Homework

- Reading
  - Tokheim, Section 13-6
- Continue mp1
  - Questions?
- Labs
  - Continue labs with your assigned section

Accessing I/O Devices

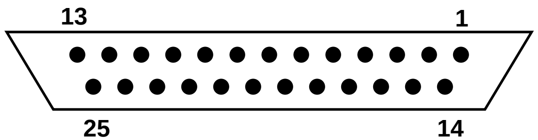
- Can’t directly access I/O devices under Unix
  - Why not?
- Can do it under Tutor
  - Why?
- Tutor allows us to learn about accessing I/O devices from “hands-on” experience

I/O Devices

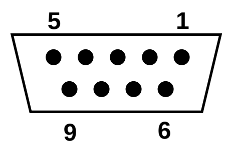
- We’ll discuss 2 types of I/O devices in detail:
  - Serial ports
  - Parallel ports
- Covering the following aspects:
  - Physical connectors
  - Overview of interface electronics
  - Handshake procedures
  - I/O addresses assigned
  - Programming procedures

Serial Ports (COM1: and COM2:)

- EIA RS-232C interface same connector as LPT1:



- COM1: a DB-9 connector on back of computer with a subset of the RS-232C signals (sufficient for async use)



- Requires a conversion cable (DB9 - DB25) to connect a PC to a standard RS-232C device such as analog modem
- “RS-232” level signals

  +3 to +15 volts is considered a logic 0

  \- 3 to - 15 volts is considered a logic 1

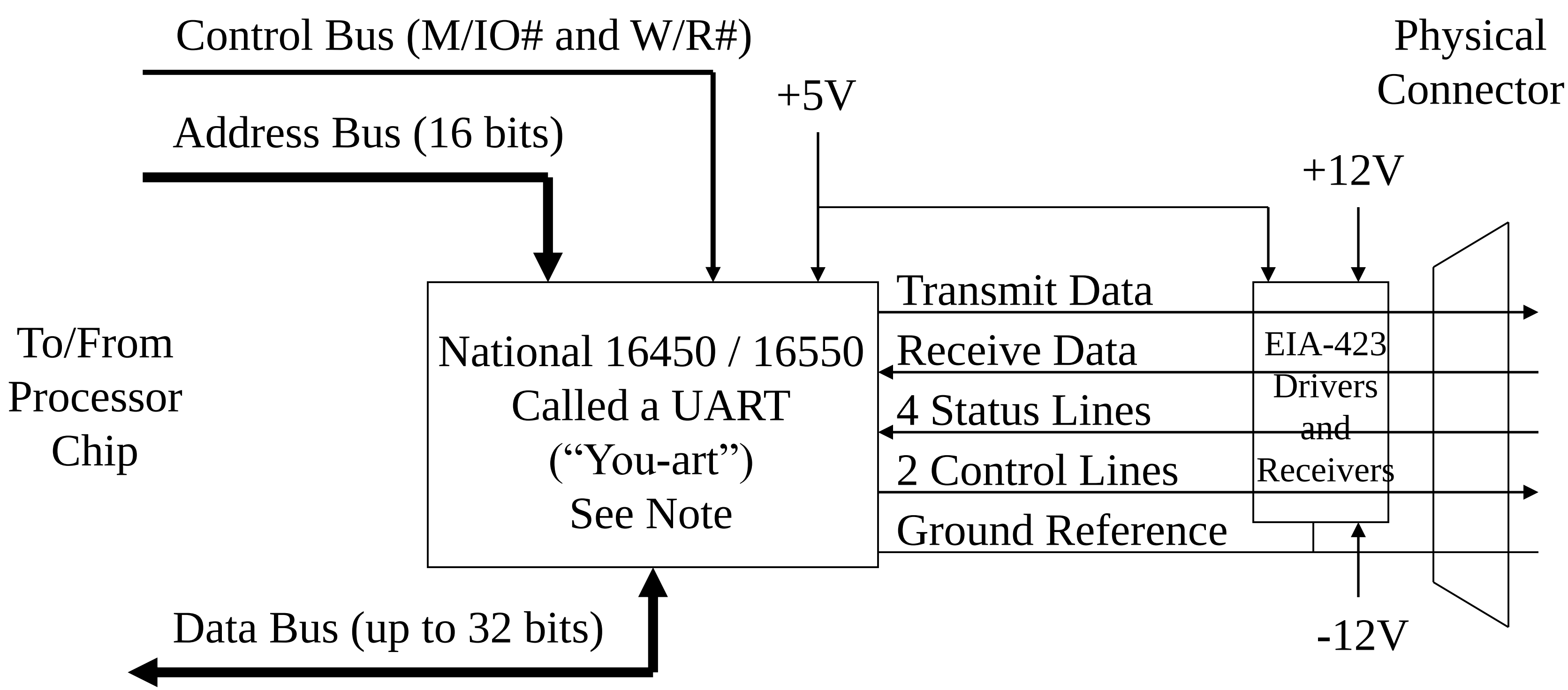
  (Note: + 12 and -12 are voltages usually used)

