Data Representation: bits, bytes, integers

Yipeng Huang

Rutgers University

Feburary 11, 2021

Table of contents

Programming assignments

Programming assignment 1

- Due date extended to: 11:59pm Monday, February 15.
- Take good advantage of this opportunity.
- Familiarity with C is a vital foundation for this class and future classes.
- Code review discussion during week of February 22 - February 26.

Programming assignment 2

- Released later today Thursday, February 11.
- Due after two weeks: February 25.
- Same techniques in programming C.
- Review of graph algorithms.

Looking ahead

Lecture plan

1. Today, Thursday, 2/11: Data representation of integers.
2. Tuesday, 2/16: Data representation of floating point numbers.
3. Thursday, 2/18: Data representation of floating point numbers.

Reading assignment

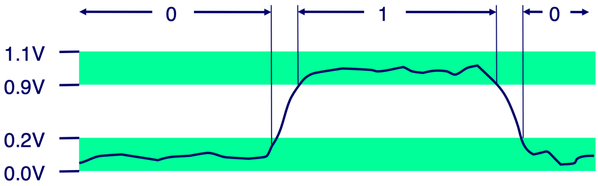
- Computer Systems: A Programmer's Perspective Chapter 2.

Table of contents

Why binary

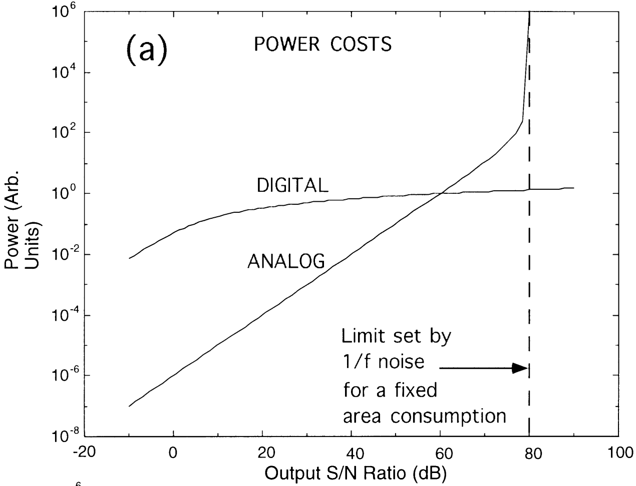
Everything is bits

- **Each bit is 0 or 1**
- **By encoding/interpreting sets of bits in various ways**
  - Computers determine what to do (instructions)
  - … and represent and manipulate numbers, sets, strings, etc…
- **Why bits? Electronic Implementation**
  - Easy to store with bistable elements
  - Reliably transmitted on noisy and inaccurate wires



Why binary

Figure: Rahul Sarpeshkar. Analog Versus Digital: Extrapolating from Electronics to Neurobiology. 1998.



Why binary

Digital encodings

Each doubling of either precision or range only needs one additional bit.

Analog encodings

Each doubling of either precision or range needs doubling of either area or power.

Decimal, binary, octal, and hexadecimal

Decimal	Binary	Octal	Hexadecimal
0	0b0000	0o0	0x0
1	0b0001	0o1	0x1
2	0b0010	0o2	0x2
3	0b0011	0o3	0x3
4	0b0100	0o4	0x4
5	0b0101	0o5	0x5
6	0b0110	0o6	0x6
7	0b0111	0o7	0x7

Decimal	Binary	Octal	Hexadecimal
8	0b1000	0o10	0x8
9	0b1001	0o11	0x9
10	0b1010	0o12	0xA
11	0b1011	0o13	0xB
12	0b1100	0o14	0xC
13	0b1101	0o15	0xD
14	0b1110	0o16	0xE
15	0b1111	0o17	0xF

In C, format specifiers for printf() and fscanf():

1. decimal: '%d'
2. binary: none
3. octal: '%o'
4. hexadecimal: '%x'

Decimal, binary, octal, and hexadecimal

How to represent the range of unsigned char in each?

Unsigned char is one byte, 8 bits.

1. decimal: 0 to 255
2. binary: 0b0 to 0b11111111
3. octal: 0 to 0o377 (group by 3 bits)
4. hexadecimal: 0x00 to 0xFF (group by 4 bits)

Bitwise operations

Why are bitwise operations important?

- Network and UNIX settings using bit masks (e.g., umask)
- Hardware and microcontroller programming (e.g., Arduinos)
- Instruction set architecture encodings (e.g., ARM, x86)

Bitwise operations

~: bitwise NOT

unsigned char a = 128

$$
\begin{aligned}
a &= 0b1000\_0000 \\
\tilde{}a &= \tilde{}0b1000\_0000 \\
&= 0b0111\_1111 \\
&= 127
\end{aligned}
$$

b	~ b
0	1
1	0

Bitwise operations

&: bitwise AND

$$
\begin{aligned}
3\&1 &= 0b11\&0b01 \\
&= 0b01 \\
&= 1
\end{aligned}
$$

a	b	a & b
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise operations

| : bitwise OR

$$\begin{aligned} 3|1 &= 0b11|0b01 \\ &= 0b11 \\ &= 3 \end{aligned}$$

$$\begin{aligned} 2|1 &= 0b10|0b01 \\ &= 0b11 \\ &= 3 \end{aligned}$$

a	b	a | b
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise operations

^: bitwise XOR

$$
\begin{aligned}
3 \wedge 1 &= 0b11 \wedge 0b01 \\
&= 0b10 \\
&= 2
\end{aligned}
$$

a	b	a ^ b
0	0	0
0	1	1
1	0	1
1	1	0

Don't confuse bitwise operators with logical operators

Bitwise operators

- ~
- &
- |
- ^

Logical operators

- !
- &&
- ||
- != (for bool type)

