The x86
Microprocessors

Introduction
Numbering and Coding Systems

- Human beings use the **decimal system** (base 10)
  - Decimal digits: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9

- Computer systems use the **binary system** (base 2)
  - Binary digits: 0, 1

- **Bit** (Binary Digit): 0 or 1

- The **Octal system** (base 8) has 8 digits
  - 0, 1, 2, 3, 4, 5, 6, 7

- The **hexadecimal system** (base 16) has 16 digits
  - 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F
## Decimal, Binary, and Hex

<table>
<thead>
<tr>
<th>Decimal</th>
<th>Binary</th>
<th>Hexadecimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>0100</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>0101</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>0110</td>
<td>6</td>
</tr>
<tr>
<td>7</td>
<td>0111</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>9</td>
<td>1001</td>
<td>9</td>
</tr>
<tr>
<td>10</td>
<td>1010</td>
<td>A</td>
</tr>
<tr>
<td>11</td>
<td>1011</td>
<td>B</td>
</tr>
<tr>
<td>12</td>
<td>1100</td>
<td>C</td>
</tr>
<tr>
<td>13</td>
<td>1101</td>
<td>D</td>
</tr>
<tr>
<td>14</td>
<td>1110</td>
<td>E</td>
</tr>
<tr>
<td>15</td>
<td>1111</td>
<td>F</td>
</tr>
</tbody>
</table>
Example: Convert \((35)_{10}\) to binary

- Dividing the decimal number by 2 repeatedly
- Keeping track of the remainders
- This process continues until the quotient becomes zero.
- The remainders are then written in reverse order to obtain the binary.

<table>
<thead>
<tr>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>35/2</td>
<td>17</td>
</tr>
<tr>
<td>17/2</td>
<td>8</td>
</tr>
<tr>
<td>8/2</td>
<td>4</td>
</tr>
<tr>
<td>4/2</td>
<td>2</td>
</tr>
<tr>
<td>2/2</td>
<td>1</td>
</tr>
<tr>
<td>1/2</td>
<td>0</td>
</tr>
</tbody>
</table>

\[(35)_{10} = (100011)_2\]
Example: Convert \((0.35)_{10}\) to binary

\[
\begin{array}{ccc}
0.35 \times 2 &=& 0.7 & + & 0 \text{ (MSB)} \\
0.7 \times 2 &=& 0.4 & + & 1 \\
0.4 \times 2 &=& 0.8 & + & 0 \\
0.8 \times 2 &=& 0.6 & + & 1 \\
0.6 \times 2 &=& 0.2 & + & 1 \\
0.2 \times 2 &=& 0.4 & + & 0 \text{ (LSB)} \\
\end{array}
\]

\((0.35)_{10} = (0.010110)_2\)
Converting from Binary to Decimal

- There is a weight associated with each digit position
- Multiply the weight of each digit position by the content of position
- Sum the weight of all digits
- Example: Convert (315)\text{10} from decimal to decimal !!!

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>1</td>
<td>5</td>
</tr>
</tbody>
</table>

\[
\begin{array}{ccc}
3 \times 10^2 & 1 \times 10^1 & 5 \times 10^0 \\
3 \times 100 & 1 \times 10 & 5 \times 1 \\
3 \times 100 + 1 \times 10 + 5 \times 1 &= 315
\end{array}
\]
Example: Convert \((110101)_2\) to Decimal

\[
\begin{array}{cccccc}
1 & 1 & 0 & 1 & 0 & 1 \\
1 \times 2^5 & 1 \times 2^4 & 0 \times 2^3 & 1 \times 2^2 & 0 \times 2^1 & 1 \times 2^0 \\
\end{array}
\]

\[
1 \times 32 + 1 \times 16 + 0 \times 8 + 1 \times 4 + 0 \times 2 + 1 \times 1 \\
(110101)_2 = (53)_{10}
\]
### Example: Convert \((0.110101)_2\) to Decimal

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 \times 2^{-1}</td>
<td>1 \times 2^{-2}</td>
<td>0 \times 2^{-3}</td>
<td>1 \times 2^{-4}</td>
<td>0 \times 2^{-5}</td>
<td>1 \times 2^{-6}</td>
</tr>
</tbody>
</table>

\[
\frac{1}{2} + \frac{1}{4} + 0/8 + \frac{1}{16} + 0/32 + \frac{1}{64} = (0.828125)_{10}
\]

\((0.110101)_2 = (0.828125)_{10}\)
Converting between Binary and Hex

- Converting from binary to hex
  - Start from the right and group 4 bits at a time
  - Replace each 4-bit binary number with its hex equivalent
  - Example: Represent binary 100111110101 in hex
    - \[1001\quad 1111\quad 0101\]
    - \[9\quad F\quad 5\]
    - Therefore, \((100111110101)_2 = (9F5)_{16}\) hexadecimal.

