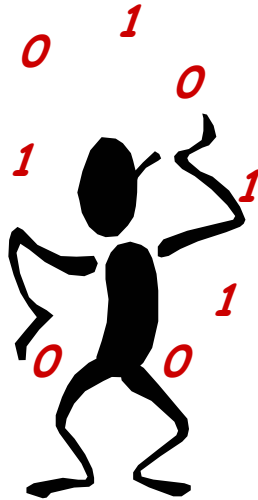


Representing Information



“Bit Juggling”

- Representing information using bits
- Number representations
- Some other bits

Announcements

TA Bud Vasile's office hours:

Monday 2-3:30pm @ SN 034

Tuesday 2-3:30pm @ SN 034



From Last Time

- 1) Computers are tools for processing information.
- 2) Information resolves uncertainty.
- 3) Information is measured in bits.
- 4) A fact that narrows N possible choices down to M provides

$\log_2(N/M)$ bits of information.

This time we'll explore how information is represented in computers and discuss specific examples.

It isn't a dream, the semester really is starting.



Encoding

- ◆ Encoding describes the process of **assigning representations to information**
- ◆ Choosing an appropriate and efficient encoding is a real engineering challenge
- ◆ Impacts design at many levels
 - Mechanism (devices, # of components used)
 - Efficiency (bits used)
 - Reliability (noise)
 - Security (encryption)



Fixed-Length Encodings

If all choices are **equally likely** (or we have no reason to expect otherwise), then a fixed-length code is often used. Such a code will use at least enough bits to represent the information content.

ex. Decimal digits $10 = \{0,1,2,3,4,5,6,7,8,9\}$
 4-bit BCD (binary code decimal)
 $\log_2(10 / 1) = 3.322 < 4\text{bits}$

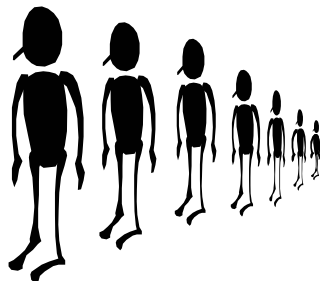
ex. ~84 English characters = {A-Z (26), a-z (26), 0-9 (11), punctuation (8), math (9), financial (4)}
 7-bit ASCII (American Standard Code for Information Interchange)
 $\log_2(84 / 1) = 6.392 < 7\text{bits}$

Encoding Positive Integers

It is straightforward to encode positive integers as a sequence of bits. Each bit is assigned a weight. Ordered from right to left, these weights are increasing powers of 2. The value of an n-bit number encoded in this fashion is given by the following formula:

$$v = \sum_{i=0}^{n-1} 2^i b_i$$

$$\begin{matrix} 2^{11} & 2^{10} & 2^9 & 2^8 & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \hline 0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \end{matrix}$$



$$\begin{array}{r} 2^4 = 16 \\ + 2^6 = 64 \\ + 2^7 = 128 \\ + 2^8 = 256 \\ + 2^9 = 512 \\ + 2^{10} = 1024 \\ \hline 2000_{10} \end{array}$$

Some Tricks with Bits

- You are going to have to get accustomed to working in binary. **Specifically for Comp 120**, but it will be helpful throughout your career as a computer scientist.
- Here are some helpful guides

1. Memorize the first 10 powers of 2

$2^0 = 1$	$2^5 = 32$
$2^1 = 2$	$2^6 = 64$
$2^2 = 4$	$2^7 = 128$
$2^3 = 8$	$2^8 = 256$
$2^4 = 16$	$2^9 = 512$

More Tricks with Bits

- You are going to have to get accustomed to working in binary. **Specifically for Comp 120**, but it will be helpful throughout your career as a computer scientist.
- Here are some helpful guides

2. Memorize the prefixes for powers of 2 that are multiples of 10

$2^{10} = \text{Kilo}$	(1024)
$2^{20} = \text{Mega}$	(1024*1024)
$2^{30} = \text{Giga}$	(1024*1024*1024)
$2^{40} = \text{Tera}$	(1024*1024*1024*1024)
$2^{50} = \text{Peta}$	(1024*1024*1024 *1024*1024)
$2^{60} = \text{Exa}$	(1024*1024*1024*1024*1024*1024)

Even More Tricks with Bits

- You are going to have to get accustomed to working in binary. Specifically for Comp 120, but it will be helpful throughout your career as a computer scientist.
- Here are some helpful guides

