Representing Data

Dov Kruger

Department of Electrical and Computer Engineering
Rutgers University

October 1, 2024

Digital Representation of Information

- This module describes how to represent data as bits
- By the end you will be able to
  - Represent unsigned and signed Integers
  - Convert binary to/from decimal
  - Convert binary to/from hexadecimal
  - Compute twos complement numbers
  - Identify when integers overflow
  - Identify how to store text on computers
  - Identify how bits are interpreted as floating point

It's all Bits

- With binary data, it's all context
- It is not obvious what bits represent
- In this module you will lean how the computer represents
- whole numbers, fractions, and text

Bits and Bytes

- Bit: Smallest unit of data (0,1)
- Byte: Group of 8 bits
- Type of data completely depends on context

- 2 bits: 00, 01, 10, 11
- 3 bits: 000, 001, 010, 011, 100, 101, 110, 111
- n bits $= 2^n$ combinations
- 8 bits $= 2^8 = 256$ combinations
- 16 bits $= 2^16 = 65536$ combinations
- 32 bits $= 2^32 = 4294967296$ combinations

Different Binary Data Representations

- Unsigned Integers
- Signed Integers
- ASCII text
- Unicode
- Fixed Point Fractions
- Floating Point

Integer Representations

- Integers come in different sizes (8, 16, 32, 64 bits)
- Signed/Unsigned
- Little-endian/Big-endian

Representing 3 bit Integers

bits	unsigned	signed
000	0	0
001	1	1
010	2	2
011	3	3
100	4	-4
101	5	-3
110	6	-2
111	7	-1

What happens if we add 1 to 7?

What happens if we subtract 1 from 0?

Twos Complement Arithmetic

- Consider just 8 bit number for simplicity
- First bit is sign 0 = positive, 1 = negative
- To negate a number
  - invert all bits
  - add 1
  - the resulting number is the negative of the original
  - Example: $5 = 00000101 \rightarrow 11111010 \rightarrow 11111011 = -5$
  - Example: $17 = 00010001 \rightarrow 11101110 \rightarrow 11101111 = -17$
  - Example: $-11 = 11110101 \rightarrow 00001010 \rightarrow 00001011 = 11$
  - Example: $-128 = 10000000 \rightarrow 01111111 \rightarrow 00000000 = -128$

Overflow and Underflow

- Overflow is when the result of a computation is too large to fit
- Underflow is the same in the negative direction
- Example: given 3-bit unsigned
  - $3 + 5 = 1000 = 8 = 000 = 0$
  - $4 + 6 = 1010 = 10 = 010 = 2$
  - $4 - 5 = 111 = 7$
- Example: given 3-bit signed
  - $3 + 2 = 101 = -2$
  - $2 - 3 = 111 = -1$
  - $3 + 1 = 100 = -4$

Integer Data Types

bits		minval	maxval
8	signed	-128	127
8	unsigned	0	255
16	signed	-32768	32767
16	unsigned	0	65535
32	signed	-2147483648	2147483647
32	unsigned	0	4294967295
64	signed	-9223372036854775808	9223372036854775807
64	unsigned	0	18446744073709551615

Overflow and Underflow

- When a result is too large, store only the low n bits
- Example: 3 bits
  - 3 + 3 = 6 (no overflow)
  - 4 + 4 = 8 (too big) = 0 (overflow)
  - 3 - 2 = 1 (no overflow)
  - 3 - 4 = -1 = 7 (underflow)

Base 16: Hexadecimal

Hex	bits	—	Hex	bits
0	0000		8	1000
1	0001		9	1001
2	0010		A	1010
3	0011		B	1011
4	0100		C	1100
5	0101		D	1101
6	0110		E	1110
7	0111		F	1111

