Digital microsystems design

Marius Marcu

2021

Objectives

- Specific objective
  - I/O ports
  - Parallel I/O design and programming
  - Serial I/O design and programming
  - Interrupts

Outline

- I/O ports
- Parallel interface
  - Introduction
  - Implementation
  - Addressing
  - Operation modes
  - Applications
- Serial interface
  - Introduction
  - Implementation
  - Addressing
  - Applications

I/O ports

- An I/O port of a microsystem is an interface between the microprocessor and peripheral devices
- Examples of peripheral devices:
  - Input devices: keyboard, switch, push button
  - Output devices: LED, LCD, printers
  - Input/output: serial communication, network
- An I/O port can be implemented by
  - Logic circuits: gates, registers, flip-flops, buffers
  - Specialized circuits: serial, parallel, network controller, video controller

I/O ports

- I/O ports implementation needs an I/O decoder
- I/O decoder implementation is based on I/O addresses map (I/O address space)
  - Similar with memory decoder and memory map
- I/O cycles of x86 processors consider 16 address lines A15-0 or 8 address lines A7-0
  - 65536 I/O ports

I/O ports

- I/O instructions

```
IN AL, PortAddress08 ; 8 bits port address
MOV DX, PortAddress16
IN AL, DX            ; 16 bits port address

OUT PortAddress08, AL
MOV DX, PortAddress16
OUT DX, AL
```

I/O ports

- I/O decoder sample
  - I/O ports map
    - P1 address 20H
    - P2 address 21H
    - P3 address 30H
    - P4 address 31H
  - 8 bis ports
  - 8 bits addresses

I/O ports

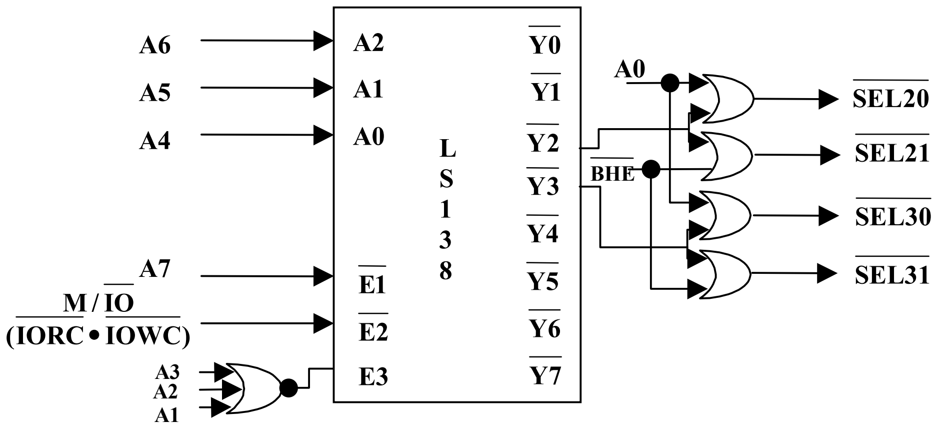
- I/O ports decoding table

A19	A18	A17	A16	A15	A14	A13	A12	A11	A10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0	P
x	x	x	x	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	P1
x	x	x	x	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	
x	x	x	x	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	P2
x	x	x	x	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	
x	x	x	x	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	P3
x	x	x	x	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	
x	x	x	x	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	P4
x	x	x	x	0	0	0	0	0	0	0	0	0	0	1	1	0	0	0	1	

I/O ports

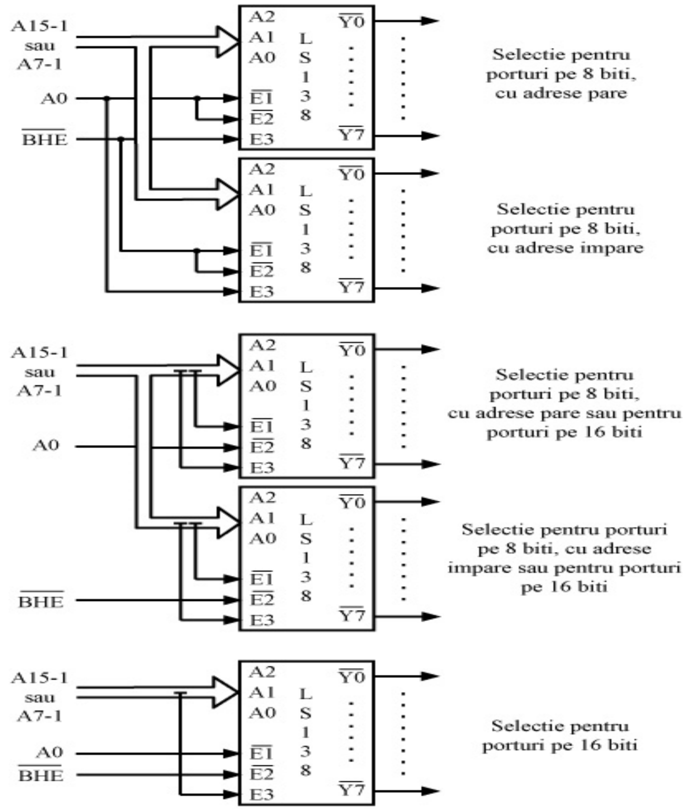
- I/O decoder sample

$\overline{BHE}$	A_0	Characteristics
0	0	Whole word
0	1	Upper byte from/to odd address
1	0	Lower byte from/to even address
1	1	None



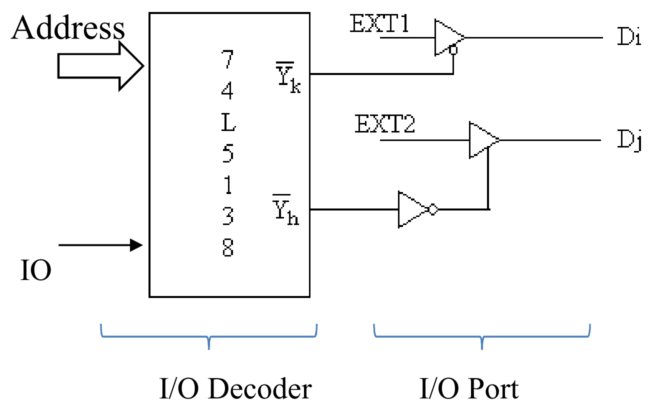
I/O ports

- I/O decoder solutions
  - 32 bits
    - BE0-4 – multiple of 4
  - 16 bits transfers
    - Even address BE0-4 (A0 = 0 , /BHE = 0)
  - 8 bits transfers
    - Odd address (A0 = 1, /BHE = 0)
    - Even address (A0 = 0, /BHE = 1)


A15-1 sau A7-1
A0
BHE
A2 A1 A0 E1 E2 E3
LS138
Y0 Y7
Selectie pentru porturi pe 8 biti, cu adrese pare
Selectie pentru porturi pe 8 biti, cu adrese impare
Selectie pentru porturi pe 8 biti, cu adrese pare sau pentru porturi pe 16 biti
Selectie pentru porturi pe 8 biti, cu adrese impare sau pentru porturi pe 16 biti
Selectie pentru porturi pe 16 biti

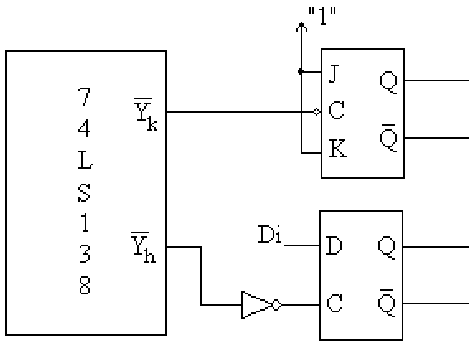
I/O ports

- Input ports
  - Tri-state gates connected to data bus or data lines
  - Tri-state gates inputs connected to peripheral lines
  - Validation signals are generated by I/O port decoder
  - Input machine cycle
    - Similar with memory read cycle
    - Initiated by an IN instruction



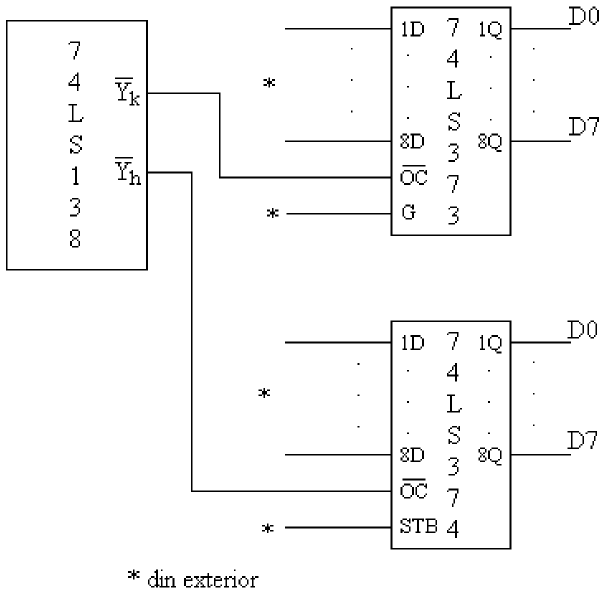
I/O ports

- Output ports
  - Flip-flops
    - Edge, level setup
  - Output machine cycle
    - Similar to memory write cycle
    - Initiated by an OUT instruction



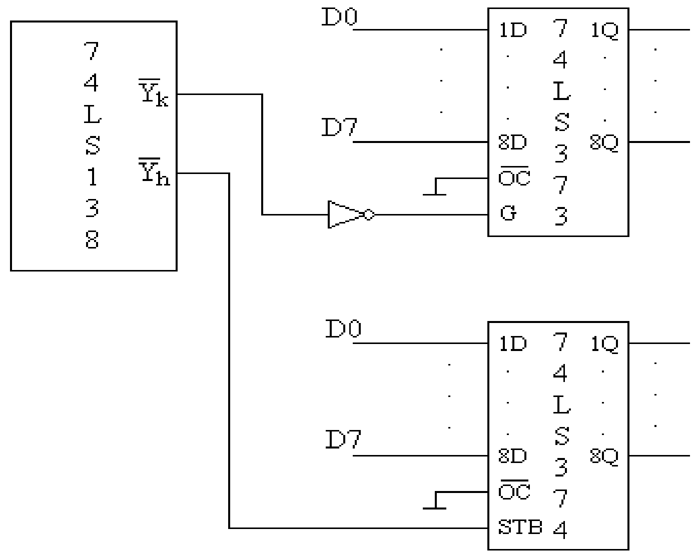
I/O ports

- Input ports
  - Registers
    - Edge, level setup



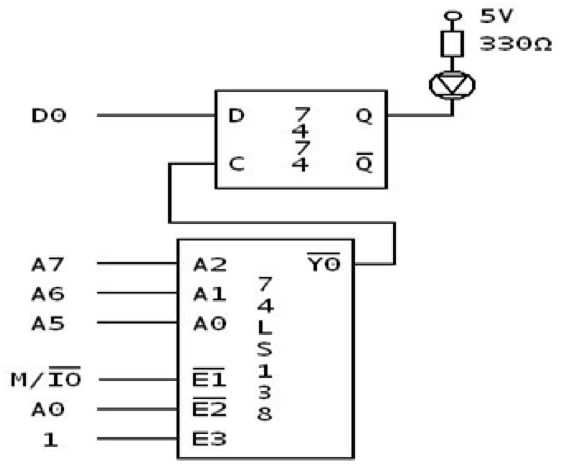
I/O ports

- Output ports
  - Registers
    - Edge, level setup



I/O ports

- Connecting a LED to a microprocessor
  - It will use an output port (register or flip-flop)
  - Example: TTL flip-flop: (I_{OL} = 16 mA, I_{OH} = 0,8 mA)
  - Determine the resistor size: R = (5 – 1,6 – 0,2) V / 10 mA = 320 Ω, usually 330 Ω



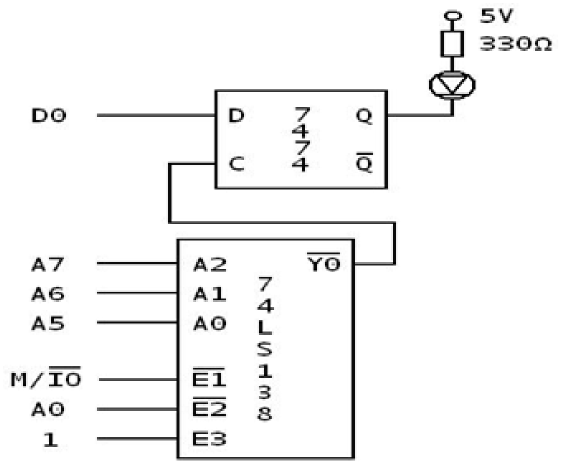
I/O ports

- Connecting a LED to a microprocessor
  - Software driver to turn the LED on

```
MOV      AL,00H
OUT      00H,AL
```