01000110000000011000000000101000

3. When you convert a binary number to decimal, first break it down into clusters of 10 bits.
4. Then compute the value of the leftmost remaining bits (1) find the appropriate prefix (GIGA) (Often this is sufficient)
5. Compute the value of and add in each remaining 10-bit cluster

Other Helpful Clusterings

Oftentimes we will find it convenient to cluster groups of bits together for a more compact representation. The clustering of 3 bits is called Octal. Octal is not that common today.

$$v = \sum_{i=0}^{n-1} 2^i d_i$$

$$\begin{array}{cccccccc} 2^{11} & 2^{10} & 2^9 & 2^8 & 2^7 & 2^6 & 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ \hline 0 & 1 & 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ \hline \end{array} = 2000_{10}$$

3 7 2 0

03720

Octal - base 8



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

$$\begin{array}{r} 0 * 8^0 = 0 \\ + 2 * 8^1 = 16 \\ + 7 * 8^2 = 448 \\ + 3 * 8^3 = 1536 \\ \hline 2000_{10} \end{array}$$

One Last Clustering

Clusters of 4 bits are used most frequently. This representation is called hexadecimal. The hexadecimal digits include 0-9, and A-F, and each digit position represents a power of 16.

$$v = \sum_{i=0}^{n-1} 16^i d_i$$

$$\underbrace{01111110100000}_{7 \quad d \quad 0} = 2000_{10}$$

0x7d0



Hexadecimal - base 16

0000 - 0	1000 - 8
0001 - 1	1001 - 9
0010 - 2	1010 - a
0011 - 3	1011 - b
0100 - 4	1100 - c
0101 - 5	1101 - d
0110 - 6	1110 - e
0111 - 7	1111 - f

$$0 * 16^0 = 0$$

$$+ 13 * 16^1 = 208$$

$$+ 7 * 16^2 = \underline{1792}$$

$$2000_{10}$$

Example: ASCII table

Dec	Hx	Oct	Char	Dec	Hx	Oct	Httrl	Chr	Dec	Hx	Oct	Httrl	Chr	Dec	Hx	Oct	Httrl	Chr
0	0	000	NUL (null)	32	20	040	0x32	Space	64	40	100	0x64	@	96	60	140	0x96	`
1	1	001	SOH (start of heading)	33	21	041	0x33	!	65	41	101	0x65	A	97	61	141	0x97	a
2	2	002	STX (start of text)	34	22	042	0x34	"	66	42	102	0x66	B	98	62	142	0x98	b
3	3	003	ETX (end of text)	35	23	043	0x35	#	67	43	103	0x67	C	99	63	143	0x99	c
4	4	004	EOT (end of transmission)	36	24	044	0x36	\$	68	44	104	0x68	D	100	64	144	0x100	d
5	5	005	ENQ (enquiry)	37	25	045	0x37	%	69	45	105	0x69	E	101	65	145	0x101	e
6	6	006	ACK (acknowledge)	38	26	046	0x38	&	70	46	106	0x70	F	102	66	146	0x102	f
7	7	007	BEL (bell)	39	27	047	0x39	'	71	47	107	0x71	G	103	67	147	0x103	g
8	8	010	BS (backspace)	40	28	050	0x40	(72	48	110	0x72	H	104	68	150	0x104	h
9	9	011	TAB (horizontal tab)	41	29	051	0x41)	73	49	111	0x73	I	105	69	151	0x105	i
10	A	012	LF (NL line feed, new line)	42	2A	052	0x42	*	74	4A	112	0x74	J	106	6A	152	0x106	j
11	B	013	VT (vertical tab)	43	2B	053	0x43	+	75	4B	113	0x75	K	107	6B	153	0x107	k
12	C	014	FF (NP form feed, new page)	44	2C	054	0x44	,	76	4C	114	0x76	L	108	6C	154	0x108	l
13	D	015	CR (carriage return)	45	2D	055	0x45	-	77	4D	115	0x77	M	109	6D	155	0x109	m
14	E	016	SO (shift out)	46	2E	056	0x46	.	78	4E	116	0x78	N	110	6E	156	0x110	n
15	F	017	SI (shift in)	47	2F	057	0x47	/	79	4F	117	0x79	O	111	6F	157	0x111	o
16	10	020	DLE (data link escape)	48	30	060	0x48	0	80	50	120	0x80	P	112	70	160	0x112	p
17	11	021	DC1 (device control 1)	49	31	061	0x49	1	81	51	121	0x81	Q	113	71	161	0x113	q
18	12	022	DC2 (device control 2)	50	32	062	0x50	2	82	52	122	0x82	R	114	72	162	0x114	r
19	13	023	DC3 (device control 3)	51	33	063	0x51	3	83	53	123	0x83	S	115	73	163	0x115	s
20	14	024	DC4 (device control 4)	52	34	064	0x52	4	84	54	124	0x84	T	116	74	164	0x116	t
21	15	025	NAK (negative acknowledge)	53	35	065	0x53	5	85	55	125	0x85	U	117	75	165	0x117	u
22	16	026	SYN (synchronous idle)	54	36	066	0x54	6	86	56	126	0x86	V	118	76	166	0x118	v
23	17	027	ETB (end of trans. block)	55	37	067	0x55	7	87	57	127	0x87	W	119	77	167	0x119	w
24	18	030	CAN (cancel)	56	38	070	0x56	8	88	58	130	0x88	X	120	78	170	0x120	x
25	19	031	EM (end of medium)	57	39	071	0x57	9	89	59	131	0x89	Y	121	79	171	0x121	y
26	1A	032	SUB (substitute)	58	3A	072	0x58	:	90	5A	132	0x90	Z	122	7A	172	0x122	z
27	1B	033	ESC (escape)	59	3B	073	0x59	;	91	5B	133	0x91	[123	7B	173	0x123	{
28	1C	034	FS (file separator)	60	3C	074	0x60	<	92	5C	134	0x92	\	124	7C	174	0x124	
29	1D	035	GS (group separator)	61	3D	075	0x61	=	93	5D	135	0x93]	125	7D	175	0x125	}
30	1E	036	RS (record separator)	62	3E	076	0x62	>	94	5E	136	0x94	^	126	7E	176	0x126	~
31	1F	037	US (unit separator)	63	3F	077	0x63	?	95	5F	137	0x95	_	127	7F	177	0x127	DEL

