Number Representation
Data input: Analog ➔ Digital

• Real world is analog!

• To import analog information, we must do two things
  • Sample
    - E.g., for a CD, every 44,100ths of a second, we ask a music signal how loud it is.
  • Quantize
    - For every one of these samples, we figure out where, on a 16-bit (65,536 tic-mark) “yardstick”, it lies.

www.joshuadysart.com/journal/archives/digital_sampling.gif

Dr Dan Garcia
How many bits to represent $\pi$?

a) 1
b) 9 ($\pi = 3.14$, so that’s 011 “.” 001 100)
c) 64 (Since Macs are 64-bit machines)
d) Every bit the machine has!
e) $\infty$
What to do with representations of numbers?

• Just what we do with numbers!
  • Add them
  • Subtract them
  • Multiply them
  • Divide them
  • Compare them

• Example: $10 + 7 = 17$

• …so simple to add in binary that we can build circuits to do it!

• subtraction just as you would in decimal

• Comparison: How do you tell if $X > Y$?
What if too big?

• Binary bit patterns above are simply **representatives** of numbers. Abstraction! Strictly speaking they are called “numerals”.

• Numbers really have an $\infty$ number of digits
  • with almost all being same (00…0 or 11…1) except for a few of the rightmost digits
  • Just don’t normally show leading digits

• If result of add (or -, *, /) cannot be represented by these rightmost HW bits, **overflow** is said to have occurred.

\[
\begin{array}{cccccccc}
00000 & 00001 & 00010 & & 11110 & 11111 \\
\end{array}
\]
How to Represent Negative Numbers?
(C’s unsigned int, C99’s uintN_t)

• So far, unsigned numbers

00000 00001 ... 01111 10000 ... 11111

• Obvious solution: define leftmost bit to be sign!
  • 0 \rightarrow + \quad 1 \rightarrow –
  • Rest of bits can be numerical value of number

• Representation called sign and magnitude

00000 00001 ... 01111

11111 ... 10001 10000

META: Ain’t no free lunch

Dr Dan Garcia
Shortcomings of sign and magnitude?

- Arithmetic circuit complicated
  - Special steps depending whether signs are the same or not

- Also, **two** zeros
  - $0x00000000 = +0_{\text{ten}}$
  - $0x80000000 = -0_{\text{ten}}$
  - What would two 0s mean for programming?

- Also, incrementing “binary odometer”, sometimes increases values, and sometimes decreases!

- Therefore sign and magnitude abandoned
Another try: complement the bits

• Example: \( 7_{10} = 00111_2 \quad -7_{10} = 11000_2 \)

• Called **One's Complement**

• Note: positive numbers have leading 0s, negative numbers have leading 1s.

• What is -00000 ? Answer: 11111

• How many positive numbers in N bits?

• How many negative numbers?

Dr Dan Garcia
Shortcomings of One’s complement?

• Arithmetic still a somewhat complicated.

• Still two zeros
  - $0x00000000 = +0_{\text{ten}}$
  - $0xFFFFFFFFF = -0_{\text{ten}}$

• Although used for a while on some computer products, one’s complement was eventually abandoned because another solution was better.
Standard Negative # Representation

- Problem is the negative mappings “overlap” with the positive ones (the two 0s). Want to shift the negative mappings left by one.
  - Solution! For negative numbers, complement, then add 1 to the result

- As with sign and magnitude, & one’s complement
  - 0s ⇒ positive, leading
  - 1s ⇒ negative

- 000000...xxx is ≥ 0, 111111...xxx is < 0
- except 1...1111 is -1, not -0 (as in sign & mag.)

- This representation is Two’s Complement
- This makes the hardware simple!
  (C’s int, aka a “signed integer”)
  (Also C’s short, long, long, ..., C99’s intN_t)
Two’s Complement Formula

- Can represent positive and negative numbers in terms of the bit value times a power of 2:
  - \( d_{31} \times -(2^{31}) + d_{30} \times 2^{30} + \ldots + d_{2} \times 2^{2} + d_{1} \times 2^{1} + d_{0} \times 2^{0} \)

- Example: \( 1101_{\text{two}} \) in a nibble?
  - \( = 1 \times -(2^{3}) + 1 \times 2^{2} + 0 \times 2^{1} + 1 \times 2^{0} \)
  - \( = -2^{3} + 2^{2} + 0 + 2^{0} \)
  - \( = -8 + 4 + 0 + 1 \)
  - \( = -8 + 5 \)
  - \( = -3_{\text{ten}} \)

Example: -3 to +3 to -3 (again, in a nibble):

\[
\begin{align*}
x & : 1101_{\text{two}} \\
x' & : 0010_{\text{two}} \\
+1 & : 0011_{\text{two}} \\
()' & : 1100_{\text{two}} \\
+1 & : 1101_{\text{two}}
\end{align*}
\]
2’s Complement Number “line”: N = 5

- $2^{N-1}$ non-negatives
- $2^{N-1}$ negatives
- one zero
- how many positives?