  - Turn the LED off

```
MOV    AL,01H
OUT    00H,AL
```



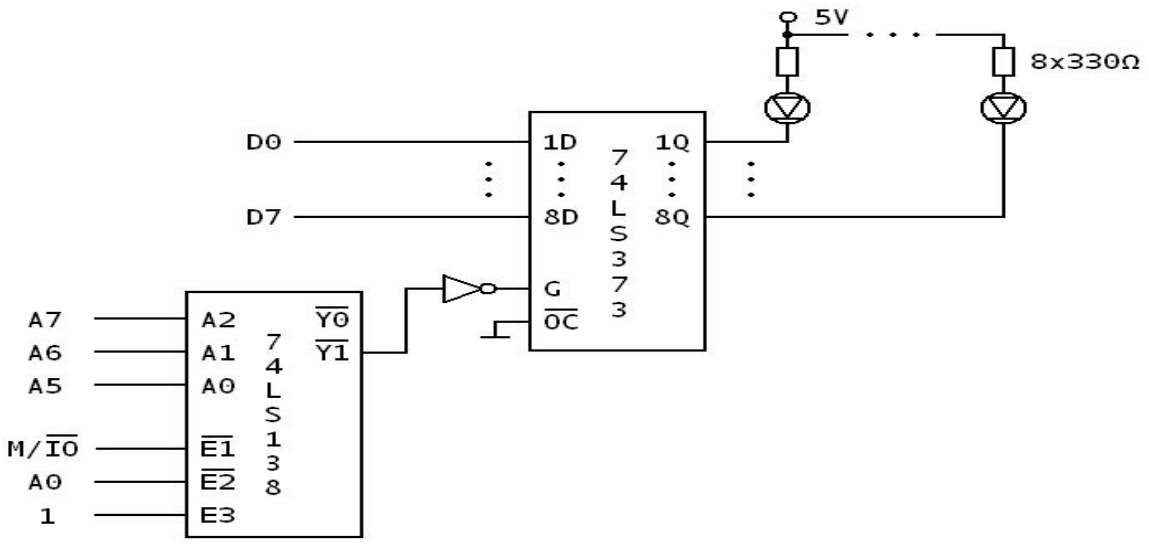
I/O ports

- Connecting several LEDs to a microprocessor
  - Turn the LEDs on:

```
MOV  AL,00H
OUT  20H,AL
```

  - Turn the LEDs off:

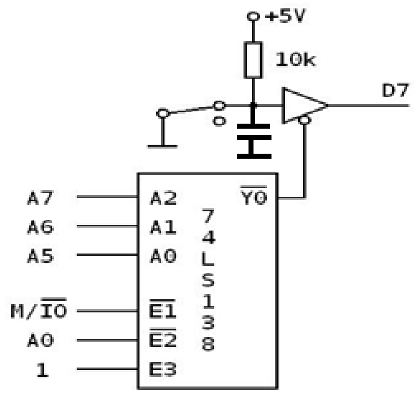
```
MOV  AL,0FFH
OUT  20H,AL
```

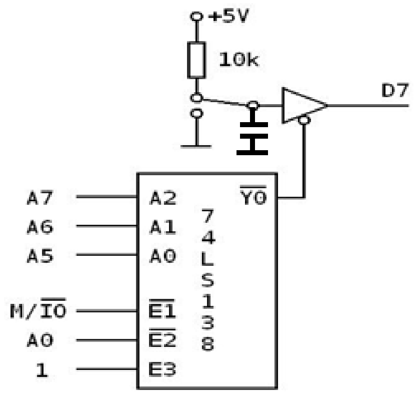


I/O ports

- Connecting a switch to a microprocessor
  - Using a tri-state input buffer

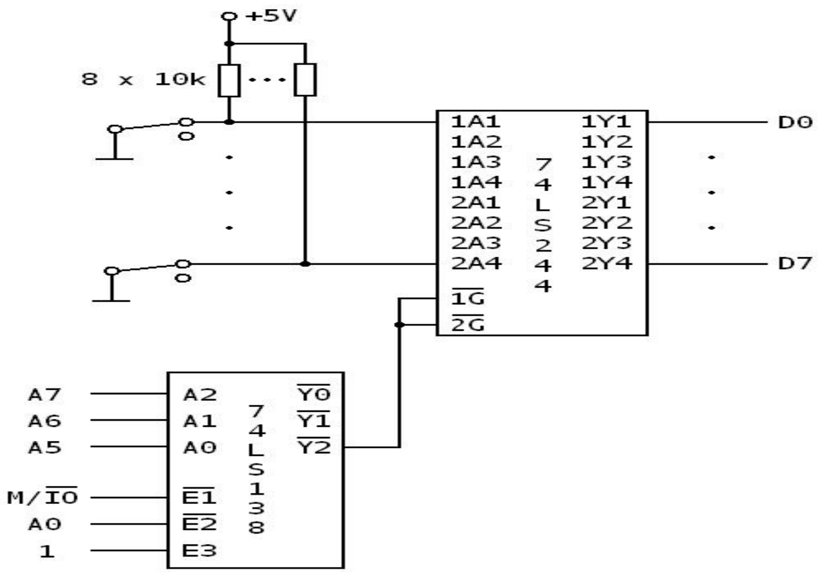
```
IN AL,00H
AND AL,80H
JZ SW_UP ; branch for the switch position represented in the first schematic bellow
; branch for the opposite switch position

IN AL,00H
AND AL,80H
JNZ SW_UP ; branch for the switch position represented in the second schematic bellow
; branch for the opposite switch position
```





I/O ports

- Connecting 8 switches to a microprocessor



Parallel port

- Parallel I/O allows transfers on multiple bits between CPU and any peripheral connected to the interface
- Parallel port has been initially designed as output port for printers and scanners.
- The parallel port is simple and easy to use
- Can be used also as a general purpose I/O
  - Develop prototypes
  - Experiments
  - Tests

Introduction

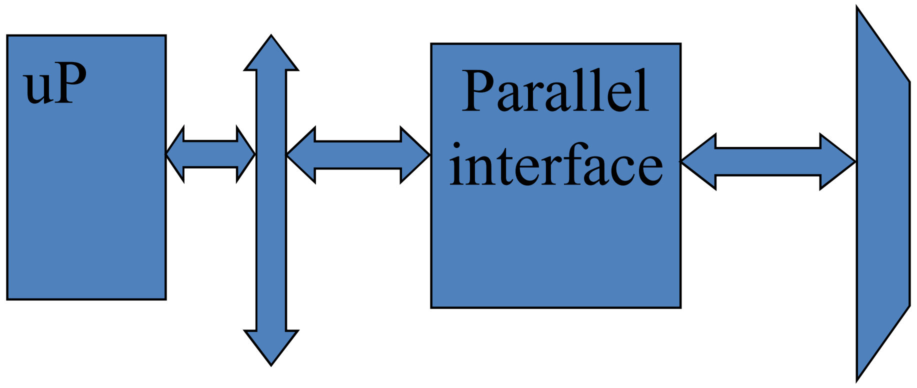
- Parallel port of a PC has been standardized in 1994 under IEEE 1284.
- The standard describes the communication between PC and peripheral through the parallel port.
- IEEE 1284 standard defines completely the characteristics of the parallel I/O interface:
  - Physical specifications: cables, connectors
  - Electrical specifications: line amplifiers, voltage levels, current ranges, receivers, line impedance, terminators
  - Data specifications: data transfer modes
  - Flow control

Introduction

- The parallel port consists of the parallel interface circuits and the parallel connector with 25 pins located at the rear side of the computer.
- The parallel interface consists of all circuits, command, data and status registers, ensuring the connectivity between CPU and peripheral device.

Introduction

- Parallel port



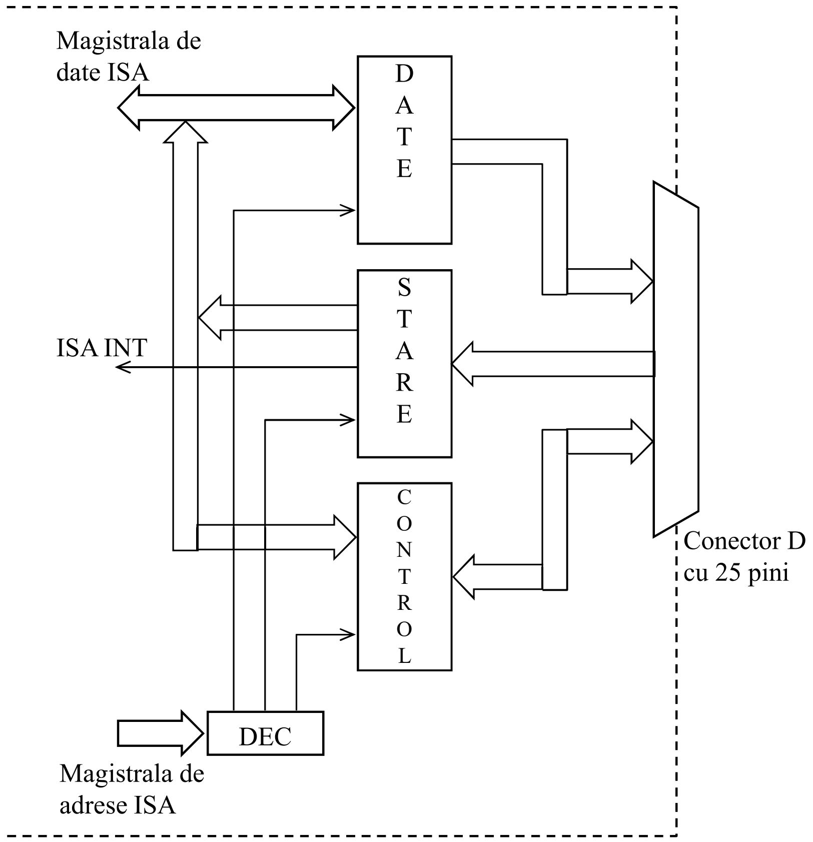
Introduction

- What hardware and software data are needed to develop a driver for a specific I/O port?

Introduction

- To access an I/O port the following information is needed:
  - Registers, input buffers
  - Meaning of registers’ content (commands, data, status)
  - Register pins connected to port connector
  - Addresses of the registers in the I/O space
  - Communication protocol
  - Interrupts
  - DMA channels

Implementation



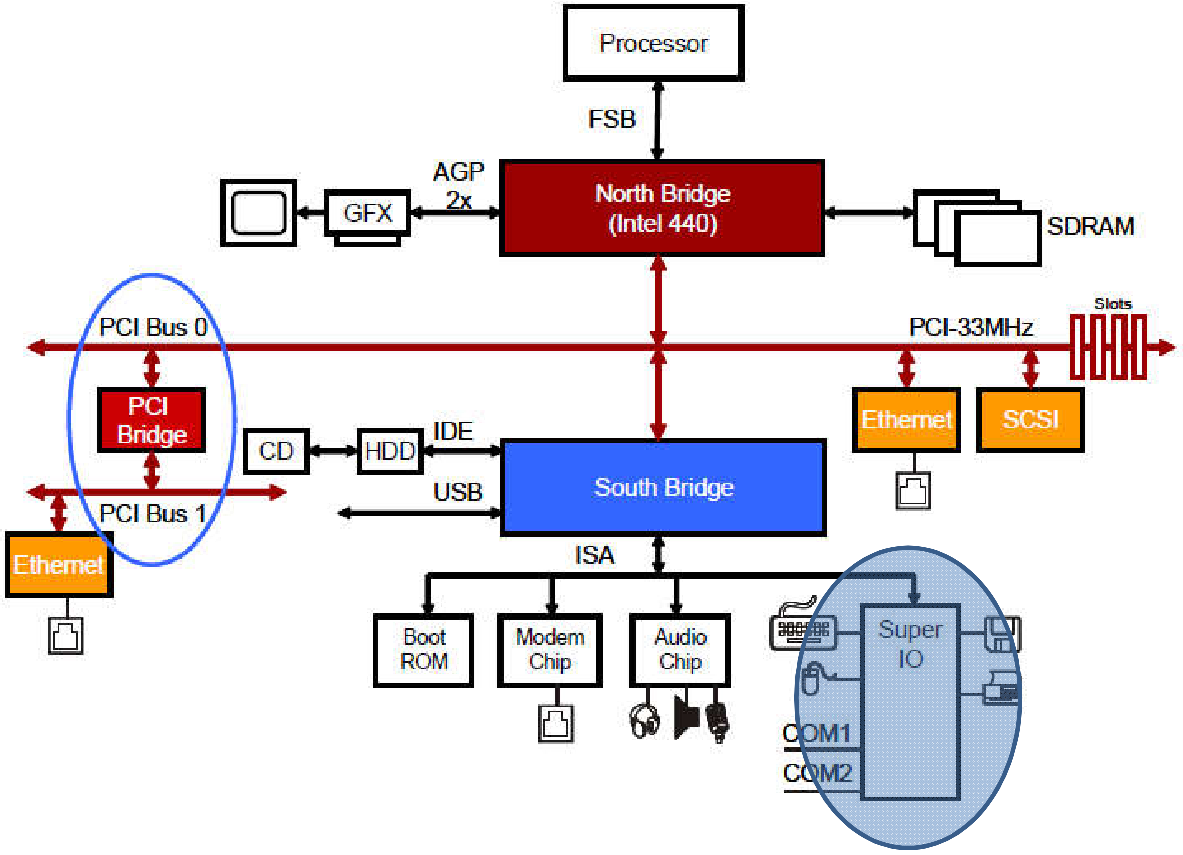
Implementation

- Logically, the parallel interface is an independent port consisting of two registers and one input buffer
- The registers have consecutive addresses in the I/O space of the processor.
- The registers can be read and written by the CPU
- The parallel port registers:
  - A bidirectional data register on 8 bits
  - A bidirectional control register on 6 bits, 4 of them connected to the parallel connector
  - An input buffer on 5 line to get the status of the peripheral.