- Converting from hex to binary
  - Replace each hex digit with its 4-bit binary equivalent
  - Example: Convert hex 29B to binary
    - \[2\quad 9\quad B\]
    - \[0010\quad 1001\quad 1011\]
    - After dropping the leading zeros, \((29B)_{16} = (1010011011)_{10}\)
Converting from Decimal to Hex

- Converting from decimal to hex could be approached in two ways:
  1. Convert to binary first and then convert to hex.
     - Example: Convert \((45)_{10}\) to hex
     - \((45)_{10} = (101101)_{2} = (0010 1101)_{2} = (2D)_{16}\)
  2. Convert directly from decimal to hex by the method of repeated division, keeping track of the remainders.
     - Example: Convert \((45)_{10}\) to hex
       - \[
       \begin{array}{c|c|c}
       \text{Quotient} & \text{Remainder} & \\
       \hline
       45/16 & 2 & 13 \text{ (hex D)} \\
       2/16 & 0 & 2 \\
       \end{array}
       \]
       - \((45)_{10} = (101101)_{2} = (0010 1101)_{2} = (2D)_{16}\)
       - Example: Convert decimal \((629)_{10}\) to hexadecimal.
         - \((629)_{10} = (0010 0111 0101)_{2} = (275)_{16}\)
Converting from Hex to Decimal

- Conversion from hex to decimal can also be approached in two ways:
  1. Convert from hex to binary and then to decimal.
     - Example: Convert hexadecimal \((6B2)_{16}\) to decimal.
     - \((6B2)_{16} = (0110 1011 0010)_{2} =\)
     - \((1 \times 2^9 + 1 \times 2^8 + 0 \times 2^7 + 1 \times 2^6 + 0 \times 2^5 + 1 \times 2^4 + 0 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0)_{10} = (1714)_{10}\)
  2. Convert directly from hex to decimal by summing the weight of all digits.
     - Example: Convert hexadecimal \((6B2)_{16}\) to decimal.
     - \((6B2)_{16} = (6 \times 16^2 + B \times 16^1 + 2 \times 16^0)_{10} = (1714)_{10}\)
Addition of Binary and Hex numbers

Add the following unsigned binary numbers

$$(1010100)_2 + (1100100)_2 = ?$$

\[
\begin{array}{c@{}c@{}c@{}c@{}c@{}c@{}c}
  & 0 & 0 & 0 & 1 & 1 & 0 \\
 0 & 0 & 1 & 0 & 0 & 1 & 0 \\
 0 & 1 & 0 & 0 & 1 & 1 & 0 \\
 1 & 0 & 0 & 1 & 1 & 1 & 1 \\
\end{array}
\]

$$(84)_{10} + (100)_{10} = (184)_{10}$$
Signed Addition

Add the following signed binary numbers

\((01010100)_2 + (01100100)_2 = ?\)

\[
\begin{array}{c}
01000100 \\
01010100 \\
01100100 \\
\hline
10111000
\end{array}
\]

2’s complement 10111000 of is 01001000 (-72)
Unsigned Subtraction

- There is no separate circuitry for subtractors.
- Instead, adders are used in conjunction with 2's complement circuitry to perform subtraction.
- To implement "x - y", the computer takes the 2's complement of y and adds it to x.

Example: Subtract the following unsigned numbers (100 - 80 = 20)
  - 1100100 - 1010000 (note that each number has the same number of digits)
    - 1100100 + 0110000 = 10010100
      - The result is +ve
      - +(0010100)₂ = +(20)

Example: Subtract the following unsigned numbers (80 - 100 = -20)
  - 1010000 - 1100100
    - 1010000 + 0011100 = 1101100 (Note: no extra digit)
      - The result is -ve
      - -(0010100)₂ = -(20)
Signed Subtraction

- Example: Subtract the following signed numbers (100 – 80 = 20)
  - (+100) – (+80)
  - 01100100 – 01010000
  - 01100100 + 10110000 = 00010100 ➔ 00010100 ➔ +20
- Any carry out of the sign-bit position is discarded
- The negative result is automatically in the 2’s complement form
- Example: Subtract the following unsigned numbers (80 - 100 = -20)
  - (+80) – (+100)
  - 01010000 - 01100100
  - 01010000 + 10011100 = 11101100 ➔ -00010100 ➔ -20
ASCII Code

