Digital Data

Patrice Koehl
Computer Science
UC Davis

10:37:49
Digital Data

Binary and hexadecimal representations

Different types of numbers: natural numbers, integers, real numbers

ASCII code and UNICODE

Sound: Sampling, and Quantitizing

Images
Digital Data

Binary and hexadecimal representations

Different types of numbers: natural numbers, integers, real numbers

ASCII code and UNICODE

Sound: Sampling, and Quantitizing

Images
Number representation

We are used to counting in base 10:

<table>
<thead>
<tr>
<th></th>
<th>1000</th>
<th>100</th>
<th>10</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Exponent</td>
<td>$10^3$</td>
<td>$10^2$</td>
<td>$10^1$</td>
<td>$10^0$</td>
</tr>
<tr>
<td></td>
<td>1000</td>
<td>100</td>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

... thousands hundreds tens units

Example:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>7</th>
<th>3</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digit</td>
<td>1000</td>
<td>100</td>
<td>10</td>
<td>1</td>
</tr>
</tbody>
</table>

\[ 1 \times 1000 + 7 \times 100 + 3 \times 10 + 2 \times 1 = 1732 \]
Number representation

Computers use a different system: base 2:

Example:

```
1 1 0 1 1 0 0 0 1 0 0 0
1024 512 256 128 64 32 16 8 4 2 1
```

1x1024 + 1x512 + 0x256 + 1x128 + 1x64 + 0x32 + 0x16 + 0x8 + 1x4 + 0x2 + 0x1 = 1732

bits
## Number representation

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>10</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>4</td>
<td>100</td>
</tr>
<tr>
<td>5</td>
<td>101</td>
</tr>
<tr>
<td>6</td>
<td>110</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>253</td>
<td>11111101</td>
</tr>
<tr>
<td>254</td>
<td>11111110</td>
</tr>
<tr>
<td>255</td>
<td>11111111</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Conversion

From base 2 to base 10:

\[
\begin{array}{cccccccccccc}
1 & 1 & 1 & 0 & 1 & 0 & 1 & 0 & 1 & 0 & 0 \\
\hline
1024 & 512 & 256 & 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \\
\end{array}
\]

\[1 \times 1024 + 1 \times 512 + 1 \times 256 + 0 \times 128 + 1 \times 64 + 0 \times 32 + 1 \times 16 + 0 \times 8 + 1 \times 4 + 0 \times 2 + 0 \times 1 = 1876\]

From base 10 to base 2:

\[
\begin{align*}
1877 \ percentage\ 2 & = 938 \ \text{Remainder}\ 1 \\
938 \ percentage\ 2 & = 469 \ \text{Remainder}\ 0 \\
469 \ percentage\ 2 & = 234 \ \text{Remainder}\ 1 \\
234 \ percentage\ 2 & = 117 \ \text{Remainder}\ 0 \\
117 \ percentage\ 2 & = 58 \ \text{Remainder}\ 1 \\
58 \ percentage\ 2 & = 29 \ \text{Remainder}\ 0 \\
29 \ percentage\ 2 & = 14 \ \text{Remainder}\ 1 \\
14 \ percentage\ 2 & = 7 \ \text{Remainder}\ 0 \\
7 \ percentage\ 2 & = 3 \ \text{Remainder}\ 1 \\
3 \ percentage\ 2 & = 1 \ \text{Remainder}\ 1 \\
1 \ percentage\ 2 & = 0 \ \text{Remainder}\ 1 \\
\end{align*}
\]

1877 (base10) = 11101010101 (base 2)
Facts about Binary Numbers

- Each “digit” of a binary number (each 0 or 1) is called a bit

- 1 byte = 8 bits

- 1 KB = 1 kilobyte = $2^{10}$ bytes = 1024 bytes (≈ 1 thousand bytes)

- 1 MB = 1 Megabyte = $2^{20}$ bytes = 1,048,580 bytes (≈ 1 million bytes)

- 1 GB = 1 Gigabyte = $2^{30}$ bytes = 1,073,741,824 bytes (≈ 1 billion bytes)

- 1 TB = 1 Terabyte = $2^{40}$ bytes = 1,099,511,627,776 bytes (≈ 1 trillion bytes)