Serial Port

- DB9 Pin Out
  - Pin 1 Data Carrier Detect (DCD) Input
  - Pin 2 Receive Data (RXD) Input
  - Pin 3 Transmit Data (TXD) Output
  - Pin 4 Data Terminal Ready (DTR) Output
  - Pin 5 Signal Ground ---
  - Pin 6 Data Set Ready (DSR) Input
  - Pin 7 Request to Send (RTS) Output
  - Pin 8 Clear to Send (CTS) Input
  - Pin 9 Ring Indicator (RI) Input
- Single wire for sending data and single wire for receiving data plus return path (i.e., ground)
- Multiple control and status signals

Serial Port

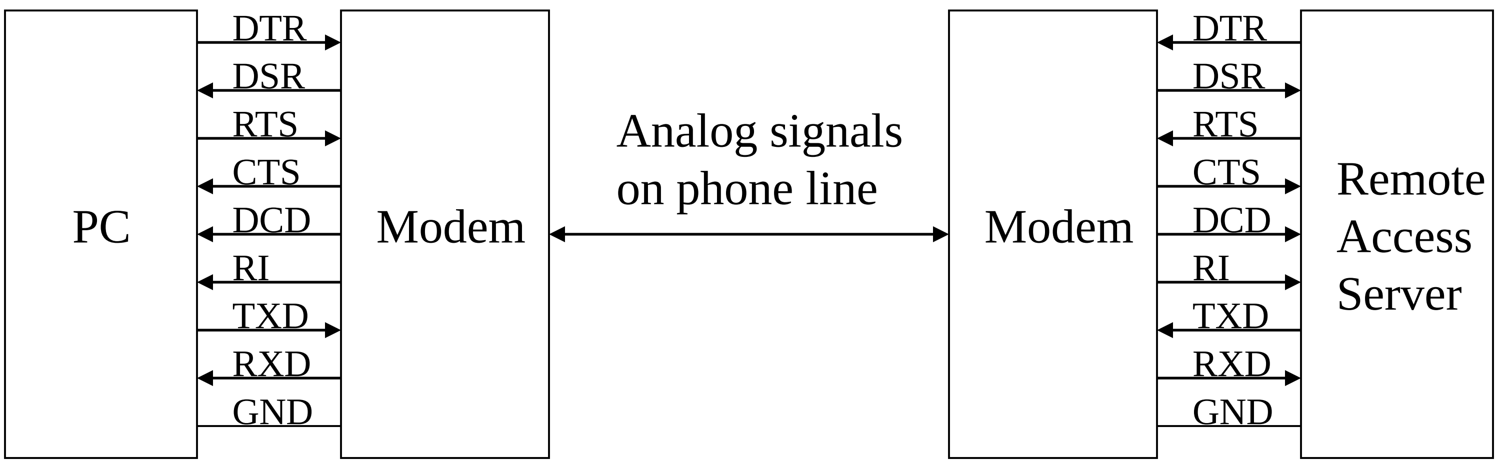
- The "inside story" on a serial port:



