ECE291

Lecture 8

Peripheral devices

Lecture outline

- Hardware I/O with ports
- 8253 timer
- PC speaker
- Serial port
- Parallel port

Port input and output

- Used to communicate with many peripheral devices built into the processor, motherboard or expansion cards
- **IN** instruction transfers data from a device on the I/O bus to AL, AX, or EAX
- **OUT** instruction transfers data from AL, AX, or EAX to a device connected to the I/O bus

Port input and output

Two forms of port addressing

- Fixed-port addressing
  - Allows an 8-bit I/O port address (use if port is 0 – 255)
  - port number is immediate (follows the instruction opcode)

```
IN    AL, 6Ah          ;data from I/O address 6Ah is input to AL
```

- Variable-port addressing
  - Allows using 16-bit port address (addresses up to FFFFh)
  - the I/O port number is stored in register DX

```
MOV DX, 0FA64h
OUT  DX, AX            ;transfers contents of AX to I/O port FA64h
```

Port I/O example

```
..START
    IN    al, 61h     ;read port 61h: PC speaker control register
    OR    al, 3       ;set bits 0 and 1
    OUT   61h, al     ;turn speaker on
    MOV   cx, 1000h   ;delay count; if the count is increased,
                      ;the beep will become longer
.L1 LOOP  .L1         ;time delay - spin 1000h times
    IN    al, 61h
    AND   al, 0fch    ;clear bits 0 and 1
    OUT   61h, al     ;turn speaker off

    MOV   ax, 4c00h   ;Normal DOS Exit
    INT   21h
```

Keyboard control

- The keyboard is connected to the 8255 Programmable Peripheral Interface (PPI) chip, a simple self-contained or embedded microprocessor
  - Contains three registers A, B, and C which are accessed on ports 60h-62h using the in/out instructions
- When a key is pressed or released two events take place:
  - an 8-bit scan code is deposited in register A of the PPI
  - an INT 09h hardware interrupt is generated
    - Default ISR gets the scan code in A, acknowledges the hardware interrupt, and passes information to the keyboard buffer (a memory buffer in BIOS data area)

Keyboard control

- Each key action is stored in the buffer as a word
  - normal characters are stored with ASCII code for the character, and the scan code for the key
  - extended codes corresponding to special keys e.g., cursor keys, shift
- The keyboard buffer is a circular buffer, when it gets full and a new key is pressed:
  - INT 09 ISR generates a “bell” character, which causes the speaker to beep
  - The data from the new key press is discarded
- Keyboard buffer holds keystrokes until read by an int 16

8253 Programmable Timer

- Intel 8253 contains 3 independent 16-bit programmable counters
- Operate on a clock frequency of 14.31818 MHz
- This frequency is divided by 12 before being sent to the timer so the pulses arrive at frequency 1.19318 MHz
- Each timer can be loaded with a sixteen bit starting count (0 – 65535)
- Once enabled, timers repeatedly count down to 0 from the starting count and generate an output signal
- The rate of the output signal is (1193180 / starting count) outputs per second

8253 Programmable Timer

- **Timer 0**
  - Generates and INT 08h when the timer expires. Can be used to trigger regularly recurring events in your programs.
  - Counts down from as high as 65535 to zero
  - The output signal occurs each $(1.19318 * 10^6)/ 65535 = 18.2$ Hz
- **Timer 1**
  - Causes the DRAM memory system to be refreshed
  - If you interfere with this function, the entire contents of the system memory could be lost
- **Timer 2**
  - General purpose, and optionally connected to the PC speaker for generating simple tones

Timer control registers

- 8253 has four byte-sized control registers which are mapped to I/O ports 40h-43h

Port	Description
40h	Timer 0 Count Register
41h	Timer 1 Count Register
42h	Timer 2 Count Register
43h	Timer Control Register

Timer

- The Control Register allows the programmer to select the counter, mode of operation, and type of operation

Bits	Description
7,6	Counter ID (00,01,10)
5,4	00: latch count for reading 01: Read/Set least significant byte 10: Read/Set most significant byte 11: Read/Set least significant byte then most significant byte
3,2,1	Count Mode (011=Square Wave)
0	Count Method: 0=Binary, 1=BCD

