CS 31: Intro to Systems
Binary Representation

Martin Gagné
Swarthmore College
January 19, 2017
Today

• Number systems and conversion

• Data storage
How many unique values can we represent with 9 bits?

- One bit: two values (0 or 1)
- Two bits: four values (00, 01, 10, or 11)
- Three bits: eight values (000, 001, …, 110, 111)

A. 18
B. 81
C. 256
D. 512
E. Some other number of values.
# How many values?

<table>
<thead>
<tr>
<th>N bits:</th>
<th>(2^N) values</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 bit:</td>
<td>0 1</td>
</tr>
<tr>
<td>2 bits:</td>
<td>00 01 10 11 10 11</td>
</tr>
<tr>
<td>3 bits:</td>
<td>000 001 010 011 100 101 110 111</td>
</tr>
<tr>
<td>4 bits:</td>
<td>0000 0001 0010 0011 0100 0101 0110 0111 16 values</td>
</tr>
<tr>
<td></td>
<td>1000 1001 1010 1011</td>
</tr>
<tr>
<td></td>
<td>1100 1101 1110 1111</td>
</tr>
</tbody>
</table>

**N bits**

- can represent number from 0 to \(2^N - 1\)
From last time:

- Decimal number system (Base 10)
- Sequence of digits in range [0, 9]

64024

Digit #4    Digit #0
Positional Notation

• The meaning of a digit depends on its position in a number.

A number, written as the sequence of digits $d_n d_{n-1} \ldots d_2 d_1 d_0$ in base $b$ represents the value $d_n \times b^n + d_{n-1} \times b^{n-1} + \ldots + d_2 \times b^2 + d_1 \times b^1 + d_0 \times b^0$. 
Decimal: Base 10

• Used by humans

A number, written as the sequence of digits $d_n d_{n-1} \ldots d_2 d_1 d_0$ where $d$ is in \{0,1,2,3,4,5,6,7,8,9\} represents the value:

$$d_n \times 10^n + d_{n-1} \times 10^{n-1} + \ldots + d_2 \times 10^2 + d_1 \times 10^1 + d_0 \times 10^0$$

64024 =
$$6 \times 10^4 + 4 \times 10^3 + 0 \times 10^2 + 2 \times 10^1 + 4 \times 10^0$$

60000 + 4000 + 0 + 20 + 4
Binary: Base 2

• Used by computers

A number, written as the sequence of digits $d_n \ldots d_2 d_1 d_0$ where $d$ is in \{0,1\}, represents the value

$$d_n \cdot 2^n + d_{n-1} \cdot 2^{n-1} + \ldots + d_2 \cdot 2^2 + d_1 \cdot 2^1 + d_0 \cdot 2^0$$
Converting Binary → Decimal

• Two methods:
  • powers of two and addition
  • multiplication by two plus position bit
Method 1: powers of two and addition

E.g. start with binary number 100101

\[ 1 \times 2^5 + 0 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 0 \times 2^1 + 1 \times 2^0 \]

\[ = 1 \times 32 + 1 \times 4 + 1 \times 1 \]

\[ = 37 \]
Method 2: multiplication by two plus position bit

E.g. start with binary number 100101
Converting Decimal → Binary

• Two methods:
  • division by two remainder
  • powers of two and subtraction
Method 1: decimal value $D$, binary result $b$ ($b_i$ is $i$th digit):

\[
i = 0 \\
\text{while } (D > 0) \\
\quad \text{if } D \text{ is odd} \\
\quad \quad \text{set } b_i \text{ to } 1 \\
\quad \text{if } D \text{ is even} \\
\quad \quad \text{set } b_i \text{ to } 0 \\
\quad \text{i++} \\
\text{D = D/2}
\]

Example: Converting 105

\[
\begin{array}{c|c|c}
D & b_i & \text{Example: } D = 105 \\
52 & 0 & b_1 = 0 \\
26 & 0 & b_2 = 0 \\
13 & 1 & b_3 = 1 \\
6 & 0 & b_4 = 0 \\
3 & 1 & b_5 = 1 \\
1 & 1 & b_6 = 1 \\
0 & 0 & b_7 = 0 \\
\end{array}
\]

\[
105 = 01101001
\]
Method 2

\[2^0 = 1, \ 2^1 = 2, \ 2^2 = 4, \ 2^3 = 8, \ 2^4 = 16, \ 2^5 = 32, \ 2^6 = 64, \ 2^7 = 128\]

