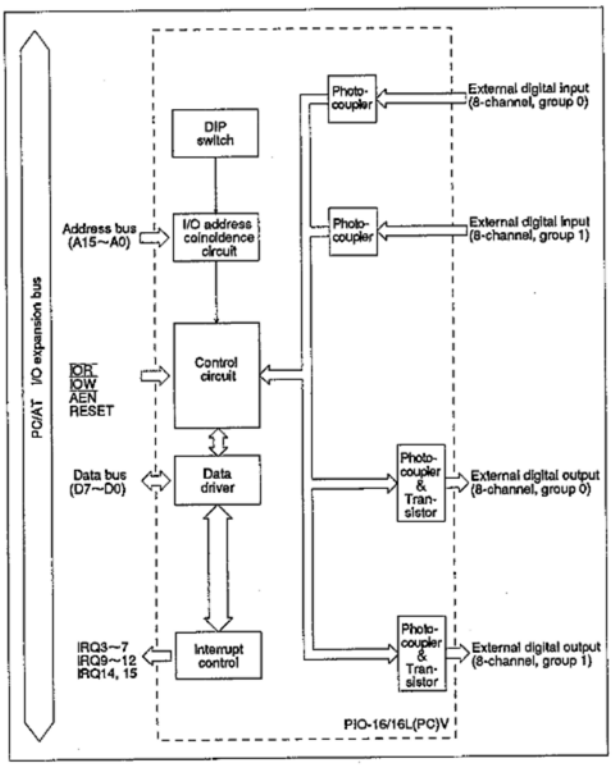


Figure 13. Block Diagram

Specifications

Table 2. lists the major specifications of this board.

Table 2. Specifications

	Item		Specification
Input section	Input system		Photocoupler-insulated current source-type input (negative logic)
	Input resistance		3kΩ
	Input ON current		3.4 mA or more
	Input OFF current		0.16 mA or less
	Number of input signal channels		16 channels (including 2 channels available for interrupts) Note: A set of 16 channels share one common power supply.
	Response time		1 ms or less
Output section	Output system		Photocoupler-insulated current sink-type output (Negative logic)
	Ratings	Withstand output voltage	35 VDC (Max.)
		Output current	200 mA max (per channel) (2 A max per common pin)
	Number of output channels		16 (A set of 16 channels shares one common power supply.)
	Response time		1 ms or less
Common section	I/O address		8 bits x 2 ports occupied
	Interrupts		Some of IRQs 3 to 7, 9 to 12, 14, and 15 (up to 2 IRQs at a time) Interrupt generated at High->Low edge
	External circuit power supply		12 to 24 VDC (±15%) Note: 4 mA/12 V to 8 mA/24 V per input channel
	Current consumption		DC5V 50mA (Max.)
	Operating conditions		0 to 50C, 20 to 90% (without condensation)
	Connecting distance		About 50 m (depending on wiring environment)
	Board dimensions		160.0 x 122.0 x 18.5mm
	Board weight		120g

APPENDIX

A. Interrupts on the PC/AT Series

Some boards provided by CONTEC can use the interrupt function of the PC. If you use these boards to use the interrupt function, refer to the information in this appendix for appropriate interrupt servicing.

The PC/AT and compatibles provides interrupt signals at interrupt levels of IRQs 3 to 7, 9 to 12, 14, and 15. (The XT bus machines supports IRQs 2 to 7.)

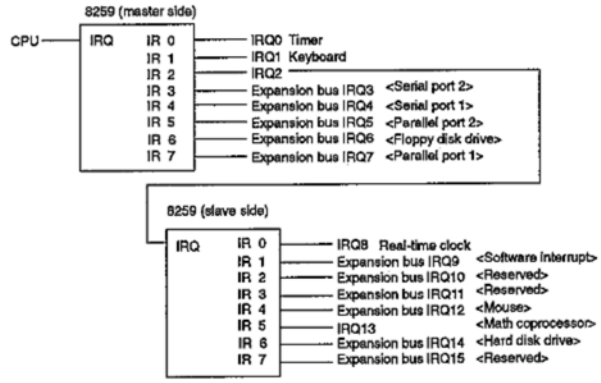
Although expansion bus interrupt levels are assigned to individual peripheral devices, expansion boards can use other interrupt levels not occupied on the system.

Note that appropriate processing is required for setting up of the PC environment, restoring the interrupt environment, and for response in interrupt handlers before you can use the interrupt function of the PC/AT or compatible.

Interrupt Levels and Interrupt Vectors

Interrupt Levels

The PC/AT and compatibles have a master/slave configuration with two interrupt controllers (8259). As illustrated in Figure 14, expansion bus IRQs 3 to 7 are assigned on the master side while IRQs 9 to 12, 14, and 15 are assigned on the slave side. Since programs using the interrupt function control the interrupt controllers, processing is different depending on each interrupt level and between the master and slave sides.



Note: Devices in angle brackets < > are standard peripheral devices to which expansion bus interrupt levels are assigned.

Figure 14. Interrupt Controllers

Note!

- *On the PC/AT and compatibles, the board can use IRQ9 but cannot use IRQ2.*
- *The interrupt controller accepts an interrupt at the rising edge from the LOW to HIGH level of the interrupt request signal from the board.*
- *Those interrupt levels cannot be used which have already been used for internal components of the PC or its peripheral devices.*
- *Different boards cannot share the same interrupt level.*

Interrupt Vectors

Interrupt levels and their respective interrupt service routines are associated with each other in an interrupt vector table. The vector table is 1K byte, starting at the low-order address (zero address) in memory. Each vector in the table contains the 4-byte starting address of an interrupt service routine. Table 3. below lists the addresses and vector numbers assigned for interrupt controllers in the interrupt vector table.

Table 3. Addresses and Vectors Assigned in Interrupt Vector Table

	Address	Vector No.	Application	
Master side	0020H to 0023H	08H	Timer	(IRQ0)
	0024H to 0027H	09H	Keyboard	(IRQ1)
	0028H to 002BH	0AH		(IRQ2)
	002CH to 002FH	0BH	Serial port 2	(IRQ3)
	0030H to 0033H	0CH	Serial port 1	(IRQ4)
	0034H to 0037H	0DH	Parallel port 2	(IRQ5)
	0038H to 003BH	0EH	Floppy disk drive	(IRQ6)
	003CH to 003FH	0FH	Parallel port 1	(IRQ7)
Slave side	01C0H to 01C3H	70H	Real-time clock	(IRQ8)
	01C4H to 01C7H	71H	Software interrupt	(IRQ9)
	01C8H to 01CBH	72H	Reserved	(IRQ10)
	01CCH to 01CFH	73H	Reserved	(IRQ11)
	01D0H to 01D3H	74H	Mouse	(IRQ12)
	01D4H to 01D7H	75H	Math coprocessor	(IRQ13)
	01D8H to 01DBH	76H	Hard disk drive	(IRQ14)
	01DCH to 01DFH	77H	Reserved	(IRQ15)

Processing for Enabling the Interrupt Function

To execute an interrupt handler in response to an interrupt request from the board, the interrupt environment of the PC must be set up to the interrupt handler. When the interrupt handler is executed, the prescribed response to the interrupt controller is performed. Upon completion of serving the interrupt, the interrupt environment of the PC is restored to the original state.