Implementation

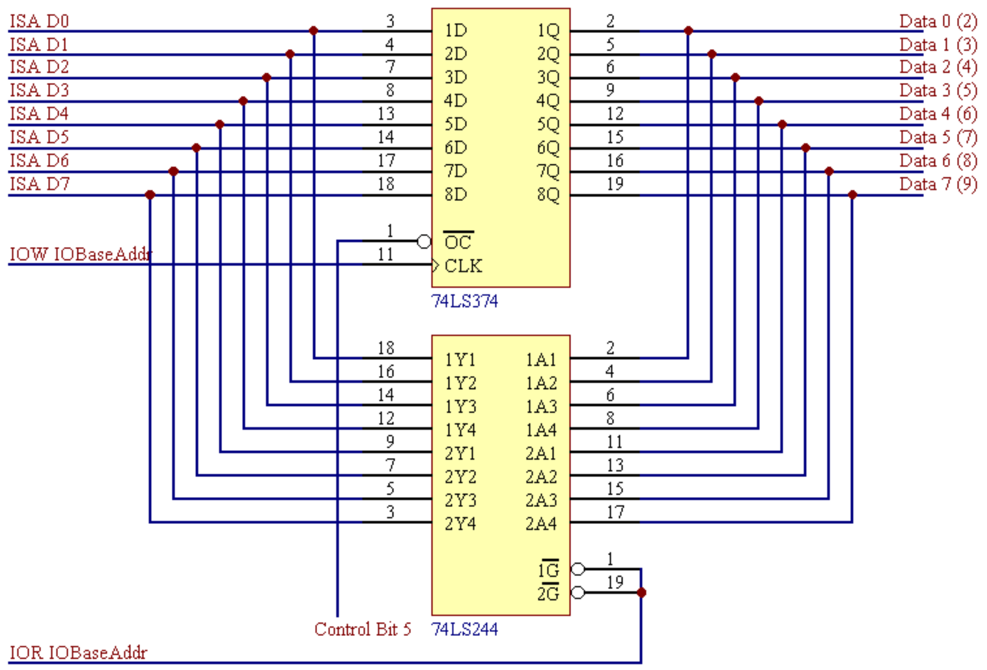
- Parallel interface lines are using TTL logical values, the current differs from one interface to another (up to 12 mA).
- Physically, the parallel interface circuits are located in the I/O chipset on the motherboard

Implementation



Implementation

- Data register



Implementation

- Data register
  - Address: IOBase+0
  - IOR:

```
MOV dx, IOBase                data = inport(IOBase);
IN al, dx
```

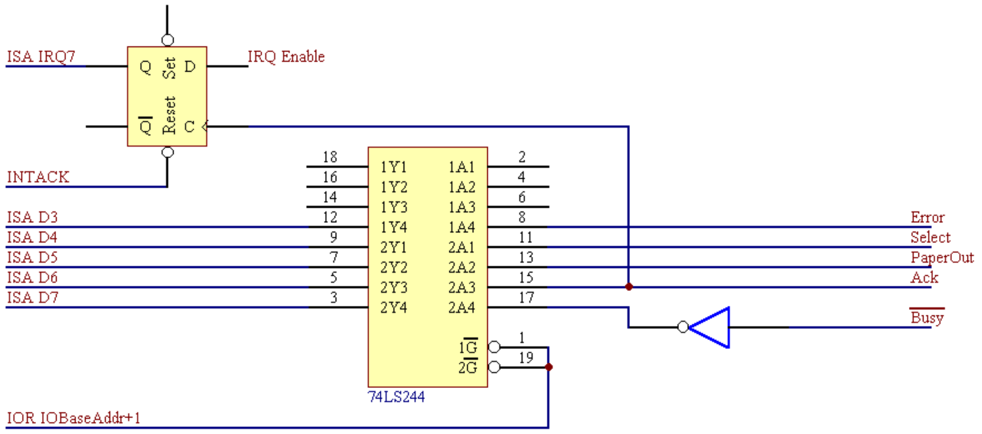
  - IOW:

```
MOV dx, IOBase                outport(IOBase, data);
OUT dx, al
```

  - The outputs of the register are connected to pins 2-9 of the parallel connector
  - Transfer direction is set with bit 5 of the control register

Implementation

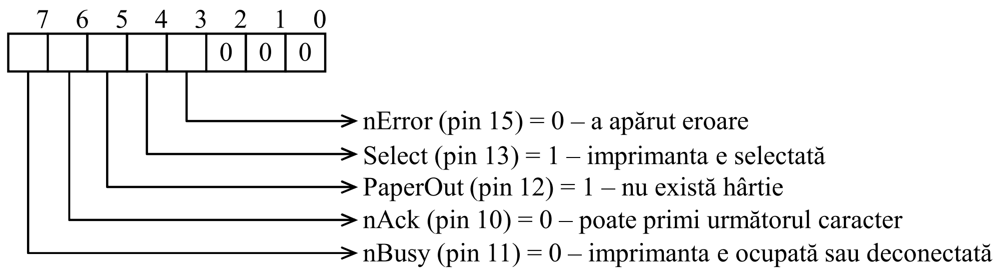
- Status buffer
  - Address: IOBase+1



Implementation

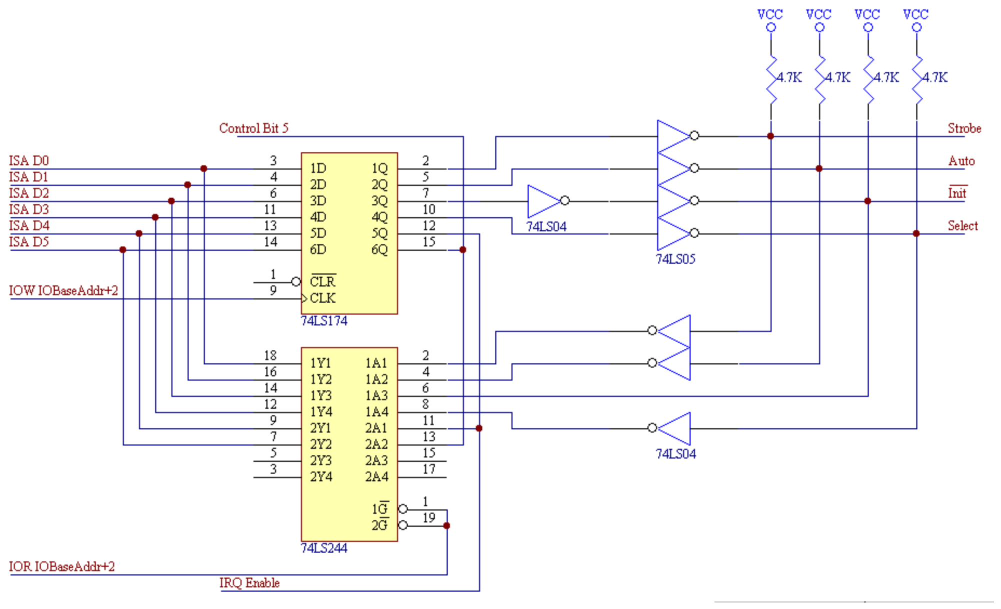
- Status register
  - IOR:

```
MOV dx, IOBase+1          status = inport(IOBase+1);
IN al, dx
```


7 6 5 4 3 2 1 0
0 0 0
nError (pin 15) = 0 – a apărut eroare
Select (pin 13) = 1 – imprimanta e selectată
PaperOut (pin 12) = 1 – nu există hârtie
nAck (pin 10) = 0 – poate primi următorul caracter
nBusy (pin 11) = 0 – imprimanta e ocupată sau deconectată

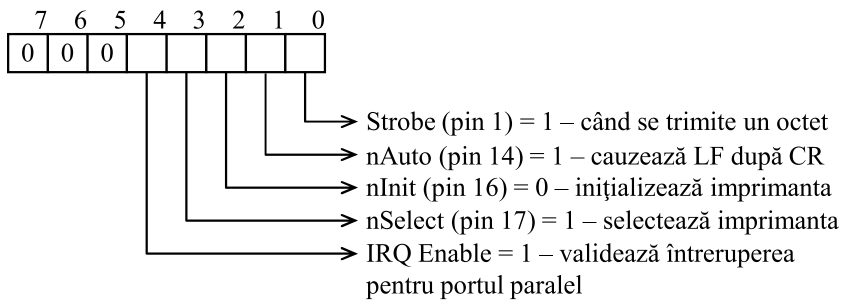
Implementation

- Control register
  - Address: IOBase+2



Implementation

- Control register
  - Register bits 4 and 5 are used to control the behavior of the parallel interface.
    - Bit 4 enables and disables the interrupts
    - Bit 5 specifies the direction of data transfers (I/O)

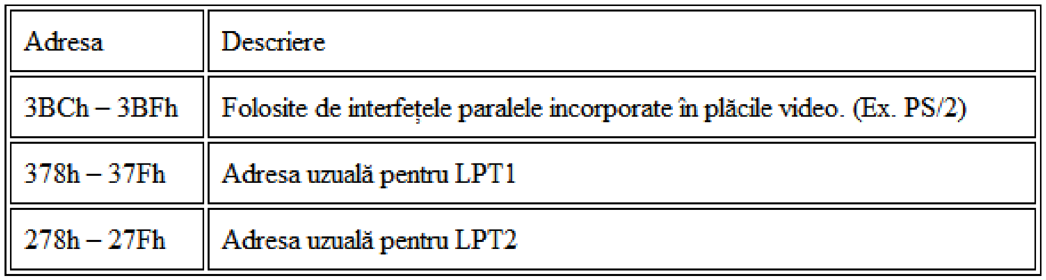

7 6 5 4 3 2 1 0
0 0 0
Strobe (pin 1) = 1 – când se trimite un octet
nAuto (pin 14) = 1 – cauzează LF după CR
nInit (pin 16) = 0 – iniţializează imprimanta
nSelect (pin 17) = 1 – selectează imprimanta
IRQ Enable = 1 – validează întreruperea pentru portul paralel

Implementation

- Control register
  - Bidirectional lines connected to the parallel connector
  - They are using Open-Colector inverters in order to allow the CPU to read the lines configured as inputs
  - In order to read input lines, the register has to be preloaded with 0100b.

Addressing

- BIOS software identifies the available parallel interfaces and it will allocate the following names: LPT1, LPT2, LPT3
- Every parallel interface will get a base address and the next few addresses
- The addresses are standardized starting with the early PCs, before plug-and-play mechanisms existed



Adresa	Descriere
3BCh – 3BFh	Folosite de interfețele paralele incorporate în plăcile video. (Ex. PS/2)
378h – 37Fh	Adresa uzuală pentru LPT1
278h – 27Fh	Adresa uzuală pentru LPT2

Operation modes

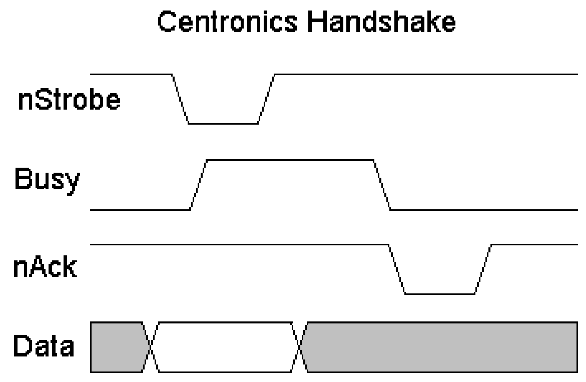
- Specifies the operation mode of the parallel port and data transfer characteristics
- Modes:
  - SPP – Standard Parallel Port
    - Compatibility
    - Nibble
    - Byte
  - EPP – Enhanced Parallel Port
  - ECP – Extended Capability Port

Operation modes

- SPP
  - Communication protocol implemented in software
  - It uses the standard parallel port registers
  - It can monitor external signals and generate output signals
  - Any protocol can be implemented, beside the standard one
  - Limited in performance

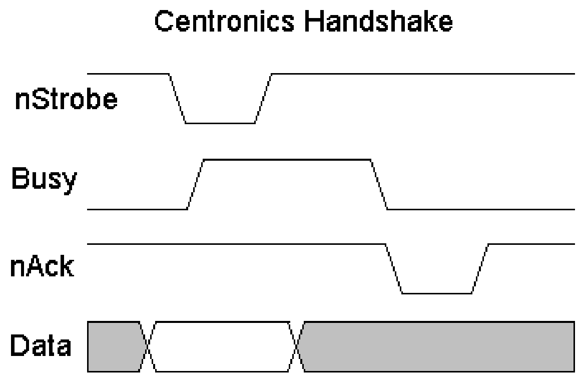
Operation modes

- SPP – Compatibility
  - Uni-directional data sent to a peripheral device (printer)
  - Centronics – standard protocol



Operation modes

- SPP – Compatibility
  - How the Centronics protocol can be implemented in software?