Representing characters

USASCII code chart

b_4	b_3	b_2	b_1	Row ↓ / Column →	0 ($b_7b_6b_5$ = 000)	1 (001)	2 (010)	3 (011)	4 (100)	5 (101)	6 (110)	7 (111)
0	0	0	0	0	NUL	DLE	SP	0	@	P	`	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	"	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	'	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	10	LF	SUB	*	:	J	Z	j	z
1	0	1	1	11	VT	ESC	+	;	K	[	k	{
1	1	0	0	12	FF	FS	,	<	L	\	l	|
1	1	0	1	13	CR	GS	-	=	M	]	m	}
1	1	1	0	14	SO	RS	.	>	N	^	n	~
1	1	1	1	15	SI	US	/	?	O	_	o	DEL

Figure: ASCII character set. Image credit Wikimedia

Table of contents

Representing negative and signed integers

Ways to represent negative numbers

1. Sign magnitude
2. 1′s complement
3. 2′s complement

Representing negative and signed integers

Sign magnitude

Flip leading bit.

Representing negative and signed integers

1′s complement

- Flip all bits
- Addition in 1′s complement is sound
- In this encoding there are 2 encodings for 0
- -0: 0b1111
- +0: 0b0000

Representing negative and signed integers

2's complement

signed char	weight in decimal
00000001	1
00000010	2
00000100	4
00001000	8
00010000	16
00100000	32
01000000	64
10000000	-128

Table: Weight of each bit in a signed char type

- what is the most positive value you can represent? 127
- what is the most negative value you can represent? -128
- how to represent -1? 11111111
- how to represent -2? 11111110

Representing negative and signed integers

2's complement

signed char	weight in decimal
00000001	1
00000010	2
00000100	4
00001000	8
00010000	16
00100000	32
01000000	64
10000000	-128

Table: Weight of each bit in a signed char type

- MSB: 1 for negative
- Take the 1's complement number + 1
- Most important; good properties for digital logic

Importance of paying attention to limits of encoding

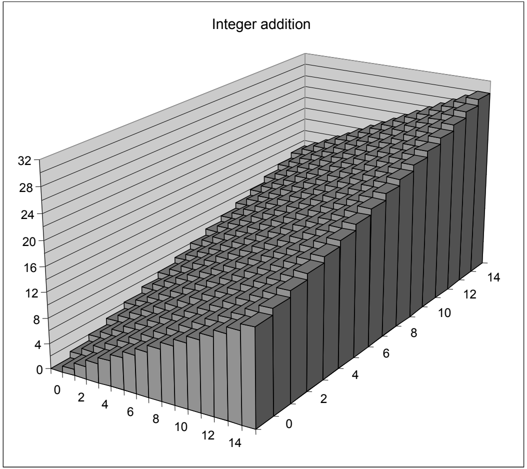


Figure: Image credit: CS:APP

Importance of paying attention to limits of encoding

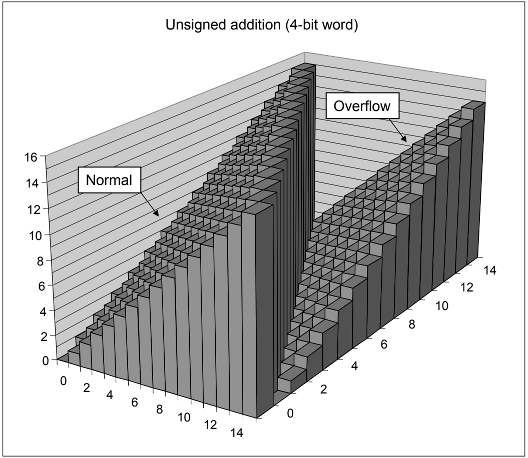


Figure: Image credit: CS:APP

Importance of paying attention to limits of encoding

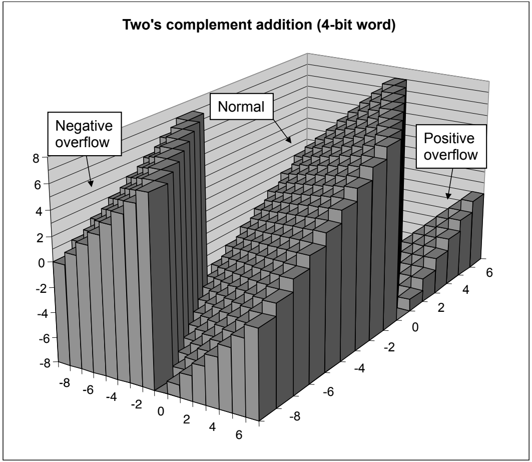


Figure: Image credit: CS:APP