The remainder of this appendix details each process.

(1) Setting up the interrupt environment

1) Details on setting up the interrupt environment

Interrupt environment setup is divided into the following two operations:

- Setting the interrupt vector table
- Setting the interrupt controllers

Setting the Interrupt Vector Table

Hold the current contents of the interrupt vector table and set the interrupt vector address (start address) of the interrupt handler. Hold the contents of the current interrupt vector table for the interrupt levels to be used so that the interrupt environment can be restored correctly. In the interrupt vector table, then, set the vector address of the interrupt handler.

Setting the Interrupt Controllers

Hold the mask status in IMR (interrupt mask register) of the interrupt controller, cancel masking set by the IMR, and clear the ISR (interrupt service register).

Hold the current mask status in the IMR for restoration of the interrupt environment. Then, cancel masking the bit corresponding to the interrupt level to be used to enable that level of interrupts. Lastly, clear the ISR, or the interrupt request currently being accepted by the interrupt controller, to restart receiving interrupt request signals. Table 4.- shows the correspondence between interrupt levels and interrupt controller data.

Table 4. Interrupt Levels and Interrupt Controller Data

	Vector No.	Interrupt signal	Canceling IMR masking		Clearing ISR	
			Port address	Data	Port address	Data
Master side	0AH	IRQ2	0021H	FBH	0020H	62H
	0BH	IRQ3	0021H	F7H	0020H	63H
	0CH	IRQ4	0021H	EFH	0020H	64H
	0DH	IRQ5	0021H	DFH	0020H	65H
	0EH	IRQ6	0021H	BFH	0020H	66H
	0FH	IRQ7	0021H	7FH	0020H	67H
Slave side	71H	IRQ9	00A1H	FDH	00A0H	61H
	72H	IRQ10	00A1H	FBH	00A0H	62H
	73H	IRQ11	00A1H	F7H	00A0H	63H
	74H	IRQ12	00A1H	EFH	00A0H	64H
	76H	IRQ14	00A1H	BFH	00A0H	66H
	77H	IRQ15	00A1H	7FH	00A0H	67H

Disable interrupts in advance to prevent interrupts from being generated when the vector address of an interrupt handler is set in the interrupt vector table, when masking set by the IMR is canceled, or when the ISR is cleared. After interrupt controller setup has all been completed, enable interrupts.

Mask
A mask is the character or bit pattern for changing or isolating information on a specific bit location in another bit pattern.

IMR (interrupt mask register)
The IMR holds the bit corresponding to the interrupt request line to be masked. The interrupt request for the IR input corresponding to the bit set in the IMR is therefore suspended.

ISR (interrupt service register)
The ISR holds the interrupt level currently being serviced. The ISR is updated the moment the command for terminating the current interrupt is issued.

2) Sample programs for setting up the interrupt environment

Example for the master side (IRQ5) (Microsoft Macro Assembler)

The sample program shown below specifies the vector number (0dh) for IRQ5, executes an MS-DOS function call to get the contents of the vector table, then stores the results in variables. After disabling interrupts, the program executes the MS-DOS function call to update the vector table to the vector address of the interrupt handler.

The program inputs the mask status in the current IMR from the port address (0021h) on the master interrupt controller and stores it in a variable. To cancel masking the bit corresponding to IRQ5, the program outputs the AND with (DFh) and enables IRQ5. The program then outputs (65h) to the port address (0020h) on the master interrupt controller to clear the ISR. After interrupt controller setup has all been completed, the program enables interrupts.

To change the interrupt level to IRQ 2, 3, 4, 6, or 7, edit the italic portions of the program listing according to Table 4.

```
mov   al, 0dhxs                ;Specify vector number
mov   ah, 35h                  ;Get current interrupt vector
int   21h                      ;MS-DOS system call
mov   orgvect_off, bx          ;Store offset address
mov   orgvect_seg, es          ;Store segment address

cli                            ;Disable interrupts

mov   ds, inthandler_segment   ;Segment address of interrupt han
                                dler
mov   dx, inthandler_offset    ;Offset address of interrupt handler
mov   l, 0dh                   ;Specify vector number
mov   ah, 25h                  ;Change interrupt vector
int   21h                      ;MS-DOS system call

in     al, 0021h               ;Input mask status in current IMR on
                                master side
mov   rgimr_ma, al             ;Store mask status in current IMR on
                                master side
and   al, 0dfh                 ;Cancel masking of corresponding bit
out   0021h, al                ;Cancel masking by IMR on master
                                side

mov   al, 65h                  ;Specify clear data for ISR on
                                master side
out   0020h, al                ;Clear ISR on master side

sti                            ;Enable interrupts
```

Example for the master side (IRQ5) (Microsoft C or C++)

The sample program shown below declares the pointer variable with an interrupt property to hold the current interrupt vector. The program then uses a _dos_getvect() function to specify the vector number (0dh) of IRQ5, get the contents of the vector table, and to store the results in variables. After disabling interrupts, the program executes a _dos_setvect() function to update the vector table to the vector address of the interrupt handler.

The program inputs the mask status in the current IMR from the port address (0021h) on the master interrupt controller and stores it in a variable. To cancel masking the bit corresponding to IRQ5, the program outputs the AND with (DFh) and enables IRQ5. The program then outputs (65h) to the port address (0020h) on the master interrupt controller to clear the ISR. After interrupt controller setup has all been completed, the program enables interrupts.

To change the interrupt level to IRQ 2, 3, 4, 6, or 7, edit the italic portions of the program listing according to Table 4.

```
void   _interrupt_far inthandler(void);
                    /* Declare prototype of interrupt handler */
void     (_interrupt_far * orgvect) (void);
                    /* Pointer variable for holding interrupt
                       vector */
orgvect = _dos_getvect(0x0d);
                    /* Get current interrupt vector */
_disable( );        /* Disable interrupts */
_dos_setvect(0x0d, inthandler);
                    /* Change interrupt vector */
outp(0x0021, (orgimr_ma=inp(0x0021))&0xdf);
                    /* Store current IMR on master side and
                       cancel masking*/
outp(0x0020, 0x65);
                    /* Clear ISR on master side */
_enable( );         /* Enable interrupts */
```

Note!