Operation modes

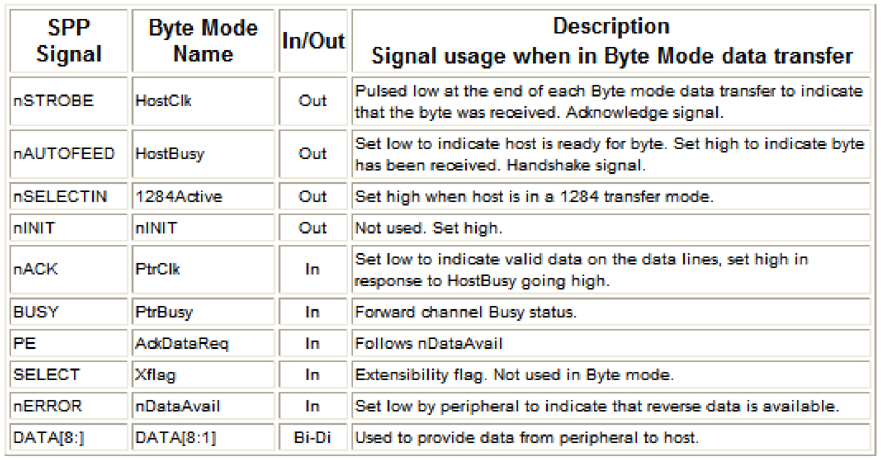
- SPP – Compatibility
  - Reads the status register and verifies the Busy line, in order to see whether the peripheral is ready
  - If the peripheral is ready
    - Writes the 8 bits data into the data register
    - Activates the Strobe line (write into the control register)
    - Deactivates the Strobe line (write into the control register)

Operation modes

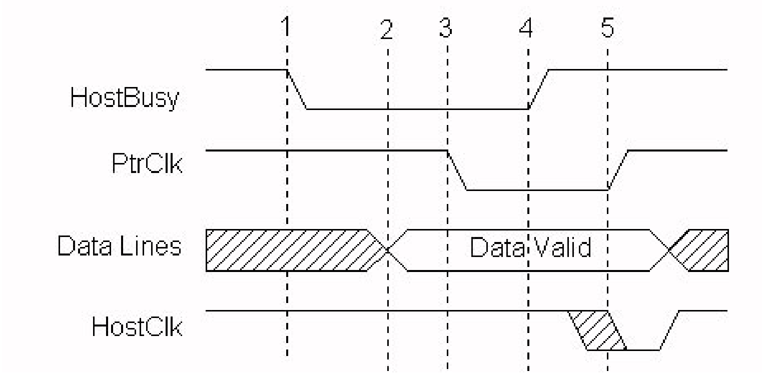
- SPP – Byte
  - Bi-directional transfers on 8 bits
  - Bit 5 of the control register will specify the transfer direction for the data register
    - Bit 5 = 1 – data register is used as input port

Operation modes

- SPP – Byte



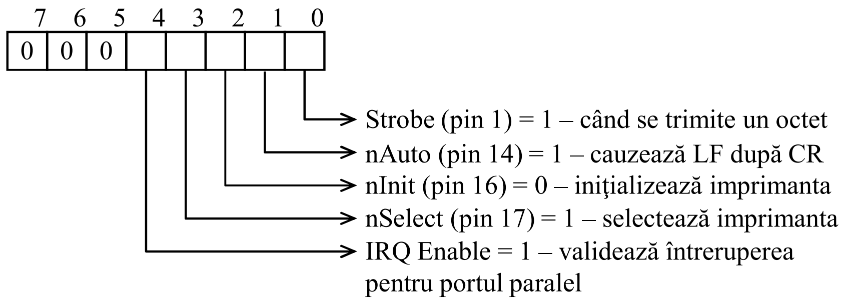
SPP Signal	Byte Mode Name	In/Out	Description Signal usage when in Byte Mode data transfer
nSTROBE	HostClk	Out	Pulsed low at the end of each Byte mode data transfer to indicate that the byte was received. Acknowledge signal.
nAUTOFEED	HostBusy	Out	Set low to indicate host is ready for byte. Set high to indicate byte has been received. Handshake signal.
nSELECTIN	1284Active	Out	Set high when host is in a 1284 transfer mode.
nINIT	nINIT	Out	Not used. Set high.
nACK	PtrClk	In	Set low to indicate valid data on the data lines, set high in response to HostBusy going high.
BUSY	PtrBusy	In	Forward channel Busy status.
PE	AckDataReq	In	Follows nDataAvail
SELECT	Xflag	In	Extensibility flag. Not used in Byte mode.
nERROR	nDataAvail	In	Set low by peripheral to indicate that reverse data is available.
DATA[8:]	DATA[8:1]	Bi-Di	Used to provide data from peripheral to host.



Applications

- Generate a clock signal at the parallel port lines

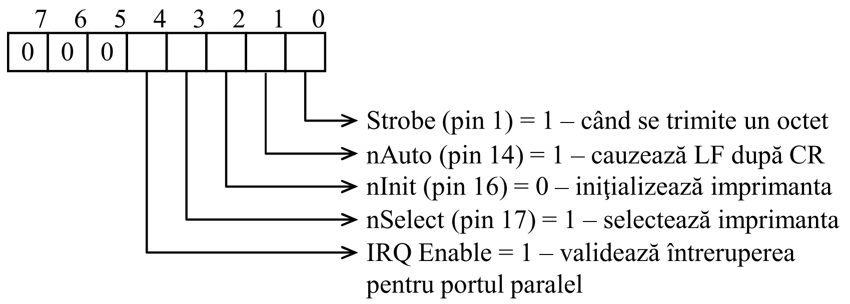
```
PARPORT_BASE    equ       378h
signal_gen      proc      near
      push      ax
      push      cx
      push      dx

      mov       cx, 2000
      mov       dx, PARPORT_BASE +2
      in        al, dx              ; read control port

next:
      and       al, 0feh            ; activate STROBE
      out       dx, al
      or        al, 01h             ; deactivate STROBE
      out       dx, al
      loop      next

      pop       dx
      pop       cx
      pop       ax
      ret
signal_gen      endp
```


7 6 5 4 3 2 1 0
0 0 0
Strobe (pin 1) = 1 – când se trimite un octet
nAuto (pin 14) = 1 – cauzează LF după CR
nInit (pin 16) = 0 – iniţializează imprimanta
nSelect (pin 17) = 1 – selectează imprimanta
IRQ Enable = 1 – validează întreruperea pentru portul paralel

Applications

- Generate a clock signal at the parallel port lines

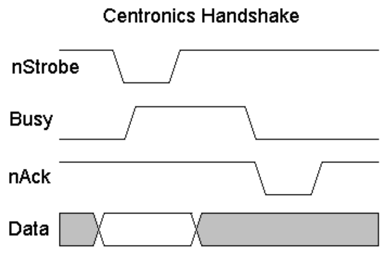
```
PARPORT_BASE    equ       378h
signal_gen      proc      near
      push      ax
      push      cx
      push      dx

      mov       cx, 2000
      mov       dx, PARPORT_BASE +2
      in        al, dx              ; citeste portul de control

next:
      xor       al, 1               ; inverseaza STROBE
      out       dx, al
      loop      next

      pop       dx
      pop       cx
      pop       ax
      ret
signal_gen      endp
```


7 6 5 4 3 2 1 0
0 0 0
Strobe (pin 1) = 1 – când se trimite un octet
nAuto (pin 14) = 1 – cauzează LF după CR
nInit (pin 16) = 0 – iniţializează imprimanta
nSelect (pin 17) = 1 – selectează imprimanta
IRQ Enable = 1 – validează întreruperea pentru portul paralel

Applications

- Print a character using Centronics protocol

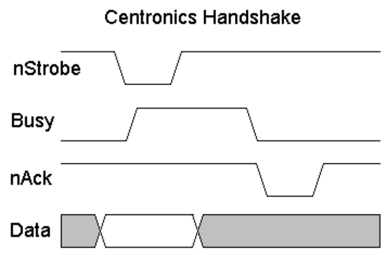
```
    PARPORT_BASE    equ       378h
    print_char      proc      near
          ; Tipareste caracterul din cl
          push      ax
          push      dx

          mov       dx, PARPORT_BASE +1
busy:     in        al, dx              ; testează linia BUSY
          test      al, 80h             ; din registrul de stare
          jz        busy

          mov       al, cl
          mov       dx, PARPORT_BASE    ; scrie caracterul în
          out       dx, al              ; registrul de date
          mov       dx, PARPORT_BASE +2
          mov       al, 0eh             ; activează STROBE
          out       dx, al
          or        al, 01h             ; dezactivează STROBE
          out       dx, al
          pop       dx
          pop       ax
          ret
    print_char      endp
```



Applications

- Print a character using Centronics protocol

```
#define PARPORT_BASE 378h
bool print_char(char ch)
{
       byte status, control;

       // testeaza linia busy
       while( (status = inport(PARPORT_BASE +1)) & 0x80 == 0 );

       // scrie caracterul in registrul de date
       outport(PARPORT_BASE, ch);

       control = inport(PARPORT_BASE +2);
       // activeaza strobe
       control = control & 0xfe;
       outport(PARPORT_BASE +2, control);
       // dezactiveaza strobe
       control = control ^ 1;
       outport(PARPORT_BASE +2, control);
}
```



Serial Interface

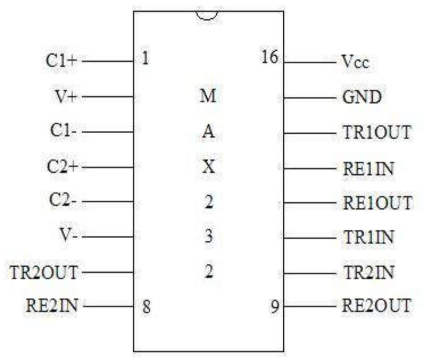
- Serial port has been proposed as an alternative for the parallel port in connecting peripheral devices in environments with perturbations at longer distances (over 3 m).
- Devices usually connected to the serial port are modems, code bars scanners and phone branch exchanges (PBX).
- The serial port is using serial communication protocols, transferring information on 2 lines: Rx and Tx
- It is easy to program and transfer data to other embedded devices
- It can be used as a virtual serial port over USB, Bluetooth or (W)LAN

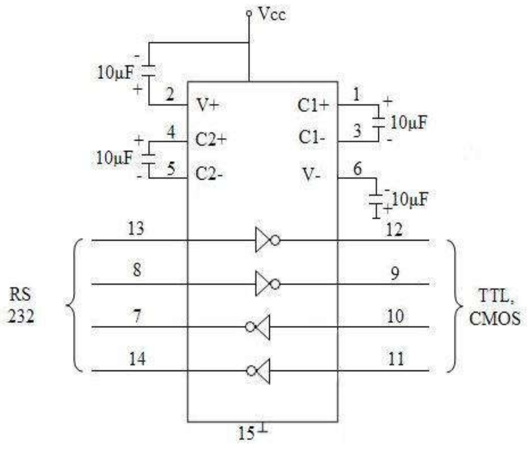
Introduction

- Serial port is described be the RS232 standard
- The standard defines the logical, electrical and mechanical requirements and specifications for serial communication interfaces.
  - Voltage levels (EIA, PC +/- 12V)
  - Transfer type: synchronous, asynchronous
  - Equipment type: DTE, DCE
  - Parameters of serial communication
  - Flow control

Introduction

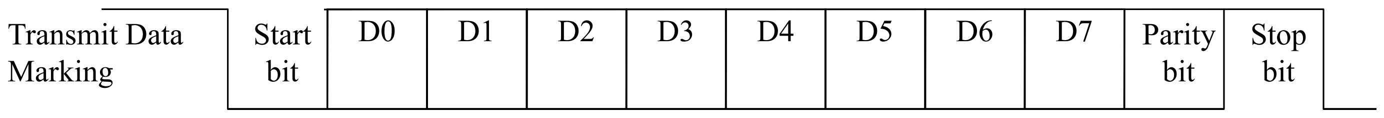
- Voltage levels:
  - EIA (Electronic Industries Association)
    - - 25V ÷ - 3V “1” logic
    - + 3V ÷ + 25V “0” logic.
- Conversion circuits are needed TTL → EIA and EIA → TTL: MAX232





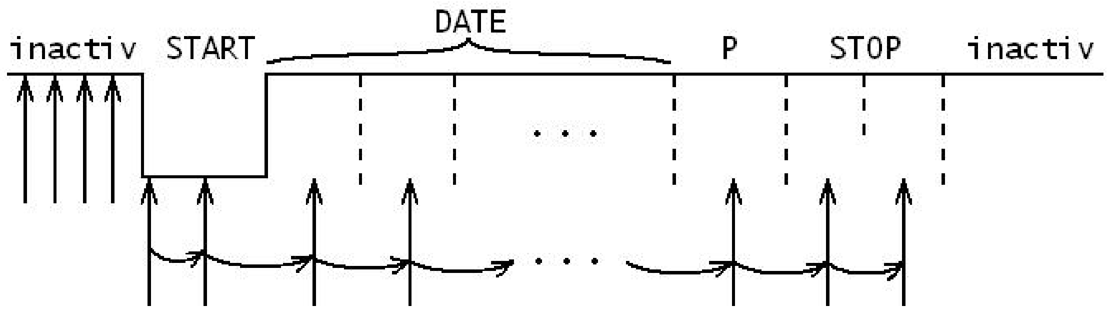
Introduction

- Asynchronous communication:
  - Transfer one character at a time, bit after bit
  - No clock signal
  - The receiver is synchronized with the emitter by setting the same communication parameters



Introduction

- Asynchronous communication:
  - Data reception
  - When the receiver knows when an new transmission is in place?
    - Data framing

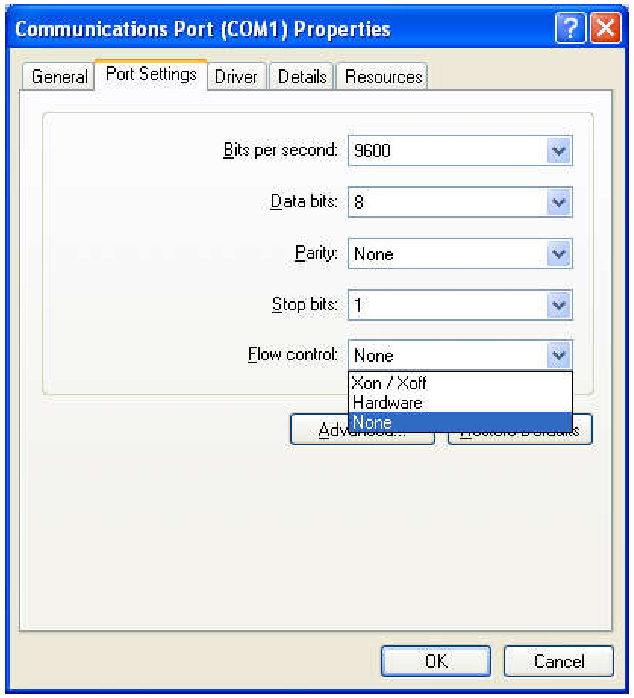


Introduction

- Asynchronous communication:
  - How synchronization is possible?
  - Communication parameters
    - Start bit
    - Data bits: 5-8
    - Parity bit
    - Stop bits
    - Baud rates: 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200
  - Both transmitter and receiver have to be configured using exactly the same parameters in order to understand each other

Introduction

- Serial communication parameters
  - Both transmitter and receiver have to be configured using the same parameters


Communications Port (COM1) Properties

Bits per second: 9600
Data bits: 8
Parity: None
Stop bits: 1
Flow control: None

Introduction

- Flow control
  - What is it?
  - Why is needed?

