

# Re: Printer port confusion

---

*Source:* <http://sci.tech-archive.net/Archive/sci.electronics.basics/2007-01/msg00139.html>

---

- *From:* Sjouke Burry <[burrynulnulfour@xxxxxxxxxxxxxxxxxxxxx](mailto:burrynulnulfour@xxxxxxxxxxxxxxxxxxxxx)>
  - *Date:* Thu, 04 Jan 2007 02:51:22 +0100
- 

Randy Day wrote:

Is there a reason why different websites list different signal states on some parallel port pins? Are some just wrong, or were there two types of parallel ports for PC's at one time?

For example:

<http://et.nmsu.edu/~etti/fall96/computer/printer/printer.html> indicates that pins 10 and 15 are inverted inputs, while

[http://ourworld.compuserve.com/homepages/Bill\\_Bowden/page6.htm#p\\_input](http://ourworld.compuserve.com/homepages/Bill_Bowden/page6.htm#p_input) has 10 and 15 as \_normal\_ inputs while \*11\* is inverted!

I just found out my PC has the latter pinout, while I had wired a circuit assuming that the first website (and another that agreed with it) were correct.

<grumble>

I just want to know if I'm going to have problems with pinouts if I want to hook a microcontroller to different PC's and their parallel ports.

well... i think the info below would be of use....  
PC Parallel Port Mini-FAQ

By Kris Heidenstrom (kheidens@xxxxxxxxxxxxxxxx), revision 4, 950403

In no event shall the author be liable for any damages whatsoever for any loss relating to this document. Use it at your own risk!

## 1. INTRO

This is a five printed page mini-FAQ with the information essential for programming the PC parallel port. Many subjects are not covered in detail. View on an 80-column monospaced screen with 8-column tab stops. Comments and suggestions to kheidens@xxxxxxxxxxxxxxxx

## Re: Printer port confusion

A parallel port links software to the real world. To software, the parallel port is three 8-bit registers occupying three consecutive addresses in the I/O space. To hardware, the port is a female 25-pin D-sub connector, carrying twelve latched outputs from the computer, accepting five inputs into the computer, with eight ground lines.

The normal function of the port is to transfer data to a parallel printer through the eight data pins, using the remaining signals as flow control and miscellaneous controls and indications.

The original port was implemented with TTL/LS logic. Modern ports are implemented in an ASIC or a combined serial/parallel port chip, but are backward compatible. Some modern ports are bidirectional.

### 2. BIOS LPT PORT TABLE

A parallel port is identified by its I/O base address, and also by its LPT port number. The BIOS power-on self-test checks specific I/O addresses for the presence of a parallel port, and builds a table of I/O addresses in the low memory BIOS data area, starting at address 0040:0008 (or 0000:0408).

This table contains up to three 16-bit words. Each entry is the I/O base address of a parallel port. The first word is the I/O base address of LPT1, the second is LPT2, etc. If less than three ports were found, the remaining entries in the table are zero. DOS, and the BIOS printer functions (accessed via int 17h), use this table to translate an LPT port number to a physical port at a certain address.

The addresses are checked in a specific order, and addresses are put into the table as they are found, so the table will never have gaps. A particular I/O address does not necessarily always equate to the same specific LPT port number, although there are conventions.

#### 2.1 ADDRESSING CONVENTIONS

The video card's parallel port is normally at 3BCh. This address is checked first by the BIOS, so if a port exists there, it will be LPT1. The BIOS then checks at 378h, then at 278h. AFAIK there is no standard address for a fourth port, and BIOSes only look for three.

### 3. DIRECT HARDWARE ACCESS

The port consists of three 8-bit registers at adjacent addresses in the processor's I/O space. The registers are defined relative to the I/O base address, and are at IOBase+0, IOBase+1, and IOBase+2 (for example if IOBase is 3BCh, then the registers are at 3BCh, 3BDh, and 3BEh). Always use 8-bit I/O accesses on these registers.

