Input/Output

Without I/O (Input/Output), there can be no interaction with the CPU, rendering it useless.

The CPU Interacts with devices

CPU to Device Communication

To enable CPUs to communicate with such devices we use a number of techniques:

- **Memory Mapped Ports**: Devices are connected to the address and data busses and appear as memory location.
- **Special Instruction**: The devices are connected to special I/O busses and accessed via special I/O instructions.
- **Interrupt Processing**: Devices communicate with CPU using interrupt I/O lines

68000

The 68000 family of CPUs use the memory mapping technique to access device hardware.

The 68332 is a **Microcontroller**:

- It has extra features designed to simplify interfacing with external devices.

Example

![Diagram of CPU and device communication](image)
Robot Memory Map

This table indicates the memory allocated to various devices on the robot. This does not include the MCU register map.

<table>
<thead>
<tr>
<th>Location</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>000000</td>
<td>Exception Vector Table</td>
</tr>
<tr>
<td>000100</td>
<td>USER Interrupt Vector Table</td>
</tr>
<tr>
<td>000140</td>
<td>TPU Interrupt Vector Table</td>
</tr>
<tr>
<td>000400</td>
<td>FREE USER RAM</td>
</tr>
<tr>
<td>01e000</td>
<td>User STACK (top of)</td>
</tr>
<tr>
<td>01f000</td>
<td>System STACK (top of)</td>
</tr>
<tr>
<td>020000</td>
<td>Robot EPROM</td>
</tr>
<tr>
<td>030000</td>
<td>LED Array Port</td>
</tr>
<tr>
<td>030800</td>
<td>LCD Registers</td>
</tr>
<tr>
<td>031000</td>
<td>DIP Switch Latch Port</td>
</tr>
<tr>
<td>040000</td>
<td>68332 MCU RAM</td>
</tr>
<tr>
<td>050000</td>
<td>RAM Expansion</td>
</tr>
<tr>
<td>070000</td>
<td>End</td>
</tr>
</tbody>
</table>

Memory Mapping

Memory mapping involves hardware devices occupying memory locations, and being accessed through those memory locations.

Example: 8 LEDs (Light Emitting Diodes)

Memory Mapping

Using the memory mapped scheme, this devices will occupy a certain area of memory (in this just a single byte at $030000$ for example).

When we write a value to $030000$
- That value is seen by the device
- In this case causes certain LEDs to light

The execution of the instruction:
- move.b #36,$030000  *00100100

Should cause the corresponding LEDs to light up, as shown in the diagram.
Memory as a Device?

- Memory itself acts as a device.
- Most of the memory map is mapped to:
  - RAM space
  - ROM space
- All device chips and memory chips are connected to the same:
  - Control bus
  - Data bus
  - Address bus.

CPU to Device Connections

- Such devices are connected to the CPU via the:
  - Address bus [a0...a23]
  - Data bus [d0...d15]
  - R/W line
  - SIZ0 and SIZ1 lines
  - Other bus arbitration lines

CPU - RAM Read & Write

MC68332
Memory Mapped Device Interface

- RAM devices must respond both to **read** and **write** request.
- The LEDs need only respond to **write** requests.
- Devices are communicated with using:
  - **read/write cycles**
  - The CPU and the devices perform **handshaking** during these cycles.
  - A well defined protocol of communication using sequences of signal assertions on different lines.

Example: READ cycle

1. CPU places address on the address bus
2. CPU specifies transfer size using SIZ0 and SIZ1
3. CPU asserts R/W -> read cycle
4. Device responds by placing the required data on the data bus.
5. Data bus is read by the CPU
6. The actual sequence is more complicated and involves bus arbitration.
7. A specific timing sequence using other lines.
8. To the CPU, memory mapped devices all appear the same.
9. i.e. RAM behaves just like the LEDs.
10. Reading from write only devices (like the LEDs) initiates a read cycle but the LEDs will not respond to the read request.
11. Invalid data is read
Problem:

- Which device should respond to each read/write request?
- Answer:
  - It depends on the address.
  - Different devices are mapped to different addresses.
  - A device should respond only if the address on the address bus is within the address space of the devices.
  - The system designer must ensure this using address decoding logic.

