Chapter 1:
Data Storage

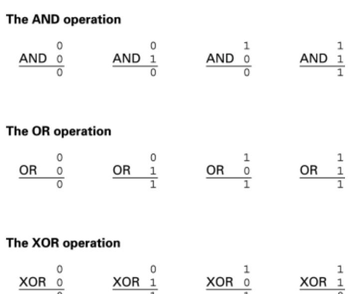
Computer Science: An Overview
Tenth Edition

by
J. Glenn Brookshear

Bits and Bit Patterns

- **Bit:** Binary Digit (0 or 1)
- Bit Patterns are used to represent information.
  - Numbers
  - Text characters
  - Images
  - Sound
  - And others

Figure 1.1 **The Boolean operations AND, OR, and XOR (exclusive or)**



Boolean operations

- AND

P * Q

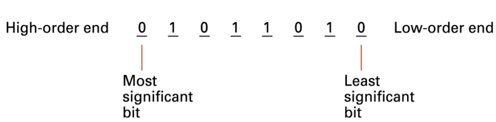
		Q: 0	Q: 1
P	0	0	0
	1	0	1

NOT

P '

P	0	1
	1	0

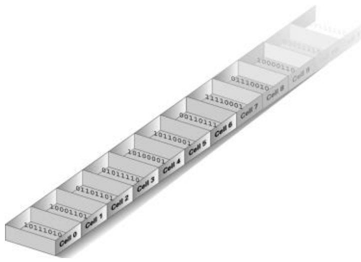
Figure 1.7 **The organization of a byte-size memory cell**



Main Memory Addresses

- **Address:** A "name" that uniquely identifies one cell in the computer's main memory
  - The names are actually numbers.
  - These numbers are assigned consecutively starting at zero.
  - Numbering the cells in this manner associates an order with the memory cells.

Figure 1.8 Memory cells arranged by address



ASCII

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
0	00	Null	32	20	Space	64	40	@	96	60	`
1	01	Start of heading	33	21	!	65	41	A	97	61	a
2	02	Start of text	34	22	"	66	42	B	98	62	b
3	03	End of text	35	23	#	67	43	C	99	63	c
4	04	End of transmit	36	24	$	68	44	D	100	64	d
5	05	Enquiry	37	25	%	69	45	E	101	65	e
6	06	Acknowledge	38	26	&	70	46	F	102	66	f
7	07	Audible bell	39	27	'	71	47	G	103	67	g
8	08	Backspace	40	28	(	72	48	H	104	68	h
9	09	Horizontal tab	41	29	)	73	49	I	105	69	i
10	0A	Line feed	42	2A	*	74	4A	J	106	6A	j
11	0B	Vertical tab	43	2B	+	75	4B	K	107	6B	k
12	0C	Form feed	44	2C	,	76	4C	L	108	6C	l
13	0D	Carriage return	45	2D	-	77	4D	M	109	6D	m
14	0E	Shift out	46	2E	.	78	4E	N	110	6E	n
15	0F	Shift in	47	2F	/	79	4F	O	111	6F	o
16	10	Data link escape	48	30	0	80	50	P	112	70	p
17	11	Device control 1	49	31	1	81	51	Q	113	71	q
18	12	Device control 2	50	32	2	82	52	R	114	72	r
19	13	Device control 3	51	33	3	83	53	S	115	73	s
20	14	Device control 4	52	34	4	84	54	T	116	74	t
21	15	Neg. acknowledge	53	35	5	85	55	U	117	75	u
22	16	Synchronous idle	54	36	6	86	56	V	118	76	v
23	17	End trans. block	55	37	7	87	57	W	119	77	w
24	18	Cancel	56	38	8	88	58	X	120	78	x
25	19	End of medium	57	39	9	89	59	Y	121	79	y
26	1A	Substitution	58	3A	:	90	5A	Z	122	7A	z
27	1B	Escape	59	3B	;	91	5B	[	123	7B	{
28	1C	File separator	60	3C	<	92	5C	\	124	7C	|
29	1D	Group separator	61	3D	=	93	5D	]	125	7D	}
30	1E	Record separator	62	3E	>	94	5E	^	126	7E	~
31	1F	Unit separator	63	3F	?	95	5F	_	127	7F	⌂

Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char	Dec	Hex	Char
128	80	Ç	160	A0	á	192	C0	└	224	E0	α
129	81	ü	161	A1	í	193	C1	┴	225	E1	ß
130	82	é	162	A2	ó	194	C2	┬	226	E2	Γ
131	83	â	163	A3	ú	195	C3	├	227	E3	π
132	84	ä	164	A4	ñ	196	C4	─	228	E4	Σ
133	85	à	165	A5	Ñ	197	C5	┼	229	E5	σ
134	86	å	166	A6	ª	198	C6	╞	230	E6	µ
135	87	ç	167	A7	º	199	C7	╟	231	E7	τ
136	88	ê	168	A8	¿	200	C8	╚	232	E8	Φ
137	89	ë	169	A9	⌐	201	C9	╔	233	E9	Θ
138	8A	è	170	AA	¬	202	CA	╩	234	EA	Ω
139	8B	ï	171	AB	½	203	CB	╦	235	EB	δ
140	8C	î	172	AC	¼	204	CC	╠	236	EC	∞
141	8D	ì	173	AD	¡	205	CD	═	237	ED	φ
142	8E	Ä	174	AE	«	206	CE	╬	238	EE	ε
143	8F	Å	175	AF	»	207	CF	╧	239	EF	∩
144	90	É	176	B0	░	208	D0	╨	240	F0	≡
145	91	æ	177	B1	▒	209	D1	╤	241	F1	±
146	92	Æ	178	B2	▓	210	D2	╥	242	F2	≥
147	93	ô	179	B3	│	211	D3	╙	243	F3	≤
148	94	ö	180	B4	┤	212	D4	╘	244	F4	⌠
149	95	ò	181	B5	╡	213	D5	╒	245	F5	⌡
150	96	û	182	B6	╢	214	D6	╓	246	F6	÷
151	97	ù	183	B7	╖	215	D7	╫	247	F7	≈
152	98	ÿ	184	B8	╕	216	D8	╪	248	F8	°
153	99	Ö	185	B9	╣	217	D9	┘	249	F9	∙
154	9A	Ü	186	BA	║	218	DA	┌	250	FA	·
155	9B	¢	187	BB	╗	219	DB	█	251	FB	√
156	9C	£	188	BC	╝	220	DC	▄	252	FC	^{n}
157	9D	¥	189	BD	╜	221	DD	▌	253	FD	^{2}
158	9E	₧	190	BE	╛	222	DE	▐	254	FE	■
159	9F	ƒ	191	BF	┐	223	DF	▀	255	FF	□

Figure 1.13 The message "Hello." in ASCII

01001000	01100101	01101100	01101100	01101111	00101110
H	e	l	l	o	.

Representing Numeric Values

- Binary notation: Uses bits to represent a number in base two
- Limitations of computer representations of numeric values
  - Overflow – occurs when a value is too big to be represented
  - Truncation – occurs when a value cannot be represented accurately

The Binary System

The traditional decimal system is based on powers of ten.

The Binary system is based on powers of two.

Figure 1.15 The base ten and binary systems

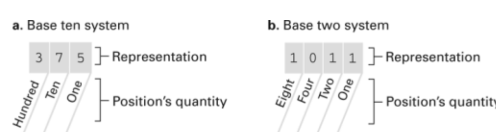


Figure 1.16 Decoding the binary representation 100101

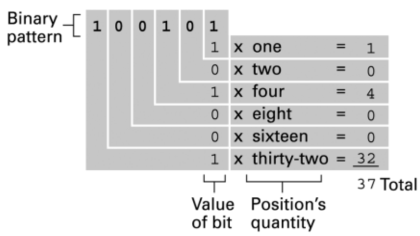


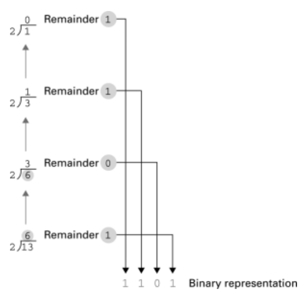
Figure 1.17 An algorithm for finding the binary representation of a positive integer

Step 1. Divide the value by two and record the remainder.

Step 2. As long as the quotient obtained is not zero, continue to divide the newest quotient by two and record the remainder.

Step 3. Now that a quotient of zero has been obtained, the binary representation of the original value consists of the remainders listed from right to left in the order they were recorded.

Figure 1.18 Applying the algorithm in Figure 1.15 to obtain the binary representation of thirteen



Hexadecimal Notation

- **Hexadecimal notation:** A shorthand notation for long bit patterns
  - Divides a pattern into groups of four bits each
  - Represents each group by a single symbol
- Example: 10100011 becomes A3

Figure 1.6 The hexadecimal coding system

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

Converting Between Binary and Hex

- To convert from binary to hex
  - starting on the right of the binary number, arrange the binary digits in groups of four
  - convert each quartet to the corresponding hex digit
  - 110 1110 1100
  - 6 E C

Hex and Binary conversion cont.

- To convert from hex to binary, replace each hex digit with its 4-bit binary equivalent
- 8 A 5
- 1000 1010 0101

Finite Precision Binary Arithmetic

- We can store *two* different values in 1 bit: 0 or 1.
- in general, in *n* bits, you can store 2^n different values.
- So with 4 bits, we can store 16 values - 0 to 15, but there are *no negative values.*
- common sense says to split the number of values in half, making half positive, half negative, not forgetting zero.
- **Two's complement notation**

Storing Integers

- **Two's complement notation:** The most popular means of representing integer values – standard use is with 8 bits
- **Excess notation:** Another means of representing integer values – used in floating point notation
- Both can suffer from overflow errors.