Speaker control

- Timer 2 output from 8253 can be enabled as an analog input to the PC speaker
- Enable timer input to speaker (port 61h, bit 1)
- Enable speaker (port 61h, bit 0)

- Write to Timer Control Port

```
mov     al,10110110b
out     43h, al
```

```
    10:  Control timer 2
    11:  Load Low Byte then High Byte
   011:  Select Square wave output
     0:  Count in binary
```

Speaker Control

- Enable Speaker

```
IN   al, 61h
OR   al, 00000011b
OUT  61h, al
```

Bit 1: Enable Timer

Bit 0: Enable Speaker

Don't change other bits

- Write count to timer

```
OUT  42h, al
MOV  al, ah
OUT  42h, al
```

AX=count (ticks) = 1193180 (ticks/sec) / Freq (1/sec)

Example

```
CR EQU 13
LF EQU 10
extrn dosxit, kbdin, dspmsg, binasc

; Define stack segment
stkseg  SEGMENT stack
   RESB   512
stktop

; Define code segment
cseg    SEGMENT CODE
; Variable declarations
pbuf    RESB  7
crlf    db    CR,LF,'$'
freq_msg  db  ' Freq = $'
count_msg db  ' Hz,  Count = $'
```

Example

```
..start
main
        mov     ax, cs          ; Initialize DS register
        mov     ds, ax

        mov     al,10110110b    ; Timer2, Load L+H, square wave,
                                ; binary
        out     43h,al          ; Write timer control byte
        in      al,61h
        or      al,00000011b    ; Enable Timer2 & Turn on speaker
        out     61h,al
.mloop
        call    kbdin
        cmp     al,'0'          ; 0 to quit
        je      .mdone
        sub     al,'0'
        mov     ah,0
```

Example

```
        mov     cl,8            ; Freq * 256  (Hz)
        shl     ax,cl

        mov     dx, freq_msg
        call    dspmsg
        mov     bx, pbuf
        call    binasc
        mov     dx, pbuf
        call    dspmsg          ; Print Freq
        mov     bx,ax
        mov     ax,34DCh        ;DX:AX=1,193,180 (tics/sec)
        mov     dx,12h          ;AX = ----------------------
        div     bx              ;BX=Freq (1/sec)
        mov     dx, count_msg
        call    dspmsg
        mov     bx, pbuf
        call    binasc
```

Example

```
   mov      dx, pbuf
   call     dspmsg  ; Print count
   mov      dx, crlf
   call     dspmsg

   out      42h,al   ; Write Low byte to Timer Channel 2
   mov      al,ah
   out      42h,al   ; Write High byte to Timer Channel 2
   jmp      .mloop
.mdone
   in       al,61h
   and      al,11111100b      ; Turn off speaker
   out      61h,al
   call     dosxit            ; Exit to DOSmain
```

Serial communication

- One of the oldest, simplest, and most widely used forms of communication in the world.
  - Smoke signals
  - Morse code
  - Early computer mice
  - Classical mainframe/terminal computing
  - USB printers, keyboards, scanners, cameras
  - Ethernet
  - The Internet
  - Digital cell phones
  - Digital satellites

Serial communication

- Can be implemented with as few as two (unidirectional) or three (bidirectional) wires
  - Common ground
  - Transmit line (Tx)
  - Receive line (Rx)
- One bit transferred at a time
  - Takes numerous transfers to move a single byte, but that's okay
  - Data must be broken down into chunks to transfer
  - Maybe we want some error checking too

Serial communication

- Serial or communications ports are nothing more than shift registers connected to specific I/O ports
  - For transmitting, write data to the port corresponding to the shift register
    - Shift register moves data out one bit at a time over a serial cable
  - For receiving, the shift register accepts bits as they come in, shifting them into the register.
    - Read the data from the shift register using the appropriate port address

Serial communication

- Start bit indicates when a communication frame begins
- Next 5-8 bits are the actual data
  - Number of data bits is configurable
  - 8-bits is very popular because of its obvious correspondence with the size of a byte