Introduction

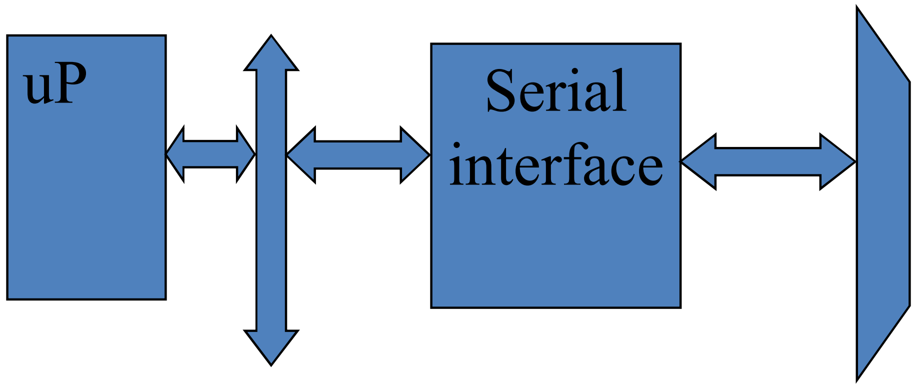
- Flow control
  - Address the synchronization of block transfers
    - How the transmitter knows if the receiver is able to receive more data or should stop transmission
  - Solutions:
    - No control
    - Software (XON/XOFF)
    - Hardware (RTS/CTS)

Introduction

- The serial port consists of the serial interface circuits and the serial connector with 9 pins located at the rear side of the computer.
- The serial interface consists of all circuits, specialized and line transceivers, ensuring the connectivity between CPU and peripheral device, using serial communication

Introduction

- Serial port

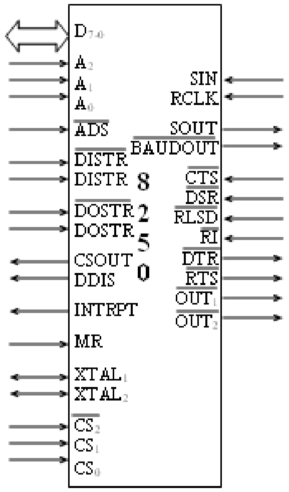


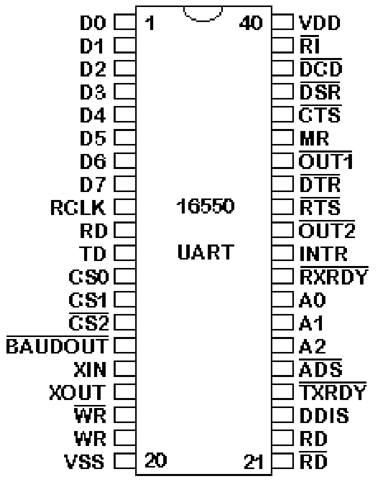
Introduction

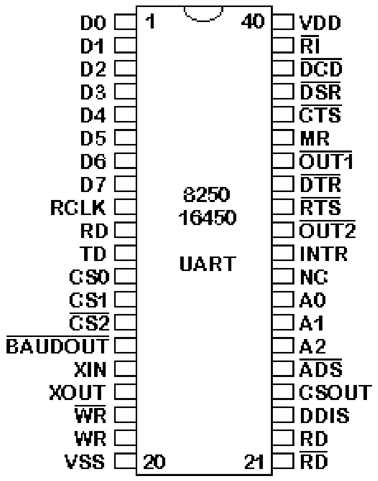
- To access an I/O port the following information is needed:
  - Registers, input buffers
  - Meaning of registers’ content (commands, data, status)
  - Register pins connected to port connector
  - Addresses of the registers in the I/O space
  - Communication protocol
  - Interrupts
  - DMA channels

Implementation

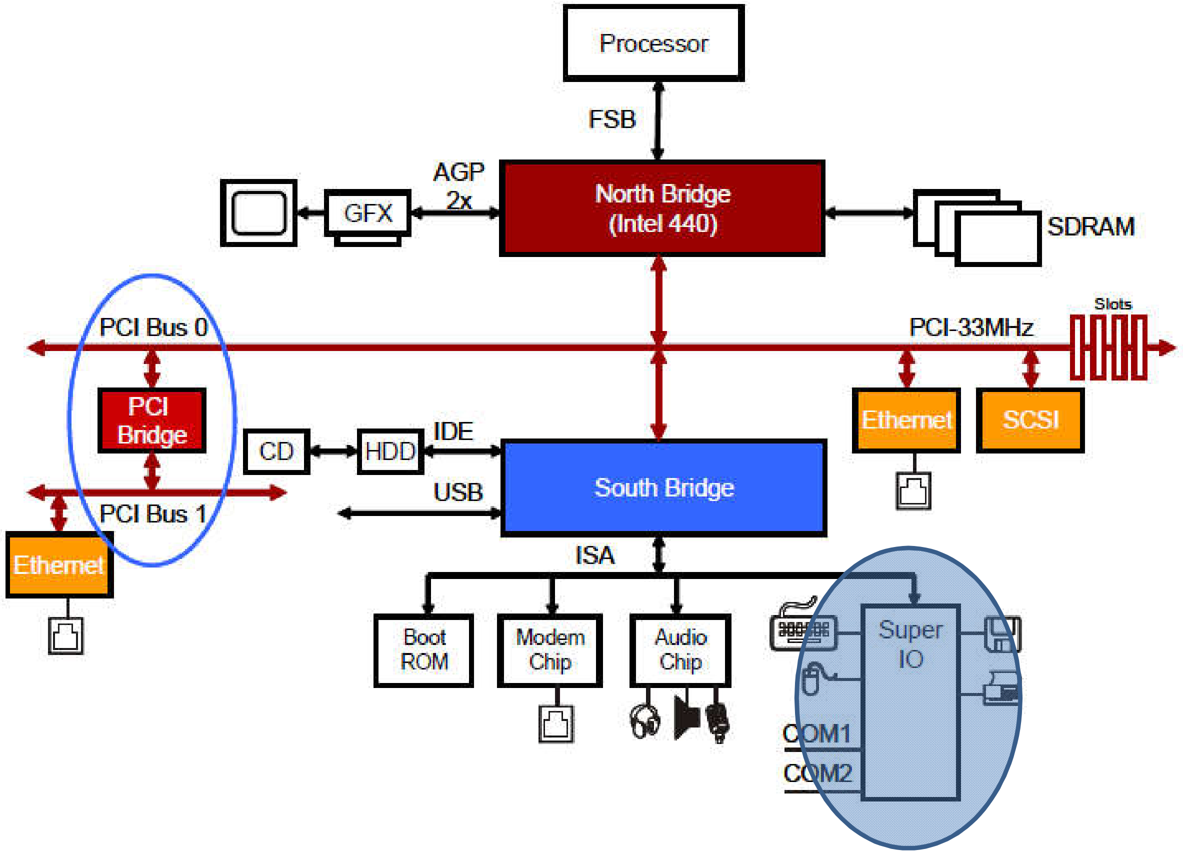
- Physically, the serial interface circuits are connected with the I/O chipset on the motherboard (Super I/O circuits)
- Circuits UART 8250, 16450, 16550.







Implementation



Implementation

- Logically, the serial interface is an independent port consisting of one ***specialized circuit*** with ten interface registers
- The registers have consecutive addresses in the I/O space of the processor.
- The registers can be read and written by the CPU
- Serial registers include:
  - Transmission and reception buffers
  - Status register
  - Control registers

Implementation

- Addressing
  - Base address

DLAB	A2	A1	A0	Register
0	0	0	0	Receiver Buffer (Read)/ Transmitter Holding Register (Write)
0	0	0	1	Interrupt Enable
x	0	1	0	Interrupt Identification Register (Read Only)
x	0	1	1	Line Control
x	1	0	0	Modem Control
x	1	0	1	Line Status
x	1	1	0	Modem Status
x	1	1	1	Scratch
1	0	0	0	Divisor Latch (Least Significant Byte)
1	0	0	1	Divisor Latch (Most Significant Byte)

Implementation

- Serial interface registers

I/O Address	DLAB	RD/WR	Description
IOBase+0	0	Write	Transmiter Holding Register
IOBase+0	0	Read	Receiver Buffer
IOBase+0	1	Read/Write	Divisor Latch Low Byte
IOBase+1	0	Read/Write	Interrupt Enable Register
IOBase+1	1	Read/Write	Divisor Latch High Byte
IOBase+2	-	Read	Interrupt Identification Register
IOBase+2	-	Write	FIFO Control Register
IOBase+3	-	Read/Write	Line Control Register
IOBase+4	-	Read/Write	Modem Control Register
IOBase+5	-	Read	Line Status Register
IOBase+6	-	Read	Modem Status Register
IOBase+7	-	Read/Write	Scratch Register

Implementation

- Transmission and reception buffers
  - Address: IOBase+0
  - IOR: (read the last received characters)

```
MOV dx, IOBase                data = inport(IOBase);
IN al, dx
```

  - IOW: (transmission of one characters)

```
MOV dx, IOBase                outport(IOBase, data);
OUT dx, al
```

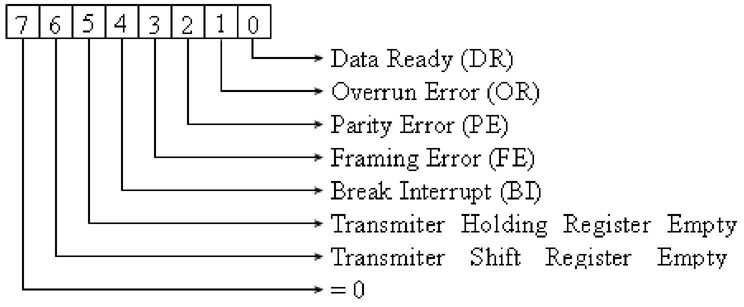
  - The serial circuit is in charge of serialization and deserialization the characters
  - Bits are serialized on TxD line and the character is reassembled at the destination from the RxD line
    - Starting with LSB

Implementation

- How exactly the application on the receiver side can detect new arrived characters?

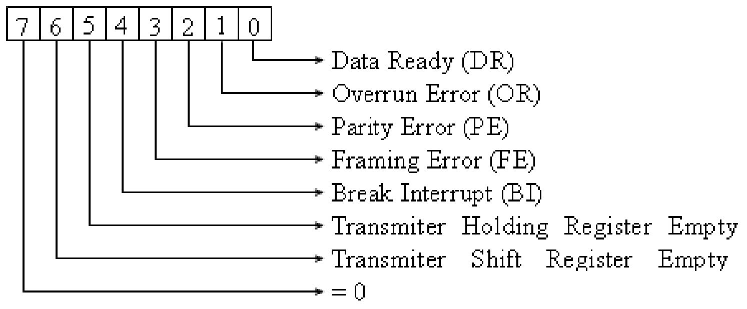
Implementation

- Status register
  - Address: IOBase+5
  - Bit 0 = 1 – new received character assembled and loaded into the receiver buffer.
  - Bit 1 = 1 – the previous character was overwritten by a new received character before if was read by the app – overrun error



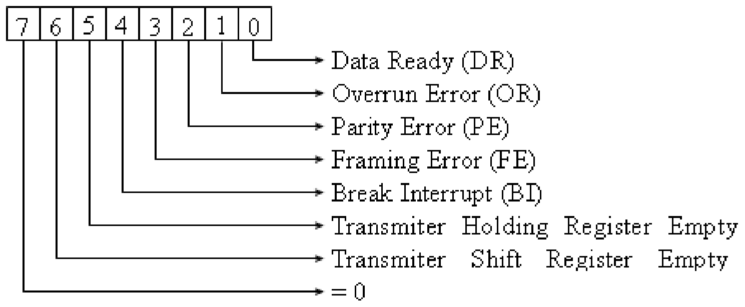
Implementation

- Status register
  - Bit 2 – parity error
  - Bit 3 – framing error – the received characters has an erroneous stop bit
  - Bit 4 – interrupt the transmission, by the transmitter while stetting the line on 0 (Set Break).



Implementation

- Status register
  - Bit 5 – transmitter holding register empty – the serial circuit is ready to accept a new character to send.
    - This bit is automatically set on 1 when the character is moved from holding register to shifting register.
    - This bit is automatically set on 0 when the CPU send an new characters.
  - Bit 6 – transmitter shift register empty. This bit is automatically set on 0 when a new character is loaded from the holding register.