Figure 1.21 **Two's complement notation systems**

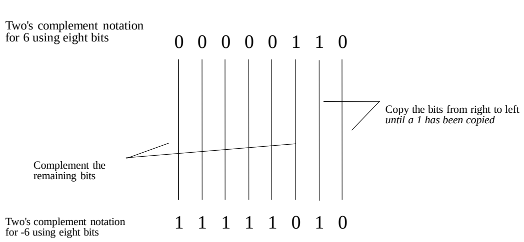
a. Using patterns of length three

Bit pattern	Value represented
011	3
010	2
001	1
000	0
111	−1
110	−2
101	−3
100	−4

b. Using patterns of length four

Bit pattern	Value represented
0111	7
0110	6
0101	5
0100	4
0011	3
0010	2
0001	1
0000	0
1111	−1
1110	−2
1101	−3
1100	−4
1011	−5
1010	−6
1001	−7
1000	−8

Figure 1.22 **Coding the value -6 in two's complement notation using eight bits**



Another method – flip all the bits and add 1 to the result

How many values

- with 4 bits
- $-8 = -2^{4-1}$ and $7 = 2^{4-1} – 1$.
- with 16 bits
- $-32,768 = -2^{16-1}$ and $32,767 = 2^{16-1} – 1$
- with 32 bits
- $-2,147,483,648 = -2^{32-1}$ and $2,147,483,647 = 2^{32-1} - 1$

Figure 1.19 **The binary addition facts**

0	1	0	1
+ 0	+ 0	+ 1	+ 1
0	1	1	10

Figure 1.23 **Addition problems converted to two's complement notation**

Problem in base ten		Problem in two's complement		Answer in base ten
3 + 2	→	0011 + 0010 = 0101	→	5
−3 + −2	→	1101 + 1110 = 1011	→	−5
7 + −5	→	0111 + 1011 = 0010	→	2

OVERFLOW – Trouble in paradise

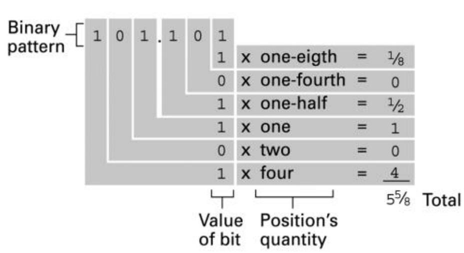
- Using 6 bits we can represent values from -32 to 31, so what happens when we try to add 19 plus 14 or -19 and -14.

19 +14 33	010011 +001110 100001	we have added two positive numbers and gotten a negative result – this is overflow
-19 -14 -33	101101 +110010 011111	we have added two negative numbers and gotten a positive result – this is overflow

Real numbers in Binary

- As the places to the left of the binary point are powers of 2, so are the places to the right of the binary point
- ... 8 4 2 1 . ½ ¼ 1/8 1/16 1/32 1/64 ...

Figure 1.20 **Decoding the binary representation 101.101**



Coding 9 11/32

- 9 = 1001
- 11 = 1011
- 32 is fifth place to the right of binary point
- so 1001.01011 with the final 1 in the 1/32 place to the right of the binary point

Figure 1.24 An excess eight conversion table

Bit pattern	Value represented
1111	7
1110	6
1101	5
1100	4
1011	3
1010	2
1001	1
1000	0
0111	−1
0110	−2
0101	−3
0100	−4
0011	−5
0010	−6
0001	−7
0000	−8

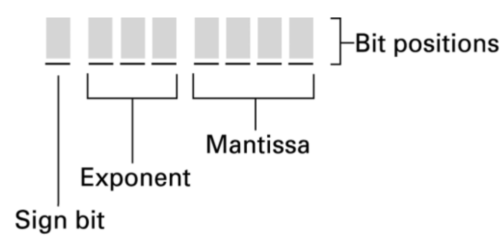
Figure 1.25 An excess notation system using bit patterns of length three

Bit pattern	Value represented
111	3
110	2
101	1
100	0
011	−1
010	−2
001	−3
000	−4

Storing Fractions

- **Floating-point Notation:** Consists of a sign bit, a mantissa field, and an exponent field.
- Related topics include
  - Normalized form
  - Truncation errors

Figure 1.26 Floating-point notation components



Converting 2 1/4 to floating point

- figure out 2 ¼ in binary = 10.01
- normalize the number (move the binary point to the left of the most significant 1 – leftmost one) and adjust the exponent = 10.01 * 2^0 = .1001 * 2^2
- Calculate the exponent by adding the excess value to the exponent value: 2 + 4 = 6 = 110 in binary
- figure out the sign – positive is 0
- put is all together = 0 110 1001

Another

- Remember if the number has a whole portion, the exponent will be positive. If the number is 0 or a fraction, the exponent will be 0 or negative
- -3/8 = .011 in binary
- normalize .11 * 2^{-1} (pad fraction = .1100)
- calculate exponent: -1 + 4 = 3 = 011
- calculate sign: 1 for negative
- put it together: 1 011 1100

Figure 1.27 **Encoding the value 2 5⁄8**