Some older versions of Microsoft C cannot use interrupt functions. Refer to the Runtime Library Reference to check the version in use.

Example for the slave side (IRQ12) (Microsoft Macro Assembler)

The interrupt levels (IRQs 9, 10, 11, 12, 14, and 15) on the slave side are connected to the CPU through the master interrupt controller. The sample program shown below therefore sets up both of the master and slave interrupt controllers.

The program specifies the vector number (74h) for IRQ12, executes an MS-DOS function call to get the contents of the vector table, then stores the results in variables. After disabling interrupts, the program executes the MS-DOS function call to update the vector table to the vector address of the interrupt handler.

After enabling the master interrupt controller, the program inputs the mask status in the current IMR from the port address (00A1h) on the slave side and stores it in a variable. To cancel masking the bit corresponding to IRQ12, the program outputs the AND with (EFh) and enables IRQ12. After clearing the ISR on the master interrupt controller, the program outputs (64h) to the port address (00A0h) on the slave side to clear the ISR. After interrupt controller setup has all been completed, the program enables interrupts.

To change the interrupt level to IRQ 9, 10, 11, 14, or 15, edit the italic portions of the program listing according to Table 4.

```
mov     al, 74h                 ;  Specify vector number
mov     ah, 35h                 ;  Get current interrupt vector
int     21h                     ;  MS-DOS system call
mov     orgvect_off, bx         ;  Store offset address
mov     orgvect_seg, es         ;  Store segment address

cli                             ;  Disable interrupts

mov     ds, inthandler_segment  ;  Segment address of interrupt
                                   handler
mov     dx, inthandler_offset   ;  Offset address of interrupt
                                   handler
mov     al, 74h                 ;  Specify vector number
mov     ah, 25h                 ;  Change interrupt vector
int     21h                     ;  MS-DOS system call

in      al, 21h                 ;  Input mask status in current IMR
                                   on master side
mov     orgimr_ma, al           ;  Store mask status in current IMR
                                   on master side
and     al, 0fbh                ;  Cancel masking of corresponding
                                   bit
out     0021h, al               ;  Input mask status in current IMR
                                   on slave side

in      al, 00a1h               ;  Store mask status in current IMR
                                   on slave side
mov   orgimr_su, al             ;  Store mask status in current IMR
                                   on slave side
and     al, 0efh          ;  Cancel masking of corresponding bit
out     00a1h, al         ;  Cancel masking by IMR on slave side

mov     al, 62h           ;  Specify clear data for ISR on master
                             side
out     0020h, al         ;  Clear ISR on master side

mov     al, 64h           ;  Specify clear datafor ISR on slavo side
out     00a0h, al         ;  Clear ISR on slave side

sti                       ;  Enable interrupts
```

Example for the slave side (IRQ12) (Microsoft C or C++)

The interrupt levels (IRQs 9, 10, 11, 12, 14, and 15) on the slave side are connected to the CPU through the master interrupt controller. The sample program shown below therefore sets up both of the master and slave interrupt controllers.

The sample program shown below declares the pointer variable with an interrupt property to hold the current interrupt vector. The program then uses a _dos_getvect() function to specify the vector number (74h) of IRQ12, get the contents of the vector table, and to store the results in variables. After disabling interrupts, the program executes a _dos_setvect() function to update the vector table to the vector address of the interrupt handler.

After enabling the master interrupt controller, the program inputs the mask status in the current IMR from the port address (00A1h) on the slave side and stores it in a variable. To cancel masking the bit corresponding to IRQ12, the program outputs the AND with (EFh) and enables IRQ12. After clearing the ISR on the master interrupt controller, the program outputs (64h) to the port address (00A0h) on the slave side to clear the ISR. After interrupt controller setup has all been completed, the program enables interrupts.

To change the interrupt level to IRQ 9, 10, 11, 14, or 15, edit the italic portions of the program listing according to Table 4.

```
void     _interrupt_far inthandler(void);
               /* Declare prototype of interrupt handler */
void     (_interrupt_far * orgvect)(void);
               /* Pointer variable for holding interrupt vector */

orgvect = _dos_getvect(0x74);
               /* Get current interrupt vector */
_disable( );   /* Disable interrupts */
_dos_setvect(0x74, inthandler);
               /* Change interrupt vector */
outp(0x0021,(orgimr_ma=inp(0x0021))&0xfb);
               /* Store current IMR on master side and cancel
                  masking*/
outp(0x00a1,(orgimr_su=inp(0x00a1))&0xef);
               /* Store current IMR on slave side and cancel
                  masking*/
outp(0x0020, 0x62);        /* Clear ISR on master side */
outp(0x00a0, 0x64);        /* Clear ISR on slave side */
_enable( );                /* Enable interrupts */
```

(2) Restoring the interrupt environment

1) Details on restoring the interrupt environment

Restoration of the interrupt environment is divided into the following operations:

- Restoring the interrupt controller
- Restoring the interrupt vector table

Restoring the Interrupt Controller

Restore the mask status in the IMR of the interrupt controller to the previous state existing prior to cancellation of masking. Set the IMR mask status preserved since cancellation of masking. Disable interrupts in advance to prevent interrupts from being generated when the interrupt vector table is set again or when the IMR mask status is restored. After interrupt controller setup has all been completed, enable interrupts.

Restoring the Interrupt Vector Table

Restore the interrupt vector table to the state existing before the last update. Set that content of the interrupt vector table which has been preserved since the last update as information corresponding to the interrupt level to be used.

2) Sample programs for restoring the interrupt environment

Example for the master side (IRQ5) (Microsoft Macro Assembler)

After disabling interrupts, the sample program shown below outputs the pre-update IMR mask status to the port address (0021h) on the master interrupt controller. The program then specifies the vector number (0dh) for IRQ5 and executes an MS-DOS function call to restore the contents of the vector table. After restoration has all been completed, the program enables interrupts.

To change the interrupt level to IRQ 2, 3, 4, 6, or 7, edit the italic portion of the program listing according to Table 4.

```
cli                             ;  Disable interrupts
mov     al, orgimr_ma           ;  Mask status of pre-update
                                   IMR on master side
out     0021h, al               ;  Restore masking in IMR on
                                   master side
mov     ds, orgvect_seg         ;  Pre-update segment address
mov     dx, orgvect_off         ;  Pre-update offset address
mov     al, 0dh                 ;  Specify vector number
mov     ah, 25h                 ;  Restore interrupt vector
int     21h                     ;  MS-DOS system call

sti                             ;  Enable interrupts
```

Example for the master side (IRQ5) (Microsoft C or C++)

After disabling interrupts, the sample program shown below outputs the pre-update IMR mask status to the port address (0021h) on the master interrupt controller. The program then executes a _dos_setvect() function to restore the contents of the vector table to the pre-update values. After restoration has all been completed, the program enables interrupts.