• To convert 105:
  • Find largest power of two that’s less than 105 (64)
  • Subtract 64 (105 – 64 = 41), put a 1 in \(d_6\)
  • Subtract 32 (41 – 32 = 9), put a 1 in \(d_5\)
  • Skip 16, it’s larger than 9, put a 0 in \(d_4\)
  • Subtract 8 (9 – 8 = 1), put a 1 in \(d_3\)
  • Skip 4 and 2, put a 0 in \(d_2\) and \(d_1\)
  • Subtract 1 (1 – 1 = 0), put a 1 in \(d_0\) (Done)

\[\begin{array}{cccccccc}
1 & 1 & 0 & 1 & 0 & 0 & 0 & 1
\end{array} \]
What is the value of 357 in binary?

A. 101100011
B. 101100101
C. 101101001
D. 101110101
E. 110100101

$2^0 = 1, \; 2^1 = 2, \; 2^2 = 4, \; 2^3 = 8, \; 2^4 = 16, \; 2^5 = 32, \; 2^6 = 64, \; 2^7 = 128$
Other (common) number systems.

- Base 10: decimal
- Base 2: binary
- Base 16: hexadecimal (memory addresses)
- Base 8: octal (ok, maybe not so common...)
- Base 64: (Commonly used on the Internet, e.g. email attachments).
- Base 60 (hours:minutes.seconds, ancient Babylon)
Hexadecimal: Base 16

• Indicated by prefacing number with 0x

• A number, written as the sequence of digits $d_n d_{n-1} \ldots d_2 d_1 d_0$ where $d$ is in $\{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}$, represents the value
  $d_n \times 16^n + d_{n-1} \times 16^{n-1} + \ldots + d_2 \times 16^2 + d_1 \times 16^1 + d_0 \times 16^0$
Hexadecimal: Base 16

- Indicated by prefacing number with 0x
- Like binary, base is power of 2
  - Fewer digits to represent same value
- Each digit is a “nibble”, or half a byte

- A number, written as the sequence of digits $d_n d_{n-1} \ldots d_2 d_1 d_0$ where $d$ is in \{0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F\}, represents the value
  $$d_n \times 16^n + d_{n-1} \times 16^{n-1} + \ldots + d_2 \times 16^2 + d_1 \times 16^1 + d_0 \times 16^0$$
Each hex digit is a “nibble”

Hex digit: 16 values, $2^4 = 16$ -> 4 bits / digit

0x 1 B 7

Four-bit value: 1
Four-bit value: B (decimal 11)
Four-bit value: 7

In binary: 0001 1011 0111
1 B 7
Converting hex and binary

• A group of four binary digits maps to one hex digit.

0x 48C1

4 → 0100
8 → 1000
C → 1100 (12)
1 → 0001

0x 48C1 = 0b 0100 1000 1100 0001
Converting hex and binary

• A group of four binary digits maps to one hex digit.

0b 110 1010 1101 0101

0101 → 5
1101 → D (13)
1010 → A (10)
0110 → 6

0b 110 1010 1101 0101 = 0x 6AD5
What is 0b101100111011 in hex?

a) 0xb3b
b) 0x59d
c) 0xc5c
d) 0x37b
e) 0x5473
Converting Hexadecimal-> Decimal

• Three methods:
  • powers of 16 and addition
  • multiplication by 16 plus position nibble
Converting Hexadecimal-> Decimal

• Three methods:
  • powers of 16 and addition
  • multiplication by 16 plus position nibble

• Just go through binary!
Converting Decimal -> Hexadecimal

• Three methods:

  • division by 16 remainder
  • powers of 16 and subtraction

• Just go through binary!
What is the value of 0x1B7 in decimal?

A. 397

B. 409

C. 419

D. 437

E. 439

$16^2 = 256$
Unsigned Integers

- Suppose we had one byte
  - Can represent $2^8$ (256) values
  - If unsigned (strictly non-negative): 0 – 255

252 = 11111100
253 = 11111101
254 = 11111110
255 = 11111111
Unsigned Integers

• Suppose we had one byte
  • Can represent $2^8$ (256) values
  • If unsigned (strictly non-negative): 0 – 255

252 = 11111100
253 = 11111101
254 = 11111110
255 = 11111111

What if we add one more?

Car odometer “rolls over”.
Unsigned Integers

• Suppose we had one byte
  • Can represent $2^8$ (256) values
  • If unsigned (strictly non-negative): 0 – 255

252 = 11111100
253 = 11111101
254 = 11111110
255 = 11111111

What if we add one more?