#### 3.1 DATA REGISTER

## Re: Printer port confusion

The data register is at IOBase+0. It may be read and written (using the IN and OUT instructions, or inportb() and outportb() or inp() and outp()). Writing a byte to this register causes the byte value to appear on pins 2 through 9 of the D-sub connector (unless the port is bidirectional and is set to input mode). The value will remain latched and stable until you write another value to the data register. Reading this register yields the state of those pins.

7 6 5 4 3 2 1 0

\* . . . . . D7 (pin 9), 1=High, 0=Low  
. \* . . . . . D6 (pin 8), 1=High, 0=Low  
.. \* . . . . . D5 (pin 7), 1=High, 0=Low  
... \* . . . . . D4 (pin 6), 1=High, 0=Low  
.... \* . . . . . D3 (pin 5), 1=High, 0=Low  
..... \* . . . . . D2 (pin 4), 1=High, 0=Low  
..... \* . . . . . D1 (pin 3), 1=High, 0=Low  
..... \* . . . . . D0 (pin 2), 1=High, 0=Low

### 3.2 STATUS REGISTER

The status register is at IOBase+1. It is read-only (writes will be ignored). Reading the port yields the state of the five status input pins on the parallel port connector at the time of the read access:

7 6 5 4 3 2 1 0

\* . . . . . Busy . . (pin 11), high=0, low=1 (inverted)  
. \* . . . . . Ack . . (pin 10), high=1, low=0 (true)  
.. \* . . . . . No paper (pin 12), high=1, low=0 (true)  
... \* . . . . . Selected (pin 13), high=1, low=0 (true)  
.... \* . . . . . Error . . (pin 15), high=1, low=0 (true)  
..... \* \* \* \* \* Undefined

### 3.3 CONTROL REGISTER

The control register is at IOBase+2. It is read/write:

7 6 5 4 3 2 1 0

\* \* . . . . . Unused (undefined on read, ignored on write)  
.. \* . . . . . Bidirectional control, see below  
... \* . . . . . Interrupt control, 1=enable, 0=disable  
.... \* . . . . . Select . . (pin 17), 1=low, 0=high (inverted)  
..... \* . . . . . Initialize (pin 16), 1=high, 0=low (true)  
..... \* . . . . . Auto Feed (pin 14), 1=low, 0=high (inverted)  
..... \* . . . . . Strobe . . (pin 1), 1=low, 0=high (inverted)

#### 3.3.1 BIDIRECTIONAL CONTROL BIT

The bidirectional control bit is only supported on true bidirectional ports – on other ports, it behaves like bits 7 and 6. On a proper bidirectional port, setting this bit to '1' causes the outputs of the buffer that drives pins 2 through 9 of the 25-pin connector to go into

## Re: Printer port confusion

a high-impedance state, so that data can be `_input_` on those pins.

In this state, values written to the data register will be stored in the latch chip, but not asserted on the connector, and reading the data register will yield the states of the pins, which may be driven by an external device without stressing or damaging the port driver.

Also note that on some machines, another port must be set correctly to enable the bidirectional features, in addition to this bit.

I suspect this applies to machines with the parallel port integrated on the motherboard as part of the motherboard chipset, but I do not have details, sorry.

On some parallel port cards, bidirectional mode must be enabled by a jumper setting. With this setting enabled, the bidirectional control bit will be able to enable the bidirectional input mode.

### 3.3.2 INTERRUPT ENABLE BIT

The parallel port interrupt was intended to be used for interrupt driven transmission of data to a parallel printer, but DOS and BIOS do not use it. Versions of OS/2 prior to Warp (3.0) did require the interrupt for printing, but from Warp onwards the interrupt is not required (though it can be used if the `/IRQ` switch is provided on the line in `CONFIG.SYS`, i.e. `BASEDEV=PRINT0x.SYS /IRQ`).

For experimenters, the interrupt facility is useful as a general purpose externally triggerable interrupt input.