To change the interrupt level to IRQ 2, 3, 4, 6, or 7, edit the italic portion of the program listing according to Table 4.

```
_disable( );                    /* Disable interrupts */
outp(0x0021, orgimr_ma);        /* Restore masking in
                                   MR on master side */
_dos_setvect(0x0d, orgvect);    /* Restore interrupt vector */
_enable( );                     /* Enable interrupts */
```

Example for the slave side (IRQ12) (Microsoft Macro Assembler)

After disabling interrupts, the sample program shown below outputs the pre-update IMR mask status to the port address (00A1h) on the slave interrupt controller. The program then specifies the vector number (74h) for IRQ12 and executes an MS-DOS function call to restore the contents of the vector table. After restoration has all been completed, the program enables interrupts.

To change the interrupt level to IRQ 9, 10, 11, 14, or 15, edit the italic portion of the program listing according to Table 4.

```
cli                             ;  Disable interrupts
mov     al, orgimr_su           ;  Mask status of pre-update IMR
                                   on slave side
out     00A1h, al               ;  Restore masking in IMR on slave
                                   side
mov     ds, orgvect_seg         ;  Pre-update segment address
mov     dx, orgvect_off         ;  Pre-update offset address
mov     al, 74h                 ;  Specify vector number
mov     ah, 25h                 ;  Restore interrupt vector
int     21h                     ;  MS-DOS system call

sti                             ;  Enable interrupts
```

Example for the slave side (IRQ12) (Microsoft C or C++)

After disabling interrupts, the sample program shown below outputs the pre-update IMR mask status to the port address (00A1h) on the slave interrupt controller. The program then executes a _dos_setvect() function to restore the contents of the vector table to the pre-update values. After restoration has all been completed, the program enables interrupts.

To change the interrupt level to IRQ 9, 10, 11, 14 or 15 edit the italic portion of the program listing according to Table 4.

```
_disable( );                    /* Disable interrupts */
outp(0x00a1, orgvect);          /* Restore masking in IMR on slave
                                   side */
_dos_setvect(0x74, orgvect);    /* Restore interruptvector */
_enable( );                     /* Enable interrupts */
```

(3) Processing of interrupt handler

1) Details on processing of an interrupt handler

Processing of an interrupt handler is divided into the following operations:

- Enabling upper-level interrupts
- Saving registers
- Interrupt handling
- Response to the interrupt controller
- Restoring the registers
- Returning from the interrupt handler

Upon starting execution of an interrupt handler, enable upper-level interrupts in consideration of operations of other programs so that upper-level interrupts can be accepted even during execution of the interrupt handler. Save the registers used by the interrupt handler and execute the process of the interrupt handler itself. When that process has been completed, perform response to the interrupt controller to notify it of termination of interrupt handling. Lastly, restore the registers used and return to the process prior to execution of the interrupt handler.

2) Sample programs for processing of interrupt handler

Example in Microsoft Macro Assemble

```
sti                 ;Enable upper-level interrupts
push    ax          ;Save registers
push    bx
push    cx
push    dx
push    si
push    es
Interrupt handling
Response to interrupt controller

pop     es          ;Restore registers
pop     si
pop     dx
pop     cx
pop     bx
pop     ax
iret                ;Return from interrupt handler
```

Example in Microsoft C or C++

```
void    _interrupt_far inthandler(void)
{
  _enable( );       /*Enable upper-level interrupts*/
  Interrupt handling
  Response to interrupt controller

}
```

(4) Response to interrupt controller
Upon completion of interrupt handling, perform response to the interrupt controller to notify it of termination of interrupt handling. This processing differ depending on whether the interrupt level used is on the master or slave side.

1) When the interrupt level used is on the master side
If the interrupt level (IRQ 2 to 7) on the master side has been used, notify only the master interrupt controller. Send EOI (End Of Interrupt) to the interrupt controller to reset the bit corresponding to the interrupt service currently in process.

Example for Response (Microsoft Macro Assembler)

```
mov     al, 20h          ;Specify data for EOI
out     0020h, al        ;Send EOI to master side
```

Example for Response (Microsoft C or C++)

```
outp(0x0020, 0x20);  /* Send EOI to master side */
```

EOI (End Of Interrupt)
EOI notifies the interrupt controller of termination of interrupt servicing to reset the interrupt service status.

2) When the interrupt level used is on the slave side

The interrupt levels (IRQs 9, 10, 11, 12, 14, and 15) on the slave side are connected to the CPU through the master interrupt controller. The interrupt handler therefore performs response to both of the master and slave interrupt controllers. The interrupt handler sends EOI (End Of Interrupt) to each interrupt controller to reset the interrupt controller bit corresponding to the interrupt service currently in process.

Note, however, that if any interrupt request other than that currently being serviced on the slave side has been left, response to the master interrupt controller is not performed. Check whether any other interrupt request has been left, then perform prescribed processing.

Example for Response (Microsoft Macro Assembler)

```
   mov   al, 20h          ;pecify data for EOI
   out   00a0h, al        ;Send EOI to slave side
   in    al, 02efh        ;Reserve interrupt controller
                           recovery time

   mov   al, 0bh          ;Input data to ISR on slave side
   out   00a0h, al        ;Send command to slave side
   in    al, 02efh        ;Reserve interrupt controller
                           recovery time
   in    al, 00a0h        ;Input data to ISR on slave side
   or    al, al           ;Check for any other interrupt request
   jnz PEND

   mov   al, 20h          ;Specify data for EOI
   out   0020h, al        ;Send EOI to master side
PEND:
```

Example for Response (Microsoft C or C++)

```
outp(0x00a0, 0x20);        /* Send EOI to slave side */
rt=inp(0x02ef);            /* Reserve interrupt controller
                              recovery time */
outp(0x00a0, 0x0b);        /* Send data input command for
                              ISR on slave side */
if (inp(0x00a0) == 0)
        /* Input data to ISR on slave side and check forany
           other interrupt request */
    outp(0x0020, 0x20);    /* Send EOI to master side */
```

Note!

"in" for an empty port reserves a recovery time of at least 0.5 microsecond. The above example reads the COM4 scratch register.

B. LSI Recovery Time

Due to the ever higher CPU clock rates used in PCs, restrictions apply when controlling a peripheral LSI device by software. Table 12 lists the LSIs used on CONTEC boards which require special consideration when accessing. Take note of the following point when accessing these LSIs.

In PCs using i386 or earlier CPUs, software waits (JMP $+2) can be used to provide a recovery time when accessing the LSI. However, software waits cannot be used to provide a recovery time in PCs with a i486 or later CPU because of the CPU cache memory function.