Source: www.asciitable.com

Signed-Number Representations

There are also schemes for representing signed integers with bits. One obvious method is to encode the sign of the integer using one bit. Conventionally, the most significant bit is used for the sign. This encoding for signed integers is called the SIGNED MAGNITUDE representation.

$$v = -1^s \sum_{i=0}^{n-2} 2^i b_i$$

s	2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	1	1	1	0	1	0	0	0

2000 -2000

Even though this approach seems straightforward, it is not used that frequently in practice (with one important exception).

One more trick...

Sum of powers of two

$$\sum_{i=0}^{n-1} 2^i = 2^n - 1$$

Complement

$$\boxed{1}^{2^{11}} = 2048_{10}$$

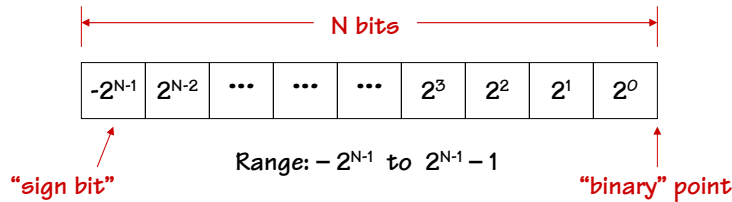
2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
0	0	0	0	0	1	0	1	1	1	1

= 47₁₀

2^{10}	2^9	2^8	2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	1	1	1	0	1	0	0	0	0

= 2000₁₀

2's Complement Integers



The 2's complement representation for signed integers is the most commonly used signed-integer representation. It is a simple modification of unsigned integers where the most significant bit represents a **negative power of 2**.

8-bit 2's complement example:

$$v = -2^{n-1}b_{n-1} + \sum_{i=0}^{n-2} 2^i b_i$$

$$\begin{aligned} 11010110 &= -2^7 + 2^6 + 2^4 + 2^2 + 2^1 \\ &= -128 + 64 + 16 + 4 + 2 = -42 \end{aligned}$$

Why 2's Complement?

If we use a two's complement representation for signed integers, the same binary addition mod 2^n procedure will work for adding positive and negative numbers (**don't need separate subtraction rules**). The same procedure will also handle unsigned numbers!

When using signed magnitude representations, adding a negative value really means to subtract a positive value. However, in 2's magnitude, adding is adding regardless of sign. In fact, you NEVER need to subtract when you use a 2's complement representation.