Modular arithmetic: Here, all values are modulo 256.
Suppose we want to support negative values too (-127 to 127). Where should we put -1 and -127 on the circle? Why?

C: Put them somewhere else.
Signed Magnitude

• One bit (usually left-most) signals:
  • 0 for positive
  • 1 for negative

For one byte:

  1 = 00000001,    -1 = 10000001

Pros: Negation is very simple!
Signed Magnitude

• One bit (usually left-most) signals:
  • 0 for positive
  • 1 for negative

For one byte:

0 = 00000000

What about 10000000?

Major con: Two ways to represent zero.
Floating Point Representation

1 bit for sign  sign |  exponent |  fraction |
8 bits for exponent
23 bits for precision

\[
\text{value} = (-1)^{\text{sign}} \times 1.\text{fraction} \times 2^{(\text{exponent} - 127)}
\]

let's just plug in some values and try it out

0x40ac49ba: 0 10000001 01011000100100110111010
sign = 0 exp = 129 fraction = 2902458

\[
= 1 \times 1.2902458 \times 2^2 = 5.16098
\]

I don't expect you to memorize this
Two’s Complement

• Borrow nice property from number line:

```
-1  0  1
```

Only one instance of zero!
Implies: -1 and 1 on either side of it.
Two’s Complement

• Borrow nice property from number line:

Only one instance of zero!
Implies: -1 and 1 on either side of it.
Two’s Complement

The Encoding comes from Definition of the 2’s complement of a number:

2’s complement of an N bit number, x, is its complement with respect to $2^N$

Can use this to find the bit encoding, y, for the negation of x:

For N bits, $y = 2^N - x$

4 bit examples:

<table>
<thead>
<tr>
<th>X</th>
<th>-X</th>
<th>$2^4 - X$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0000</td>
<td>0000</td>
<td>$10000 - 0000 = 0000$ (only 4 bits)</td>
</tr>
<tr>
<td>0001</td>
<td>1111</td>
<td>$10000 - 0001 = 1111$</td>
</tr>
<tr>
<td>0010</td>
<td>1110</td>
<td>$10000 - 0010 = 1110$</td>
</tr>
<tr>
<td>0011</td>
<td>1101</td>
<td>$10000 - 0011 = 1101$</td>
</tr>
</tbody>
</table>
Two’s Complement

• Only one value for zero
• With N bits, can represent the range:
  • $-2^{N-1}$ to $2^{N-1} – 1$
• First bit still designates positive (0) /negative (1)

• Negating a value is slightly more complicated:
  $1 = 00000001$,       $-1 = 11111111$

From now on, unless we explicitly say otherwise, we’ll assume all integers are stored using two’s complement! This is the standard!
Two’s Complement

• Each two’s complement number is now:

\[-2^{n-1}d_{n-1} + 2^{n-2}d_{n-2} + \ldots + 2^1d_1 + 2^0d_0\]

Note the negative sign on just the first digit. This is why first digit tells us negative vs. positive.
What is 11001 in decimal?

- Each two’s complement number is now:
  \[ -2^{n-1}d_{n-1} + 2^{n-2}d_{n-2} + \ldots + 2^1d_1 + 2^0d_0 \]

A. -2
B. -7
C. -9
D. -25
Negative Two’s Complement to Decimal

• Two methods:
  • powers of two and addition (largest power is negative)

• trick:
  • flip all the bits
  • convert to decimal
  • add one
  • add minus sign
Negative Two’s Complement to Decimal

11001

• flip all the bits: 00110
• convert to decimal: 6
• add one : 7
• add minus sign: -7
Negative Decimal to Two’s Complement

• Two methods:
  • powers of two and subtraction (largest power is negative)

• trick:
  • remove minus sign
  • subtract one
  • convert to binary
  • flip all the bits
Negative Decimal to Two’s Complement

-11 (assume 5 bit two’s complement)

• remove minus sign: 11
• subtract one: 10
• convert to binary: 01010
• flip all the bits: 10101
What is -7 in two’s complement?