- A byte can represent numbers up to 255: 11111111 (base 2) = 255 (base 10)

- The largest number represented by a binary number of size N is $2^N - 1$
Big Data: Volume

Byte   Kilobyte (KB)   Megabyte (MB)   Gigabyte (GB)   Terabyte (TB)   Petabyte (PB)   Exabyte (EB)   Zettabyte (ZB)   Yottabyte (YB)
1000 bytes  1000 KB  1000 MB  1000 GB  1000 TB  1000 PB  1000 EB  1000 ZB  1000 YB
Big Data: Volume

- One page of text: 30KB
- One song: 5 MB
- One movie: 5 GB
- 6 million books: 1 TB
- 55 storeys of DVD: 1 PB
- Data up to 2003: 5 EB
- Data in 2011: 1.8 ZB
- NSA data center: 1 YB

<table>
<thead>
<tr>
<th>Unit</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte</td>
<td>1000 bytes</td>
</tr>
<tr>
<td>Kilobyte KB</td>
<td>1000 KB</td>
</tr>
<tr>
<td>Megabyte MB</td>
<td>1000 MB</td>
</tr>
<tr>
<td>Gigabyte GB</td>
<td>1000 GB</td>
</tr>
<tr>
<td>Terabyte TB</td>
<td>1000 TB</td>
</tr>
<tr>
<td>Petabyte PB</td>
<td>1000 PB</td>
</tr>
<tr>
<td>Exabyte EB</td>
<td>1000 EB</td>
</tr>
<tr>
<td>Zettabyte ZB</td>
<td>1000 ZB</td>
</tr>
<tr>
<td>Yottabyte YB</td>
<td>1000 YB</td>
</tr>
</tbody>
</table>
Big Data: Volume

<table>
<thead>
<tr>
<th>Unit</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Byte (KB)</td>
<td>One page of text</td>
</tr>
<tr>
<td>Kilobyte (MB)</td>
<td>One song</td>
</tr>
<tr>
<td>Megabyte (GB)</td>
<td>One movie</td>
</tr>
<tr>
<td>Gigabyte (TB)</td>
<td>6 million books</td>
</tr>
<tr>
<td>Terabyte (PB)</td>
<td>55 storeys of DVD</td>
</tr>
<tr>
<td>Petabyte (EB)</td>
<td>Data up to 2003</td>
</tr>
<tr>
<td>Exabyte (ZB)</td>
<td>Data in 2011</td>
</tr>
<tr>
<td>Zettabyte (YB)</td>
<td>NSA data center</td>
</tr>
</tbody>
</table>

- 30KB
- 5 MB
- 5 GB
- 1 TB
- 1 PB
- 5 EB
- 1.8 ZB
- 1 YB

- 1000 bytes
- 1000 KB
- 1000 MB
- 1000 GB
- 1000 TB
- 1000 PB
- 1000 ZB
- 1000 YB

- 1s
- 20 mins
- 11 days
- 30 years
- 300 centuries
- 30 million years
- 30 billion years
Hexadecimal numbers

While base 10 and base 2 are the most common bases used to represent numbers, others are also possible: base 16 is another popular one, corresponding to hexadecimal numbers.

The “digits” are: 0 1 2 3 4 5 6 7 8 9 A B C D E F

Example: 2x256 + 10x16 + 15x1 = 687
Hexadecimal numbers

Everything we have learned in base 10 should be studied again in other bases!!

Example: multiplication table in base 16:
Hexadecimal numbers

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
<th>Base 16</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>
Conversion: From base 2 to base 16, and back

This is in fact easy!!