The following describes one method of providing the recovery time when using an i486 or later CPU.

In the PC/AT and compatible computers, executing an IN instruction for the port at I/O address 2EFh (COM4 scratch register) takes a minimum of 0.5µs. As this time does not depend on the CPU type or clock rate, the time can be used to provide the recovery time. After accessing any of the devices listed in Table 5, execute the IN instruction for the 2EFh port the required number of times to provide the recovery time.

Table 5. Number of Times the IN Instruction Must be Executed for the 2EFh Port After Accessing the LSI

LSI Device	Output	Input
i8237 or equivalent	None	None
i8254 or equivalent	Once	Once
i8255 or equivalent	Once	Once
i8259 or equivalent	Once	Once
NS16550 or equivalent	Once	Once
µPD7210C	Once	None

Example program (for accessing an i8254 or equivalent)

•Microsoft Macro Assembler

```
OUT DX, AL          ; Access to the i8254
IN  AL, 2EFH        ; Execute IN AL, 2EFH once to provide
                      the recovery time
```

•Microsoft C/C++

```
outp(port, byte); /* Access to the i8254 */
rt=inp(0x2ef);    /* Execute rt=inp(0x2ef); once to
                     provide the recovery time */
```

C. Sample Programs

Sample Input Program in Q-BASIC

This program is written in Q-BASIC. The program inputs data from individual ports and displays the values in hexadecimal on the screen.

Specification

The program inputs data from the 0300H, 0301H, 0302H, and 0303H ports and displays their values on the screen. This board only occupies two ports; the input data at 0302H and 0303H are FFH.

Preparation

Perform the following preparation before you can run the program.

(1) Remove all strapping connectors from the JP1 because the program use any interrupt.
(2) Set the port address to 0300H using the DIP switches (SW1 and SW2) as shown below:

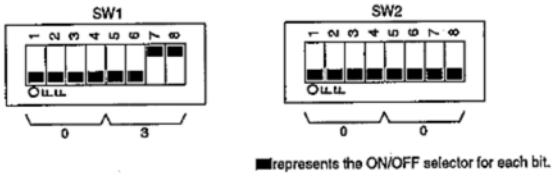


Figure 15. DIP Switch Settings

(3) Leave the JP2 set by factor default (connected between terminals 2 and 3).

Figure 16. Function Select Jumper Setting

Display Screen and Flowchart

(1) Display screen

```
**********************************
*     Byte Data Input Sample     *
**********************************

PORT      INPUT DATA    D7 D6 D5 D4 D3 D2 D1 D0
0300H          AAH       1  0  1  0  1  0  1  0
0301H          55H       0  1  0  1  0  1  0  1
0302H          FFH       1  1  1  1  1  1  1  1
0303H          FFH       1  1  1  1  1  1  1  1

CONTINUE    ---> PRESS ANY KEY
END         ---> PRESS ESC KEY
```

(2) Flowchart

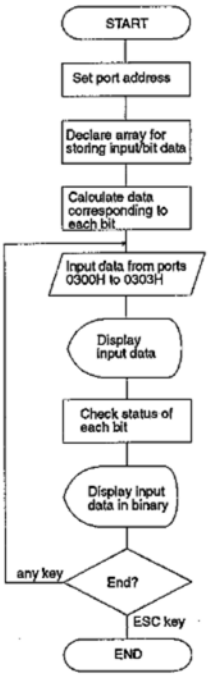


Figure 17. Flowchart for Sample Input Program

```
'      ************************************
'      *     Sample Program               *
'      *           data input    Ver 1.0  *
'      ************************************

    CLS :    WIDTH 80, 25
    DIM IN.DATA(4), BIT(8)
    SETPORT = &H300
    LOCATE 3, 20: PRINT "************************************"
    LOCATE 4, 20: PRINT "*     Byte Data Input Sample     *"
    LOCATE 5, 20: PRINT "************************************"

    FOR J = 0 TO 7: BIT(J) = 2 ^ J: NEXT J
LOOP1:
    LOCATE 9, 17: PRINT "port     input data   D7 D6 D5 D4 D3 D2 D1
D0"
    LOCATE 19, 25: PRINT "End       ---> Push ESC key"
    LOCATE 18, 25: PRINT "Input     ---> Push any key"
WAIT1:
    X$ = INKEY$: IF X$ = "" THEN GOTO WAIT1 ELSE IF ASC(X$) = &H1B
THEN GOTO PEND

'     **********************
'     *    Input  data     *
'     **********************

    FOR I = 0 TO 3
        IN.DATA(I) = INP(SETPORT + I)
        LOCATE I + 10, 17
        PRINT USING "&          &       "; RIGHT$("000" + HEX$(SETPORT
+ I), 4); RIGHT$("0" + HEX$(IN.DATA(I)), 2);
        FOR J = 7 TO 0 STEP -1
            IF (IN.DATA(I) AND BIT(J)) <> 0 THEN PRINT "  1";  ELSE
PRINT "  0";
        NEXT J
    NEXT I
    LOCATE 18, 25: PRINT "Continue ---> Push any key"
WAIT2:
    X$ = INKEY$: IF X$ = "" THEN GOTO WAIT2 ELSE IF ASC(X$) = &H1B
THEN GOTO PEND

    FOR I = 10 TO 13: LOCATE I, 17: PRINT SPACE$(60): NEXT I
    GOTO LOOP1
PEND:
    END
```

Sample Input Program in Microsoft C

This program is the Microsoft C version of the sample program provided in "Sample Input Program in Q-BASIC." Since the C version is the same as the Q-BASIC version in specification, make the same switch and jumper settings as those made for the Q-BASIC version. The two programs are also the same in display screen and flowchart.

```
/*******************************/
/*        SAMPLE PROGRAM       */
/*   MS-C DATA INPUT  VER 1.00 */
/*******************************/

#include<stdio.h>
#include<conio.h>
#include<dos.h>
#include<math.h>
#define cls()           printf("\033[2J")
#define locate(x,y)     printf("\033[%d;%dH", y+1, x+1)

main()
{
    unsigned char a;
    unsigned portadd = 0x0300;
    unsigned char A[4], Bit[8];
    char HOW_DO = '1';
    char i,j;

    cls();

    locate(20,3);  printf("**********************************\n");
    locate(20,4);  printf("*     Byte Data Input Sample     *\n");
    locate(20,5);  printf("**********************************\n");

    locate(17,9);
    printf("port    input data   D7 D6 D5 D4 D3 D2 D1 D0\n");

    while (HOW_DO != 0x1b)
    {
        locate(25,18);
        printf("Input     ----->   Push any key\n");
        locate(25,19);
        printf("End       ----->   Push ESC key\n");
        for (i=0; i<=4; i++)
            {
            locate(16,10+i);
            printf("                                                  ");
            }
        HOW_DO = getch();

        if (HOW_DO != 0x1b)
        {
```

```
            for (i=0; i<=3; i++)
              {
              a = 0x80;
              A[i] = inp (portadd + i);
              locate(17,10+i);
              printf("%xH          %xH\n", portadd +i, A[i]);
              for (j=7; j>=0; j--)
                  {
                  if ((A[i] & a) > 0) Bit[j] = 1;
                      else Bit[j] = 0;
                  locate(39+(7-j)*3,10+i);
                  printf("%x  ",Bit[j]);
                  a = (a>>1);
                  }
              }
              locate(25,18);
              printf("Continue ---->   Push any key\n");
              HOW_DO = getch();
          }
      }
      cls();
  }
```

Sample Output Program in Q-BASIC

This program is written in Q-BASIC. The program inputs hexadecimal data from the keyboard, outputs the data in individual ports, then display the values in hexadecimal.