- **ASCII**: American Standard Code for Information Interchange
- The great advantage of this system is that it is used by most computers, so that information can be shared among computers.
- The ASCII code assigns binary patterns for:
  - numbers 0 to 9,
  - uppercase (capital) letters A to Z
  - lowercase letters a to z
  - Another characters `{ " ' !@$
  - many control codes
- The ASCII system uses a total of 7 bits to represent each code.
- Example
  - 100 0001 ➔ uppercase letter "A"
  - 110 0001 ➔ lowercase letter "a".
- Often, a zero is placed in the most significant bit position to make it an 8-bit code.
Note That

- The pattern of ASCII codes was designed to allow for easy manipulation of ASCII data.
- Digits \((0)_{10}\) through \((9)_{10}\) are represented by ASCII codes 30H or \((0011\ 0000)_{2}\) through 39H or \((0011\ 1001)_{2}\).
- This enables a program to easily convert ASCII to decimal by masking off the "3" in the upper nibble.
- There is a relationship between the uppercase and lowercase letters.
- Uppercase letters are represented by ASCII codes 41H or \((0100\ 0001)_{2}\) through 5AH or \((0101\ 1010)_{2}\) while lowercase letters are represented by ASCII codes 61H or \((0110\ 0001)_{2}\) through 7AH or \((0111\ 1010)_{2}\).
- Looking at the binary code, the only bit that is different between uppercase "A" and lowercase "a" is bit 5.
- Therefore conversion between uppercase and lowercase is as simple as changing bit 5 of the ASCII code.
Some Important Terminology

- **Bit**: a binary digit that can have the value 0 or 1.
- **Byte**: 8 bits.
- **Nibble**: half a byte or 4 bits.
- **Word**: two bytes, or 16 bits.
- **Kilobyte**: $2^{10}$ bytes, which is 1024 bytes.
  - The abbreviation K is often used (356K bytes of data).
- **Megabyte**: $2^{20}$ bytes, which is a little over 1 million bytes; it is exactly 1,048,576.
- **Gigabyte**: $2^{30}$ bytes (over 1 billion)
- **Terabyte**: $2^{40}$ bytes (over 1 trillion).
- **RAM**: random access memory (sometimes called *read/write memory*)
  - The data in RAM is lost when the computer is turned off (*volatile memory*).
- **ROM**: read-only memory
  - The information in ROM is permanent, cannot be changed by the user, and is not lost when the power is turned off (*nonvolatile memory*).
The internal working of every computer can be broken down into three parts:

1. **CPU** (central processing unit): to execute (process) information stored in memory.
2. **Memory**: to store programs and data
3. **I/O** (input/output) devices: to provide a means of communicating with the CPU.

The CPU is connected to memory and I/O through strips of wire called a **bus**. The bus inside a computer carries information from place to place.
Computer Buses
Address Bus, Data Bus, and Control Bus.

- **Address bus,**
  - The number of address buses for a CPU determines the number of locations with which CPU can communicate.
  - The number of locations is always equal to $2^x$, where $x$ is the number of address lines, regardless of the size of the data bus.
  - A CPU with 16 address lines can provide a total of 65,536 ($2^{16}$) or 64K bytes of addressable memory. Each location can have a maximum of 1 byte of data.

- **Data bus:** used to carry information in and out of a CPU,
  - The more data buses available, the better the CPU.
  - More data buses mean a more expensive CPU and computer.
  - The average size of data buses in CPUs varies between 8 and 64.
  - Data buses are bidirectional, since the CPU must use them either to receive or to send data.
  - The processing power of a computer is related to the size of its buses, since an 8-bit bus can send out one byte a time, but a 16-bit bus can send out 2 bytes at a time, which is twice as fast.

- **Control bus:** used to provide read or write signals to the device to indicate if the CPU is asking for information or sending it information.
Internal Organization of Computers

- For the CPU to process information, the data must be stored in RAM or ROM.
- The CPU gets the information to be processed, first from RAM (or ROM).
- Only if it is not there does the CPU seek it from a mass storage device such as a disk, and then it transfers the information to RAM.
- For this reason, RAM and ROM are sometimes referred to as *primary memory* and disks are called *secondary memory*.
Internal Block Diagram of a CPU

Decoder: interpret the instruction fetched into the CPU

PC (IP): point to the address of the next instruction to be executed

Arithmetic/logic unit: perform arithmetic functions such as add, subtract, multiply, and divide, and logic functions such as AND, OR, and NOT

Store information temporarily

CPU

Fetches instructions from memory
Executes fetched instructions
The x86 Microprocessors
Processor Architecture Impacting Factors