Note: Implemented inside a "mother board" chip today, but backward compatible

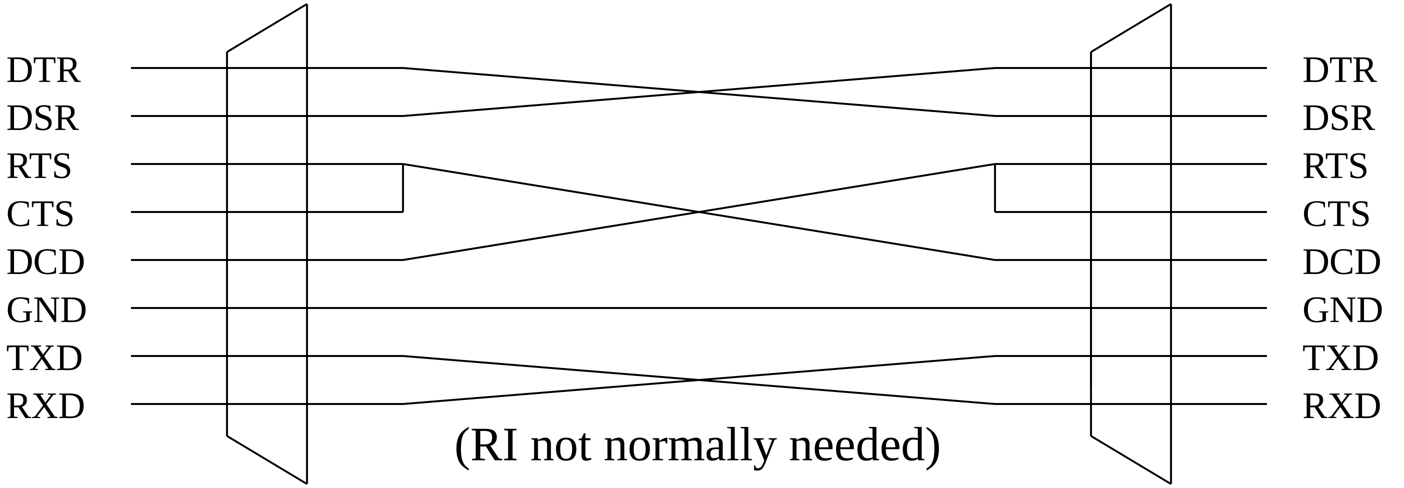
Serial Port Handshake

- Connecting PC to an access server via a pair of modems
- Control / Status Lines (two straight-through cables)
  - Data Terminal Ready indicates that PC is on and ready
  - Data Set Ready indicates that modem is on and ready
  - With Request to Send, PC tells modem to turn on its carrier
  - With Clear to Send, the modem indicates that carrier is on
  - With Data Carrier Detect, the modem indicates carrier seen
  - With Ring Indicator, modem indicates incoming call



Serial Port Handshake

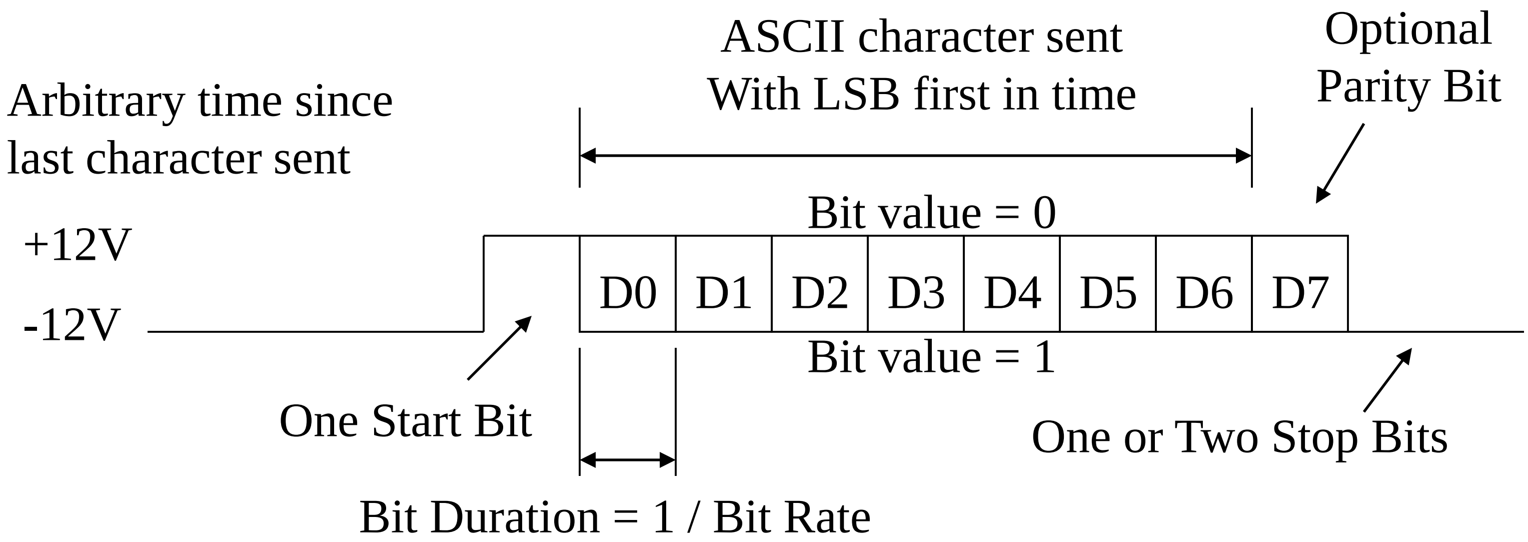
- Connecting two PCs via a NULL modem cable
  - Behaves like a pair of modems
  - Control / status lines are “cross-connected”
  - Transmit and receive data are “cross-connected”