Serial communication

- The parity bit indicates if the number of ones was even (0) or odd (1). Helpful with error checking.
- Stop bits (1, 1.5 or 2) indicate the end of a data frame

Serial communication

- Bit or baud rate determines how quickly the bits are shifted out of the serial port
  - Can be as low as 110 bits per second
  - Can be as high as 115000 bits per second with standard serial ports (USB is much faster)
- Actual rate of data transfer depends port settings
  - 8 data bits, no parity, one stop bit means that 8 of every 10 bits that transfer contain useful data
  - 5 data bits, parity, 2 stop bits means that only 5 of every 9 bits contain useful data

Serial communication

- For serial ports to work, both ends must agree on all the previous settings
- Once that's done, the hardware takes care of the messy details
- From the programming side of things, you have to configure the port, then know how to write data properly

Serial communication

- Programming the serial port
  - First you should configure the port
    - Stop bits, data bits, parity, bit rate
  - To read, check status register until a byte is ready to be read, then read it
  - To write, check status register until serial port is ready, then write the byte
- This particular paradigm is called polling
- Serial ports also generate hardware interrupts when they receive data, but we won't be going that far in MP2

Serial communication - regs

- For configuring the port
  - Line control (LCR) – Specifies the data frame (word length, number of stop bits, and type of parity) for transmitting and receiving
  - Divisor latch low (DLL) – Stores lower 8 bits of the baud-rate generator divisor
  - Divisor latch high (DLH) – Stores the upper 8 bits of the baud-rate generator divisor
- For transferring data
  - Line status (LSR) – Contains the transmitter empty, transmit buffer empty, receive buffer full, and receive error flags
  - Receive buffer (RBR) – Holds the byte received
  - Transmit buffer (TBR) – Holds the byte to transmit

Serial communication - regs

- Each serial port has a unique Port I/O address

Device	Serial port base address
COM0:	03F8h
COM1:	02F8h
COM2:	03E8h
COM3:	02E8h

The serial port base addresses are stored at 0000:0400 – 0407. You can view these addresses directly using *debug* or *td.*

- At the command prompt, type “debug” to get the “-” debug prompt
- type **d0:0400** (and hit return)
- you get : **0000:0400 F8 03 F8 02 E8 03 E8 02**
- the first four words are addresses of the installed comm ports

Register addresses

Table 11-5. SIO Registers

Register	Expanded Address	PC/AT Address	Function
PINCFG (read/write)	F826H	—	Pin Configuration: Connects the SIO1 transmit data (TXD1), data terminal ready (DTR1#), and request to send (RTS1#) signals to package pins.
P1CFG (read/write)	F820H	—	Port 1 Configuration: Connects the SIO0 ring indicator (RI0#), data set ready (DSR0#), data terminal ready (DTR0#), request to send (RTS0#), and data carrier detected (DCD0#) signals to package pins.
P2CFG (read/write)	F822H	—	Port 2 Configuration: Connects the SIO0 clear to send (CTS0#), transmit data (TXD0), and receive data (RXD0) signals to package pins.
P3CFG (read/write)	F824H	—	Port 3 Configuration: Connects COMCLK to the package pin.
SIOCFG (read/write)	F836H	—	SIO and SSIO Configuration: Connects the SIO*n* modem input signals internally or to package pins and connects either the internal SERCLK signal or the COMCLK pin to the SIO*n* baud-rate generator input.
DLL0 DLL1 (read/write)	F4F8H F8F8H	03F8H 02F8H	Divisor Latch Low: Stores the lower 8 bits of the SIO*n* baud-rate generator divisor.
DLH0 DLH1 (read/write)	F4F9H F8F9H	03F9H 02F9H	Divisor Latch High: Stores the upper 8 bits of the SIO*n* baud-rate generator divisor.
TBR0 TBR1 (write only)	F4F8H F8F8H	03F8H 02F8H	Transmit Buffer: Holds the data byte to transmit.
RBR0 RBR1 (read only)	F4F8H F8F8H	03F8H 02F8H	Receiver Buffer: Holds the data byte received.
LCR0 LCR1 (write only)	F4FBH F8FBH	03FBH 02FBH	Line Control: Specifies the data frame (word length, number of stop bits, and type of parity) for transmissions and receptions. Allows the transmitter to transmit a break condition.
LSR0 LSR1 (read only)	F4FDH F8FDH	03FDH 02FDH	Line Status: Contains the transmitter empty, transmit buffer empty, receive buffer full, and receive error flags.