Specification

When data to be output to the 0300H, 0301H, 0302H, and 0303H ports is entered in byte units from the keyboard, the program outputs the data to the individual ports. This boad only occupies two ports; the output data to 0302H and 0303H are made invalid.

Preparation

Perform the following preparation before you can run the program.

(1) Set the port address to 0300H using the DIP switches (SW1 and SW2) as shown below:

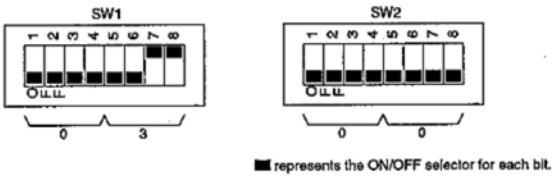


Figure 18. DIP Switch Settings

(2) Leave the JP2 set by factor default (connected between terminals 2 and 3).

Figure 19. Function Select Jumper Setting

Display Screen and Flowchart

(1) Display screen

```
**********************************
*    Byte Data Output Sample     *
**********************************

ENTER OUTPUT DATA FOR 300H PORT (00~FF) --> AA
ENTER OUTPUT DATA FOR 301H PORT (00~FF) --> 55
ENTER OUTPUT DATA FOR 302H PORT (00~FF) --> 00
ENTER OUTPUT DATA FOR 303H PORT (00~FF) --> 00

DATA HAS BEEN OUTPUTTED.

CONTINUE    ---> PRESS ANY KEY
END         ---> PRESS ESC KEY
```

(2) Flowchart

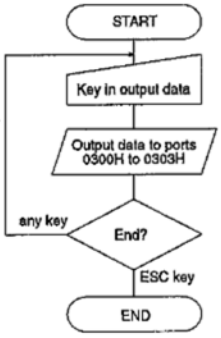


Figure 20. Flowchart for Sample Output Program

```
************************************
*         Sample Program           *
*      Byte data output   Ver 1.0  *
************************************

     CLS :    WIDTH 80, 25
     LOCATE 3, 20: PRINT "**********************************"
     LOCATE 4, 20: PRINT "*     Byte Data Output Sample    *"
     LOCATE 5, 20: PRINT "**********************************"

 '      *********************************
 '      *    Data input  by keyboard    *
 '      *********************************

     LOCATE  9, 15: PRINT "ENTER a 2-digit Hex number at 300H port --
> "
     LOCATE 10, 15: PRINT "ENTER a 2-digit Hex number at 301H port --
> "
     LOCATE 11, 15: PRINT "ENTER a 2-digit Hex number at 302H port --
> "
     LOCATE 12, 15: PRINT "ENTER a 2-digit Hex number at 303H port --
> "

WHILE (1)

     PD0$ = ""
     WHILE (LEN(PD0$) < 2)
        LOCATE 9, 60: LINE INPUT PD0$                    '300H port
DATA INPUT
     WEND
     LOCATE 9, 60: PRINT RIGHT$("0" + PD0$, 2); SPACE$(5)
     PD0 = VAL("&H" + RIGHT$(PD0$, 2))

     PD1$ = ""
     WHILE (LEN(PD1$) < 2)
        LOCATE 10, 60: LINE INPUT PD1$                   '301H port
DATA INPUT
     WEND
     LOCATE 10, 60: PRINT RIGHT$("0" + PD1$, 2); SPACE$(5)
     PD1 = VAL("&H" + RIGHT$(PD1$, 2))

     PD2$ = ""
     WHILE (LEN(PD2$) < 2)
        LOCATE 11, 60: LINE INPUT PD2$                   '302H port
DATA INPUT
     WEND
     LOCATE 11, 60: PRINT RIGHT$("0" + PD2$, 2); SPACE$(5)
     PD2 = VAL("&H" + RIGHT$(PD2$, 2))

     PD3$ = ""
     WHILE (LEN(PD3$) < 2)
        LOCATE 12, 60: LINE INPUT PD3$                   '303H port
DATA INPUT
     WEND
     LOCATE 12, 60: PRINT RIGHT$("0" + PD3$, 2); SPACE$(5)
```

```
     PD3 = VAL("&H" + RIGHT$(PD3$, 2))


     LOCATE 18, 25: PRINT "Output   ---> Push any key"
     LOCATE 19, 25: PRINT "End      ---> Push ESC key"
WAIT1:
     X$ = INKEY$: IF X$ = "" THEN GOTO WAIT1 ELSE IF ASC(X$) = &H1B
THEN GOTO PEND
                                    'WAITING PUSH KEY

 '       **********************
 '       *    Output data     *
 '       **********************

     OUT &H300, PD0:                '300H port's data output
     OUT &H301, PD1:                '301H port's data output
     OUT &H302, PD2:                '302H port's data output
     OUT &H303, PD3:                '303H port's data output

     LOCATE 18, 25: PRINT "Continue ---> Push any key"
WAIT2:
     X$ = INKEY$: IF X$ = "" THEN GOTO WAIT2 ELSE IF ASC(X$) = &H1B
THEN GOTO PEND
                                    'WAITING PUSH KEY
     FOR I = 18 TO 19: LOCATE I, 25: PRINT SPACE$(30): NEXT I
'DELETE COMENT
     FOR I = 9 TO 12: LOCATE I, 60: PRINT SPACE$(10): NEXT I
'DELETE COMENT
WEND
PEND:
    END
```