Encoding a byte in Hexadecimal

- D9 $=$ 11011001
- AF $=$ 10101111
- 8C $=$ 10001100

Converting Between Binary and Decimal

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
128	64	32	16	8	4	2	1

- `Method 1: sum powers of 2`
- $10010010 = 128 + 16 + 2 = 146$
- `Method 2: start from left`
    - `Start with 1`
    - `For each digit, multiply by 2`
    - `If the digit is 1, add 1`
    - `Example:` $(((((1 * 2) * 2) * 2) + 1) * 2 * 2 * 2 + 1) * 2 = 146$

ASCII Encoding

- ASCII: American Standard Code for Information Interchange
- Maps characters to binary values
- ASCII table overview `https://www.ascii-code.com/`
- Example: 'A' $=$ 01000001 $=$ 65
- Example: 'B' $=$ 01000010 $=$ 66
- Example: 'a' $=$ 01100001 $=$ 97

Unicode UTF-8 Encoding

- Unicode is a standard for representing text of all languages
- Originally fit into 16 bits
- Now requires 19 bits because of emoji
- UTF-8 is a variable length encoding
- Useful when most of the text is English
- `https://symbl.cc/en/unicode-table/`
- `https://r12a.github.io/app-conversion/`

1st byte	2nd byte	3rd byte	4th byte	19-bit value
0xxxxxxx				000000000000000xxxxxx
110yyyyy	10xxxxxx			0000000000yyyyyxxxxxx
1110zzzz	10yyzzzz	10yyyyyy	10xxxxxx	00000zzzzyyyyyyxxxxxx
11110uuu	10uuzzzz	10yyyyyy	10xxxxxx	uuuuuzzzzyyyyyyxxxxxx

Unicode UTF-8 Encoding

Let's decode the following bytes

```
00000000: 74 65 73 74 0a ce b5 cf 85 cf 87 ce b1 cf 81 ce b
00000010: cf 83 cf 84 cf 8e 0a
```

Fixed Point Fractions

$$2^3 \mid 2^2 \mid 2^1 \mid 2^0 \mid . \mid 2^{-1} \mid 2^{-2}$$

- Fractions in binary are represented as negtive powers
- $\frac{1}{2}, \frac{1}{4}, \frac{1}{8}$
- Examples

$$\begin{aligned}
101.1 &= 4 + 1 + \tfrac{1}{2} = 5.5 \\
1.01 &= 1 + \tfrac{1}{4} = 1.25 \\
110.11 &= 4 + 2 + \tfrac{1}{2} + \tfrac{1}{4} \\
1001.001 &= 8 + 1 + \tfrac{1}{8}
\end{aligned}$$

Floating Point

- Fixed point represents fractions, but only a single size
- Floating point can represent values wildly different values
- IEEE-754

Single precision	32 bits
Double precision	64 bits
Quad Precision	128 bits (not yet in hardware)
Half Precision	16 bits (GPUs)
fp8	8 bits (GPUs)

float (32-bit) Representation

value	seeeeeeeemmmmmmmmmmmmmmmmmmmmmmm	hex
1.0	00011111100000000000000000000000	3f800000
2.0	01000000000000000000000000000000	40000000
1.5	00011111110000000000000000000000	3fc00000
0.1	00111101110011001100110011001101	3dcccccd
1234567.8	01001001100101101011010000111110	4996b43e
1.2345678e+10	01010000001101111111011100000110	5037f706
6.023e+23	01100110111111110001010101111111	66ff157f
6.674e-11	00101110100100101100001101001000	2e92c348
-1234567.8	11001001100101101011010000111110	c996b43e
1.234e-30	00001101110010000011101001011011	0dc83a5b
NaN	01111111110000000000000000000000	7fc00000
Infinity	01111111100000000000000000000000	7f800000
-Infinity	11111111100000000000000000000000	ff800000

Unsigned Integers

-
-
-

Signed Integer Representations

-
-
-

American Standard Code for Information Interchange (ASCII)

- Represent characters with 7-bit codes
-
-
-
-

UTF-8 Text

-
-
-
-
-

Fixed Point Representations

Floating Point Representation

-
-

Roundoff Errors

-
-

Limits of Floating Point

-
-