-From base 2 to base 16:

Example: 11011000100

Step 1: break into groups of 4 (starting from the right):

110  1100  0100

Step 2: pad with 0, if needed:

0110  1100  0100

Step 3: convert each group of 4, using table:

6   C   4

Step 4: regroup:

6C4

11011000100 (base 2) = 6C4 (base 16)
Conversion: From base 2 to base 16, and back

From base 16 to base 2:

Example: 4FD

Step 1: split:

\[
\begin{array}{ccc}
4 & F & D \\
\end{array}
\]

Step 2: convert each “digit”, using table:

\[
\begin{array}{ccc}
0100 & 1111 & 1101 \\
\end{array}
\]

Step 3: Remove leading 0, if needed

\[
\begin{array}{ccc}
100 & 1111 & 1101 \\
\end{array}
\]

Step 4: regroup:

\[
\begin{array}{ccc}
1001111101 \\
\end{array}
\]

4FD (base 16) = 1001111101 (base 2)
Digital Data

Binary and hexadecimal representations

Different types of numbers: natural numbers, integers, real numbers

ASCII code and UNICODE

Sound: Sampling, and Quantitizing

Images
### The different set of numbers

<table>
<thead>
<tr>
<th>Set</th>
<th>Description</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \mathbb{N} )</td>
<td>Natural numbers</td>
<td>1, 2, 3, 4...</td>
</tr>
<tr>
<td>( \mathbb{Z} )</td>
<td>Integers</td>
<td>... , -4, -3, -2, -1, 0, 1, 2, 3, 4, ...</td>
</tr>
<tr>
<td>( \mathbb{Q} )</td>
<td>Rational numbers</td>
<td>( \frac{a}{b} ) where ( a ) and ( b ) are integers and ( b ) is not zero</td>
</tr>
<tr>
<td>( \mathbb{R} )</td>
<td>Real numbers</td>
<td>The limit of a convergent sequence of rational numbers</td>
</tr>
<tr>
<td>( \mathbb{C} )</td>
<td>Complex numbers</td>
<td>( a + ib ) where ( a ) and ( b ) are real numbers and ( i ) is the square root of ( -1 )</td>
</tr>
</tbody>
</table>
Representing Integers

Unsigned integers (natural numbers):

Sizes

➢ Char: 1 bit
➢ Unsigned short: 16 bits (2 bytes)
➢ Unsigned int: 32 bits (4 bytes)
Representing Integers

Signed integers

<table>
<thead>
<tr>
<th>S</th>
<th>Num</th>
</tr>
</thead>
</table>

S:
- sign bit: 0 means positive, 1 means negative

Num:
- If $s = 0$, direct representation of the number in binary form
- If $s = 1$, two's complement of the number

Sizes
- Char: 1 bit
- Short: 16 bits (2 bytes)
- int: 32 bits (4 bytes)
The two's complement of an $N$-bit number is defined as its complement with respect to $2^N$.

The sum of a number and its two's complement is $2^N$.

For instance, for the three-bit number 010, the two's complement is 110, because $010 + 110 = 1000 (= 2^3 = 8)$.

The two's complement is calculated by inverting the bits and adding one.
<table>
<thead>
<tr>
<th>Bits</th>
<th>Unsigned value</th>
<th>Two's complement value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 0000</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0000 0001</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0000 0010</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>0111 1110</td>
<td>126</td>
<td>126</td>
</tr>
<tr>
<td>0111 1111</td>
<td>127</td>
<td>127</td>
</tr>
<tr>
<td>1000 0000</td>
<td>128</td>
<td>-128</td>
</tr>
<tr>
<td>1000 0001</td>
<td>129</td>
<td>-127</td>
</tr>
<tr>
<td>1000 0010</td>
<td>130</td>
<td>-126</td>
</tr>
<tr>
<td>1111 1110</td>
<td>254</td>
<td>-2</td>
</tr>
<tr>
<td>1111 1111</td>
<td>255</td>
<td>-1</td>
</tr>
</tbody>
</table>
IEEE Standard 754

- Established in 1985 as uniform standard for floating point arithmetic
  Before that, many idiosyncratic formats

- Supported by all major CPUs

Driven by Numerical Concerns

- Nice standards for rounding, overflow, underflow
- Hard to make go fast
  - Numerical analysts predominated over hardware types in defining standard
IEEE Floating Point Representation

Numerical Form

\((-1)^s M 2^E\)

- Sign bit \(s\) determines whether number is negative or positive
- Significand \(M\) normally a fractional value in range \([1.0, 2.0)\).
- Exponent \(E\) weights value by power of two

Encoding

<table>
<thead>
<tr>
<th>MSB</th>
<th>exp</th>
<th>frac</th>
</tr>
</thead>
</table>