- DLL = Base + 0
  - Depends on a setting in LCR
- TBR = Base + 0
  - Write only
- RBR = Base + 0
  - Read only
- DLH = Base + 1
- LCR = Base + 3
- LSR = Base + 5

Line control register

Serial Line Control
LCR0, LCR1
(write only)

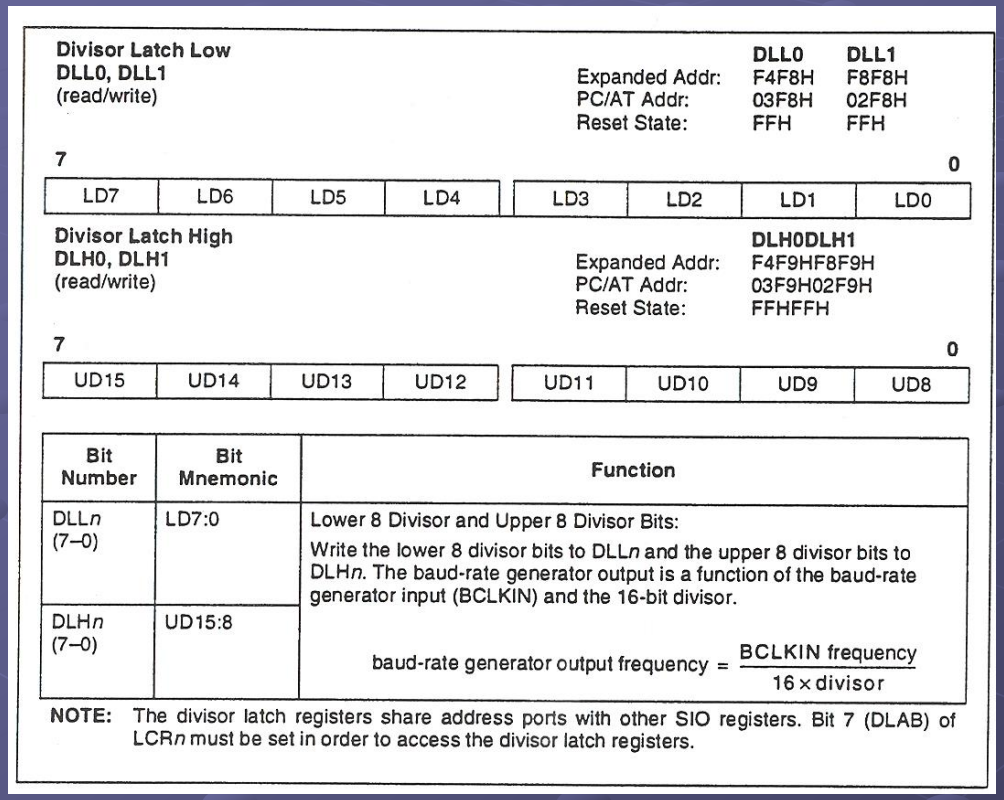
	LCR0	LCR1
Expanded Addr:	F4FBH	F8FBH
PC/AT Addr:	03FBH	02FBH
Reset State:	00H	00H