(RI not normally needed)

Serial Port Handshake

- Bits are sent on TXD and RXD serially (one at a time)
  - Bit Rate needs to be specified
  - When the sequence starts and stops has to be specified
  - How the bits are serialized has to be specified



Accessing the Serial Port

- PC specification allows up to four serial ports
  - COM1: base address is 0x3F8
  - COM2: base address is 0x2F8
  - Each has up to eight port addresses
  - Usually use six of these addresses
    - Base: Receive buffer on read / Transmit buffer on write
    - Base+1 Interrupts and FIFO buffer
    - Base+2: Interrupt ID
    - Base+3: Line control (set up by Tutor for us)
    - Base+4: Modem control
    - Base+5: Line status
    - Base+6: Modem Status

Accessing the Serial Port

- Examples:
  - Send an ‘A’ out on COM2: (port mtip connected to)

    `ps 2f8 41` (ASCII A = 0x41)
  - And you will see:

    `ATutor>` (Character A then prompt)
  - Read a character from COM2:

    `pd 2f8`
  - And you will see:

    `02f8 00 00 c1 03 0b 00 00 00  ff ff ff ff ff ff ff ff`

Accessing the Serial Port

- Port access support for C programs
  - Can use functions specific to the PC
  - We have our own library ($pcinc/cpu.h)
- Look at example $pcex/echo.c
- Function prototypes are in cpu.h

```
void outpt(int port, unsigned char outbyte);
unsigned char inpt(int port);
```

  - Port address < 0xFFFF
  - Unsigned char is the 8-bit character
  - Example for COM2:

```
outpt(0x2F8, 0x41);
```

Accessing the Serial Port

- Don't want to use hard coded numbers!
- Look at $pcinc/serial.h for symbolic constants

```
#define COM1_BASE   0x3f8
#define COM2_BASE   0x2f8
#define UART_TX     0  /* send data */
#define UART_RX     0  /* recv data */
. . .
#define UART_LCR    3  /* line control */
#define UART_MCR    4  /* modem control */
#define UART_LSR    5  /* line status */
#define UART_MSR    6  /* modem status */
```

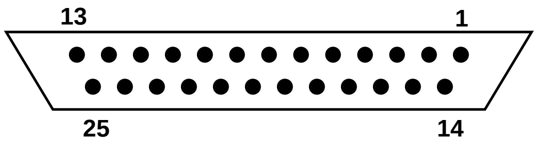
Accessing the Serial Port

- Construct addresses using symbolic constants

```
unsigned char status;
outpt(COM1_BASE + UART_TX, ‘A’);
status = inpt(COM1_BASE + UART_LSR);
```

Parallel Port (LPT1:)

