General Microprocessor System Architecture

- Three parts:
  1. CPU ($\mu$Processor)
  2. Memory
  3. I/O

- Connected to each other by a BUS.

$\mu$Processor:

- Controls memory and I/O through bus.

- Three main tasks:
  1. data transfer: MOV, Push*, Pop*, IN, OUT.
  2. arithmetic and logic: ADD, SUB, MUL*, DIV*, AND, OR, XOR, NOT, NEG, shift, rotate.
  3. program flow: Branch, Jump, Trap, Loop. (possibly based on flags ZERO, SIGN, CARRY, OVERFLOW, PARITY).

* Depends on architecture.
• program is stored in memory as binary data.

• data widths are:
  – byte (8-bits) (octet in network world)
  – word (16-bits) *
  – double word (32-bits) (a.k.a. long) *

* Depends on architecture.

Memory:

• Each location is typically 1 byte of binary data.

• Each memory element (byte) has an address, usually specified in hexadecimal notation.

• Memory size chart:

  1K  $2^{10}$  1,024 bytes
  1M  $2^{20}$  1,048,576 bytes
  1G  $2^{30}$  1,073,741,824 bytes

• ex: $64K = 64 \times 2^{10} = 65536$ bytes.

• $64K = 2^{16}$: need 16 address lines.
• (See text for MSDOS memory map.) HW fixes interrupt vectors.

Bus:

• a common group of traces or wires that interconnect components.

• Typically three:
  1. Address
  2. Data
  3. Control.

• Busses can be multiplexed to save pins/wires.
Address bus:

- selects a location in memory or I/O space for reading or writing.

- $N$ address lines can access $2^N$ locations.
  - 8086/88, $N = 20$, $2^{20}$, 1M byte.
  - 80286/386, 68000, $N = 24$, $2^{24}$, 16M byte.
  - 80386DX/486, 68020, $N = 32$, $2^{32}$, 4G byte.
  - Pentium II, $N = 36$, $2^{36}$, 64G byte.

Data bus:

- transfers information between the µP and memory or I/O.

- data transfers vary in size:
  - 8088, 68008, 8 bits
  - 8086, 80286, some 386, 68000, 68010, 16 bits
  - 386DX, 486, 68020, Pentium..., 32 bits
  - Pentium..., 64 bits*

* Fetches are to cache.
Control bus:

- In most ix86 systems, these four control signals are found:
  
<table>
<thead>
<tr>
<th>Signal</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MRDC</td>
<td>Memory Read Control</td>
</tr>
<tr>
<td>MWTC</td>
<td>Memory Write Control</td>
</tr>
<tr>
<td>IORC</td>
<td>I/O Read Control</td>
</tr>
<tr>
<td>IOWC</td>
<td>I/O Write Control</td>
</tr>
</tbody>
</table>

  Note: All signals are active low.

- Example read cycle:
  1. \(\mu P\) puts address on address bus.
  2. drops MRDC to cause memory to place data on data bus.
  3. \(\mu P\) reads data from data bus.

Number Systems.

- Review of binary, decimal and hexadecimal numbers. (see pages 28-34).

- Binary Coded Hexadecimal (nibbles).

<table>
<thead>
<tr>
<th>Hex</th>
<th>Dec</th>
<th>BCH</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0000</td>
<td>7fe</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0001</td>
<td>0111 1111 1110</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>0010</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>0011</td>
<td>3a9</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>0100</td>
<td>0011 1010 1001</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>0101</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>0110</td>
<td>6b4c</td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>0111</td>
<td>0110 1011 0100 1100</td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>1000</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>1001</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>10</td>
<td>1010</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>11</td>
<td>1011</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>12</td>
<td>1100</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>13</td>
<td>1101</td>
<td></td>
</tr>
<tr>
<td>E</td>
<td>14</td>
<td>1110</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td>15</td>
<td>1111</td>
<td></td>
</tr>
</tbody>
</table>
- Hex conversion

\[
\begin{array}{c|c|c|c|c}
7 & 2^7 & 128 & 0 \\
6 & 2^6 & 64 & 1 \\
5 & 2^5 & 32 & 1 \\
4 & 2^4 & 16 & 1 \\
3 & 2^3 & 8 & 1 \\
2 & 2^2 & 4 & 0 \\
1 & 2^1 & 2 & 1 \\
0 & 2^0 & 1 & 1 \\
\end{array}
\]