- MSB is sign bit
- exp field encodes \(E\)
- frac field encodes \(M\)
IEEE Floating Point Representation

<table>
<thead>
<tr>
<th>s</th>
<th>exp</th>
<th>frac</th>
</tr>
</thead>
</table>

**Encoding**
- MSB is sign bit
- \( \text{exp} \) field encodes \( E \)
- \( \text{frac} \) field encodes \( M \)

**Sizes**
- Single precision: 8 exp bits, 23 frac bits (32 bits total)
- Double precision: 11 exp bits, 52 frac bits (64 bits total)
- Extended precision: 15 exp bits, 63 frac bits
  - Only found in Intel-compatible machines
  - Stored in 80 bits (1 bit wasted)
IEEE Floating Point Representation

**Special value:**

\[ \text{exp} = 111...1 \]

- \[ \text{exp} = 111...1, \text{frac} = 000...0 \]
  - Represents value \( \infty \) (infinity)
  - Operation that overflows
  - Both positive and negative
  - E.g., \( 1.0/0.0 = -1.0/-0.0 = +\infty, \ 1.0/-0.0 = -\infty \)

- \[ \text{exp} = 111...1, \text{frac} \neq 000...0 \]
  - Not-a-Number (NaN)
  - Represents case when no numeric value can be determined
  - E.g., \( \sqrt{-1}, \infty - \infty \)
## Floating Point Operations

### Conceptual View
- First compute exact result
- Make it fit into desired precision
  - Possibly overflow if exponent too large
  - Possibly round to fit into $\text{frac}$

### Rounding Modes (illustrate with $\$ \$ \$ \$ \$ \$$ rounding)

<table>
<thead>
<tr>
<th></th>
<th>$1.40$</th>
<th>$1.60$</th>
<th>$1.50$</th>
<th>$2.50$</th>
<th>$-1.50$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Round down ($-\infty$)</td>
<td>$1$</td>
<td>$1$</td>
<td>$1$</td>
<td>$2$</td>
<td>$-2$</td>
</tr>
<tr>
<td>Round up ($+\infty$)</td>
<td>$2$</td>
<td>$2$</td>
<td>$2$</td>
<td>$3$</td>
<td>$-1$</td>
</tr>
<tr>
<td>Nearest Even</td>
<td>$1$</td>
<td>$2$</td>
<td>$2$</td>
<td>$2$</td>
<td>$-2$</td>
</tr>
</tbody>
</table>

**Note:**
1. **Round down:** rounded result is close to but no greater than true result.
2. **Round up:** rounded result is close to but no less than true result.
Computers encounter noise!

The Ariane 5 tragedy: On June 1996, the first Ariane 5 was launched… and exploded after 37 seconds.

The failure of the Ariane 501 was caused by the complete loss of guidance and altitude information 37 seconds after start… due to a numerical error.

Digital Data

Binary and hexadecimal representations

Different types of numbers: natural numbers, integers, real numbers

ASCII code and UNICODE

Sound: Sampling, and Quantitizing

Images
So far, we have seen how computers can handle numbers.

What about letters / characters?

The ASCII code was designed for that: it assigns a number to each character:

