Number Representations

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
```

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., \( 0x12, 0xa3 \))

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
```

What about fractions?

- Positional number systems work on both sides of the decimal point (radix point).
  
  \[ \text{val} = a_{n-1} \cdot r^{n-1} + a_{n-2} \cdot r^{n-2} + \ldots + a_0 \cdot r^0 + a_{-1} \cdot r^{-1} + a_{-2} \cdot r^{-2} + a_{m} \cdot r^m \]

- If radix is \( r \) (n integer digits, m fractional digits):
  
  \[ \text{val} = a_{n-1} \cdot r^{n-1} + a_{n-2} \cdot r^{n-2} + \ldots + a_0 \cdot r^0 + a_{-1} \cdot r^{-1} + a_{-2} \cdot r^{-2} + a_{m} \cdot r^m \]

- e.g., \( wx.yz_{16} = w \cdot 16 + x + y \cdot 16^{-1} + z \cdot 16^{-2} \)
  
  or  \( wx.yz_2 = w \cdot 2 + x + y \cdot 2^{-1} + z \cdot 2^{-2} \)

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 1000
 4  8  = 0x48
```

- Or, e.g., \( 110101110 \)

```
 1  1010  1110
 1  a  e  = 0x1ae
```
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
  – 0.1001110 is 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 0100.1100, 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·xyz = w · (1) + x · (1/2) + y · (1/4) + z · (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: \( x \times 10^y \) 0.32 \times 10^{-3} = 0.00032

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

\[
\begin{array}{l|c|c}
| s | \text{exponent (e)} | \text{fraction (f)} |
\hline
1 & 8 \text{ bits} & 23 \text{ bits}
\end{array}
\]

value = \((-1)^s\times2^{e-127}\times1.f\)

hidden “1”

range = \( \pm 2 \times 10^{\pm38} \)

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)

\[
\begin{array}{l|c|c|c}
31 & 30 & 23 & 22 & 0
\hline
| s | \text{exponent (e)} | \text{fraction (f)} |
\hline
1 & 8 \text{ bits} & 23 \text{ bits}
\end{array}
\]

\[
\begin{array}{r}
\text{value} = (-1)^s \times 2^{e-127} \times 1.f \\uparrow \text{hidden “1”}
\end{array}
\]

\[
\begin{array}{l}
\text{range} = \pm 2 \times 10^{\pm38}
\end{array}
\]

• s = 0, e = 0, f = 0 \Rightarrow \text{value} = 0
• e = 255, f = 0 \Rightarrow \text{value} = (-1)^1 \times \text{infinity}
• e = 255, f \neq 0 \Rightarrow \text{value} = “not a number” triggers exception
• e = 0, f \neq 0 \Rightarrow \text{denormalized}
  \[
  \text{value} = (-1)^s \times 2^{126} \times 0.f \\uparrow \text{hidden “0”}
  \]
• Note use of sign-magnitude for entire number, and excess notation (excess 127) for exponent
Double precision (64 bits)

\[
\begin{array}{c|c|c|c|c}
| s | \text{exponent (e)} | \text{fraction (f)} | 0 \\hline
1 & 11 \text{ bits} & 52 \text{ bits} & \hline
\end{array}
\]

value = \((-1)^s \times 2^{e-1023} \times 1.f \uparrow \text{hidden "1"}

range = \pm 2 \times 10^{\pm 308}

e = 0, f \neq 0 \Rightarrow \text{denormalized}

value = (-1)^s \times 2^{-1022} \times 0.f

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 – 8- or 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

Java Strings

- Strings are represented via the class “String”
- String objects are immutable
- The character encoding is system specific, e.g., either UTF-8 or UTF-16 (typical).
- The length is an instance variable in the object (in most implementations)
- The characters are stored in a char[] array (again, in most implementations)

Unicode

- Standard for character representation
  - Supports wide variety of languages, symbols
- UTF-8
  - Variable length code with 8-bit code units
  - U+0000 to U+007F are the same as ASCII
- UTF-16
  - Uses 16-bit code units, also variable length
  - Latin + Greek + Cyrillic + Coptic + Armenian + Hebrew + Arabic + Syriac + Tāna + N’Ko fit in 16 bits
- UTF-32
  - Uses 32-bit code units, fixed length

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:

\[ \ldots \ldots \ldots \ldots \ldots \ldots \ldots \]

- Arrange in rows, one byte per row:

```
  .   .   .   .   .   .   .   .
  .   .   .   .   .   .   .   .
  .   .   .   .   .   .   .   .
  .   .   .   .   .   .   .   .
  .   .   .   .   .   .   .   .
```

- Each bit is a “pixel” in the image

Images

- Common approach is row, column multiplexing

- Extend with intensity control for each pixel
  - 8 bits \( \rightarrow \) 0 is “off”, 255 (or 0xff) is “on”
Row-based Multiplexed Control

\[
\begin{align*}
\text{for } r &= 1 \text{ to } 7 \\
& \quad \text{wait until next row time} \\
& \quad \text{set row, LOW} \\
& \quad \text{set all other rows HIGH} \\
& \quad \text{for } c = 1 \text{ to } 5 \\
& \quad \quad \text{set column}_c \text{ to value for row}_r \\
& \quad \quad \text{(HIGH for on, LOW for off)} \\
& \quad \text{end for} \\
& \text{end for} \\
\end{align*}
\]

This needs series resistors on each column

Column-based Multiplexed Control

\[
\begin{align*}
\text{for } c &= 1 \text{ to } 5 \\
& \quad \text{wait until next column time} \\
& \quad \text{set column}_c \text{ HIGH} \\
& \quad \text{set all other columns LOW} \\
& \quad \text{for } r = 1 \text{ to } 7 \\
& \quad \quad \text{set row}_r \text{ to value for column}_c \\
& \quad \quad \text{(LOW for on, HIGH for off)} \\
& \quad \text{end for} \\
& \text{end for} \\
\end{align*}
\]

This needs series resistors on each row

Add color and more pixels

Color

- Additive color – primaries Red, Green, Blue
- Position close together and put diffuser above – This builds one pixel