Bits, Bytes, and Words

- Instead, computers represent numbers in digital format.
  - A number is a sequence of bits, each of which can have the values 0 or 1. Voltages above a threshold are 1, below are 0.
  - A collection of 8 bits is known as a byte. A byte has its own address, and the addresses n and n + 1 refer to consecutive bytes.

Representing Integers

- So far we’ve ignored how the computer represents numbers. I assumed that numbers could be transmitted over wires, but let’s talk about numbers.
- One possibility is to use the wire’s voltage to represent the number, e.g. 5 volts = 17, etc. This is an analog representation.
- Problems:
  1. It’s dangerous to compute Bill Gate’s worth.
  2. You can increase your salary by scuffing your feet and touching the payroll computer.

Bits, Bytes, and Words...

- A word is a larger collection of bits, usually 2 or 4 bytes. On a 32-bit computer a word is 4 bytes or 32 bits. The word size is usually determined by the number of bits in a register.
**Positional Number Systems . . .**

- In general, the sequence of digits:
  \[ d_{n-1} \cdots d_2 d_1 d_0 \]
represents the value:
  \[ d_{n-1} * r^{n-1} + \cdots + d_0 * r^0. \]
r is called the radix (or the base).
- For decimal numbers the radix is 10.
- The radix for a number is usually indicated as a subscript, e.g. \( 101_2 \) is base 2, \( 101_8 \) is base 8, etc.

**Positional Number Systems . . .**

- Unlike English text, numbers are read from right-to-left. The least-significant digit is on the right, and represents the radix to the zeroth power, while the most-significant digit is on the left and represents the radix to the highest power.
- Computers use bits to represent numbers, so the numbers are necessarily base 2. These are binary numbers.
- This is a very important point – all computers store numbers in binary format.

**Positional Number Systems . . .**

- That’s great, but how do we use a bunch of 1s and 0s to represent a number? First, let’s look at numbering systems in general.
- Most numbering systems are positional number systems, in which a sequence of digits represents a number, such that the position of a digit within the sequence indicates the magnitude of its value.

**Positional Number Systems . . .**

- Example: The value two-hundred and thirty-eight is represented by the sequence ”238” (that is, the digit ”2”, followed by the digit ”3”, followed by the digit ”8”). The position of each digit indicates its value
  \[
  ”238” = 2 \times 100 + 3 \times 10 + 8 \times 1 \\
  = 2 \times 10^2 + 3 \times 10^1 + 8 \times 10^0 \\
  = 238
  \]
- Base 10 numbering is called decimal. It’s what humans use, probably because we have ten fingers.
Converting Binary to Decimal

- Multiply each digit by the radix raised to the appropriate power, and sum:

\[
325_7 = 3 \times 7^2 + 2 \times 7^1 + 5 \times 7^0
\]

\[
= 3 \times 49 + 2 \times 7 + 5
\]

\[
= 147 + 14 + 5
\]

\[
= 166_{10}
\]

Converting Decimal to Binary

- Repeatedly divide by the target radix. Each remainder is the next least significant digit of the number.

<table>
<thead>
<tr>
<th>Division</th>
<th>Quotient</th>
<th>Remainder</th>
</tr>
</thead>
<tbody>
<tr>
<td>489/5</td>
<td>97</td>
<td>4</td>
</tr>
<tr>
<td>97/5</td>
<td>19</td>
<td>2</td>
</tr>
<tr>
<td>19/5</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>3/5</td>
<td>0</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ 489_{10} = X_{2} \]

Converting Between Bases

- Convert to decimal first, then to the target base.

Octal and Hexadecimal

- Sometimes the conversion is easy, if both radices are powers of the same number. This allows numbers to be converted by grouping digits.

- For example, to convert a decimal number to a base 100 number each pair of decimal digits corresponds to a single base 100 digit.
Octal and Hexadecimal...

- Note that octal uses the digits 0-7, and hexadecimal 0-F. Hexadecimal needs 6 more digits than decimal, and instead of inventing new symbols the letters A-F are used.
- Each octal digit represents three binary digits, and each hex digit represents four. Digits are grouped from the right (least-significant), and leading zeros are added to achieve the correct grouping.

Converting between powers of 2

- Convert 25378 into binary

\[
2537_8 = 010 101 011 111_2 \\
= 01010101111_2
\]

- Convert 1011010010102 into octal

\[
101101001010_2 = 101 101 001 010_2 \\
= 5512_8
\]

Octal and Hexadecimal...

- Computers use binary numbers, but it’s unpleasant for humans to deal with long strings like 11101012.
- The larger the base, the fewer digits it takes to represent the number.
- Decimal representations are shorter (11101012 is 11710) but it’s not easy to convert between binary and decimal.
- Therefore, people often use octal (base 8) and hexadecimal (base 16) because they are powers of two, and can be converted to and from binary easily.
Converting between powers of 2...

- Convert $3A_{16}$ into binary:
  $3A_{16} = 011110101001_2$
- Convert $101101001010_2$ into hexadecimal:
  $101101001010_2 = B4A_{16}$

- Convert $401_{10}$ into hexadecimal:
  $401_{10} = 10000000110_2$
  $= 80B_{16}$

---

Fractions

- Fractions are expressed via digits to the right of the radix point. These digits represent decreasing negative powers of the radix.
- The number $0.d_1d_2...d_n$ represents the value $d_1 \times r^{-1} + d_2 \times r^{-2} + ... + d_n \times r^{-n}$.