A-Z: 65-90  
a-z: 97-122  
0-9: 48-57
# ASCII

American Standard Code for Information Interchange

<table>
<thead>
<tr>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
<th>Dec</th>
<th>Hex</th>
<th>Char</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>C0</td>
<td>Null</td>
<td>32</td>
<td>20</td>
<td>Space</td>
<td>64</td>
<td>40</td>
<td>Æ</td>
<td>95</td>
<td>69</td>
<td>]</td>
</tr>
<tr>
<td>1</td>
<td>C1</td>
<td>Start of heading</td>
<td>33</td>
<td>21</td>
<td>!</td>
<td>65</td>
<td>41</td>
<td>A</td>
<td>97</td>
<td>69</td>
<td>a</td>
</tr>
<tr>
<td>2</td>
<td>C2</td>
<td>Start of text</td>
<td>34</td>
<td>22</td>
<td>&quot;</td>
<td>66</td>
<td>42</td>
<td>B</td>
<td>98</td>
<td>68</td>
<td>b</td>
</tr>
<tr>
<td>3</td>
<td>C3</td>
<td>End of text</td>
<td>35</td>
<td>23</td>
<td>#</td>
<td>67</td>
<td>43</td>
<td>C</td>
<td>99</td>
<td>67</td>
<td>c</td>
</tr>
<tr>
<td>4</td>
<td>C4</td>
<td>End of transmit</td>
<td>36</td>
<td>24</td>
<td>$</td>
<td>68</td>
<td>44</td>
<td>D</td>
<td>100</td>
<td>66</td>
<td>d</td>
</tr>
<tr>
<td>5</td>
<td>C5</td>
<td>Enquiry</td>
<td>37</td>
<td>25</td>
<td>%</td>
<td>69</td>
<td>45</td>
<td>E</td>
<td>101</td>
<td>65</td>
<td>e</td>
</tr>
<tr>
<td>6</td>
<td>C6</td>
<td>Acknowledge</td>
<td>38</td>
<td>26</td>
<td>&amp;</td>
<td>70</td>
<td>46</td>
<td>F</td>
<td>102</td>
<td>64</td>
<td>f</td>
</tr>
<tr>
<td>7</td>
<td>C7</td>
<td>Audible alert</td>
<td>39</td>
<td>27</td>
<td>'</td>
<td>71</td>
<td>47</td>
<td>G</td>
<td>103</td>
<td>67</td>
<td>g</td>
</tr>
<tr>
<td>8</td>
<td>C8</td>
<td>Backspace</td>
<td>40</td>
<td>28</td>
<td>(</td>
<td>72</td>
<td>48</td>
<td>H</td>
<td>104</td>
<td>68</td>
<td>h</td>
</tr>
<tr>
<td>9</td>
<td>C9</td>
<td>Vertical tab</td>
<td>41</td>
<td>29</td>
<td>)</td>
<td>73</td>
<td>49</td>
<td>I</td>
<td>105</td>
<td>67</td>
<td>i</td>
</tr>
<tr>
<td>10</td>
<td>CA</td>
<td>Line feed</td>
<td>42</td>
<td>2A</td>
<td>^</td>
<td>74</td>
<td>4A</td>
<td>J</td>
<td>106</td>
<td>68</td>
<td>j</td>
</tr>
<tr>
<td>11</td>
<td>CB</td>
<td>Vertical tab</td>
<td>43</td>
<td>2B</td>
<td>+</td>
<td>75</td>
<td>4B</td>
<td>K</td>
<td>107</td>
<td>69</td>
<td>k</td>
</tr>
<tr>
<td>12</td>
<td>CC</td>
<td>Form feed</td>
<td>44</td>
<td>2C</td>
<td>,</td>
<td>76</td>
<td>4C</td>
<td>L</td>
<td>108</td>
<td>69</td>
<td>l</td>
</tr>
<tr>
<td>13</td>
<td>CD</td>
<td>Carriage return</td>
<td>45</td>
<td>2D</td>
<td>-</td>
<td>77</td>
<td>4D</td>
<td>M</td>
<td>109</td>
<td>68</td>
<td>m</td>
</tr>
<tr>
<td>14</td>
<td>CE</td>
<td>Shift out</td>
<td>46</td>
<td>2E</td>
<td>.</td>
<td>78</td>
<td>4E</td>
<td>N</td>
<td>110</td>
<td>6B</td>
<td>n</td>
</tr>
<tr>
<td>15</td>
<td>CF</td>
<td>Shift in</td>
<td>47</td>
<td>2F</td>
<td>/</td>
<td>79</td>
<td>4F</td>
<td>O</td>