Sample Output Program in Microsoft C

This program is the Microsoft C version of the sample program provided in "Sample Output Program in Q-BASIC." Make the same switch and jumper settings as those made for the Q-BASIC version. The two programs are also the same in display screen and flowchart.

```
/*********************************/
/*        SAMPLE PROGRAM         */
/*   MS-C DATA OUTPUT  VER 1.00 */
/*********************************/

#include<stdio.h>
#include<conio.h>
#include<dos.h>

#define  off   0
#define  on    1
#define cls()           printf("\033[2J")
#define locate(x,y)     printf("\033[%d;%dH", y+1, x+1)

main()
{
    unsigned portadd = 0x0300;
    unsigned char      pd0,pd1,pd2,pd3;
    int                i;
    char               HOW_DO = '1';

    cls();

    locate(17,3);  printf("**********************************\n");
    locate(17,4);  printf("*     Byte Data Output Sample    *\n");
    locate(17,5);  printf("**********************************\n");

    locate(15, 9);  printf("ENTER a 2-dight Hex number at 300H port -
-> ");
    locate(15,10);  printf("ENTER a 2-dight Hex number at 301H port -
-> ");
    locate(15,11);  printf("ENTER a 2-dight Hex number at 302H port -
-> ");
    locate(15,12);  printf("ENTER a 2-dight Hex number at 303H port -
-> ");


    while (HOW_DO != 0x1b)
    {
        locate(25,16); printf("                                ");
        locate(25,17); printf("                                ");

        locate(60, 9); scanf("%x",&pd0);
        locate(60,10); scanf("%x",&pd1);
        locate(60,11); scanf("%x",&pd2);
        locate(60,12); scanf("%x",&pd3);
```

```
        locate(25,16); printf("Output   ---->   Push any key\n");
        locate(25,17); printf("End      ---->   Push ESC key\n");
        locate( 1,25); HOW_DO = getch();


        if (HOW_DO != 0x1b)
        {
            outp(portadd  ,pd0);
            outp(portadd+1,pd1);
            outp(portadd+2,pd2);
            outp(portadd+3,pd3);

            locate(25,16); printf("Continue ---->   Push any key\n");
            locate( 1,25); HOW_DO = getch();

            for (i = 9 ; i <= 12 ; i++)
            {
                locate(60,i);printf("      ");
            }
        }
    }
    cls();
}
```

Sample Interrupt Program in Q-BASIC

This program registers and uses a machine language program (Microsoft Mace Assembler) from Q-BASIC as an interrupt processing program.

Specification

The program causes the machine language program to count the number of interrupts and BASIC to display the count each time an interrupt signal (IRQ5) is generated.

Preparation

Perform the following preparation before you can run the program.

(1) Set the port address to 0300H using the DIP switches (SW1 and SW2) as shown below:

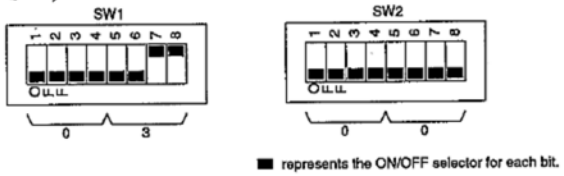


Figure 21. DIP Switch Settings

(2) Leave the JP2 set by factor default (connected between terminals 2 and 3).

Figure 22. Function Select Jumper Setting

(3) Since IRQ5 is used as the interrupt level, connect SIG1 to IRQ5 on the JP1 as shown below:

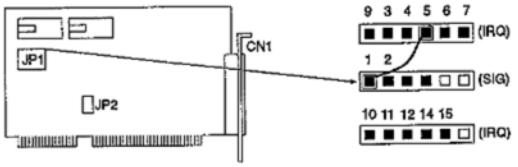


Figure 23. Interrupt Jumper Settings

Display Screen and Flowchart

(1) Display screen

```
**********************************
**      Interrupt Sample          **
**********************************

INTERRUPT COUNTER OF IRQ5 = 54

END    ---> PRESS ESC KEY
```

(2) Flowchart

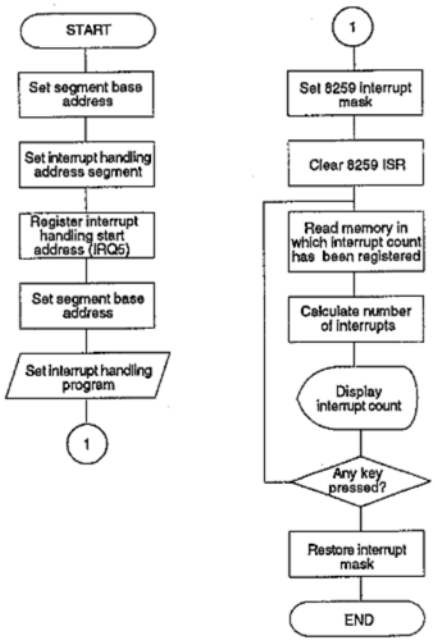


Figure 24. Flowchart for Sample Interrupt Program

```
'   ********************************
'   **   INTERRUPT SAMPLE PROGRAM  **
'   **              Ver. 1.00      **
'   ********************************

    DIM MBUFF%(&H190): CLS
    WIDTH 80, 25: KEY OFF

' *****  GET SEGMENT  *****
    LOCATE 3, 20: PRINT "**********************************"
    LOCATE 4, 20: PRINT "*     Interrupt Sample           *"
    LOCATE 5, 20: PRINT "**********************************"
    LOCATE 11, 20: PRINT "Interrupt counter of IRQ5 ="
    LOCATE 18, 20: PRINT "END ---> Push ESC key   "

    MCHSEG = VARSEG(MBUFF%(0))
    SEGH = (INT(MCHSEG AND &HFF00) / &H100) AND &HFF
    SEGL = MCHSEG AND &HFF

' *****  SET INTERRUPT TABLE ADDRESS  *****
    DEF SEG = &H0
    POKE &H34, &H0
    POKE &H35, &H0
    POKE &H36, SEGL
    POKE &H37, SEGH

' *****  STORE MACHINE LANGUAGE  *****
    DEF SEG = MCHSEG
    RESTORE ASM
    FOR I = 0 TO 20
    READ DAT: POKE I, DAT
    NEXT I

ASM:
    DATA &HFB                 : ' STI
    DATA &H1E                 : ' POP    DS
    DATA &H50                 : ' POP    AX
    DATA &H53                 : ' PUSH   BX
    DATA &H8C, &HC8           : ' MOV    AX, CS
    DATA &H8E, &HD8           : ' MOV    DS, AX
    DATA &HBB, &H02, &H03     : ' MOV    BX, 302H
    DATA &HFF, &H07           : ' INC    WORD PTR [BX]
    DATA &HB0, &H20           : ' MOV    AL, 20H
    DATA &HE6, &H20           : ' MOV    20H, AL
    DATA &H5B                 : ' POP    BX
    DATA &H58                 : ' POP    AX
    DATA &H1F                 : ' POP    DS
    DATA &HCF                 : ' IRET
```

```
' *****  IRT COUNT SET  *****
    POKE &H303, 0
    POKE &H302, 0

' *****  CLEAR MASK REGISTER  *****
    A = INP(&H21)
    A = (A AND &HDF)
    OUT &H21, A
    OUT &H20, &H65

' *****  LOOP  *****
WHILE (A$ <> CHR$(&H1B))
    A$ = INKEY$
    DEF SEG = MCHSEG
    A = PEEK(&H303) * 256 + PEEK(&H302)
    LOCATE 11, 47: PRINT A
WEND

' *****  DISABLE INTERRUPT  *****
    A = INP(&H21)
    A = A OR &H20
    OUT &H21, A

END
```

Sample Interrupt Program in Microsoft C

This program is the Microsoft C version of the sample program provided in "Sample Interrupt Program in Q-BASIC." Since the C version is the same as the Q-BASIC version in specification, make the same switch and jumper settings as those made for the Q-BASIC version. The two programs are also the same in display screen and flowchart.