7							0
DLAB	SB	SP	EPS	PEN	STB	WLS1	WLS0

Bit Number	Bit Mnemonic	Function
7	DLAB	Divisor Latch Access Bit: This bit determines which of the multiplexed registers is accessed. Setting this bit allows access to the divisor latch registers (DLL*n* and DLH*n*). Clearing this bit allows access to the receiver and transmit buffer registers (RBR*n* and TBR*n*) and the interrupt control register (IER*n*).
6	SB	Set Break: Setting SB forces the TXD*n* pin to the spacing (logic 0) state for an entire transmission time (time of start bit + data bits + parity bit + stop bit).
5	SP	Sticky Parity, Even Parity Select, and Parity Enable: These bits determine whether the control logic produces (during transmission) or checks for (during reception) even, odd, no, or forced parity. **SP EPS PEN Function** X X 0 parity disabled (no parity option) 0 0 1 produce or check for odd parity 0 1 1 produce or check for even parity 1 0 1 produce or check for forced parity (parity bit = 0) 1 1 1 produce or check for forced parity (parity bit = 1)
4	EPS	
3	PEN	
2	STB	Stop Bits: This bit specifies the number of stop bits transmitted and received in each serial character. 0 = 1 stop bit 1 = 2 stop bits (1.5 stop bits for 5-bit characters)
1–0	WLS1:0	Word Length Select: These bits specify the number of data bits in each transmitted or received serial character. 00 = 5-bit character 01 = 6-bit character 10 = 7-bit character 11 = 8-bit character

- Must set DLAB to access DLL and DLH

Baud rate divisor



Divisor Latch Low
DLL0, DLL1
(read/write)

	DLL0	DLL1
Expanded Addr:	F4F8H	F8H
PC/AT Addr:	03F8H	02F8H
Reset State:	FFH	FFH

7							0
LD7	LD6	LD5	LD4	LD3	LD2	LD1	LD0

Divisor Latch High
DLH0, DLH1
(read/write)

	DLH0DLH1
Expanded Addr:	F4F9HF8F9H
PC/AT Addr:	03F9H02F9H
Reset State:	FFHFFH

7							0
UD15	UD14	UD13	UD12	UD11	UD10	UD9	UD8

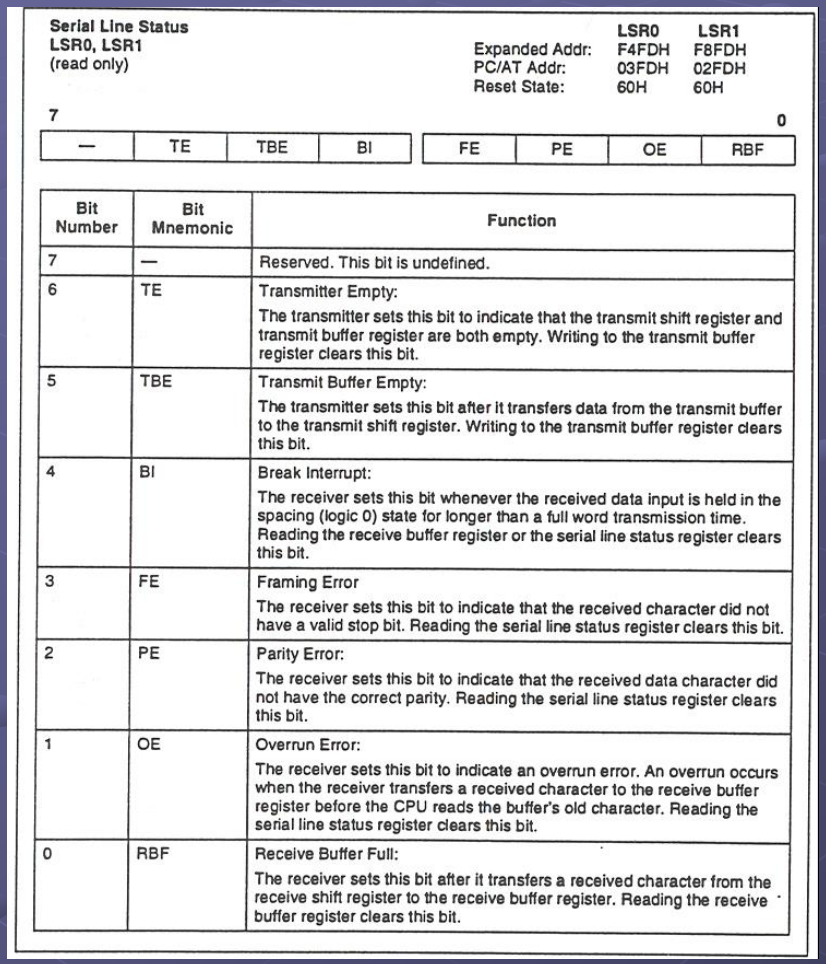
Bit Number	Bit Mnemonic	Function
DLL*n* (7–0)	LD7:0	Lower 8 Divisor and Upper 8 Divisor Bits: Write the lower 8 divisor bits to DLL*n* and the upper 8 divisor bits to DLH*n*. The baud-rate generator output is a function of the baud-rate generator input (BCLKIN) and the 16-bit divisor.
DLH*n* (7–0)	UD15:8	baud-rate generator output frequency = $\frac{\text{BCLKIN frequency}}{16 \times \text{divisor}}$

