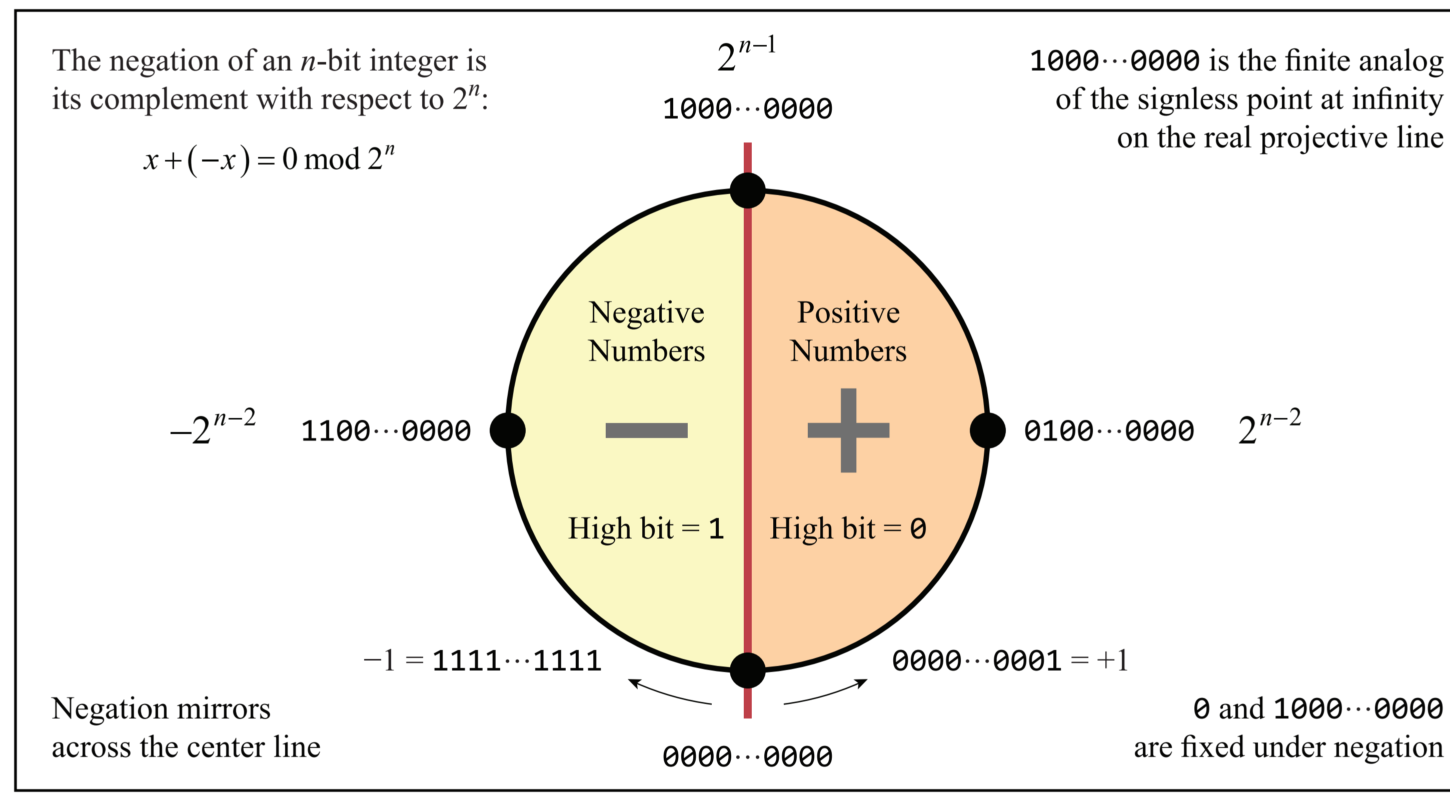
Binary Fundamentals

Powers of Two

Power	Decimal	Hexadecimal
2^1	2	0x00000002
2^2	4	0x00000004
2^3	8	0x00000008
2^4	16	0x00000010
2^5	32	0x00000020
2^6	64	0x00000040
2^7	128	0x00000080
2^8	256	0x00000100
2^9	512	0x00000200
2^{10}	1024	0x00000400
2^{11}	2048	0x00000800
2^{12}	4096	0x00001000

Power	Decimal	Hexadecimal
2^{13}	8192	0x00002000
2^{14}	16,384	0x00004000
2^{15}	32,768	0x00008000
2^{16}	65,536	0x00010000
2^{17}	131,072	0x00020000
2^{18}	262,144	0x00040000
2^{19}	524,288	0x00080000
2^{20}	1,048,576	0x00100000
2^{21}	2,097,152	0x00200000
2^{22}	4,194,304	0x00400000
2^{23}	8,388,608	0x00800000
2^{24}	16,777,216	0x01000000