<td>111</td>
<td>6F</td>
<td>o</td>
</tr>
<tr>
<td>16</td>
<td>10</td>
<td>Dtab ink escape</td>
<td>49</td>
<td>30</td>
<td>0</td>
<td>80</td>
<td>50</td>
<td>p</td>
<td>112</td>
<td>70</td>
<td>p</td>
</tr>
<tr>
<td>17</td>
<td>11</td>
<td>Device control 1</td>
<td>49</td>
<td>31</td>
<td>1</td>
<td>81</td>
<td>51</td>
<td>q</td>
<td>113</td>
<td>71</td>
<td>q</td>
</tr>
<tr>
<td>18</td>
<td>12</td>
<td>Device control 2</td>
<td>49</td>
<td>32</td>
<td>2</td>
<td>82</td>
<td>52</td>
<td>r</td>
<td>114</td>
<td>72</td>
<td>r</td>
</tr>
<tr>
<td>19</td>
<td>13</td>
<td>Device control 3</td>
<td>49</td>
<td>33</td>
<td>3</td>
<td>83</td>
<td>53</td>
<td>s</td>
<td>115</td>
<td>73</td>
<td>s</td>
</tr>
<tr>
<td>20</td>
<td>14</td>
<td>Device control 4</td>
<td>49</td>
<td>34</td>
<td>4</td>
<td>84</td>
<td>54</td>
<td>t</td>
<td>116</td>
<td>74</td>
<td>t</td>
</tr>
<tr>
<td>21</td>
<td>15</td>
<td>Neg. acknowledge</td>
<td>53</td>
<td>35</td>
<td>5</td>
<td>85</td>
<td>55</td>
<td>u</td>
<td>117</td>
<td>75</td>
<td>u</td>
</tr>
<tr>
<td>22</td>
<td>16</td>
<td>Synchronous idle</td>
<td>54</td>
<td>36</td>
<td>6</td>
<td>86</td>
<td>56</td>
<td>v</td>
<td>118</td>
<td>76</td>
<td>v</td>
</tr>
<tr>
<td>23</td>
<td>17</td>
<td>End of block</td>
<td>55</td>
<td>37</td>
<td>7</td>
<td>87</td>
<td>57</td>
<td>w</td>
<td>119</td>
<td>77</td>
<td>w</td>
</tr>
<tr>
<td>24</td>
<td>18</td>
<td>Cancel</td>
<td>56</td>
<td>38</td>
<td>8</td>
<td>88</td>
<td>58</td>
<td>x</td>
<td>120</td>
<td>78</td>
<td>x</td>
</tr>
<tr>
<td>25</td>
<td>19</td>
<td>End of medium</td>
<td>57</td>
<td>39</td>
<td>9</td>
<td>89</td>
<td>59</td>
<td>y</td>
<td>121</td>
<td>7A</td>
<td>y</td>
</tr>
<tr>
<td>26</td>
<td>20</td>
<td>Substitution</td>
<td>58</td>
<td>3A</td>
<td>:</td>
<td>90</td>
<td>5A</td>
<td>z</td>
<td>122</td>
<td>7B</td>
<td>z</td>
</tr>
<tr>
<td>27</td>
<td>21</td>
<td>Escape</td>
<td>59</td>
<td>3B</td>
<td>;</td>
<td>91</td>
<td>5B</td>
<td>(</td>
<td>123</td>
<td>7C</td>
<td>(</td>
</tr>
<tr>
<td>28</td>
<td>22</td>
<td>File separator</td>
<td>60</td>
<td>3C</td>
<td>&lt;</td>
<td>92</td>
<td>5C</td>
<td>)</td>
<td>124</td>
<td>7D</td>
<td>)</td>
</tr>
<tr>
<td>29</td>
<td>23</td>
<td>Group separator</td>
<td>61</td>
<td>3D</td>
<td>=</td>
<td>93</td>
<td>5D</td>
<td>1</td>
<td>125</td>
<td>7E</td>
<td>1</td>
</tr>
<tr>
<td>30</td>
<td>24</td>
<td>Record separator</td>
<td>62</td>
<td>3E</td>
<td>&gt;</td>
<td>94</td>
<td>5E</td>
<td>~</td>
<td>126</td>
<td>7F</td>
<td>~</td>
</tr>
<tr>
<td>31</td>
<td>25</td>
<td>Unit separator</td>
<td>63</td>
<td>3F</td>
<td>?</td>
<td>95</td>
<td>5F</td>
<td>0</td>
<td>127</td>
<td>7F</td>
<td>0</td>
</tr>
</tbody>
</table>
ASCII only contains 127 characters (though an extended version exists with 257 characters).