Implementation

- Why two transmitter status flags?
  - Transmiter Holding Register Empty
  - Transmiter Shift Register Empty

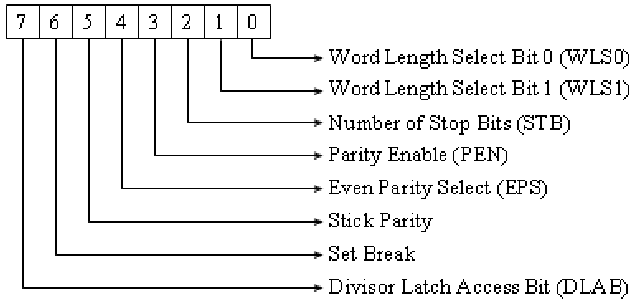
Implementation

- Control register
  - Address: IOBase + 3
  - IOR:

```
MOV dx, IOBase+3            data = inport(IOBase+3);
IN al, dx
```

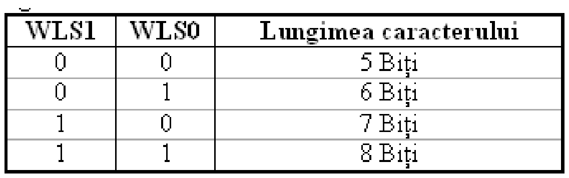
  - IOW:

```
MOV dx, IOBase+3            outport(IOBase+3, data);
OUT dx, al
```


7 6 5 4 3 2 1 0
Word Length Select Bit 0 (WLS0)
Word Length Select Bit 1 (WLS1)
Number of Stop Bits (STB)
Parity Enable (PEN)
Even Parity Select (EPS)
Stick Parity
Set Break
Divisor Latch Access Bit (DLAB)

Implementation

- Control register
  - Bits 1,0 – size of the character



WLS1	WLS0	Lungimea caracterului
0	0	5 Biţi
0	1	6 Biţi
1	0	7 Biţi
1	1	8 Biţi

  - Bit 2 number of stop bits:
    - STB = 0 – 1 stop bit;
    - STB = 1 and the char length is 5 - 1 ½ stop bits;
    - STB = 1 şi and the char length > 5 – 2 stop bits.

Implementation

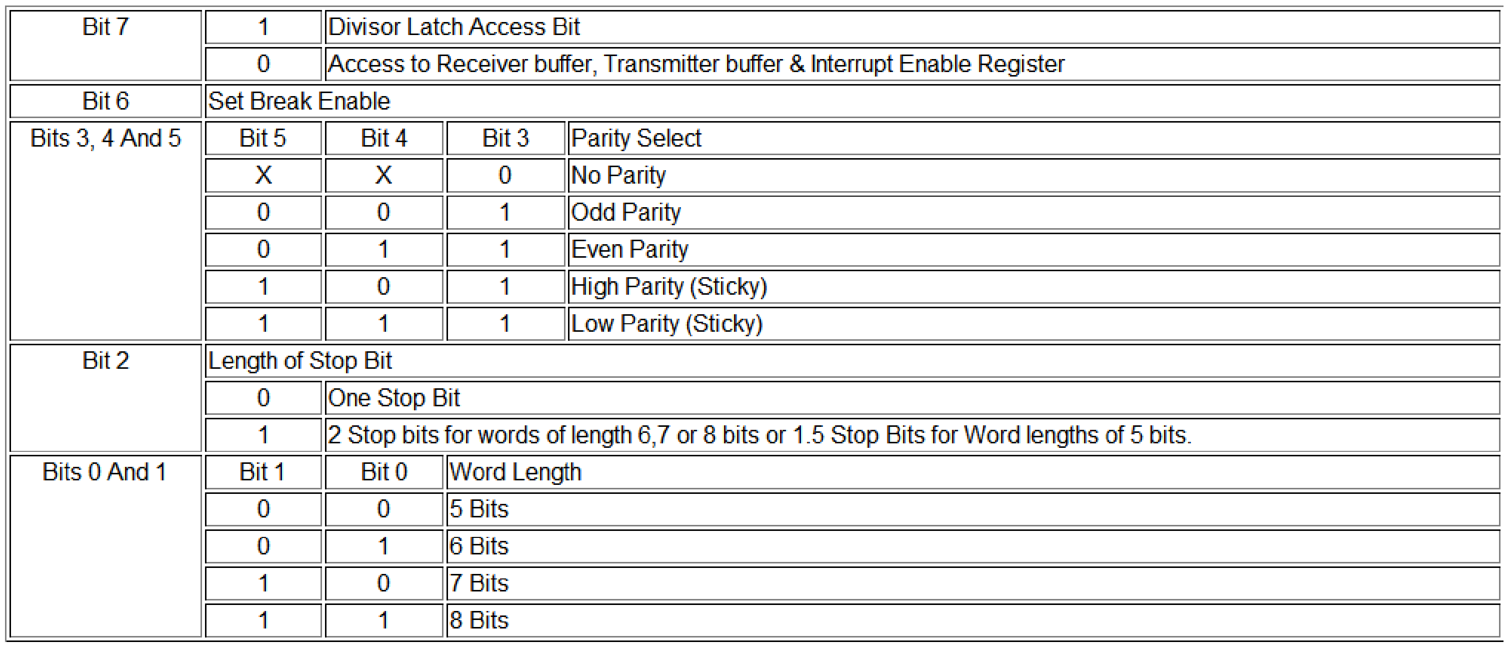
- Control register
  - Bit 3 usage of the parity bit in transmission.
    - PEN = 0 – no parity
    - PEN = 1 – use parity bit in transmission.
  - Bit 4 set the parity type.
    - EPS = 0 and PEN = 1 – odd parity (odd number of 1s in the packet);
    - EPS = 1 and PEN = 1 – even parity (even number of 1s in the packet).

Implementation

- Control register
  - Bit 5 – sticky parity (fixed parity bit) - 0 if EPS=1 or 1 if EPS=0.
  - Bit 6 - set break control bit.
  - Bit 7 divisor latch enable bit:
    - DLAB = 1 – allow access to divisor registers;
    - DLAB = 0 – allow access to reception/transmission buffers and interrupt register.

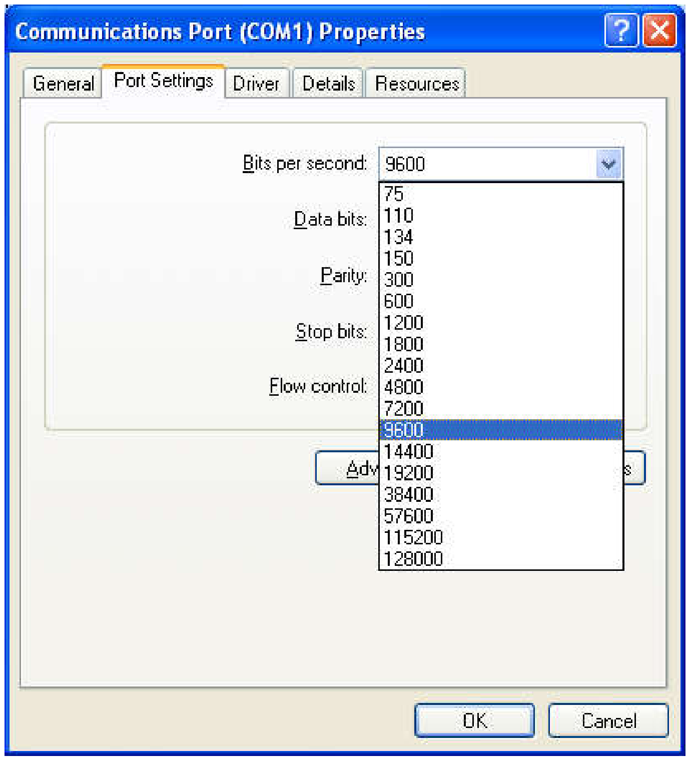
Implementation

- Control register



Bit 7	1	Divisor Latch Access Bit		
	0	Access to Receiver buffer, Transmitter buffer & Interrupt Enable Register		
Bit 6	Set Break Enable			
Bits 3, 4 And 5	Bit 5	Bit 4	Bit 3	Parity Select
	X	X	0	No Parity
	0	0	1	Odd Parity
	0	1	1	Even Parity
	1	0	1	High Parity (Sticky)
	1	1	1	Low Parity (Sticky)
Bit 2	Length of Stop Bit			
	0	One Stop Bit		
	1	2 Stop bits for words of length 6,7 or 8 bits or 1.5 Stop Bits for Word lengths of 5 bits.		
Bits 0 And 1	Bit 1	Bit 0	Word Length	
	0	0	5 Bits	
	0	1	6 Bits	
	1	0	7 Bits	
	1	1	8 Bits	

Implementation



Communications Port (COM1) Properties

General | Port Settings | Driver | Details | Resources

Bits per second: 9600

Data bits:

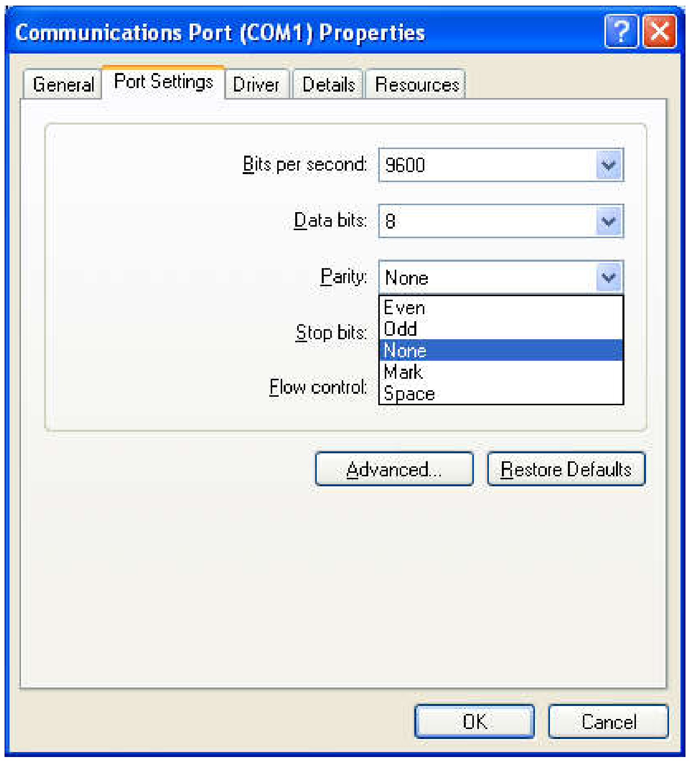
Parity:

Stop bits:

Flow control:

75
110
134
150
300
600
1200
1800
2400
4800
7200
9600
14400
19200
38400
57600
115200
128000

OK | Cancel



Communications Port (COM1) Properties

General | Port Settings | Driver | Details | Resources

Bits per second: 9600

Data bits: 8

Parity: None

Stop bits:

Flow control:

Even
Odd
None
Mark
Space

Advanced... | Restore Defaults

OK | Cancel

Implementation

- Divisor registers
  - Address: IOBase+0 şi IOBase+1
  - Divide the input tack with a divisor between 1 and $2^{16} - 1$
  - The divisor is loaded into the divisor register
  - The frequency of the generator is 16x baud rate.
  - The input frequency is fixed to 1.84MHz

```
divisor = (input frequency) / (baud rate x 16)
```

Implementation

- Divisor registers

Viteza (bps)	Rgistrul de divizare mai semnificativ	Registrul de divizare mai puţin semnificativ
50	09h	00h
300	01h	80h
600	00h	C0h
2400	00h	30h
4800	00h	18h
9600	00h	0Ch
19200	00h	06h
38400	00h	03h
57600	00h	02h
115200	00h	01h

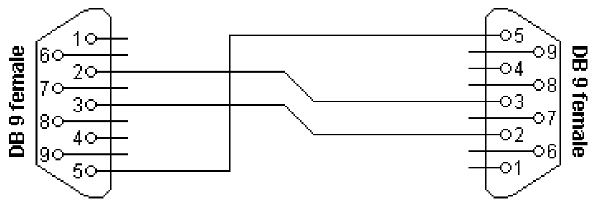
```
outport(IOBase+3, 0x80);
outport(IOBase+1, 0x00);
outport(IOBase+0, 0x06);
outport(IOBase+3, ...);
```

Implementation

- Classes of devices
  - DTE – data terminal equipment (terminal nodes – computers, peripherals)
  - DCE – data communication equipment (communication nodes – modems)
- Cables and connectors
  - DTE-DCE
  - DTE-DTE

Implementation

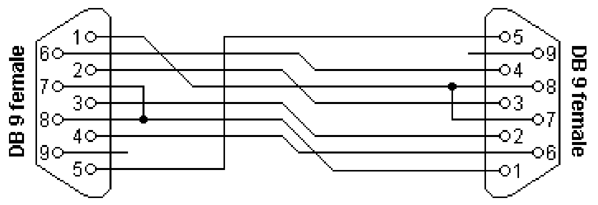
- Cables and connectors (DTE-DTE)
  - Null modem without dialog



Connector 1	Connector 2	Function
2	3	Rx ← Tx
3	2	Tx → Rx
5	5	Signal ground

Implementation

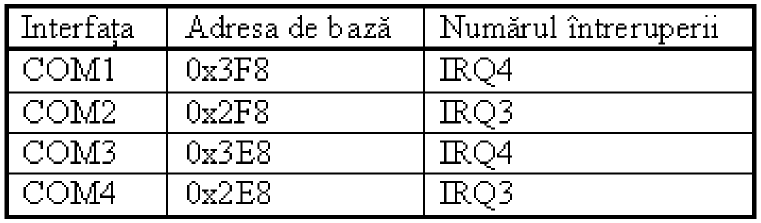
- Cables and connectors (DTE-DTE)
  - Null modem with partial dialog



Connector 1	Connector 2	Function
1	7 + 8	$RTS_2 \rightarrow CTS_2 + CD_1$
2	3	$Rx \leftarrow Tx$
3	2	$Tx \rightarrow Rx$
4	6	$DTR \rightarrow DSR$
5	5	Signal ground
6	4	$DSR \leftarrow DTR$
7 + 8	1	$RTS_1 \rightarrow CTS_1 + CD_2$

