Number Representations

CSE 361S

Make binary more human friendly

• Hexadecimal representation – base 16
• Commonly called “hex” but don’t be confused, it is not base 6, it is base 16
• Character set 0-9, a-f (alternately A-F)
  – a=10, b=11, c=12, d=13, e=14, and f=15
• C notation is to prefix hex with symbol 0x
  (e.g., 0x12, 0xa3)

Positional notation applies

xyz_{16} = x \cdot 16^2 + y \cdot 16^1 + z \cdot 16^0
= x \cdot 256 + y \cdot 16 + z

So 02c_{16} = 0 \cdot 256 + 2 \cdot (16) + 12 = 44_{10}

or 0x02c, which is the shorthand I will typically use in class

Benefits of Hex

• Real beauty of hex notation is ease with which one can move back and forth between hex and binary, since 16 = 2^4
• To transform hex number (e.g., 0x3d50) to binary we expand each hex digit to 4 bits of binary:
  3  d  5  0
  0011 1101 0101 0000

Binary to Hex Transformation

• To transform binary number (e.g., 1001000) to hex we group into 4-bit groups (starting from right) and rewrite each group in hex
  100 \quad 1000
  4 \quad 8 \quad = 0x48
• Or, e.g., 110101110
  1 \quad 1010 \quad 1110
  1 \quad a \quad e \quad = 0x1ae

What about fractions?

• Positional number systems work on both sides of the decimal point (radix point).
• If radix is r (n integer digits, m fractional digits):
  \text{val} = a_n \cdot r^n + a_{n-1} \cdot r^{n-1} + \ldots + a_0 \cdot r^0 + a_{-1} \cdot r^{-1} + a_{-2} \cdot r^{-2} + a_{-m} \cdot r^{-m}
• e.g., wx.yz_{16} = w \cdot 16 + x + y \cdot 16^{-1} + z \cdot 16^{-2}
Two kinds of numbers

- **Integers** – radix point is assumed to be at the far right end of the digits:
  - E.g. 01001110.

- **Fixed point** – radix point is at a given, fixed location:
  - E.g. 0100.1110
  - Often a common representation on digital signal processors

Q notation

- **Qn.m** means a number with n+m bits (digits), n integer and m fractional. Sign bit is often in addition to this.
- E.g., Q3.4 for 0100.1110, with value 4.75
- Qm means a number with m+1 bits, m are fractional
  - E.g., Q3 notation would have 4 bits and the following values
    - wxyz = \( w \cdot (1/2) + y \cdot (1/4) + z \cdot (1/8) \)
  - range is now -1 to +7/8, with resolution 1/8

Floating point representation

What about the reals? Use scientific notation.

In base 10: \( x \cdot 10^y \)

\( 0.32 \times 10^{-3} = 0.00032 \)

In base 2: \( x \cdot 2^y \) called floating point

\( \frac{\text{exponent}}{\text{mantissa}} \)

IEEE Floating Point

- Limited range of x and y (fixed # of bits) means we cannot represent every real number exactly

- IEEE std. 754 describes a standard form for floating point number representations
  - Single precision is 32 bits in size
  - Double precision is 64 bits in size

Single precision (32 bits)

<table>
<thead>
<tr>
<th>31 30</th>
<th>23 22</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>s</td>
<td>exponent (e)</td>
<td>fraction (f)</td>
</tr>
<tr>
<td>1</td>
<td>8 bits</td>
<td>23 bits</td>
</tr>
</tbody>
</table>

\[ \text{value} = (-1)^s \times 2^{e-127} \times \frac{1}{2} \times f \]

\[ \text{range} = \pm 2 \times 10^{\pm 38} \]
Double precision (64 bits)

\[
\begin{array}{ccc}
63 & 62 & 52 \\
| s | exponent (e) | fraction (f) |
\end{array}
\]

\[
11 \text{ bits} \quad 52 \text{ bits}
\]

\[
\text{value} = (-1)^s \times 2^{e-1023} \times 1.f
\]

\[
\begin{array}{ccc}
31 & 30 & 23 \quad 22 \quad 0 \\
| s | exponent (e) | fraction (f) |
\end{array}
\]

\[
\text{value} = (-1)^s \times 2^{e-127} \times 1.f
\]

<table>
<thead>
<tr>
<th>s</th>
<th>e</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>01000001110000000000000000000000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>e-127 =</td>
<td>mantissa =</td>
<td></td>
</tr>
<tr>
<td>value =</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Text – Characters and Strings

• ASCII – American Standard Code for Information Interchange
  – 7-bit codes representing basic Latin characters and numbers [A-Z, a-z, 0-9], some common punctuation, and control characters
  – There are a number of extensions to 8 bits, but only the 7-bit codes really standard.

• Unicode – 16-bit codes extending to a much wider set of languages
  – The first 128 codes are equivalent to the 7-bit ASCII standard

C Strings

• Strings are sequences of ASCII characters, stored one byte per character (8 bits), terminated by a NULL (zero) character
• E.g., “Hello!”
  01001000 ‘H’ 0x48
  01100101 ‘e’ 0x65
  01101100 ‘l’ 0x6c
  01101100 ‘l’ 0x6c
  01101111 ‘o’ 0x6f
  00100001 ‘!’ 0x21
  00000000 NULL 0x00

ASCII Facts

• Numerical digits are assigned in order of increasing value
  i.e., ‘0’ = 0x30
  ‘1’ = 0x31
  ‘2’ = 0x32
  ‘9’ = 0x39

• For single character, value conversion is simply a difference of 0x30
More ASCII Facts

• Letters are also assigned in lexicographical order:
  'A' = 0x41
  'B' = 0x42
  'Z' = 0x5a
  'a' = 0x61
  'b' = 0x62
  'z' = 0x7a

• Upper/lower case conversion is simply a difference of 0x20

Still More ASCII Facts

• First 32 characters (0-0x1f) are control codes:
  0x00 ^@ null (C string terminator)
  0x07 ^G bell
  0x0a ^J line feed
  0x0c ^L form feed
  0x0d ^M carriage return

Line breaks are not standardized

• End of line conventions differ by operating system:
  – In MS Windows: 0x0a, 0x0d is end of line
  – In Unix/Linux: 0x0a is end of line
  – 0x0a, linefeed, is sometimes called ‘newline’
• In C, ^n is mapped to OS end of line termination convention

Images

• Consider the following bits:
  0x002400081881423c
  0000 0000 0010 0100 0000 0000 0000 1000
  0001 1000 1000 0001 0100 0010 0011 1100
• Make 1 dark and 0 light:
  
Add color and more pixels

• Arrange in rows, one byte per row:

  ![Image](image1.png)

  Each bit is a “pixel” in the image