Unit 3
Binary Representation

ANALOG VS. DIGITAL

Analog vs. Digital
• The analog world is based on continuous events. Observations can take on (real) any value.

• The digital world is based on discrete events. Observations can only take on a finite number of discrete values

Q. Which is better?
A. Depends on what you are trying to do.

• Some tasks are better handled with analog data, others with digital data.
  – Analog means continuous/real valued signals with an infinite number of possible values
  – Digital signals are discrete [i.e. 1 of n values]
Analog vs. Digital

• How much money is in my checking account?
  – Analog: Oh, some, but not too much.
  – Digital: $243.67

Analog vs. Digital

• How much do you love me?
  – Analog: I love you with all my heart!!!!
  – Digital: $3.2 \times 10^3$ MegaHearts

The Real (Analog) World

• The real world is inherently analog.
• To interface with it, our digital systems need to:
  – Convert analog signals to digital values (numbers) at the input.
  – Convert digital values to analog signals at the output.
• Analog signals can come in many forms
  – Voltage, current, light, color, magnetic fields, pressure, temperature, acceleration, orientation

Digital is About Numbers

• In a digital world, numbers are used to represent all the possible discrete events
  – Numerical values (5.7, 1923.8, ...)
  – Computer instructions (ADD, SUB, BLE, ...)
  – Characters ('a', 'b', 'c', ...)
  – Conditions (on, off, ready, paper jam, ...)
• Numbers allow for easy manipulation
  – Add, multiply, compare, store, ...
• Results are repeatable
  – Each time we add the same two number we get the same result
DIGITAL REPRESENTATION

Interpreting Binary Strings

- Given a string of 1’s and 0’s, you need to know the representation system being used, before you can understand the value of those 1’s and 0’s.

\[ \text{01000001 = ?} \]

**Binary Representation Systems**

- **Integer Systems**
  - Unsigned
    - Unsigned (Normal) binary
  - Signed
    - Signed Magnitude
    - 2’s complement
    - Excess-\(N\)^*'
    - 1’s complement^*
- **Floating Point**
  - For very large and small (fractional) numbers

**Codes**

- **Text**
  - ASCII / Unicode
- **Decimal Codes**
  - BCD (Binary Coded Decimal) / (8421 Code)

* = Not fully covered in this class
4 Skills

- We will teach you 4 skills that you should know and be able to apply with confidence
  - Convert a number in any base (base r) to decimal (base 10)
  - Convert a decimal number (base 10) to binary
  - Use the shortcut for conversion between binary (base 2) and hexadecimal (base 16)
  - Understand the finite number of combinations that can be made with n bits (binary digits) and its implication for codes including ASCII and Unicode

**BASE R TO BASE 10**

Using positional weights/place values

### Number Systems

- Number systems consist of
  1. ________________
  2. ___ coefficients [__________]
- Human System: Decimal (Base 10): 0,1,2,3,4,5,6,7,8,9
- Computer System: Binary (Base 2): 0,1
- Human systems for working with computer systems (shorthand for human to read/write binary)
  - ________________
  - ________________

### Anatomy of a Decimal Number

- A number consists of a string of explicit coefficients (digits).
- Each coefficient has an implicit place value depending on its position in the number and is a ________ of the base.
- The value of a decimal number (a string of decimal coefficients) is the sum of each coefficient times its place value

\[
(934)_{10} = 9\times + 3\times + 4\times = __________
\]

\[
(3.52)_{10} = 3\times + 5\times + 2\times = __________
\]
**Anatomy of a Binary Number**

- Same as decimal but now the coefficients are 1 and 0 and the place values are the powers of 2

\[ (1011)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \]

**General Conversion From Base \( r \) to Decimal**

- A number in base \( r \) has place values/weights that are the powers of the base
- Denote the coefficients as: \( a_i \)

\[
N_r = a_3r^3 + a_2r^2 + a_1r^1 + a_0r^0 + a_{-1}r^{-1} + a_{-2}r^{-2}
\]

\[ N_r \Rightarrow \_\_\_\_\_\_\_\_ \Rightarrow D_{10} \]

**Examples**

- \((746)_8 = \)
- \((1A5)_{16} = \)
- \((AD2)_{16} = \)

**Binary Examples**

- \((1001.1)_2 = \)
- \((10110001)_2 = \)
Powers of 2

- $2^0 = 1$
- $2^1 = 2$
- $2^2 = 4$
- $2^3 = 8$
- $2^4 = 16$
- $2^5 = 32$
- $2^6 = 64$
- $2^7 = 128$
- $2^8 = 256$
- $2^9 = 512$
- $2^{10} = 1024$

Unique Combinations

- Given $n$ digits of base $r$, how many unique numbers can be formed? __
  - What is the range? [________