Markets

Processor Architecture

Technology

Applications
Evolution of Intel’s Processors

Moore’s law: The number of transistors per integrated circuit would double every 18 months
# Evolution of Intel’s Processors

<table>
<thead>
<tr>
<th>Product</th>
<th>4004</th>
<th>8008</th>
<th>8080</th>
<th>8085</th>
<th>8086</th>
<th>8088</th>
<th>80286</th>
<th>80386</th>
<th>80486</th>
<th>Pentium</th>
<th>P. Pro</th>
</tr>
</thead>
<tbody>
<tr>
<td>MHz</td>
<td>.108</td>
<td>.5-8</td>
<td>2-3</td>
<td>3-8</td>
<td>5-10</td>
<td>5-8</td>
<td>6-16</td>
<td>16-33</td>
<td>25-50</td>
<td>60, 66</td>
<td>150</td>
</tr>
<tr>
<td># Pins</td>
<td>18</td>
<td>18</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>40</td>
<td>68</td>
<td>132</td>
<td>168</td>
<td>273</td>
<td>387</td>
</tr>
<tr>
<td># Tran K</td>
<td>2.9</td>
<td>3</td>
<td>4.5</td>
<td>6.5</td>
<td>29</td>
<td>29</td>
<td>130</td>
<td>275</td>
<td>1,200</td>
<td>3,100</td>
<td>5,500</td>
</tr>
<tr>
<td>Memory</td>
<td>4K</td>
<td>16K</td>
<td>64K</td>
<td>64K</td>
<td>1M</td>
<td>1M</td>
<td>16M</td>
<td>4G</td>
<td>4G</td>
<td>4G</td>
<td>64G</td>
</tr>
<tr>
<td>Int.</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>16</td>
<td>16</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>Ext.</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>16</td>
<td>8</td>
<td>16</td>
<td>32</td>
<td>32</td>
<td>64</td>
<td>64</td>
</tr>
<tr>
<td>Add bus</td>
<td>4</td>
<td>8</td>
<td>16</td>
<td>16</td>
<td>20</td>
<td>20</td>
<td>24</td>
<td>32</td>
<td>32</td>
<td>32</td>
<td>36</td>
</tr>
<tr>
<td>Data type</td>
<td>4</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td>8, 16</td>
<td>8, 16</td>
<td>8, 16</td>
<td>8, 16</td>
<td>8, 16, 32</td>
<td>8, 16, 32</td>
<td>8, 16, 32</td>
</tr>
</tbody>
</table>
Evolution from 8080/8085 to 8086

- In 1978, Intel Corp. introduced a 16-bit microprocessor called the 8086.
- 8086 was a major improvement over 8080/8085 in several ways:
  1. The 8086's capacity of 1 megabyte of memory exceeded the 8080/8085's capability of handling a maximum of 64K bytes.
  2. The 8080/8085 was an 8-bit system (work on only 8 bits of data at a time). Data larger than 8 bits had to be broken into 8-bit pieces to be processed by the CPU.
  3. The 8086 was a pipelined processor, as opposed to the non-pipelined 8080/8085.
    - In a system with pipelining, the data and address buses are busy transferring data while the CPU is processing information, thereby increasing the effective processing power of the microprocessor.
Evolution from 8086 to 8088

- The 8086 is a microprocessor with a 16-bit data bus internally and externally
  - All registers are 16 bits wide and there is a 16-bit data bus to transfer data in and out of the CPU.
- At that time all peripherals were designed around an 8-bit microprocessor instead of 16-bit external data bus.
- A printed circuit board with a 16-bit data bus was much more expensive.
  - Therefore, Intel came out with the 8088 version.
  - 8088 is identical to the 8086 as far as programming is concerned, but externally it has an 8-bit data bus instead of a 16-bit bus.
  - 8088 has the same memory capacity, 1 megabyte.
Other Microprocessors: 80286, 80386, 80486

- Intel introduced the 80286 in 1982.
- Its features included
  - **16-bit internal and external** data buses
  - **24 address** lines, which give 16 megabytes of memory ($2^{24} = 16$ megabytes); and most significantly,
  - **Virtual memory**
    - The 80286 can operate in one of two modes: real mode or protected mode.
    - Virtual memory is a way of fooling the microprocessor into thinking that it has access to an almost unlimited amount of memory by swapping data between disk storage and RAM.
    - **Real mode** is simply a faster 8088/8086 with the same maximum of 1 megabyte of memory.
    - **Protected mode** allows for 16M of memory but is also capable of protecting the operating system and programs from accidental or deliberate destruction by a user.
32-bit Microprocessor