The interrupt control bit controls a tristate buffer that drives the IRQ line. Setting the bit to '1' enables the buffer, and an IRQ will be triggered on each falling edge (high to low transition) of the Ack signal on pin 10 of the 25-pin connector. Disabling the interrupt allows other devices to use the IRQ line.

The actual IRQ number is either hardwired (by convention, the port at 3BCh uses IRQ7), or is jumper-selectable (IRQ5 is a common choice). Sound cards, in particular, tend to use IRQ7 for their own purposes.

To use the IRQ you must also enable the interrupt via the interrupt mask register in the interrupt controller, at I/O address 21h, and your interrupt handler must send an EOI on exit. DOS technical programming references have notes on writing interrupt handlers.

### 3.3.3 PRINTER CONTROL BITS

The bottom four bits are latched and presented on the parallel port connector, much like the data register. Three of them are inverted, so writing a '1' will output a low voltage on the port pin for them.

These four outputs are open collector outputs with pullup resistors,

## Re: Printer port confusion

so an external device can force them low without stressing the driver in the PC, and they can even be used as inputs.

To use them as inputs, write 0100 binary to the bottom four bits of the control register. This sets the outputs all high, so they are pulled high by the pullup resistors (typically 4700 ohms). An external device can then pull them low, and you can read the pin states by reading the control register. Remember to allow for the inversion on three of the pins.

If you are using this technique, the control register is not strictly 'read/write', because you may not read what you write (or wrote).

### 4 SAMPLE PROGRAM – DETERMINE WHETHER A PORT IS BIDIRECTONAL

This program reports for LPT1, LPT2, and LPT3 whether the port exists and whether it is bidirectional, i.e. setting the bidirectional control bit causes the port to go into high-impedance (tri-state) mode. This program is written for Borland C. Change `outportb()` and `inportb()` to `outp()` and `inp()` for Microsoft C, I think. I did not have a machine with a bidirectional port with which to test this program fully, so please drop me a line if you can confirm that it does or doesn't work.

Save this code to `BIDIR.C` and compile with:

```
bcc -I<include_path> -L<library_path> bidir.c
```

```
----- snip snip snip -----
```

```
#include <dos.h>
#include <process.h>
#include <stdio.h>
```

```
/* The following function returns the I/O base address of the nominated
parallel port. The input value must be 1 to 3. If the return value
is zero, the specified port does not exist. */
```

```
unsigned int get_lptport_iobase(unsigned int lptport_num) {
return *((unsigned int far *)MK_FP(0x40, 6) + lptport_num);
}
```

```
/* Checks whether the port's data register retains data, returns 1 if
so, 0 if not. The data register retains data on non-bidirectional
ports, but on bidirectional ports in high impedance (tri-state)
mode, the data register will not retain data. */
```

```
unsigned int test_retention(unsigned int iobase) {
outportb(iobase, 0x55); /* Write a new value */
(void) inportb(iobase); /* Delay */
if (inportb(iobase) != 0x55) {
return 0; /* Did not retain data */
```

## Re: Printer port confusion

```
}
outportb(iobase, 0xAA); /* Write another new value */
(void) inportb(iobase); /* Delay */
if (inportb(iobase) != 0xAA) {
return 0; /* Did not retain data */
}
return 1; /* Retained data alright */
}
```

```
void report_port_type(unsigned int portnum) {
unsigned int iobase, oldctrl, oldval;
iobase = get_lptport_iobase(portnum);
if (iobase == 0) {
printf("LPT%d does not exist\n", portnum);
return;
}
oldctrl = inportb(iobase+2);
outportb(iobase+2, oldctrl & 0xDF); /* Bidir off */
(void) inportb(iobase); /* Delay */
oldval = inportb(iobase); /* Keep old data */
if (test_retention(iobase) == 0) {
printf("LPT%d appears to be faulty!\n", portnum);
outportb(iobase+2, oldctrl);
outportb(iobase, oldval);
return;
}
outportb(iobase+2, oldctrl | 0x20); /* Bidir on */
printf("LPT%d %s bidirectional\n", portnum,
(test_retention(iobase)) ? "is not" : "is");
outportb(iobase+2, oldctrl); /* Put it back */
outportb(iobase, oldval); /* Restore data */
return;
}
```

```
void main(void) {
unsigned int portnum;
for (portnum = 1; portnum < 4; ++portnum)
report_port_type(portnum);
exit(0);
}
```