NOTE: The divisor latch registers share address ports with other SIO registers. Bit 7 (DLAB) of LCR*n* must be set in order to access the divisor latch registers.

- Setting the baud rate
  - Based off 1.8432 MHz clock frequency
  - Baud rate = frequency / (16 x divisor)
  - You set the divisor in a special control register within the serial port

Divisor	Baud Rate
1	115200
2	57600
3	38400
4	28800
6	19200
8	14400
12	9600
24	4800
384	300

Line status register



Serial Line Status
LSR0, LSR1
(read only)

	LSR0	LSR1
Expanded Addr:	F4FDH	F8FDH
PC/AT Addr:	03FDH	02FDH
Reset State:	60H	60H

7							0
—	TE	TBE	BI	FE	PE	OE	RBF

Bit Number	Bit Mnemonic	Function
7	—	Reserved. This bit is undefined.
6	TE	Transmitter Empty: The transmitter sets this bit to indicate that the transmit shift register and transmit buffer register are both empty. Writing to the transmit buffer register clears this bit.
5	TBE	Transmit Buffer Empty: The transmitter sets this bit after it transfers data from the transmit buffer to the transmit shift register. Writing to the transmit buffer register clears this bit.
4	BI	Break Interrupt: The receiver sets this bit whenever the received data input is held in the spacing (logic 0) state for longer than a full word transmission time. Reading the receive buffer register or the serial line status register clears this bit.
3	FE	Framing Error The receiver sets this bit to indicate that the received character did not have a valid stop bit. Reading the serial line status register clears this bit.
2	PE	Parity Error: The receiver sets this bit to indicate that the received data character did not have the correct parity. Reading the serial line status register clears this bit.
1	OE	Overrun Error: The receiver sets this bit to indicate an overrun error. An overrun occurs when the receiver transfers a received character to the receive buffer register before the CPU reads the buffer's old character. Reading the serial line status register clears this bit.
0	RBF	Receive Buffer Full: The receiver sets this bit after it transfers a received character from the receive shift register to the receive buffer register. Reading the receive buffer register clears this bit.

- Check RBF to see if a byte has been received
- Check TE or TBE to see of a byte has been transmitted

Parallel interface

- Originally designed for use with printers only
- Provides 8-bit data transfer (originally output only, now bi-directional)
- Provides TTL (5V) external I/O
- When the computer boots it detects any parallel ports and assigns ‘logical’ port names to them (LPT1, LPT2, LPT3)
- Each parallel port uses three processor I/O ports
  - data output (to send a byte of data to a printer you send the data to the output data register)
  - status signals from the printer
  - control/commands to the printer