<table>
<thead>
<tr>
<th>Base 10</th>
<th>Base 2</th>
<th>Base 16</th>
</tr>
</thead>
<tbody>
<tr>
<td>1024</td>
<td>1010</td>
<td>1000010</td>
</tr>
<tr>
<td>512</td>
<td>1000</td>
<td>330</td>
</tr>
<tr>
<td>256</td>
<td>1001</td>
<td>126</td>
</tr>
<tr>
<td>128</td>
<td>1010</td>
<td>80</td>
</tr>
<tr>
<td>64</td>
<td>1011</td>
<td>40</td>
</tr>
<tr>
<td>32</td>
<td>1100</td>
<td>20</td>
</tr>
<tr>
<td>16</td>
<td>1110</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>1000</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>1001</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>1010</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Main Point: Given $n$ digits of base $r$, ___ unique numbers can be made with the range [________

Approximating Large Powers of 2

- Often need to find decimal approximation of a large powers of 2 like $2^{16}$, $2^{32}$, etc.
- Use following approximations:
  - $2^{10} = \approx 1024$
  - $2^{20} = \approx 1048576$
  - $2^{30} = \approx 1073741824$
  - $2^{40} = \approx 1099511627776$

- For other powers of 2, decompose into product of $2^{10}$ or $2^{20}$ or $2^{30}$ and a power of 2 that is less than $2^{10}$
  - 16-bit half word: 64K numbers
  - 32-bit word: 4G numbers
  - 64-bit dword: 16 million trillion numbers

"Making change"

BASE 10 TO BASE 2 OR BASE 16
Decimal to Unsigned Binary

• To convert a decimal number, \( x \), to binary:
  – Only coefficients of 1 or 0. So simply find place values that add up to the desired values, starting with larger place values and proceeding to smaller values and place a 1 in those place values and 0 in all others.

\[
\begin{array}{cccccc}
25_{10} &=& 32 & 16 & 8 & 4 & 2 & 1 \\
\end{array}
\]

Decimal to Unsigned Binary

\[
\begin{array}{cccccc}
73_{10} &=& 128 & 64 & 32 & 16 & 8 & 4 & 2 & 1 \\
87_{10} &=& & & & & & & & \\
145_{10} &=& & & & & & & & \\
0.625_{10} &=& & & & & & .5 & .25 & .125 & .0625 & .03125 \\
\end{array}
\]

Decimal to Another Base

• To convert a decimal number, \( x \), to base \( r \):
  – Use the place values of base \( r \) (powers of \( r \)). Starting with largest place values, fill in coefficients that sum up to desired decimal value without going over.

\[
\begin{array}{cccccc}
75_{10} &=& 256 & 16 & 1 & \text{hex} \\
\end{array}
\]

Shortcuts for Converting Binary (\( r=2 \)), Hexadecimal (\( r=16 \)) and Octal (\( r=8 \))

**SHORTHAND FOR BINARY**
Binary, Octal, and Hexadecimal

- Octal (base $8 = 2^3$)
  - 1 Octal digit ($\_)_8$ can represent: _______
  - 3 bits of binary ($\_\_\_)_2$ can represent: 000-111 = _______
  - Conclusion...
    __Octal digit = ___ bits

- Hex (base $16 = 2^4$)
  - 1 Hex digit ($\_)_16$ can represent: 0-F (_____)
  - 4 bits of binary ($\_\_\_\_)_2$ can represent: 0000-1111 = _______
  - Conclusion...
    __Hex digit = ___ bits

Binary to Octal or Hex

- Make groups of 3 bits starting from radix point and working outward
- Add 0’s where necessary
- Convert each group of 3 to an octal digit

Octal or Hex to Binary

- Expand each octal digit to a group of 3 bits
  
    $317.2_8$
    
        $4 \ 2 \ 1 \ 4 \ 2 \ 1 \ 4 \ 2 \ 1 \ 4 \ 2 \ 1$

- Expand each hex digit to a group of 4 bits
  
    $D93.8_{16}$
    
        $8 \ 4 \ 2 \ 1 \ 8 \ 4 \ 2 \ 1 \ 8 \ 4 \ 2 \ 1 \ 8 \ 4 \ 2 \ 1$

Hexadecimal Representation

- Since values in modern computers are many bits, we use hexadecimal as a shorthand notation (4 bits = 1 hex digit)
  - 11010010 = D2 hex or $0xD2$ if you write it in C/C++
  - 0111011011001011 = 76CB hex or $0x76CB$ if you write it in C/C++
Binary Representation Systems

- Integer Systems
  - Unsigned
    - Unsigned (Normal) binary
  - Signed
    - Signed Magnitude
    - 2's complement
    - 1's complement* 
    - Excess-N*

- Floating Point
  - For very large and small (fractional) numbers

* = Not covered in this class

Codes

- Text
  - ASCII / Unicode

- Decimal Codes
  - BCD (Binary Coded Decimal) / (8421 Code)

---

Binary Codes

- Using binary we can represent any kind of information by coming up with a code
- Using $n$ bits we can represent $2^n$ distinct items

Colors of the rainbow:
- Red = 000
- Orange = 001
- Yellow = 010
- Green = 100
- Blue = 101
- Purple = 111
- *Z* = 11001

Letters:
- *A* = 00000
- *B* = 00001
- *C* = 00010

---

BCD (If Time Permits)