Address Space Implementation

- Memory mapped devices are implemented by mapping their internal register space to an assigned address space.
- Most such devices have a number of pins which facilitate this:
  - CS: Chip Select
  - RS: Register Select

Example: A Device with 8 Registers

The device will have 3 register select pins (usually):

![Diagram of a device with 8 registers and 3 register select pins]

If we wish to select register 3 we assert the RS0 and RS1 lines and set CS high. R/W is used to select a read or write operation.

Implement a Memory Map

1. Choose device address space: $ffff00-ffff07
2. Design the CS circuit:

![Diagram of address decoding logic]

The CS signal is high only when A23-A8=1 AND A7-A3=0.

1111 1111 1111 1111 0000 0XXX
Implement a Memory Map

3. Map the address lines to the register selects:

<table>
<thead>
<tr>
<th>d0</th>
<th>d1</th>
<th>d2</th>
<th>d3</th>
<th>d4</th>
<th>d5</th>
<th>d6</th>
<th>d7</th>
</tr>
</thead>
<tbody>
<tr>
<td>DB0</td>
<td>RS0</td>
<td>a0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB1</td>
<td>RS1</td>
<td>a1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2</td>
<td>RS2</td>
<td>a2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>CS</td>
<td>CS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB7</td>
<td>R/W</td>
<td>R/W</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

8-bit Device

4. Finally the R/W line is tied to the CPU R/W pin and the DB0-7 lines connected to the data bus d0-7 lines.

Implement a Memory Map

- If we execute the instruction
  ```
  move.b #10,$ffff03
  ```
- The address placed on the address bus is $ffff03 which pushed CS for this device high.
- 10 is placed on the data-bus
- A write-cycle begins and register 3 is selected.

The write cycle will clear R/W causing the device to read the contents of the data-bus which are then stored, by the device, in register 3.

$ffff03 is known as a port

- Memory locations = ports on the RAM

68332 Architecture

- The MC68332 has 3 main components:
  - CPU32
    - The 32bit CPU
  - TPU
    - Time processor unit
  - SIM
    - System integration module
System Integration Module

- The SIM provides the following functionality:
  - **PIT**: periodic interval timer
  - **Clock Synthesiser**: generates a range of clock signals
  - **System Configuration & Protection**: monitors bus and handles reset configuration
  - **External Bus Interface**: allows connection to other CPUs and devices
  - **Test Facilities**
  - **Chip Select**: manages processor address space and device selection

68332 Chip Selects

The 68332 has 12 programmable CS lines which can be mapped to address spaces from 2k - 1Mb in size.

This eliminates the need for address decoding circuitry.

Each chip select has a number of parameters:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>CS asserted for upper or lower byte access</td>
</tr>
<tr>
<td>R/W</td>
<td>CS asserted for read or write access</td>
</tr>
<tr>
<td>SPACE</td>
<td>Controls access privilege to address space</td>
</tr>
<tr>
<td>DSACK</td>
<td>Specifies memory/device access speed</td>
</tr>
<tr>
<td>BLKSZ</td>
<td>Size of memory block (2Kb → 1Mb.)</td>
</tr>
<tr>
<td>BASE</td>
<td>Base address of block</td>
</tr>
</tbody>
</table>

Note: minimum block size = 2kb → further decoding necessary for devices with few registers.

Example: LED latch at $30000

- **CS4 assignment**:
  - Byte = Lower
  - R/W = Write
  - SPACE = User Privilege
  - DSACK = Slow
  - BLKSZ = 2Kb
  - BASE = $30000

Now CS4 may be used in place of the decoding hardware.