----- snip snip snip -----

### 5. FILE TRANSFER PROGRAM CABLES

The parallel-to-parallel cable is used by DRDOS's INTERLNK program. Apparently Laplink and FastLynx cables are the same. The pin-to-pin connection between two male 25-pin D-sub connectors is: 2-15, 3-13, 4-12, 5-10, 6-11, and the reverse: 15-2, 13-3, 12-4, 10-5, and 11-6, and ground: 25-25. This requires eleven wires. If you have spare wires, link some extra grounds together. Pins 18 to 25 inclusive are

## Re: Printer port confusion

grounds. A very long cable may be unreliable, limit it to 10 metres.

### 6. TRANSFERRING DATA BETWEEN TWO COMPUTERS USING PARALLEL PORTS

The normal method uses the file transfer program cable described above, which sends data port bits 0–4 of one machine to status port bits 3–7 (respectively) on the other, and vice versa. Data is sent four bits at a time, using the fifth lines in each direction as data strobe and acknowledge respectively. Some protocol is required to control the data direction.

Here are some sample functions that will send and receive a byte of data using the above cable. One program must be the sender and one must be the receiver. The `input_value()` function will be used on both sender and receiver. The `receive_byte()` function will be used only on the receiver. The `transmit_byte()` function will be used only on the sender, and will wait without returning, until the byte has been received and acknowledged by the receiver.

----- snip snip snip -----

```
static unsigned int lpt_base; /* Set to base I/O address */

/* Return input value as 5-bit number. If input has changed since this
function was last called, verify that the input is stable. */

unsigned int input_value(void) {
    static unsigned char last_value = 0xFF;
    auto unsigned char new1, new2;
    new1 = inportb(lpt_base + 1) & 0xF8;
    if (new1 != last_value) {
        for (;;) {
            new2 = inportb(lpt_base + 1) & 0xF8;
            if (new2 == new1) /* Wait for stable value */
                break;
            new1 = new2;
        }
        last_value = new1;
    }
    return (last_value ^ 0x80) >> 3;
}

/* Receive an 8-bit byte value, returns -1 if no data available yet */

signed int receive_byte(void) {
    unsigned int portvalue, bytevalue;
    portvalue = input_value(); /* Read input */
    if ((portvalue & 0x10) == 0)
        return -1; /* Await high flag */
    outportb(lpt_base, 0x10); /* Assert reverse flag */
    bytevalue = portvalue & 0x0F; /* Keep low nibble */
}
```

Re: Printer port confusion

```
do {
portvalue = input_value();
} while ((portvalue & 0x10) != 0); /* Await low flag */
outportb(lpt_base, 0); /* Deassert reverse flag */
bytevalue |= (portvalue << 4); /* High nibble */
return bytevalue & 0xFF;
}

/* Transmit an 8-bit byte value, won't return until value is sent */

void transmit_byte(unsigned int val) {
val &= 0xFF;
outportb(lpt_base, (val & 0x0F) | 0x10); /* Set nibble flag */
while ((input_value() & 0x10) == 0)
; /* Await returned flag high */
outportb(lpt_base, val >> 4); /* Clear nibble flag */
while ((input_value() & 0x10) != 0)
; /* Await returned flag low */
return;
}
```

----- snip snip snip -----

End of the PC Parallel Port Mini-FAQ.

Kris

--

Kris Heidenstrom kheidens@xxxxxxxxxxxxx Wellington, New Zealand  
"Minister can I just begin by asking why, with manufacturing output  
down and Sterling showing a steep decline on world markets, are you  
wearing a giant rabbit costume?" -- Interview, Alexei Sayle's Stuff

.