Addressing

- BIOS software identifies the available serial interfaces and it will allocate the following names: COM1, COM2, COM3
- Every serial interface will get a base address and the next few addresses
- The addresses are standardized starting with the early PCs, before plug-and-play mechanisms existed

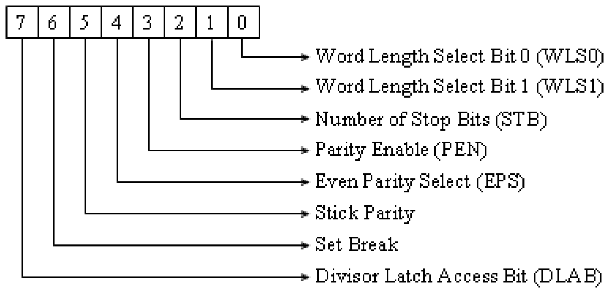


Interfața	Adresa de bază	Numărul întreruperii
COM1	0x3F8	IRQ4
COM2	0x2F8	IRQ3
COM3	0x3E8	IRQ4
COM4	0x2E8	IRQ3

Applications

- Programming the serial communication parameters

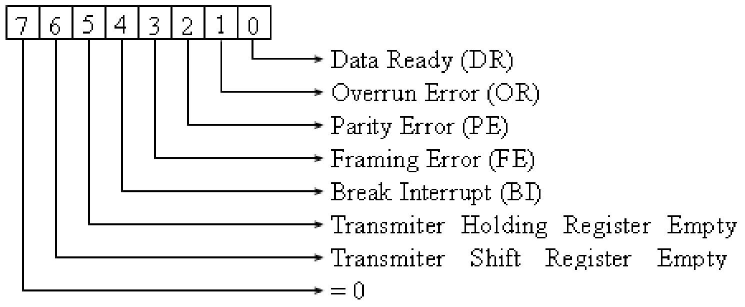
```
COMPORT_BASE    equ       3f8h
com_config      proc      near
      push      ax
      push      dx
      mov       dx, COM_ADDR + 3    ; control register
      mov       al, 80h             ; DLAB enable
      out       dx, al
      mov       dx, COM_ADDR + 1    ; divisor register
      mov       al, 0               ; MSB
      out       dx, al
      mov       dx, COM_ADDR        ; divisor register
      mov       al, 03h             ; LSB
      out       dx, al
      mov       dx, COM_ADDR + 3    ; control register
      mov       al, 1bh             ; 00011011b = 8E1
      out       dx, al
      pop       dx
      pop       ax
      ret
signal_gen      endp
```


7
6
5
4
3
2
1
0
Word Length Select Bit 0 (WLS0)
Word Length Select Bit 1 (WLS1)
Number of Stop Bits (STB)
Parity Enable (PEN)
Even Parity Select (EPS)
Stick Parity
Set Break
Divisor Latch Access Bit (DLAB)

Applications

- Serial reception of one character

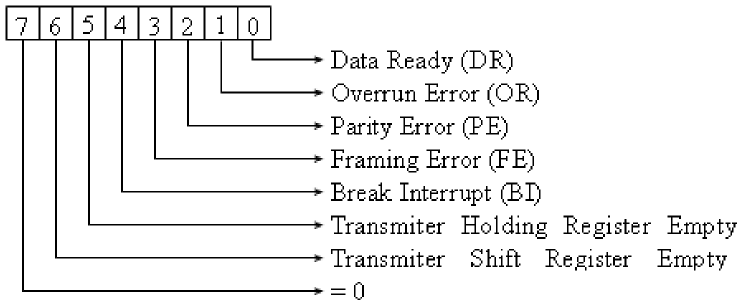
```
   com_recv      proc
   ; output: al – received char
        push     dx
        mov      dx, 3fdh         ; status register address
recv:   in       al, dx           ; read status register
        test     al, 01h          ; is new char received
        jz       recv
        mov      dx, 3f8h         ; reception buffer address
        in       al, dx           ; read received data
        pop      dx
        ret
com_recv endp
```



Applications

- Serial transmission of one character

```
    com_send        proc
    ; input: al - char to send
          push      dx
          push      ax
          mov       dx, 3fdh  ; status register address
send:     in        al, dx    ; read status register
          test al, 20h        ; is THRE?
          jz        send
          pop       ax
          mov       dx, 3f8h  ; transmission buffer address
          out       dx, al    ; send data
          pop       dx
          ret
com_send  endp
```

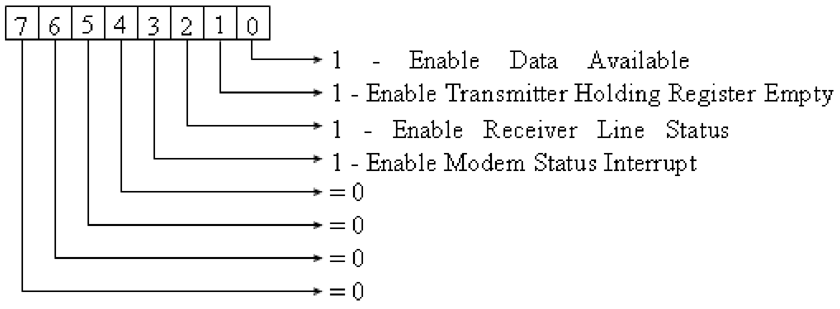


Applications

- Data transfers
  - Polling
  - Interrupts
  - DMA – Direct memory access

Applications

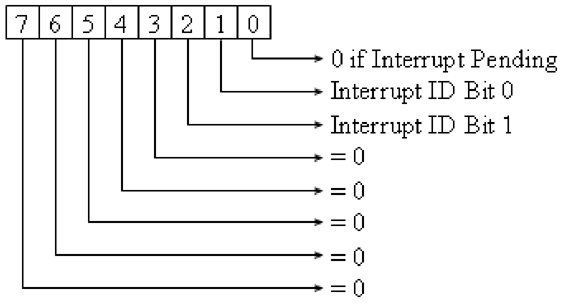
- Interrupt enable register
  - Address: IOBase+1
  - Enables events that can generate interrupts



7
6
5
4
3
2
1
0
1 - Enable Data Available
1 - Enable Transmitter Holding Register Empty
1 - Enable Receiver Line Status
1 - Enable Modem Status Interrupt
= 0
= 0
= 0
= 0

Applications

- Interrupt identification register
  - Address: IOBase+2
  - Identifies source of the interrupt



Interrupts

- Interrupts and exceptions
- Interrupt acknowledge cycle
- Interrupt handling
- Interrupt service routines

Interrupts

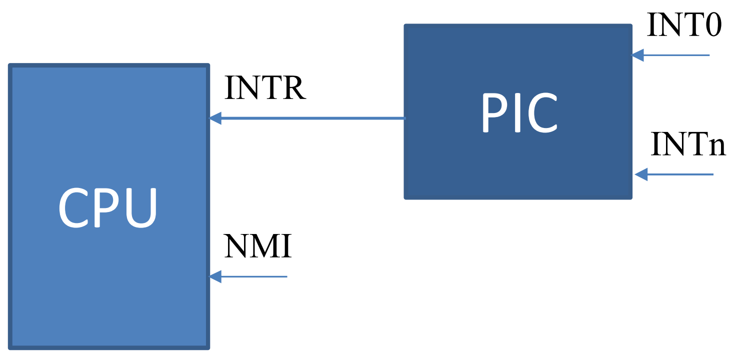
- The main goal of the interrupt mechanism is to permit peripherals to ask for immediate attention from the processor
  - Peripherals can interrupt at runtime the execution of code in order to start a dedicated handling routine
- Interrupt requests are sent to the processor using interrupt pins
- Peripherals’ interrupts can be raised anytime – asynchronous events
  - They are not synchronized with the current software execution

Interrupts

- External interrupts – Hardware generated interrupts
    - Maskable interrupts – can be programmatically ignored by the processor
        - INTR pin
        - Ignored when IF (FLAGS) is 0 by using STI and CLI instructions
    - Non-maskable interrupts – they are not ignored by the processor
        - NMI pin
    - Inter-processor interrupts – in multi-processor systems

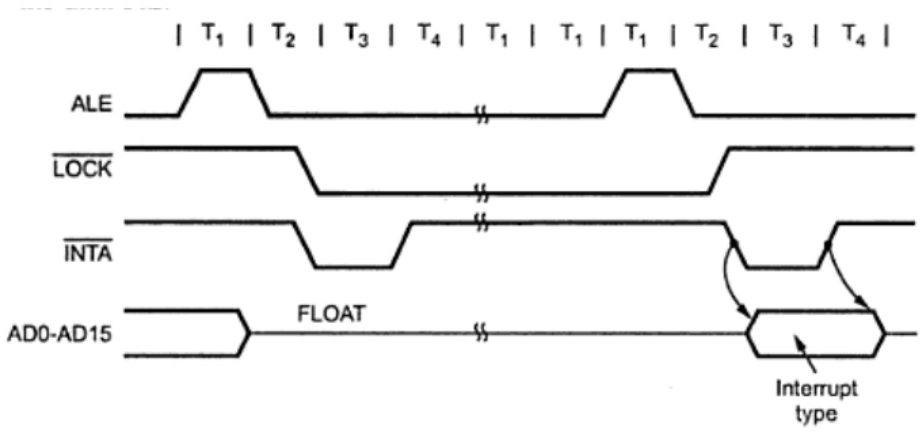
Interrupts

- External interrupts
  - They are connected to interrupt pins of the processor using an (A)PIC (Advanced Programmable Interrupt Controller)
    - Arbitration of peripheral interrupts
    - Prioritization
    - Level vs. edge triggered interrupts



Interrupts

- External interrupts
  - Interrupt acknowledge machine cycle
    - 8086 initiates two machine cycles when an interrupt request occurs
      - Interrupt acknowledge (INTA signal)
      - Interrupt type receive (INTA signal + vector number)



Interrupts

- External interrupts
  - When an interrupt occurs (hardware):
    - the processor will wait the current cycle to finish – instructions are atomic (they cannot be interrupted)
    - It will initiate the first interrupt acknowledge cycle when it will float data bus lines – it shows that an interrupt request is accepted by the processor
    - It will continue with the second interrupt acknowledge cycle when it will receive the interrupt type (vector)

Interrupts

- External interrupts
  - When an interrupt occurs (software):
    - The processor stores the FLAGS register into stack
    - It disables any further interrupts (clears TF and IF)
    - Pushes the CS and IP values of the next instruction on the stack
    - The vector is used as an index into the interrupt vectors table – it reads the address in the interrupt vector table from offset = 4*vector and segment = 4*vector+2
    - Call the interrupt service routine whose address is read from the vector table
    - Interrupt service routine should return with the IRET instruction

Interrupts

- External interrupts
  - Return from interrupt service routine
    - Pops return address from the stack (CS and IP of the instruction following the interrupted one)
    - Restores the FLAGS content from the stack
    - Continues the execution of the original code from the point it was interrupted

Interrupts

- Internal interrupts – exceptions, CPU-generated
  - Fault – correctable error
  - Trap – programmer initiated (not an error)
  - Abort – severe error

Interrupts

- Internal interrupts
  - Software interrupts – called by program code
    - INT nn instructions
    - nn = 0 .. 255
  - Exceptions – raised when a software and hardware error condition occurs
    - Divide by zero
    - Debug (single step, break point)
    - Bus error
    - Memory error

Interrupts

- Exceptions
  - The processor detects an error condition while executing an instruction
  - Examples (Faults)
    - div ebx, eax
      - divide by 0 when EAX is 0
    - mov eax, [ebx]
      - page fault or segment violation if EBX is un-mapped virtual address
    - jmp label
      - General protection fault if label is invalid address in the code segment

Interrupts

- Software interrupts
  - Are synchronous – called every time the instruction is executed
  - Interrupt service routines can be invoked from software using INT instruction
    - e.g. INT 2 will call the ISR associated to NMI
  - Are treated the same way like hardware interrupts except external cycles

Interrupts

- Interrupt vector table

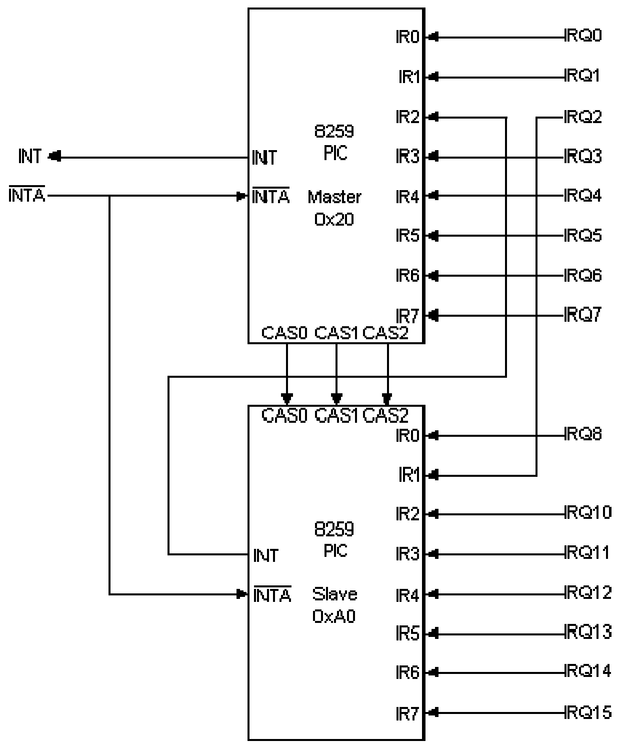
Vector	Type	Description	Address
0	Fault	Divide error	0000H
1	Exception	Single step	0004H
2	Interrupt	NMI interrupt	0008H
3	Exception	Breakpoint	000CH
4	Exception	Overflow	0010H
5			0014H
6	Fault	Invalid opcode	0018H
7-12			001CH-0030H
13	Fault	General protection	0034H
14	Fault	Page fault	0038H
15-16			003CH-0040H
17			0044H
18-31			0048H-007CH
32-255	Interrupt	User defined (INTR, INT nn)	0080H-03FCH

Interrupts

- APIC
  - Has 16 interrupt request lines IRQ0-IRQ15
  - It can be programmed to map IRQ lines to vector numbers
  - It activates the INTR of the processor when at least one IRQ line is active
    - During interrupt acknowledge cycle it passes the vector to the processor
  - Prioritizes the IRQ requests when more of them are active
    - When several IRQ lines are active the controller will serialize their transfer to the processor INTR line
    - How exactly the PIC will know when the previous interrupt is finished in order to send the next highest priority active IRQ to CPU?

