Basic Processor Architecture

Controller

Operation → Result → Latch → Read, Write → Latch → Registers → Address Reg. → Memory

Shared Bus
Typical Processor System

Diagram showing a typical processor system with connections to memory and peripherals, with labels for R/W, Enable, Address, and Data signals.
Simple Bus Timing

Read Cycle

Write Cycle

R/W

Enable

Addr

Data

R/W

Enable

Addr

Data
Strobe vs. Handshake

**Strobe**
- Req
- Data

**Handshake**
- Req
- Ack
- Data
1982: The IBM PC
The ISA Bus: Memory Read

- **CLK**: Clock signal.
- **Addr**: Address signal.
- **BALE**: Bus Active Low Enable signal.
- **MEMR**: Memory Read signal.
- **IOCHRDY**: I/O Channel Ready signal.
- **Data**: Data output signal.

The timing diagram illustrates the sequence of events during a memory read operation on the ISA bus.
The PC/104 Form Factor: ISA Lives

3.8in

Embedded System Legos. Stack ’em and go.
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</th>
</tr>
</thead>
<tbody>
<tr>
<td>-Strobe</td>
<td>1</td>
</tr>
<tr>
<td>E</td>
<td>P</td>
</tr>
<tr>
<td>X</td>
<td>A</td>
</tr>
<tr>
<td>T</td>
<td>R</td>
</tr>
<tr>
<td>E</td>
<td>A</td>
</tr>
<tr>
<td>R</td>
<td>L</td>
</tr>
<tr>
<td>N</td>
<td>L</td>
</tr>
<tr>
<td>A</td>
<td>E</td>
</tr>
<tr>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>-Acknowledge</td>
<td>10</td>
</tr>
<tr>
<td>D</td>
<td>A</td>
</tr>
<tr>
<td>E</td>
<td>D</td>
</tr>
<tr>
<td>V</td>
<td>A</td>
</tr>
<tr>
<td>I</td>
<td>P</td>
</tr>
<tr>
<td>C</td>
<td>T</td>
</tr>
<tr>
<td>E</td>
<td>E</td>
</tr>
<tr>
<td>-Select Input</td>
<td>R</td>
</tr>
<tr>
<td>-Select</td>
<td></td>
</tr>
<tr>
<td>Ground</td>
<td>18-25</td>
</tr>
</tbody>
</table>

Centronics Handshake

- nStrobe
- Busy
- nAck
- Data
## 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>0x378</th>
</tr>
</thead>
<tbody>
<tr>
<td>Busy</td>
<td>Ack</td>
<td>Paper</td>
<td>Sel</td>
<td>Err</td>
<td>Sel</td>
<td>Init</td>
<td>Auto</td>
<td>Strobe</td>
</tr>
</tbody>
</table>

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 */
}
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?
Interrupt Polling

Processor receives interrupt
ISR polls all potential interrupt sources
Prioritizes incoming requests & notifies processor
ISR reads 8-bit interrupt vector number of winner
IBM PC/AT: two 8259s; became standard