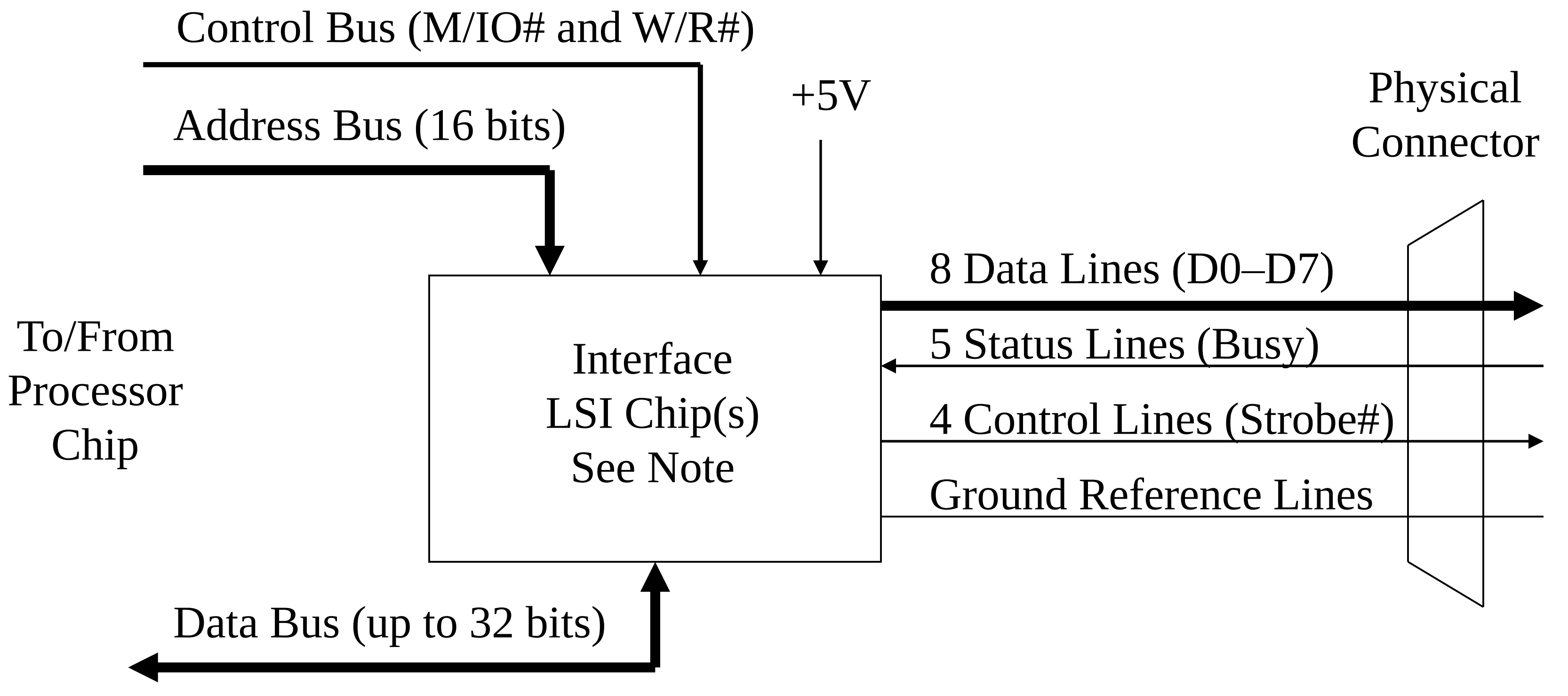
- LPT1: a DB25 connector on back of computer



  - Data appears on pins 2-9
  - Control/Status on pins 1 & 10-17
  - Pins 18-25 are ground
  - "TTL" level signals
    - 0-1volts is considered low and a logic 0
    - 3-5volts is considered high and a logic 1
- Very simple interface to understand and use
  - Provides 8 bits of output (one byte at a time)
  - No transformation of data
  - Simple handshake protocol