Parallel interface

- Each logical device has a unique Port I/O address

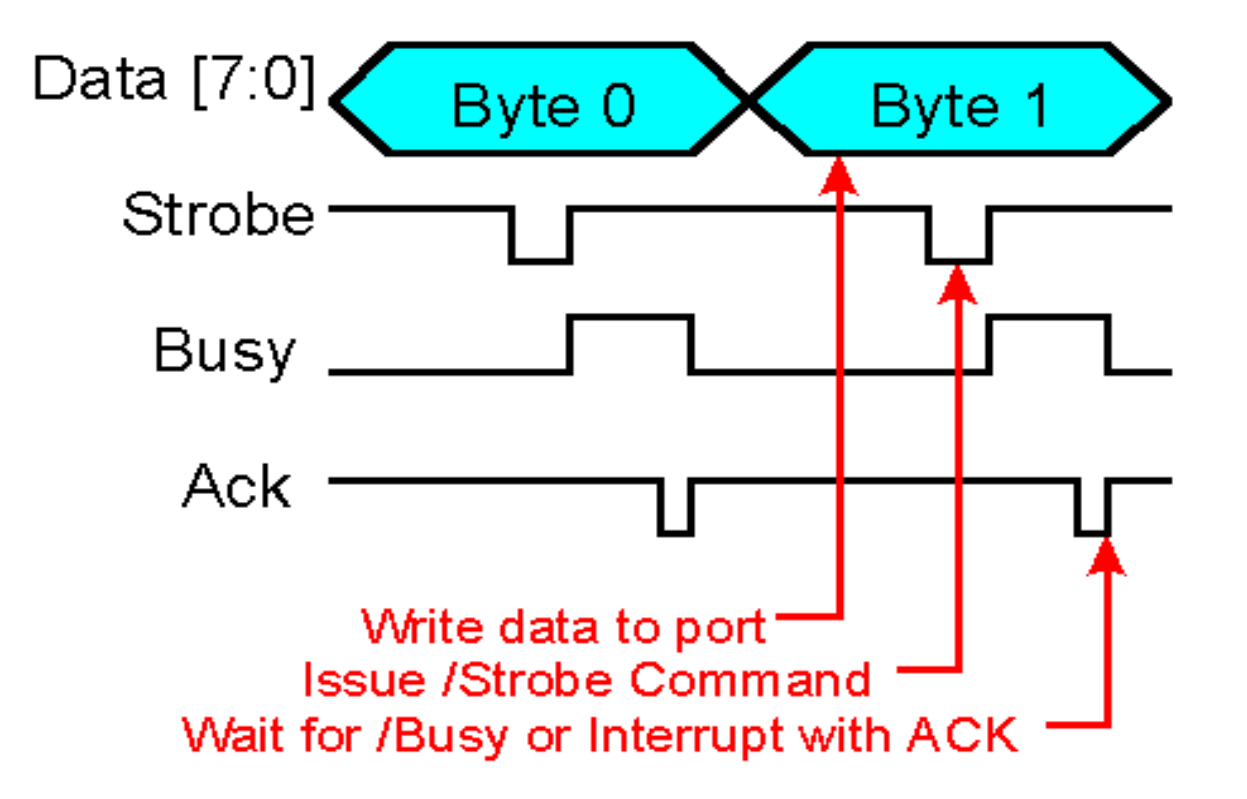
Device	Data Port	Status Port	Control Port
LPT0:	3BCh	3BDh	3BEh
LPT1:	378h	379h	37Ah
LPT2:	278h	279h	27Ah

- The parallel port base addresses are stored at 40:0008h, and can be viewed by using *debug* or *td*
  - At the command prompt, type “debug” to get the “-” debug prompt
  - type **d40:0008** (and hit return)
  - you get : **0040:0008 BC 03 78 03 78 02** …
  - the first six bytes are addresses of the installed printer adapters

Parallel interface major signals

Signal(s)	Input/Output	Notes
Data[7:0]	Bi-directional	Byte-wide data bus
Strobe	To device	Write signal
Busy	From device	Don't send more data
Ack	From device	Acknowledge interrupts
Initialize	To device	Initialize external device
Out of Paper	From device	Status signal

Writing to the parallel interface



Control & Status Ports

Control Port		Status Port	
Bit	**Description**	**Bit**	**Description**
7-5	Unused	7	Busy [inverted]
4	IRQ Enable	6	Ack
3	Select [inverted]	5	Out-of-Paper
2	Initialize	4	Selected
1	AutoFeed [inverted]	3	I/O Error
0	Strobe [inverted]	2-0	Unused