Example:

$$\begin{array}{r} 55_{10} = 00110111_2 \\ + 10_{10} = 00001010_2 \\ \hline 65_{10} = 01000001_2 \end{array}$$

$$\begin{array}{r} 55_{10} = 00110111_2 \\ + -10_{10} = 11110110_2 \\ \hline 45_{10} = 100101101_2 \end{array}$$



2's Complement Tricks

- Negation – changing the sign of a number
 - First complement every bit (i.e. $1 \rightarrow 0, 0 \rightarrow 1$)
 - Add 1
 - Example: $20 = 00010100, -20 = 11101011 + 1 = 11101100$

- Sign-Extension – aligning different sized 2's complement integers

16-bit version of 42 = 0000 0000 0010 1010

8-bit version of -2 = 1111 1111 1111 1110



signed-number vs. 2's complement

$b_7b_6b_5b_4$	Unsigned	Sign and Magnitude	Two's complement
1111	15	-7	-1
1110	14	-6	-2
1101	13	-5	-3
1100	12	-4	-4
1011	11	-3	-5
1010	10	-2	-6
1001	9	-1	-7
1000	8	-0	-8
0111	7	7	7
0110	6	6	6
0101	5	5	5
0100	4	4	4
0011	3	3	3
0010	2	2	2
0001	1	1	1
0000	0	0	0

Fixed-Point Numbers

By moving the implicit location of the “binary” point, we can represent signed fractions too. This has no effect on how operations are performed, assuming that the operands are properly aligned.

-2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
--------	-------	-------	-------	----------	----------	----------	----------



$$\begin{aligned} 1101.0110 &= -2^3 + 2^2 + 2^0 + 2^{-2} + 2^{-3} \\ &= -8 + 4 + 1 + 0.25 + 0.125 \\ &= -2.625 \end{aligned}$$

OR

$$1101.0110 = -42 * 2^{-4} = -42/16 = -2.625$$

Repeated Binary Fractions

Not all fractions can be represented exactly using a finite representation. You’ve seen this before in decimal notation where the fraction $1/3$ (among others) requires an infinite number of digits to represent ($0.3333\dots$).

–

In Binary, a great many fractions that you’ve grown attached to require an infinite number of bits to represent exactly.

$$\begin{aligned} \text{EX: } 1/10 &= 0.1_{10} = \overline{.00011}_2 \\ 1/5 &= 0.2_{10} = \overline{.0011}_2 = 0.33\overline{3}_{16} \end{aligned}$$

Bias Notation

There is yet one more way to represent signed integers, which is surprisingly simple. It involves subtracting a fixed constant from a given unsigned number. This representation is called "Bias Notation".

$$v = \sum_{i=0}^{n-1} 2^i b_i - \text{Bias}$$

2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0
1	1	0	1	0	1	1	0

EX: (Bias = 127)

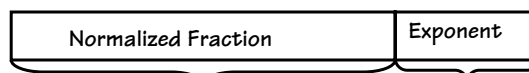
$$\begin{array}{r}
 6 * 1 = 6 \\
 13 * 16 = 208 \\
 \hline
 - 127 \\
 \hline
 87
 \end{array}$$

Why? Monotonicity

Floating Point Numbers

Another way to represent numbers is to use a notation similar to Scientific Notation. This format can be used to represent numbers with fractions (3.90×10^{-4}), very small numbers (1.60×10^{-19}), and large numbers (6.02×10^{23}). This notation uses two fields to represent each number. The first part represents a normalized fraction (called the significand), and the second part represents the exponent (i.e. the position of the "floating" binary point).

$$\text{Normalized Fraction} \times 2^{\text{Exponent}}$$



"bits of accuracy" "dynamic range"

IEEE 754 Format

- Single precision format

$v = -1^S \times 1.\text{Significand} \times 2^{\text{Exponent} - 127}$

- Double precision format

$v = -1^S \times 1.\text{Significand} \times 2^{\text{Exponent} - 1023}$

Comp120 - Spring 2005 1/18/05 L02 - Representing Information 23

Summary

- 1) Selecting the encoding of information has important implications on **how this information can be processed**, and **how much space it requires**.
- 2) Computer arithmetic is constrained by **finite representations**, this has advantages (it allows for complement arithmetic) and disadvantages (it allows for overflows, numbers too big or small to be represented).
- 3) Bit patterns can be interpreted in an endless number of ways, however important standards do exist
 - Two's complement
 - IEEE 754 floating point