(assume 5 bits two’s complement)

a) 11000
b) 11001
c) 10111
d) 10110
e) ﾑ(ﾂ)〇/
Data Storage
You can view binary file contents

**xxd (or hexdump –C) to view binary file values:**

```
xxd a.out       # a binary executable file

Address:  value of the next 16 bytes in memory
00000000: 7f45 4c46 0201 0100 0000 0000 0000 0000
00000010: 0200 3e00 0100 0000 3007 4000 0000 0000
00000020: 4000 0000 0000 0000 084d 0000 0000 0000
...

xxd myprog.c    # binary ascii encoding of C source:

00000000: 2369 6e63 6c75 6465 3c73 7464 696f 2e68
  #i   nc   lu   de   <s   td   io   .h
00000010: 3e0a 696e 7420 6d61 696e 2829 207b 0a20
  >
  in   t    ma   in   ()    {  
...
```
Data Storage

• Lots of technologies out there:
  • Magnetic (hard drive, floppy disk)
  • Optical (CD / DVD / Blu-Ray)
  • Electronic (RAM, registers, …)

• Focus on electronic for now
  • We’ll see (and build) digital circuits soon

• Relatively easy to differentiate two states
  • Voltage present / light present / magnetized up
  • Voltage absent / light absent / magnetized down
Bits and Bytes

• Bit: a 0 or 1 value (binary)
  • 1: the presence of voltage (high voltage)
  • 0: the absence of voltage (low voltage)

• Byte: 8 bits, the smallest addressable unit
  Memory:  01010101  10101010  00001111  …  
  (address) [0]  [1]  [2]  …

• Other names:
  • 4 bits: nibble
  • “Word”: Depends on system, often 4 bytes
A Quick Note on Multiples of Bytes

In this course (and information tech. in general)

<table>
<thead>
<tr>
<th>Byte Unit</th>
<th>Symbol</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kilobyte</td>
<td>kB</td>
<td>(2^{10})</td>
</tr>
<tr>
<td>Megabyte</td>
<td>MB</td>
<td>(2^{20})</td>
</tr>
<tr>
<td>Gigabyte</td>
<td>GB</td>
<td>(2^{30})</td>
</tr>
<tr>
<td>Terabyte</td>
<td>TB</td>
<td>(2^{40})</td>
</tr>
<tr>
<td>Petabyte</td>
<td>PB</td>
<td>(2^{50})</td>
</tr>
<tr>
<td>Exabyte</td>
<td>EB</td>
<td>(2^{60})</td>
</tr>
<tr>
<td>Zetabyte</td>
<td>ZB</td>
<td>(2^{70})</td>
</tr>
<tr>
<td>Yottabyte</td>
<td>YB</td>
<td>(2^{80})</td>
</tr>
</tbody>
</table>
A Quick Note on Multiples of Bytes

However, in SI and IEC *(which we do not follow here)*

<table>
<thead>
<tr>
<th>Kilobyte</th>
<th>kB</th>
<th>10^3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Megabyte</td>
<td>MB</td>
<td>10^6</td>
</tr>
<tr>
<td>Gigabyte</td>
<td>GB</td>
<td>10^9</td>
</tr>
<tr>
<td>Terabyte</td>
<td>TB</td>
<td>10^{12}</td>
</tr>
<tr>
<td>Petabyte</td>
<td>PB</td>
<td>10^{15}</td>
</tr>
<tr>
<td>Exabyte</td>
<td>EB</td>
<td>10^{18}</td>
</tr>
<tr>
<td>Zetabyte</td>
<td>ZB</td>
<td>10^{21}</td>
</tr>
<tr>
<td>Yottabyte</td>
<td>YB</td>
<td>10^{24}</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kibibyte</th>
<th>KiB</th>
<th>2^{10}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mebibyte</td>
<td>MiB</td>
<td>2^{20}</td>
</tr>
<tr>
<td>Gibibyte</td>
<td>GiB</td>
<td>2^{30}</td>
</tr>
<tr>
<td>Tebibyte</td>
<td>TiB</td>
<td>2^{40}</td>
</tr>
<tr>
<td>Pebibyte</td>
<td>PiB</td>
<td>2^{50}</td>
</tr>
<tr>
<td>Exbibyte</td>
<td>EiB</td>
<td>2^{60}</td>
</tr>
<tr>
<td>Zebibyte</td>
<td>ZiB</td>
<td>2^{70}</td>
</tr>
<tr>
<td>Yobibyte</td>
<td>YiB</td>
<td>2^{80}</td>
</tr>
</tbody>
</table>
Up Next

- Binary arithmetic
  - adding binary numbers
  - subtracting binary numbers
- Bitwise operations
- Characters and strings in C