Interrupts

- APIC



Interrupts

- Interrupt service routine
  - Disable interrupts
  - Call old interrupt handler
  - Advertise end of interrupt handling
  - Enable interrupts

```
/* interrupt pointer for old timer ISR */
void interrupt (* oldhandler)();

void interrupt yourisr()   /* Interrupt Service Routine (ISR) */
{
            disable();

            /* Body of ISR goes here */
            …

            oldhandler();

            outportb(0x20,0x20);     /* Send EOI to PIC1 */
            enable();
}
```

Interrupts

- Interrupt service routine
  - Save old interrupt handler
  - Set new interrupt handler
  - Enable APIC interrupt line
  - Wait for interrupts
  - Disable APIC interrupt line
  - Restore old interrupt handler

```
#define INTNO 0x0B                     /* Interupt Number - See Table 1 */

void main(void)
{
 oldhandler = getvect(INTNO);           /* Save Old Interrupt Vector */
 setvect(INTNO, yourisr);               /* Set New Interrupt Vector Entry */
 outportb(0x21,(inportb(0x21) & 0xF7)); /* Un-Mask (Enable) IRQ3 */

 /* Set Card - Port to Generate Interrupts */

 /* Body of Program Goes Here */

 /* Reset Card - Port as to Stop Generating Interrupts */

 outportb(0x21,(inportb(0x21) | 0x08)); /* Mask (Disable) IRQ3 */
 setvect(INTNO, oldhandler);      /* Restore old Interrupt Vector Before Exit */
}
```

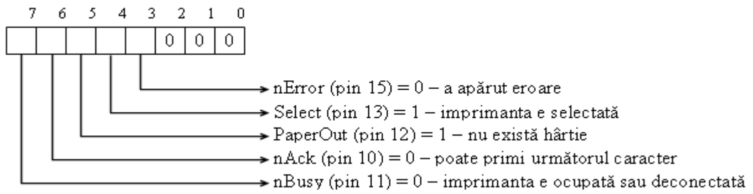
Interrupts

- Usage of interrupts
  - Device drivers
  - BIOS functions
  - OS kernel functions

Summary

Applications

- Connecting a mini-keyboard to parallel port



7 6 5 4 3 2 1 0
0 0 0
nError (pin 15) = 0 – a apărut eroare
Select (pin 13) = 1 – imprimanta e selectată
PaperOut (pin 12) = 1 – nu există hârtie
nAck (pin 10) = 0 – poate primi următorul caracter
nBusy (pin 11) = 0 – imprimanta e ocupată sau deconectată

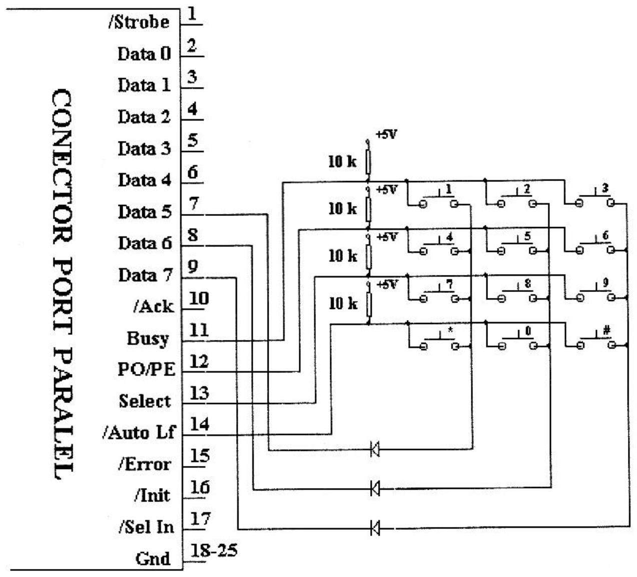


Fig.1.46. Conectarea unei minitastaturi mecanice la portul paralel - soluția 1

Applications

- Connecting a mini-keyboard to parallel port

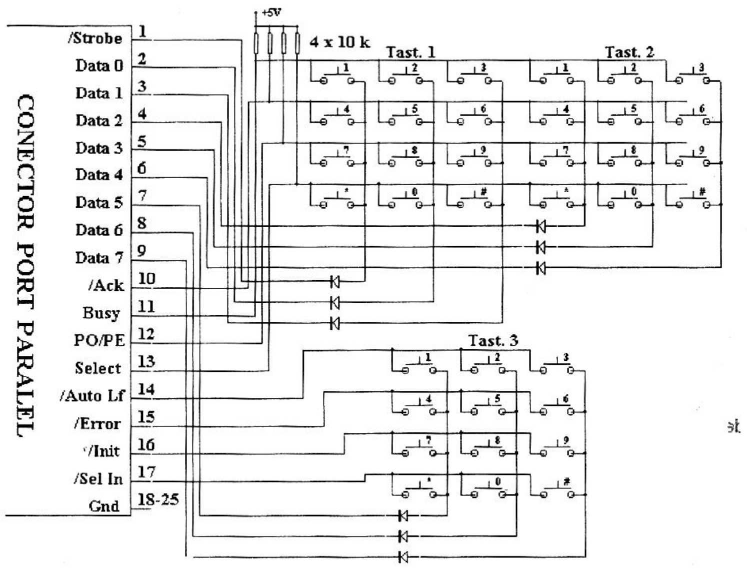
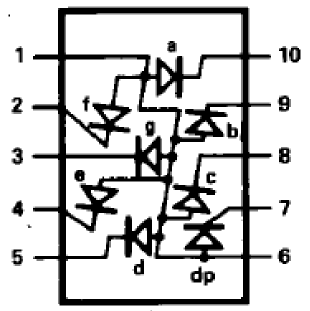
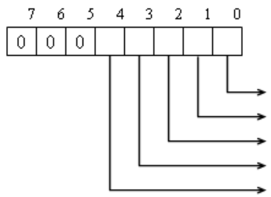


Fig.1.49. Conectarea a 3 minitastaturi mecanice la portul paralel

Applications

- Connecting segment displays to parallel port





Strobe (pin 1) = 1 – când se trimite un octet
nAuto (pin 14) = 1 – cauzează LF după CR
nInit (pin 16) = 0 – inițializează imprimanta
nSelect (pin 17) = 1 – selectează imprimanta
IRQ Enable = 1 – validează întreruperea pentru portul paralel

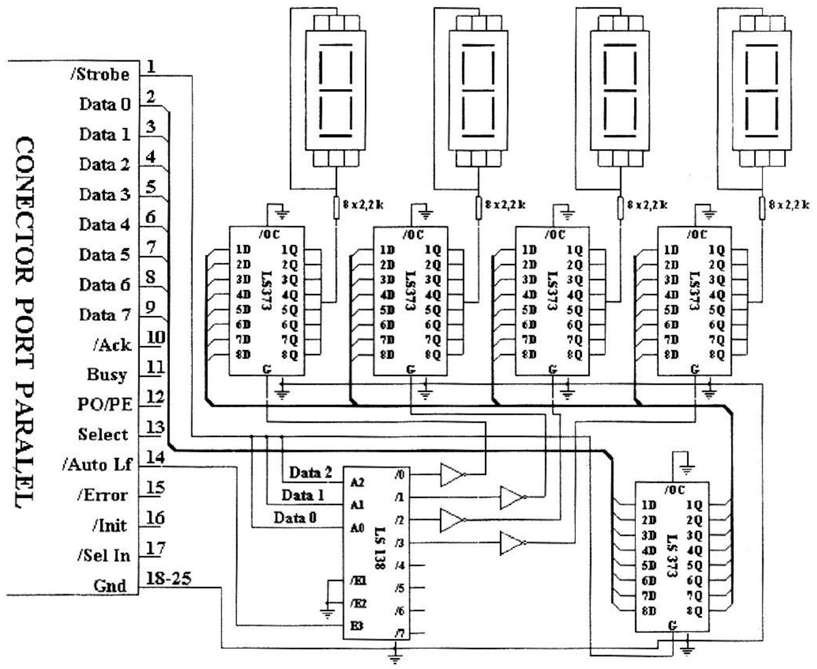


Fig.1.53. Conectarea unui modul de afişare cu segmente la portul paralel

Applications

- Connecting a LCD to the parallel port

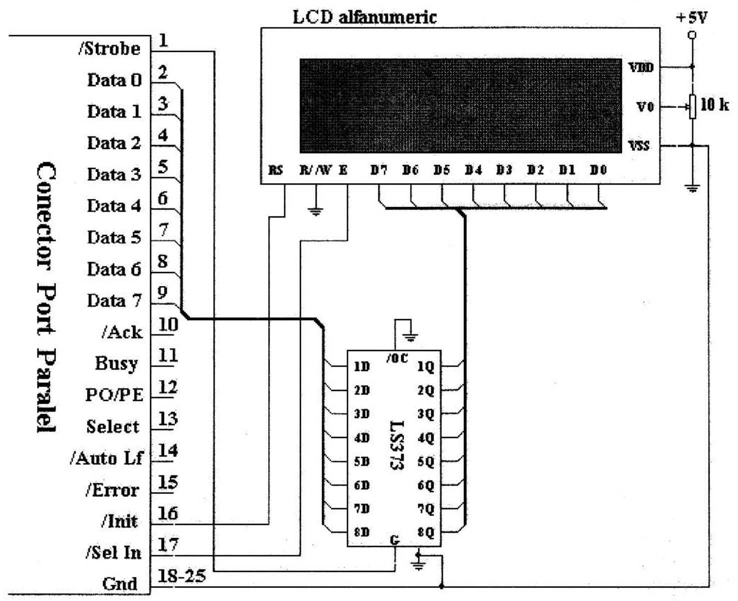
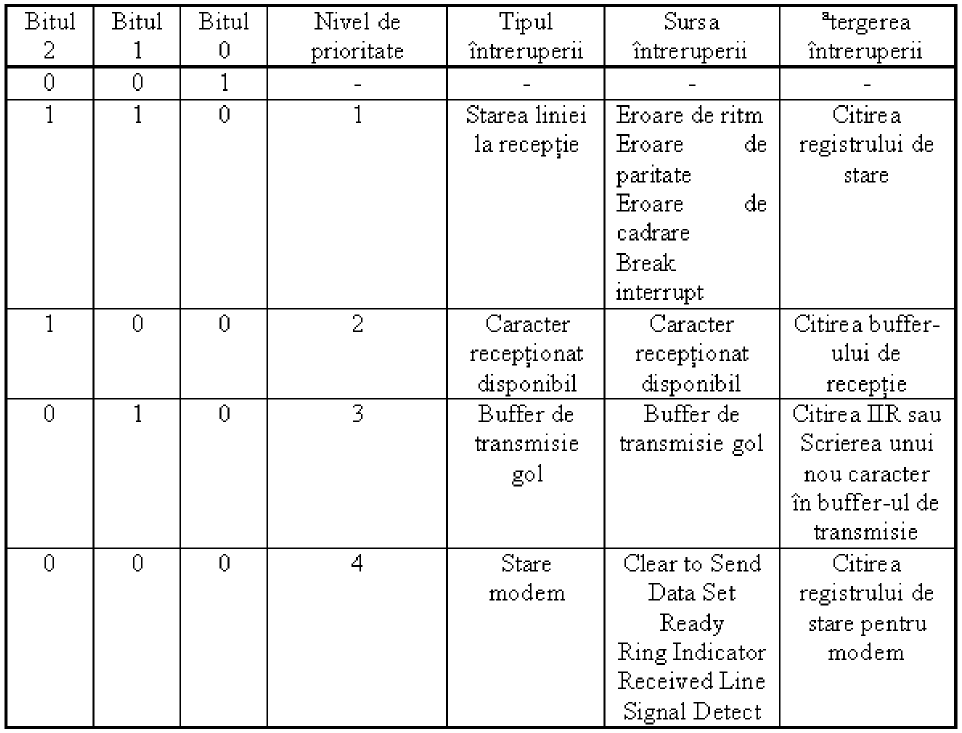


Fig.1.54. Conectarea unui modul LCD alfanumeric la portul paralel

Implementation

- Registrul de identificare a întreruperilor



Bitul 2	Bitul 1	Bitul 0	Nivel de prioritate	Tipul întreruperii	Sursa întreruperii	ªtergerea întreruperii
0	0	1	-	-	-	-
1	1	0	1	Starea liniei la recepţie	Eroare de ritm Eroare de paritate Eroare de cadrare Break interrupt	Citirea registrului de stare
1	0	0	2	Caracter recepţionat disponibil	Caracter recepţionat disponibil	Citirea buffer-ului de recepţie
0	1	0	3	Buffer de transmisie gol	Buffer de transmisie gol	Citirea IIR sau Scrierea unui nou caracter în buffer-ul de transmisie
0	0	0	4	Stare modem	Clear to Send Data Set Ready Ring Indicator Received Line Signal Detect	Citirea registrului de stare pentru modem