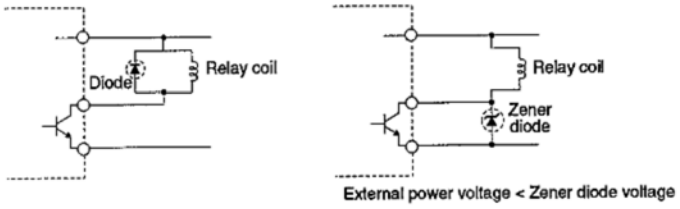
```
/************************************/
/*        SAMPLE PROGRAM            */
/*   MS-C INTERRUPT COUNT  VER 1.00 */
/************************************/

#include<stdio.h>
#include<conio.h>
#include<dos.h>

    static  unsigned  int port = 0x300;
    static  int cnt = 0;
    static  int dat = 0;

#define cls()           printf("\033[2J")
#define locate(x,y)     printf("\033[%d;%dH", y+1, x+1)

void cursor(int m)
{
    union REGS r;
    switch(m)
    {
        case 0:
            r.h.ah = 1;
            r.h.ch = 0x20;
            r.h.cl = 0x20;
            break;
        case 1:
            r.h.ah = 1;
            r.h.ch = 06;
            r.h.cl = 07;
            break;
    }
    int86(0x10,&r,&r);
}

void interrupt far int5_c(void)
{
    _disable();
    cnt++;
    outp(0x20,0x20);
    _enable();
```

```
void main(void)
{
    int masdat;

    cls();
    cursor(0);
    locate(20,3);  printf("**********************************\n");
    locate(20,4);  printf("**     Interrupt Sample          *\n");
    locate(20,5);  printf("**********************************\n");
    locate(20,18);
    printf("End ----> Push ESC key");
    locate(20,11);
    printf("Interrupt counter of IRQ5 = 0 ");

    _dos_setvect(0xD,(void(interrupt far *)())int5_c);
    masdat = inp(0x21);
    outp(0x21, masdat & 0xDF);
    outp(0x20, 0x65);

    do
    {
        locate(48,11);
        printf("%d",cnt);
    }while(kbhit() == 0 || getch() != 0x1b);

    outp(0x21,masdat);
    cursor(1);
}
```

D. Measures Against Voltages

When connecting a load which may generate a surge voltage or current, for example an inductive load (relay coil) or incandescent lamp, to digital outputs, suitable protection measures are required to prevent damage to the output stage or a malfunction due to noise. Instantaneously interrupting current flowing through a coil including a relay causes the sudden generation of a high-voltage pulse. If its voltage exceeds the withstand voltage of the transistor, the performance of the transistor may be degraded or it may be damaged. To prevent this, be sure to connect a surge absorption element when driving an inductive load including a relay coil. Examples of mea sures against surge voltages are as shown in the Figure 25 below.

■Examples of use of relay coil



■Examples of use of lamp

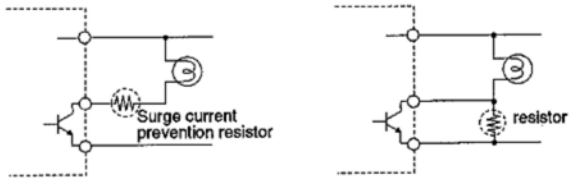


Figure 25. Samples of Surge Voltage Protection

Note!

The protection circuit will not be effective unless it is installed less than 50 cm from the load and contact.

INDEX

CONTEC Group

JAPAN : Headquarters
CONTEC Co., LTD.
3-9-31, Himesato, Nishiyodogawa-ku, Osaka 555-0025, Japan
Tel : +81 (6) 6477-5219 Fax : +81 (6) 6477-1692
E-mail : intsales@osaka.contec.co.jp

U.S.A. : CONTEC MICROELECTRONICS U.S.A. INC.
744 South Hillview Drive, Milpitas, CA 95035 U.S.A.
Tel : +1 (408) 719-8200 Fax : +1 (408) 719-6750
E-mail : tech_support@contecusa.com

EUROPE : CONTEC MICROELECTRONICS EUROPE B.V.
Binnenweg 4, 2132 CT, Hoofddorp, The Netherlands
Tel : +31 (23) 567-3030 Fax : +31 (23) 567-3035
E-mail : tech_support@conteceu.nl

KOREA : HYOJIN CONTEC Co., LTD.
Ki-im Bldg. #399, Shindolim-Dong, Kuro-ku, Seoul, Korea
Tel : +82 (2) 2636-4277/8 Fax : +82 (2) 2636-4279
E-mail : product@conteck.com

CHINA : INTERNATIONAL CONTEC TECHNOLOGY CO., LTD.
B-8F, Hua Tong Building, No. B19, Che Gong Zhuang West Road,
Hai Dian District, Beijing 100044, China
Tel : +86(10)8801-8228 Fax : +86 (10)8801-8209
E-mail : ict@ict.com.cn

SHANGHAI CONTEC MICROELECTRONICS CORP.
No. 481 Gui Ping Road, Cao He Jing Hi-Tech Park Shanghai, 200233, China
Tel : +86 (21) 6485-1907 Fax : +86 (21) 6485-0330
E-mail : contec@contec.com.cn

SHENYANG CONTEC MICROELECTRONICS Co., LTD.
No. 169, Qingnian Street, Shenhe District, Shenyang 110015, China
Tel : +86 (24) 2392-9771 Fax : +86 (24) 2392-9773

TAIWAN : MACROMATE CORP.
8F, Universal Center, No.179, Ta-Tung Rd., Sec.1 Hsi-Chih, Taipei Hsien, Taiwan,
R.O.C
Tel : +886 (2) 2647-9353 Fax : +886 (2) 2647-9373
E-mail : intl@macromate.com.tw

A-46-368 Ver. 2001. 02. 06
200307