Liquid Crystal Diode Displays

Motorola have recently introduced the DragonBall™ 68328 processor to facilitate interfacing with LCD displays:

68328 = 68332 - TPU + LCD Driver

This is a low power device designed for mobile phones and PDAs (Personal Digital Assistant).
Example: 3Com PalmPilot

Case Study: Hitachi 44780 LCD

- The **LCD** (Liquid Crystal Diode) used in the robot is a good example of a general purpose LCD device.
- **Features:**
  - Hitachi 44780 compatible controller
  - 16×2 character array
  - 5×8 dot characters
  - Full ASCII character set + many extra symbols
  - 8 programmable characters
  - 128 bytes of **Display Data RAM (DD RAM)**
  - 64 bytes of **Character Generator RAM (CG RAM)**

**Case Study: Hitachi 44780 LCD**

**DD RAM**

The 128 bytes of DD RAM is split into 2 rows of 64 bytes.

- Line1: $00 - $3f
- Line2: $40 - $7f

At any one time, only 16 characters are visible on a given line.

Usually, you will only use address $00-$0f and $40-$4f.

The display is a window on a larger region.
Using the LCD

- There are 2 register selected by the Register Select (RS) line.
  - RS = 0 selects control register
  - RS = 1 selects data register
- The chip select associated with the LCD has a base address of $30800.
- RS is tied to address bus A0 line
  - When an odd address is accessed, RS = 1, and the data register is selected.
  - move b, d0, $30800 * move d0 into control register
  - move b, d0, $30801 * move d0 into data register
- Moving a value to the control register is interpreted as a command.

Command Summary (part one)

<table>
<thead>
<tr>
<th>Command</th>
<th>RS</th>
<th>Address</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Clear Display</td>
<td>0</td>
<td>00000001</td>
<td>1.64ms</td>
</tr>
<tr>
<td>HOME CURSOR</td>
<td>0</td>
<td>0000010x</td>
<td>1.64ms</td>
</tr>
<tr>
<td>ENTRY SET MODE</td>
<td>0</td>
<td>0000101s</td>
<td>40us</td>
</tr>
</tbody>
</table>

Sets cursor move direction and specifies whether or not to shift display.

- i=1 -> increments DD RAM address by 1 after each DD RAM read/write
- i=0 -> decrements DD RAM address by 1 after each DD RAM read/write
- s=1 -> display scrolls in direction specified by i when cursor is at edge of window.

Command Summary (part two)

<table>
<thead>
<tr>
<th>Command</th>
<th>RS</th>
<th>Address</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>ON/OFF CONTROL</td>
<td>0</td>
<td>0001dcb</td>
<td>40us</td>
</tr>
<tr>
<td>CURSOR/SHIFT</td>
<td>0</td>
<td>0001srxx</td>
<td>40us</td>
</tr>
</tbody>
</table>

Turn display off/on, turn cursor off/on, blink character at cursor off/on.

- b=1 -> display on
- c=1 -> cursor on
- d=1 -> blink character at cursor position

Move cursor or scroll display without changing display data RAM.

- r=0 -> left
- r=1 -> right
- s=0 -> move cursor
- s=1 -> scroll display

Command Summary (part three)

<table>
<thead>
<tr>
<th>Command</th>
<th>RS</th>
<th>Address</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>DD RAM ADDRESS SET</td>
<td>0</td>
<td>1aaaaaa</td>
<td>40us</td>
</tr>
<tr>
<td>CG RAM ADDRESS SET</td>
<td>0</td>
<td>01aaaaaa</td>
<td>40us</td>
</tr>
</tbody>
</table>

To read or write display data. Data written to or read from the LCD after this command will be to/from DD RAM.

aaaaaa = 7-bit DD RAM address

To read or write custom characters. Data written to or read from the LCD after this command will be to/from CG RAM.

aaaaaa = 6-bit CG RAM address
Command Summary (part four)

Write Data (DD/CG)  RS=1 aaaaaaaaa  40/120ns.

Data is written to the current cursor position and DD/CG
RAM address (depending on the last address set
command). RAM address incremented/decremented by 1,
as set in entry mode set command.

aaaaaaaaa = 8-bit character code