- 123_{10} = 0111 \ 1011 = 7B_{16}

- Hex addition.

\[
\begin{array}{c|c|c|c|c|c|c}
7EBD & 7 & 14 & 11 & 13 \\
4AC & 4 & 10 & 12 \\
\hline 
8369 & 7 & 18 & 21 & 25 \\
\hline 
& 16 + 2 & 16 + 5 & 16 + 9 \\
& 7 + 1 & 2 + 1 & 5 + 1 & 9 \\
& 8 & 3 & 6 & 9 \\
\end{array}
\]

- Representation of negative numbers.

\begin{itemize}
\item Defn: Radix means number base
\end{itemize}

\begin{itemize}
\item decimal \quad \text{Radix} = 10
\item binary \quad \text{Radix} = 2
\item hexadecimal \quad \text{Radix} = 16
\item octal \quad \text{Radix} = 8
\end{itemize}

- "Radix -1 complement" - each digit of the number is subtracted from the radix.

\begin{itemize}
\item Ex. 1's complement of 01001100₂
\end{itemize}

\[
\begin{array}{c|c|c|c|c|c|c|c|c}
\text{radix (2) - 1} & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
\text{number} & 01001100 & \hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c|c}
& 10110011 \\
\end{array}
\]

- Ex. 15's complement of 4AB₁₆

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c|c}
\text{radix (16) - 1} & F & F & F \\
\text{number} & 4 & A & B & \hline
\end{array}
\]

\[
\begin{array}{c|c|c|c|c|c|c|c|c|c|c}
& B & 5 & 4 \\
\end{array}
\]

- Radix-1 complement is not used (two zeros!), except floating point.
• Radix compliment is used and computed as
  \[ \text{radix compliment} = (\text{radix-1 compliemt}) + 1. \]

  - Ex. 2's complement of \(01001100_2\)

    \[
    \begin{array}{c|c|c}
    \text{radix (2)} & \text{01001100} & 11111111 \\
    \text{number} & \text{01001100} & \hline \\
    \text{+} & 1 & \text{10110011} \\
    \text{=} & 10110100 & \\
    \end{array}
    \]

Data types.

• integers (signed/unsigned)
  1. byte
  2. word
  3. double

• Binary coded decimal. nibble stores 0-9. Funky instructions. (who cares?)

• Floating point. Deferred.

• ASCII (EBCDIC). Characters and strings.
Bytes (signed or unsigned)

<table>
<thead>
<tr>
<th>Bit</th>
<th>7</th>
<th>6</th>
<th>5</th>
<th>4</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
</table>

- In a signed byte, high order bit is sign and carries a weight of -128.
  
  **unsigned:** 00H-FFH 0 – 255  
  **signed:** 00H-7FH 0 – 127  
  80H-FFH  -128 – -1

Words (signed or unsigned)

- formed from two bytes.

- “Byte Sex”

“Little Endian” (Intel)

- Least significant byte stored in lower-numbered memory location

- Consistent with BIT ordering.

- Ex: AC43H

<table>
<thead>
<tr>
<th>2002H</th>
<th>2001H</th>
<th>AC</th>
<th>High order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>2000H</td>
<td>43</td>
<td></td>
<td>Low order byte</td>
</tr>
<tr>
<td>1FFFH</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>1FFFH</th>
<th>2000H</th>
<th>2001H</th>
<th>2002H</th>
</tr>
</thead>
<tbody>
<tr>
<td>43</td>
<td>AC</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

- Needs byte flipping in network code.
“Big Endian” (Motorola, *Network*)

- Most significant byte stored in lower-numbered memory location.

- When incrementing through memory, highest (most significant) byte is first (like writing a number).

<table>
<thead>
<tr>
<th>2002H</th>
<th>AC</th>
<th>Low order byte</th>
</tr>
</thead>
<tbody>
<tr>
<td>2001H</td>
<td>43</td>
<td>High order byte</td>
</tr>
<tr>
<td>2000H</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1FFFH</td>
<td>2000H</td>
<td>2001H</td>
</tr>
<tr>
<td></td>
<td>AC</td>
<td>43</td>
</tr>
</tbody>
</table>

- Double (long) words – four bytes.

Memory Organization

- Memory devices are arranged in bytes of 8-bits (modulo parity/ECC).

- \(\mu P\) may have 8, 16, 32 or 64 data lines.
- What happens if we add 32 more data lines?

Questions: 10, 23, 24, 25, 30, 31, 59, 60, 70