Analogue vs. Digital

- The analog world is based on continuous events. Observations can take on any (real) 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: $243.67

Digital is About Numbers

- In a digital world, numbers are used to represent all the possible discrete events
  - Numerical values
  - 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

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
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.

\[10010001 = ?\]

Number Systems

- Number systems consist of
  1. _____________
  2. ___ coefficients/digits: _____________

- 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)
  - _______________
  - _______________

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
Positional Number Systems (Unsigned)

- Base r:
  - Implicit Place Values:
  - Explicit Coefficients:

  Left-most digit = Right-most digit =

  \[ a_3 \quad a_2 \quad a_1 \quad a_0 \quad a_{-1} \quad a_{-2} \quad \ldots \]

Decimal:  
Binary:

\[ N_r = \]

Examples

\[(746)_8 = \]
\[(1A5)_{16} = \]
\[(\underline{1001}.1)_2 = \]
\[(\underline{10110001})_2 = \]

Powers of 2

- Decimal equivalent is…
  ... the sum of each coefficient multiplied by its implicit place value (power of the base)

  \[ = \sum a_i \cdot r^i \quad [a_i = \text{coefficient}, \ r = \text{base}] \]

\[(11010)_2 = \]
\[(6523)_8 = \]
\[(\text{AD}2)_{16} = \]

Practice On Your Own
Unique Combinations

- Given \( n \) digits of base \( r \), how many unique numbers can be formed? ____
  - What is the range? _________

2-digit, decimal numbers (\( r=10, n=2 \))

3-digit, decimal numbers (\( r=10, n=3 \))

4-bit, binary numbers (\( r=2, n=4 \))

6-bit, binary numbers (\( r=2, n=6 \))

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:
  - \( 2^{10} \approx \) ____________
  - \( 2^{20} \approx \) ____________
  - \( 2^{30} \approx \) ____________
  - \( 2^{40} \approx \) ____________

- 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 \( 2^{10} \)

Decimal to Unsigned Binary

- Now lets convert from decimal to 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

\[
\begin{align*}
25_{10} &= 0 & 1 & 0 & 0 & 0 \\
73_{10} &= 1 & 0 & 0 & 0 & 1 & 1 \\
87_{10} &= 1 & 0 & 0 & 0 & 1 & 0 & 1 \\
145_{10} &= 1 & 0 & 0 & 0 & 1 & 1 & 0 \\
0.625_{10} &= 0 & 0 & 0 & 1 & 0 & 0 & 1 \\
\end{align*}
\]
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.

$$75_{10} = \_\_\_\_\_\ hex$$

$$\_\_\_\_\ 256$$
$$\_\_\_\ 16$$
$$\_\_\ 1$$

Unsigned and Signed

• Normal (unsigned) binary can only represent positive numbers
  – All place values are positive

• To represent negative numbers we must use a modified binary representation that takes into account sign (pos. or neg.)
  – We call these *signed* representations

Signed Number Representation

• 2 Primary Systems
  1.
  2.

Signed numbers

• All systems used to represent negative numbers split the possible binary combinations in half (half for positive numbers / half for negative numbers)

• In both signed magnitude and 2’s complement, positive and negative numbers are separated using the ______
  – _____ = 1 means __________
  – _____ = 0 means __________
Signed Magnitude System

- Use binary place values but now MSB represents the sign (1 if negative, 0 if positive)

<table>
<thead>
<tr>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-bit Unsigned</td>
<td>8</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>4-bit Signed Magnitude</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

0 to 15

-7 to +7

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit Signed Magnitude</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

-127 to +127

Signed Magnitude Examples

4-bit Signed Magnitude

<table>
<thead>
<tr>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

-5

<table>
<thead>
<tr>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

-7

Notice that +3 in signed magnitude is the same as in the unsigned system

8-bit Signed Magnitude

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

-19

Important: Positive numbers have the same representation in signed magnitude as in normal unsigned binary

2’s Complement System

- Normal binary place values except ________
  - MSB of 1 means ________________

<table>
<thead>
<tr>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-bit Unsigned</td>
<td>8</td>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>4-bit 2’s complement</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

0 to 15

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>8-bit 2’s complement</td>
<td>64</td>
<td>32</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Important: ________ numbers have the same representation in 2’s complement as in normal unsigned binary

2’s Complement Examples

4-bit 2’s complement

<table>
<thead>
<tr>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

= 0 0 1 1

1 1 1 1

8-bit 2’s complement

<table>
<thead>
<tr>
<th>Bit 7</th>
<th>Bit 6</th>
<th>Bit 5</th>
<th>Bit 4</th>
<th>Bit 3</th>
<th>Bit 2</th>
<th>Bit 1</th>
<th>Bit 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

= 1 0 0 0 0 0 0 1

Important: ________ numbers have the same representation in 2’s complement as in normal unsigned binary
2’s Complement Range

• Given n bits...
  – Max positive value =
  – Max negative value =

Range with n-bits of 2’s complement

  – Side note – What decimal value is 111...11?