Binary odometer

Dr Dan Garcia
Bias Encoding: $N = 5$ (bias = -15)

• # = unsigned + bias

• Bias for $N$ bits chosen as $-(2^{N-1}-1)$

• one zero

• how many positives?

Binary odometer

Dr Dan Garcia
How best to represent -12.75?

a) 2s Complement (but shift binary pt)
b) Bias (but shift binary pt)
c) Combination of 2 encodings
d) Combination of 3 encodings
e) We can’t

Shifting binary point means “divide number by some power of 2. E.g.,
11_{10} = 1011.0_2 \rightarrow 10.110_2 = (11/4)_{10} = 2.75_{10}
And in summary...

- We represent “things” in computers as particular bit patterns: \( N \) bits \( \Rightarrow 2^N \) things
- These 5 integer encodings have different benefits; 1s complement and sign/mag have most problems.
  - unsigned (C99’s `uintN_t`):
    
    \[
    \begin{align*}
    \text{00000} & \quad \text{00001} & \ldots & \text{01111} & \text{10000} & \ldots & \text{11111} \\
    \text{10000} & \ldots & \text{11110} & \text{11111}
    \end{align*}
    \]
  - 2’s complement (C99’s `intN_t`) universal, learn!
    
    \[
    \begin{align*}
    \text{00000} & \quad \text{00001} & \ldots & \text{01111}
    \end{align*}
    \]
- Overflow: numbers \( \propto \) computers finite, errors!

META: We often make design decisions to make HW simple

META: Ain’t no free lunch

Dr Dan Garcia
REFERENCE: Which base do we use?

• **Decimal**: great for humans, especially when doing arithmetic

• **Hex**: if human looking at long strings of binary numbers, its much easier to convert to hex and look 4 bits/symbol
  - Terrible for arithmetic on paper

• **Binary**: what computers use; you will learn how computers do +, -, *, /
  - To a computer, numbers always binary
  - Regardless of how number is written:
    - $32_{\text{ten}} = 32_{10} = 0x20 = 100000_{2} = 0b100000$
  - Use subscripts “ten”, “hex”, “two” in book, slides when might be confusing

Dr Dan Garcia
# Two’s Complement for N=32

<table>
<thead>
<tr>
<th>Binary</th>
<th>Two’s Complement</th>
<th>Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000 ... 0000 0000 0000 0000&lt;sub&gt;two&lt;/sub&gt;</td>
<td>0&lt;sub&gt;ten&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>0000 ... 0000 0000 0000 0001&lt;sub&gt;two&lt;/sub&gt;</td>
<td>1&lt;sub&gt;ten&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>0000 ... 0000 0000 0000 0010&lt;sub&gt;two&lt;/sub&gt;</td>
<td>2&lt;sub&gt;ten&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>0111 ... 1111 1111 1111 1101&lt;sub&gt;two&lt;/sub&gt;</td>
<td>2,147,483,645&lt;sub&gt;ten&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>0111 ... 1111 1111 1111 1110&lt;sub&gt;two&lt;/sub&gt;</td>
<td>2,147,483,646&lt;sub&gt;ten&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>0111 ... 1111 1111 1111 1111&lt;sub&gt;two&lt;/sub&gt;</td>
<td>2,147,483,647&lt;sub&gt;ten&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>1000 ... 0000 0000 0000 0000&lt;sub&gt;two&lt;/sub&gt;</td>
<td>−2,147,483,648&lt;sub&gt;ten&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>1000 ... 0000 0000 0000 0001&lt;sub&gt;two&lt;/sub&gt;</td>
<td>−2,147,483,647&lt;sub&gt;ten&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>1000 ... 0000 0000 0000 0010&lt;sub&gt;two&lt;/sub&gt;</td>
<td>−2,147,483,646&lt;sub&gt;ten&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>1111 ... 1111 1111 1111 1101&lt;sub&gt;two&lt;/sub&gt;</td>
<td>−3&lt;sub&gt;ten&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>1111 ... 1111 1111 1111 1110&lt;sub&gt;two&lt;/sub&gt;</td>
<td>−2&lt;sub&gt;ten&lt;/sub&gt;</td>
<td></td>
</tr>
<tr>
<td>1111 ... 1111 1111 1111 1111&lt;sub&gt;two&lt;/sub&gt;</td>
<td>−1&lt;sub&gt;ten&lt;/sub&gt;</td>
<td></td>
</tr>
</tbody>
</table>

- One zero; 1st bit called **sign bit**
- 1 “extra” negative: no positive 2,147,483,648<sub>ten</sub>
Two’s comp. shortcut: Sign extension

• Convert 2’s complement number rep. using n bits to more than n bits

• Simply replicate the most significant bit (sign bit) of smaller to fill new bits
  • 2’s comp. positive number has infinite 0s
  • 2’s comp. negative number has infinite 1s
  • Binary representation hides leading bits; sign extension restores some of them

• \((-4_{\text{ten}})\) 16-bit to 32-bit:

\[
\begin{array}{cccccccccccc}
1111 & 1111 & 1111 & 1100_{\text{two}} \\
1111 & 1111 & 1111 & 1111 & 1111 & 1111 & 1111 & 1100_{\text{two}}
\end{array}
\]