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 diagram illustrates the timing for a memory read operation on the ISA bus. It shows the following signals:

- **Clk**: Clock signal.
- **Addr**: Address signal.
- **BALE**: Bus Alignment signal.
- **MEMR**: Memory Read signal.
- **IOCHRDY**: I/O Channel Ready signal.
- **Data**: Data signal.

The sequence is divided into five phases:

- **C1**: Initial setup.
- **C2**: Address and control signals settle.
- **Wait**: Awaiting for the data to be stable.
- **C3**: Data is transferred.
- **C4**: Completion of the transfer.

The diagram visualizes the timing and interaction of these signals during a memory read operation.
The ISA Bus: Memory Write

The diagram illustrates the timing sequence for a memory write operation on the ISA bus. The sequence includes phases C1, C2, Wait, C3, and C4.

- **Clk**: The clock signal is active during the C1 and C2 phases.
- **Addr**: The address is valid during the C1 phase.
- **BALE**: The bus activate line is asserted during the C1 and C2 phases.
- **MEMR**: The memory enable signal is active during the C1 phase.
- **IOCHRDY**: The input/output chip ready signal is active during the C1 and C2 phases.
- **Data**: The data is stable during the C2 and C3 phases.

The diagram shows the synchronization of these signals across the different phases, highlighting the timing and sequence of operations in a memory write cycle on the ISA bus.
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>E</td>
<td>2</td>
</tr>
<tr>
<td>+Data Bit 0</td>
<td>P</td>
</tr>
<tr>
<td>X</td>
<td>3</td>
</tr>
<tr>
<td>+Data Bit 1</td>
<td>A</td>
</tr>
<tr>
<td>T</td>
<td>4</td>
</tr>
<tr>
<td>+Data Bit 2</td>
<td>R</td>
</tr>
<tr>
<td>E</td>
<td>5</td>
</tr>
<tr>
<td>+Data Bit 3</td>
<td>A</td>
</tr>
<tr>
<td>R</td>
<td>6</td>
</tr>
<tr>
<td>+Data Bit 4</td>
<td>L</td>
</tr>
<tr>
<td>N</td>
<td>7</td>
</tr>
<tr>
<td>+Data Bit 5</td>
<td>L</td>
</tr>
<tr>
<td>A</td>
<td>8</td>
</tr>
<tr>
<td>+Data Bit 6</td>
<td>E</td>
</tr>
<tr>
<td>L</td>
<td>9</td>
</tr>
<tr>
<td>+Data Bit 7</td>
<td>L</td>
</tr>
<tr>
<td>Ack</td>
<td>10</td>
</tr>
<tr>
<td>-Acknowledge</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>11</td>
</tr>
<tr>
<td>+Busy</td>
<td>A</td>
</tr>
<tr>
<td>E</td>
<td>12</td>
</tr>
<tr>
<td>+Paper End</td>
<td>D</td>
</tr>
<tr>
<td>V</td>
<td>13</td>
</tr>
<tr>
<td>+Select</td>
<td>A</td>
</tr>
<tr>
<td>I</td>
<td>14</td>
</tr>
<tr>
<td>-Auto Feed</td>
<td>P</td>
</tr>
<tr>
<td>C</td>
<td>15</td>
</tr>
<tr>
<td>-Error</td>
<td>T</td>
</tr>
<tr>
<td>E</td>
<td>16</td>
</tr>
<tr>
<td>-Initialize</td>
<td>E</td>
</tr>
<tr>
<td>Select Input</td>
<td>17</td>
</tr>
<tr>
<td>Ground</td>
<td>18-25</td>
</tr>
</tbody>
</table>

Strobe

Busy

Ack

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></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>

-  **0x378**
-  **0x379**
-  **0x37A**

1. Write Data
2. Assert Strobe
3. Wait for Busy to clear
4. Wait for Acknowledge
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 */
}
```
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: two 8259s; became standard