Parallel Port

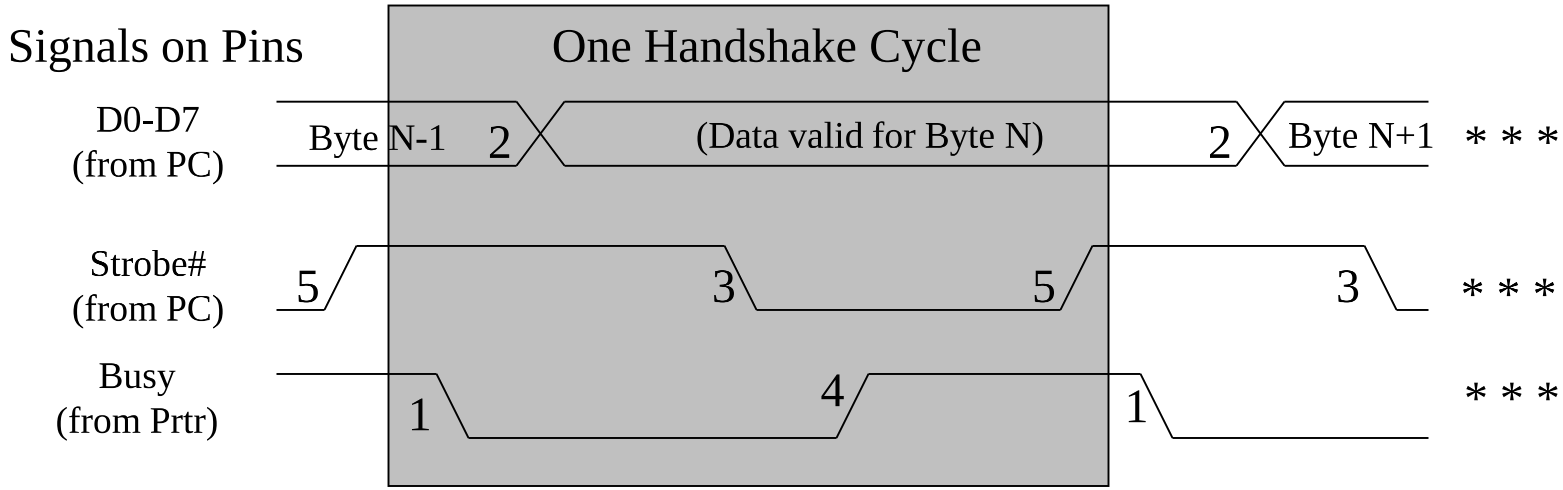
- The “inside story” on a parallel port:



Note: Implemented inside a “mother board” chip today, but backward compatible

Parallel Port Printer Handshake

- Data byte sent to parallel data port (all 8 bits at once)
  1. Printer indicates ready for next data byte (Busy = 0)
  2. PC sets up data bits on data lines D0-D7
  3. PC tells printer that data is ready Strobe# = 0
  4. Printer acknowledges or “acks” (Busy = 1) and takes data
  5. PC sets Strobe# =1 to be ready for next cycle



Accessing Parallel Port

- IBM defines up to three parallel port addresses
- We will use “LPT1:” with 0x378 as base address
  - Base used to send data to printer (D0-D7)
  - Base+1 used to get status byte (with MSB = Busy# signal)
  - Base+2 used for control (with LSB = Strobe signal)
- Can access parallel port using Tutor ‘ps’ command
  - ps 378 FF to set all data output bits to ones
  - ps 378 0   to set all data output bits to zeros

Write

0x378	D7	D6	D5	D4	D3	D2	D1	D0
0x379	- - -							
0x37a			D	IQ	SI	IN#	AF	ST

Read

0x378	D7	D6	D5	D4	D3	D2	D1	D0
0x379	Bsy#	Ack#	PE	SL	Err#	IRQ		
0x37a			D	IQ	SI	IN#	AF	ST

Accessing the Parallel Port

- Examples:
  - Note that status port address is “read only”

```
pd 378
 0378 00 7F E0 . . .
ps 378 55
pd 378
 0378 55 7F E0 . . . {has effect on 378}
ps 379 66
pd 378
 0378 55 7F E0 . . . {no effect on 379}
```

Accessing Parallel Port

- Port access support for C programs
  - Can use functions specific to the PC
  - We have our own library ($pcinc/cpu.h)
- Look at example $pcex/testlp.c
- Function prototypes are in $pcinc/cpu.h

```
void outpt(int port, unsigned char outbyte);
unsigned char inpt(int port);
```

  - Port address < 0xFFFF
  - Unsigned char is the 8-bit character
  - Example:

```
outpt(0x378, 0xFF);
```

Accessing the Parallel Port

- Don’t want to use hard coded numbers!
- Look at $pcinc/lp.h for symbolic constants

```
#define LPT1_BASE     0x378
#define LP_DATA       0    /* 8 bits of data */
#define LP_STATUS     1    /* in: status bits */
#define LP_CNTRL      2    /* in, out: control bits*/
```

- Construct addresses using symbolic constants

```
unsigned char cntrl, status;
outpt(LPT1_BASE + LP_CNTRL, cntrl);
status = inpt(LPT1_BASE + LP_STATUS);
```