Comparison of Systems

Signed Mag.

Unsigned and Signed Variables

• In C, unsigned variables use ______________ (normal power-of-2 place values) to represent numbers

1 0 0 1 0 0 1 1 = +147
128 64 32 16 8 4 2 1

• In C, signed variables use the ______________ system (Neg. MSB weight) to represent numbers

1 0 0 1 0 0 1 1 = -109
-128 64 32 16 8 4 2 1

IMPORTANT NOTE

• All computer systems use the 2’s complement system to represent signed integers!

• So from now on, if we say an integer is signed, we are actually saying it uses the 2's complement system unless otherwise specified
  – We will not use "signed magnitude" unless explicitly indicated
Zero and Sign Extension

- Extension is the process of increasing the number of bits used to represent a number without changing its value.

Unsigned = 

111011 = ___111011

2’s complement = 

pos. 011010 = ___011010
neg. 110011 = ___110011

Zero and Sign Truncation

- Truncation is the process of decreasing the number of bits used to represent a number without changing its value.

Unsigned = 

0011011 = __________

2’s complement = 

pos. 00011010 = __________
neg. 1110011 = __________

Review

<table>
<thead>
<tr>
<th>Unsigned</th>
<th>Signed Mag.</th>
<th>2’s Complement</th>
</tr>
</thead>
<tbody>
<tr>
<td>11010011</td>
<td></td>
<td></td>
</tr>
<tr>
<td>00101101</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Decimal Equivalent

<table>
<thead>
<tr>
<th>Unsigned</th>
<th>2’s comp.</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x0D</td>
<td></td>
</tr>
<tr>
<td>0xEF</td>
<td></td>
</tr>
</tbody>
</table>

Data Representation

- In C/C++ variables can be of different types and sizes.
  - Integer Types (signed and unsigned)
    - [unsigned] char
    - [unsigned] short [int]
    - [unsigned] long [int]
    - [unsigned] long long [int]

<table>
<thead>
<tr>
<th>C Type</th>
<th>Bytes</th>
<th>Bits</th>
<th>MIPS Name</th>
<th>ATmega328</th>
</tr>
</thead>
<tbody>
<tr>
<td>[unsigned] char</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[unsigned] short [int]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[unsigned] long [int]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>[unsigned] long long [int]</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

  ^ Can emulate but has no single-instruction support

  - Floating Point Types
    - float
    - double
**Binary, Octal, and Hexadecimal**

**Octal**
- Octal (base 8 = $2^3$)
- 1 Octal digit ($\_\_\_\_$) can represent: ______
- 3 bits of binary ($\_ \_ \_\_\_$) can represent: _______________
- Conclusion...

**Hex**
- Hex (base 16 = $2^4$)
- 1 Hex digit ($\_\_\_\_\_$) can represent: ___________
- 4 bits of binary ($\_ \_ \_\_\_\_\_$) can represent: _______________
- Conclusion...

**SHORTHAND FOR BINARY**

**Octal**
- Octal (base 8 = $2^3$)
- 1 Octal digit ($\_\_\_\_$) can represent: ______
- 3 bits of binary ($\_ \_ \_\_\_$) can represent: _______________
- Conclusion...

**Hex**
- Hex (base 16 = $2^4$)
- 1 Hex digit ($\_\_\_\_\_$) can represent: ___________
- 4 bits of binary ($\_ \_ \_\_\_\_\_$) can represent: _______________
- Conclusion...

**Binary to Octal or Hex**
- Make groups of ___ bits starting from radix point and working outward
- Add 0’s where necessary
- Convert each group to an octal digit
  
  101001110.11

**Octal or Hex to Binary**
- Expand each octal digit to a group of ___ bits
  
  $317.2_8$

- Expand each hex digit to a group of ___ bits
  
  $D93.8_{16}$
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++

• Important Point: To interpret the value of a hex number, you must know what underlying binary system is assumed (unsigned, 2’s comp. etc.)
  – D2 hex (what if underlying binary system is unsigned?)
  – D2 hex (what if underlying binary system is signed?)
  – D2 hex ( = 'Ọ' in Arial Unicode font)

Binary Representation Systems

• Codes
  – Text
    • ASCII / Unicode
  – Decimal Codes
    • BCD (Binary Coded Decimal) / (8421 Code)

• 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

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

Letters:
- ‘A’ = 00000
- ‘B’ = 00001
- ‘C’ = 00010
- ‘Z’ = 11001
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 character

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>p</td>
<td></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>&quot;</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>N</td>
<td>^</td>
<td>n</td>
<td>~</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
  - 8-bit code => 256 characters
  - It would be nice to have one code that represented more alphabets/characters for common languages used around the world
- Unicode
  - 16-bit Code => 65,536 characters
  - Represents many languages alphabets and characters
  - Used by Java as standard character code

Unicode hex value
(i.e. FB52 => 1111101101010010)