Processor System Block Diagram

- Processor
- Memory
- Peripheral
- Peripheral

Address Bus
Data Bus
Simple Bus Timing

**Read Cycle**

- **R/W**
- **Enable**
- **Addr**
- **Data**

**Write Cycle**

- **R/W**
- **Enable**
- **Addr**
- **Data**
Strobe vs. Handshake

**Strobe**
- Req
- Data

**Handshake**
- Req
- Ack
- Data
1982: The IBM PC/XT
The ISA Bus: Memory Read
# The ISA Bus: Memory Write

<table>
<thead>
<tr>
<th></th>
<th>C1</th>
<th>C2</th>
<th>Wait</th>
<th>C3</th>
<th>C4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clk</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Addr</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BALE</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>MEMW</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>IOCHRDY</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Data</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The PC/104 Form Factor: ISA Lives

Embedded System Legos. Stack ‘em and go.
Memory-Mapped I/O

- To a processor, everything is memory.
- Peripherals appear as magical memory locations.
- Status registers: when read, report state of peripheral
- Control registers: when written, change state of peripheral
### Typical Peripheral: PC Parallel Port

#### At Standard TTL Levels

<table>
<thead>
<tr>
<th>Signal Name</th>
<th>Adapter Pin Number</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strobe</td>
<td>1</td>
</tr>
<tr>
<td>+Data Bit 0</td>
<td>2</td>
</tr>
<tr>
<td>+Data Bit 1</td>
<td>3</td>
</tr>
<tr>
<td>+Data Bit 2</td>
<td>4</td>
</tr>
<tr>
<td>+Data Bit 3</td>
<td>5</td>
</tr>
<tr>
<td>+Data Bit 4</td>
<td>6</td>
</tr>
<tr>
<td>+Data Bit 5</td>
<td>7</td>
</tr>
<tr>
<td>+Data Bit 6</td>
<td>8</td>
</tr>
<tr>
<td>+Data Bit 7</td>
<td>9</td>
</tr>
<tr>
<td>Acknowledge</td>
<td>10</td>
</tr>
<tr>
<td>+Busy</td>
<td>11</td>
</tr>
<tr>
<td>+Paper End</td>
<td>12</td>
</tr>
<tr>
<td>+Select</td>
<td>13</td>
</tr>
<tr>
<td>-Auto Feed</td>
<td>14</td>
</tr>
<tr>
<td>-Error</td>
<td>15</td>
</tr>
<tr>
<td>-Initialize</td>
<td>16</td>
</tr>
<tr>
<td>-Select Input</td>
<td>17</td>
</tr>
<tr>
<td>Ground</td>
<td>18-25</td>
</tr>
</tbody>
</table>
### Parallel Port Registers

<table>
<thead>
<tr>
<th>D7</th>
<th>D6</th>
<th>D5</th>
<th>D4</th>
<th>D3</th>
<th>D2</th>
<th>D1</th>
<th>D0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Busy</td>
<td>Ack</td>
<td>Paper</td>
<td>Sel</td>
<td>Err</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Sel</th>
<th>Init</th>
<th>Auto</th>
<th>Strobe</th>
</tr>
</thead>
</table>

- 0x378
- 0x379
- 0x37A

1. Write Data
2. Assert Strobe
3. Wait for Busy to clear
4. Wait for Acknowledge
#define DATA 0x378
#define STATUS 0x379
#define CONTROL 0x37A

#define NBSY 0x80
#define NACK 0x40
#define OUT 0x20
#define SEL 0x10
#define NERR 0x08
#define STROBE 0x01

#define INVERT (NBSY | NACK | SEL | NERR)
#define MASK (NBSY | NACK | OUT | SEL | NERR)
#define NOT_READY(x) ((inb(x)^INVERT)&MASK)

void write_single_character(char c) {
    while (NOT_READY(STATUS)) ;
    outb(DATA, c);
    outb(CONTROL, control | STROBE); /* Assert STROBE */
    outb(CONTROL, control ); /* Clear STROBE */
}
The Parallel Port Schematic
Interrupts and Polling

Two ways to get data from a peripheral:
- Polling: “Are we there yet?”
- Interrupts: Ringing Telephone
Interrupts

Basic idea:
1. Peripheral asserts a processor’s interrupt input
2. Processor temporarily transfers control to interrupt service routine
3. ISR gathers data from peripheral and acknowledges interrupt
4. ISR returns control to previously-executing program
Many Different Interrupts

What’s a processor to do?
Many Different Interrupts

What’s a processor to do?
ISR polls all potential interrupt sources, then dispatches handler.
Prioritizes incoming requests & notifies processor
ISR reads 8-bit interrupt vector number of winner
IBM PC/AT uses 8259, became a standard