- In 1985 Intel introduced the 80386 (sometimes called 80386DX), internally and externally a **32-bit microprocessor** with a 32-bit address bus.
- It is capable of handling physical memory of up to **4 gigabytes** ($2^{32}$).
- Virtual memory was increased to 64 terabytes ($2^{46}$).
- Intel introduced numeric data processing chips, called math coprocessors, such as the 8087, 80287, and 80387.
- Later Intel introduced the 386SX, which is internally identical to the 80386 but has a 16-bit external data bus and a 24-bit address bus which gives a capacity of 16 megabytes ($2^{24}$) of memory.
- This makes the 386SX system much **cheaper**.
- With the introduction of the 486 in 1989, Intel put a greatly enhanced version of the 386 and the math coprocessor on a single chip plus additional features such as **cache memory**.
  - Cache memory is static RAM with a very fast access time.
Pentium

- In 1992 Intel introduced the Pentium.
- The Pentium had speeds of 60 and 66 MHz, but new design features made its processing speed twice that of the 66-MHz 80486.
- Although the Pentium has a **64-bit data bus**, its registers are 32-bit and it has a **32-bit address bus** capable of addressing 4 gigabytes of memory.
- In 1995 Intel introduced the Pentium Pro, the sixth generation of the x86 family.
- Pentium Pro is an enhanced version of the Pentium.
Pipelining

- There are two ways to make the CPU process information faster:
  1. Increase the working frequency or
  2. Change the internal architecture of the CPU

- The first option is technology dependent, meaning that the designer must use whatever technology is available at the time, with consideration for cost.

- The technology determines the working frequency, power consumption, and the number of transistors packed into a single-chip microprocessor.

- The second option for improving the processing power of the CPU has to do with the internal working of the CPU.

- In the 8085 microprocessor, the CPU could either fetch or execute at a given time.

Fetch Execute Fetch Execute
Pipelining (Cont.)

The idea of pipelining in its simplest form is to allow the CPU to fetch and execute at the same time.

- Non pipelined
- 2-stage pipeline
- 3-stage pipeline
Pipelining in the 8088/86

- Intel implemented the concept of pipelining in the 8088/86 by splitting the internal structure of the microprocessor into two sections:
  - the execution unit (EU) and
  - the bus interface unit (BIU).

- These two sections work simultaneously. The BIU accesses memory and peripherals while the EU executes instructions previously fetched.

- This works only if the BIU keeps ahead of the EU; thus the BIU of the 8088/86 has a buffer, or queue.

- The buffer is 4 bytes long in the 8088 and 6 bytes in the 8086.

- If any instruction takes too long to execute, the queue is filled to its maximum capacity and the buses will sit idle.

- The BIU fetches a new instruction whenever the queue has room for 2 bytes in the 6-byte 8086 queue, and for 1 byte in the 4-byte 8088 queue.

- In some circumstances, the microprocessor must flush out the queue.
Internal Block Diagram of the 8088/86 CPU

EXECUTION UNIT (EU)    BUS INTERFACE UNIT (BIU)

\[
\begin{array}{c|c}
    AH & AL \\
    BH & BL \\
    CH & CL \\
    DH & DL \\
    BP & \\
    DI & \\
    SI & \\
    SP & \\
\end{array}
\]

\[
\begin{array}{c}
    CS \\
    ES \\
    SS \\
    DS \\
    IP \\
\end{array}
\]

multiplexed bus

operands

ALU

flags

address generation and bus control

instruction queue

The 80x86 Microprocessors 1.35 Assembly Language
8088/86 Registers

- Registers: used to store information temporarily.
- Information: data (8/16-bit) to be processed or the address of data.

<table>
<thead>
<tr>
<th>Category</th>
<th>Bits</th>
<th>Register Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>General</td>
<td>16</td>
<td>AX (accumulator), BX (base addressing), CX (counter), DX (point to data in I/O operations)</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>AH, AL, BH, BL, CH, CL, DH, DL</td>
</tr>
<tr>
<td>Pointer</td>
<td>16</td>
<td>SP (stack pointer), BP (base pointer)</td>
</tr>
<tr>
<td>Index</td>
<td>16</td>
<td>SI (source index), DI (destination index)</td>
</tr>
<tr>
<td>Segment</td>
<td>16</td>
<td>CS (code segment), DS (data segment), SS (stack segment), ES (extra segment)</td>
</tr>
<tr>
<td>Instruction</td>
<td>16</td>
<td>IP (instruction pointer)</td>
</tr>
<tr>
<td>Flag</td>
<td>16</td>
<td>FR (flag register)</td>
</tr>
</tbody>
</table>
8-bit/16-bit Registers

8-bit registers

16-bit registers

The 80x86 Microprocessors

Assembly Language