- Rather than convert a decimal number to binary which may lose some precision (i.e. $0.1_{10}$ = infinite binary fraction), BCD represents each decimal digit as a separate group of bits (exact decimal precision)
  - Each digits is represented as a ________ number (using place values 8,4,2,1 for each dec. digit)
  - Often used in financial and other applications where decimal precision is needed

Important: Some processors have specific instructions to operate on #'s represented in BCD

$(439)_{10}$

BCD Representation:

Unsigned Binary Rep.: $\ 110110111_2$
**ASCII Code**

- Used for representing text characters
- Originally 7-bits but usually stored as 8-bits = 1-byte in a computer
- Example:
  - "Hello\n";
  - Each character is converted to ASCII equivalent
    - ‘H’ = 0x48, ‘e’ = 0x65, ...
    - \n = newline character is represented by either one or two ASCII characters

**ASCII Table**

<table>
<thead>
<tr>
<th>LSD/MSD</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NULL</td>
<td>DLW</td>
<td>SPACE</td>
<td>0</td>
<td>@</td>
<td>P</td>
<td>‘</td>
<td>p</td>
</tr>
<tr>
<td>1</td>
<td>SOH</td>
<td>DC1</td>
<td>‘</td>
<td>1</td>
<td>A</td>
<td>Q</td>
<td>a</td>
<td>q</td>
</tr>
<tr>
<td>2</td>
<td>STX</td>
<td>DC2</td>
<td>“</td>
<td>2</td>
<td>B</td>
<td>R</td>
<td>b</td>
<td>r</td>
</tr>
<tr>
<td>3</td>
<td>ETX</td>
<td>DC3</td>
<td>#</td>
<td>3</td>
<td>C</td>
<td>S</td>
<td>c</td>
<td>s</td>
</tr>
<tr>
<td>4</td>
<td>EOT</td>
<td>DC4</td>
<td>$</td>
<td>4</td>
<td>D</td>
<td>T</td>
<td>d</td>
<td>t</td>
</tr>
<tr>
<td>5</td>
<td>ENQ</td>
<td>NAK</td>
<td>%</td>
<td>5</td>
<td>E</td>
<td>U</td>
<td>e</td>
<td>u</td>
</tr>
<tr>
<td>6</td>
<td>ACK</td>
<td>SYN</td>
<td>&amp;</td>
<td>6</td>
<td>F</td>
<td>V</td>
<td>f</td>
<td>v</td>
</tr>
<tr>
<td>7</td>
<td>BEL</td>
<td>ETB</td>
<td>‘</td>
<td>7</td>
<td>G</td>
<td>W</td>
<td>g</td>
<td>w</td>
</tr>
<tr>
<td>8</td>
<td>BS</td>
<td>CAN</td>
<td>[</td>
<td>8</td>
<td>H</td>
<td>X</td>
<td>h</td>
<td>x</td>
</tr>
<tr>
<td>9</td>
<td>TAB</td>
<td>EM</td>
<td>]</td>
<td>9</td>
<td>I</td>
<td>Y</td>
<td>i</td>
<td>y</td>
</tr>
<tr>
<td>A</td>
<td>LF</td>
<td>SUB</td>
<td>*</td>
<td>:</td>
<td>J</td>
<td>Z</td>
<td>j</td>
<td>z</td>
</tr>
<tr>
<td>B</td>
<td>VT</td>
<td>ESC</td>
<td>+</td>
<td>;</td>
<td>K</td>
<td></td>
<td>k</td>
<td></td>
</tr>
<tr>
<td>C</td>
<td>FF</td>
<td>FS</td>
<td>.</td>
<td>&lt;</td>
<td>L</td>
<td>\</td>
<td>l</td>
<td></td>
</tr>
<tr>
<td>D</td>
<td>CR</td>
<td>GS</td>
<td>-</td>
<td>=</td>
<td>M</td>
<td>]</td>
<td>m</td>
<td>)</td>
</tr>
<tr>
<td>E</td>
<td>SO</td>
<td>RS</td>
<td>&gt;</td>
<td>&gt;</td>
<td>N</td>
<td>^</td>
<td>n</td>
<td>~</td>
</tr>
<tr>
<td>F</td>
<td>SI</td>
<td>US</td>
<td>/</td>
<td>?</td>
<td>O</td>
<td>_</td>
<td>o</td>
<td>DEL</td>
</tr>
</tbody>
</table>

**UniCode**

- ASCII can represent only the English alphabet, decimal digits, and punctuation
  - 7-bit code => $2^7 = 128$ characters
  - It would be nice to have one code that represented more alphabets/characters for common languages used around the world
- Unicode
  - Up to 32-bit Code => $2^{32}$ combinations
  - 137,000 character defined for many languages
  - Used by Java as standard character code

**Summary**

- Convert Base $r$ to Base 10
  - Apply place values (powers of $r$)
  - $N_r \Rightarrow \Sigma (a_i \cdot r^i) \Rightarrow D_{10}$
- Convert Base 10 to Base $r$
  - "Make change" using powers of $r$ as the weights/denominations
- Base 2 (Bin) $\leftrightarrow$ Base 16 (Hex)
  - Group or expand 1 hex digit to/from 4 bits
  - Start at binary point and work outward