Processor System Block Diagram

- Processor
- Address Bus
- Data Bus
- Memory
- Peripheral
- Peripheral
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

![Diagram showing the timing of Clk, Addr, BALE, MEMW, IOCHRDY, and Data signals during memory write cycles on the ISA bus.](image-url)
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
# 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>
<th>Address</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>
<td>0x378</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0x379</td>
</tr>
</tbody>
</table>

Options:
- Sel
- Init
- Auto
- Strobe

<table>
<thead>
<tr>
<th>0x37A</th>
</tr>
</thead>
</table>

**Procedure**

1. Write Data
2. Assert Strobe
3. Wait for Busy to clear
4. Wait for Acknowledge

![Waveforms]

**Waveforms**

- Strobe
- Busy
- Ack
- Data
A Parallel Port Driver

```c
#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 */
}
```
Interrupts and Polling

Two ways to get data from a peripheral:

- Polling: “Are we there yet?”
- Interrupts: Ringing Telephone
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?
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: two 8259s; became standard