Example:

- $0.234 = 2 \times 10^{-1} + 3 \times 10^{-2} + 4 \times 10^{-3}$

Convert non-decimal fractions to decimal:

- Multiply each digit by the radix to the appropriate power and sum.
- Convert $0.117_8$ into decimal:
  $0.117_8 = 1 \times 8^{-1} + 1 \times 8^{-2} + 7 \times 8^{-3}$
  $= \frac{1}{8} + \frac{1}{64} + \frac{7}{512}$
  $= 0.154296875_{10}$
**Binary-Coded Decimal (BCD) ...**

- The ‘+’ and ‘-’ signs also have 4-bit encodings, allowing positive and negative numbers to be represented:
- Convert 37 to BCD: 0011 0111
- Convert -42 to BCD: 1011 0100 0010
- Convert 1001 0111 from BCD to decimal: 97
- Note that some 4-bit values do not represent BCD digits: 1100, 1101, 1110, and 1111. BCD is therefore less "dense" than binary; a 4-bit binary number can represent 16 values, a 4-bit BCD number only 10.

**Convert decimal fractions to non-decimal**

- Multiply the fraction by the target radix.
- Integer portion is next digit, starting with most-significant.
- Repeat with remaining fraction. Note that this may never terminate.
- Convert 0.380859375₁₀ into hex:
  
  \[
  0.380859375₁₀ \times 16 = \boxed{6} \cdot 0.9375 \quad 0.6₁₆ \\
  0.09375 \times 16 = \boxed{1} \cdot 5 \quad 0.6₁₆ \\
  0.5 \times 16 = \boxed{8} \cdot 0 \quad 0.618₁₆
  \]

**Character Encodings**

- Strings of characters are represented by grouping bits together so that each group represents one character.
- ASCII (American Standard Code for Information Interchange) is the most common. Each character is represented by a 7-bit quantity.
- Because computers typically have 8-bits per byte, each ASCII digit is stored in one byte with the MSB=0.
- Convert the following sequence of bytes into characters:
  
  01001000 01100101 01101100 01101110
  
  ‘H’ ‘E’ ‘L’ ‘L’ ‘0’

**Binary-Coded Decimal (BCD)**

- Another way of encoding decimal numbers in a binary computer is binary-coded decimal. Its (only) advantage is that it’s easy to convert between BCD and decimal. Each digit of the decimal number is encoded using a 4-bit binary representation.

<table>
<thead>
<tr>
<th>Digit</th>
<th>Binary</th>
<th>Digit</th>
<th>Binary</th>
<th>Digit</th>
<th>Binary</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0000</td>
<td>4</td>
<td>0100</td>
<td>8</td>
<td>1000</td>
</tr>
<tr>
<td>1</td>
<td>0001</td>
<td>5</td>
<td>0101</td>
<td>9</td>
<td>1001</td>
</tr>
<tr>
<td>2</td>
<td>0010</td>
<td>6</td>
<td>0110</td>
<td>+</td>
<td>1010</td>
</tr>
<tr>
<td>3</td>
<td>0011</td>
<td>7</td>
<td>0111</td>
<td>-</td>
<td>1011</td>
</tr>
</tbody>
</table>
**Big Endian vs. Little Endian**

- There is an issue of how to store a 4-byte integer in memory. Assume we have a 4-byte integer, where byte ‘A’ is the most-significant and byte ‘B’ is the least.

\[
\begin{array}{cccc}
  A & B & C & D \\
\end{array}
\]

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
</table>

- The same issue arises when organizing bits in a byte.

**Big Endian vs. Little Endian . . .**

- When storing this integer there are many possible ways of assigning its four bytes to four bytes of memory. There are two popular ones:

<table>
<thead>
<tr>
<th>Big-Endian</th>
<th>Little-Endian</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 1 2 3</td>
<td>0 1 2 3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>MSB</th>
<th>LSB</th>
<th>MSB</th>
<th>LSB</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>B</td>
<td>D</td>
<td>C</td>
</tr>
<tr>
<td>D</td>
<td>C</td>
<td>B</td>
<td>A</td>
</tr>
</tbody>
</table>

**Character Encodings . . .**

- Unicode is gradually replacing ASCII. Unicode contains non-English characters, allowing Greek, Hebrew, etc. alphabets to be encoded. Each Unicode character is 16-bits; basically, the upper byte indicates the alphabet and the lower byte the character, although some alphabets have more than 256 characters and use several values for the upper byte. If the upper byte is 0, the lower byte is an ASCII character, providing some backwards-compatibility.

- EBCDIC was originally used in IBM mainframes. Perhaps it still is.

**Negative numbers**

- So far we haven’t discussed encoding negative numbers in binary. This is intentional. We’ll cover negative numbers when we get to arithmetic, where they are needed.
Big Endian vs. Little Endian...

- Big-endian: the most-significant byte (A) is stored in the given memory address, and least-significant byte (D) at the address + 3.
- Little-endian: the least-significant byte (D) is stored at the given memory address, and the most-significant at the address + 3.
- Neither one is clearly better than the other.
- You can configure most CPUs to do it either way. The MIPS is usually big-endian, the x86 family little-endian.

Readings and References

- See Table 1.7 in Maccabe for the full ASCII character set.
- Read Maccabe, Chapter 1, section 1.1–1.3.1, pp. 3–19.