This is by far not enough as it is too restrictive to the English language.

UNICODE was developed to alleviate this problem: the latest version, UNICODE 14.0 (September 2021) contains more than 140,000 characters, covering most existing languages.

For more information, see:

http://www.unicode.org/versions/Unicode14.0.0/
Digital Data

- Binary and hexadecimal representations
- Different types of numbers: natural numbers, integers, real numbers
- ASCII code and UNICODE
- Sound: Sampling, and Quantitizing
- Images
Digital Sound

Sound is produced by the vibration of a media like air or water. Audio refers to the sound within the range of human hearing. Naturally, a sound signal is analog, i.e. continuous in both time and amplitude.

To store and process sound information in a computer or to transmit it through a computer network, we must first convert the analog signal to digital form using an analog-to-digital converter (ADC); the conversion involves two steps:

1. **sampling**, and
2. **quantization**.

![Sound pressure vs. time diagram](http://example.com/sound_diagram.png)
Sampling is the process of examining the value of a continuous function at regular intervals.

Sampling usually occurs at uniform intervals, which are referred to as sampling intervals. The reciprocal of sampling interval is referred to as the sampling frequency or sampling rate.

If the sampling is done in time domain, the unit of sampling interval is second and the unit of sampling rate is Hz, which means cycles per second.
Note that choosing the sampling rate is not innocent:

A higher sampling rate usually allows for a better representation of the original sound wave. However, when the sampling rate is set to twice the highest frequency in the signal, the original sound wave can be reconstructed without loss from the samples. This is known as the Nyquist theorem.
Quantization

Quantization is the process of limiting the value of a sample of a continuous function to one of a predetermined number of allowed values, which can then be represented by a finite number of bits.
Quantization

The number of bits used to store each intensity defines the accuracy of the digital sound:

Adding one bit makes the sample twice as accurate
Audio Sound

**Sampling:**

The human ear can hear sound up to 20,000 Hz: a sampling rate of 40,000 Hz is therefore sufficient. The standard for digital audio is 44,100 Hz.

**Quantization:**

The current standard for the digital representation of audio sound is to use 16 bits (i.e. 65536 levels, half positive and half negative)
Audio Sound

How much space do we need to store one minute of music?

- 60 seconds
- 44,100 samples
- 16 bits (2 bytes) per sample
- 2 channels (stereo)

\[ S = 60 \times 44100 \times 2 \times 2 = 10,534,000 \text{ bytes} \approx 10 \text{ MB}!! \]

1 hour of music would be more than 600 MB!
Analog Recording

source medium (air) mic. analog representation

analog representation speaker medium (air) receivers

www.atpm.com/6.02/digitalaudio.shtml
DIGITAL RECORDING

www.atpm.com/6.02/digitalaudio.shtml
Advantages of digital recording:

- Faithful
  - can make multiple identical copies
- Can be processed
  - compression (MP3)
Digital Data

Binary and hexadecimal representations

Different types of numbers: natural numbers, integers, real numbers

ASCII code and UNICODE

Sound: Sampling, and Quantitizing

Images
Digital Images
Digital Images

Sampling:
Images are broken down into little squares: pixels
Resolution: Number of squares along each direction

Quantization:
Each pixel is characterized either as

• A binary number (0 or 1) to indicate black or white
• A natural number between 0 and 255, to indicate a gray scale
• A set of three numbers, each between 0 and 255, to indicate the amount of Red (R), Green (G), and Blue (B)

“True Color”: a pixel is represented by 24 bits, corresponding to 16,777,216 possible colors
Digital Images

The RGB color model (used for most digital representations of images)

<table>
<thead>
<tr>
<th>Notation</th>
<th>RGB triplet</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arithmetic</td>
<td>(1.0, 0.0, 0.0)</td>
</tr>
<tr>
<td>Percentage</td>
<td>(100%, 0%, 0%)</td>
</tr>
<tr>
<td>Digital 8-bit per channel</td>
<td>(255, 0, 0) or sometimes</td>
</tr>
<tr>
<td></td>
<td>#FF0000 (hexadecimal)</td>
</tr>
</tbody>
</table>

Mixing colors:
Digital Images

The CMYK color model (used by color printers)