Two's Complement



Logical Complement

NOT

Bitwise NOT

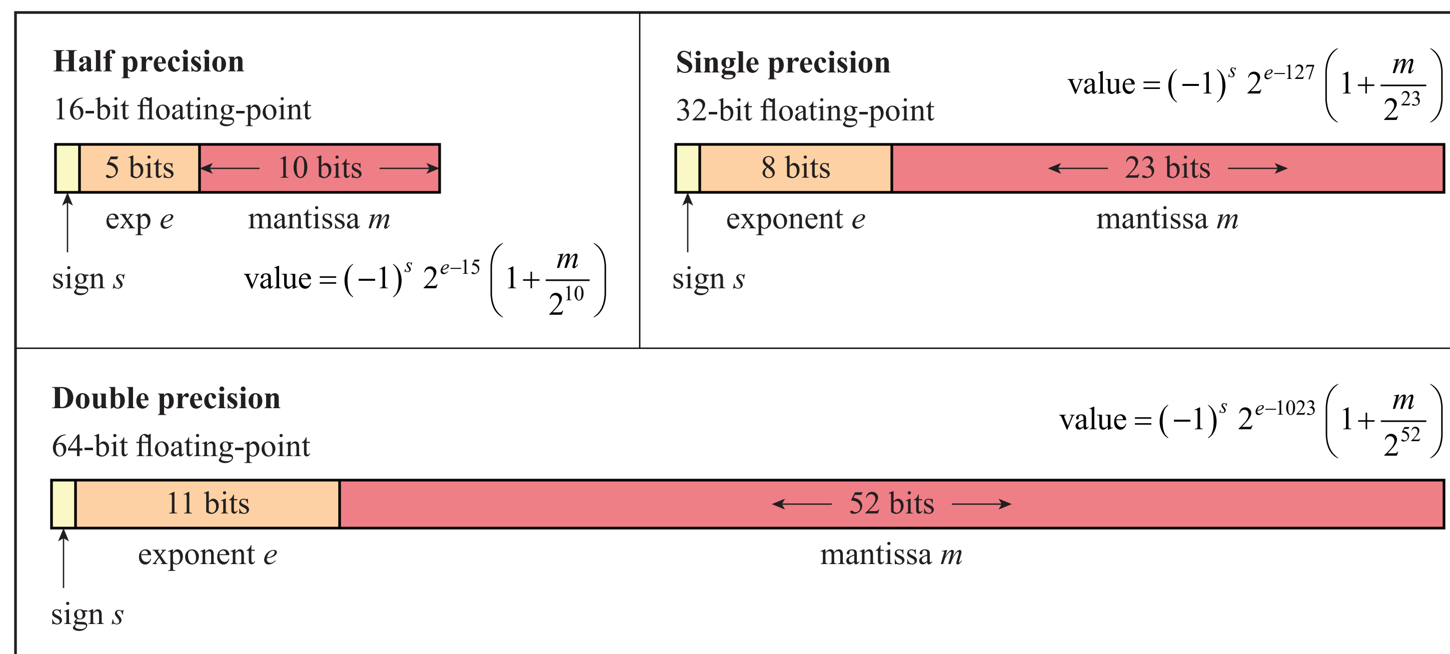
`~x`

x	~x
0	1
1	0

Logical Identities

Unary	Binary
`~x = -x - 1`	`~(x & y) = ~x | ~y`
`-x = ~x + 1`	`~(x | y) = ~x & ~y`
`-~x = x + 1`	`~(x ^ y) = { ~x ^ y ; x ^ ~y }`
`~-x = x - 1`	

Floating-Point



Special Floating-Point Value	Half	Float	Double
+0.0	0x0000	0x00000000	0x00000000_00000000
+1.0	0x3C00	0x3F800000	0x3FF00000_00000000
Positive infinity	0x7C00	0x7F800000	0x7FF00000_00000000
Smallest positive normalized value	0x0400	0x00800000	0x00100000_00000000
Upper limit of non-integer values	0x6400	0x4B000000	0x43300000_00000000
Largest representable positive value	0x7BFF	0x7F7FFFFF	0x7FEFFFFF_FFFFFFFF

Binary Logical Operations

AND — Bitwise AND — `x & y`

x	y	x & y
0	0	0
0	1	0
1	0	0
1	1	1

OR — Bitwise OR — `x | y`

x	y	x | y
0	0	0
0	1	1
1	0	1
1	1	1

NAND — Not AND — `~(x & y)`

x	y	~(x & y)
0	0	1
0	1	1
1	0	1
1	1	0

NOR — Not OR — `~(x | y)`

x	y	~(x | y)
0	0	1
0	1	0
1	0	0
1	1	0

ANDC — AND with complement — `x & ~y`

x	y	x & ~y
0	0	0
0	1	0
1	0	1
1	1	0

ORC — OR with complement — `x | ~y`

x	y	x | ~y
0	0	1
0	1	0
1	0	1
1	1	1

XOR — Exclusive OR — `x ^ y`

x	y	x ^ y
0	0	0
0	1	1
1	0	1
1	1	0

XNOR — Exclusive NOR — `~(x ^ y)`

x	y	~(x ^ y)
0	0	1
0	1	0
1	0	0
1	1	1

Mask Creation

Formula	Operation / Effect	Illustration
`~x | (x - 1)`	Create mask for all bits other than lowest 1 bit. 000···000 becomes 111···111.	10111000 → 11110111
`x | ~(x + 1)`	Create mask for all bits other than lowest 0 bit. 111···111 is unchanged.	01101111 → 11110111
`x | -x`	Create mask for bits left of lowest 1 bit, inclusive. 000···000 is unchanged.	10111000 → 11111000
`x ^ -x`	Create mask for bits left of lowest 1 bit, exclusive. 000···000 is unchanged.	10111000 → 11110000
`~x | (x + 1)`	Create mask for bits left of lowest 0 bit, inclusive. 111···111 becomes 000···000.	01101111 → 11111000
`~x ^ (x + 1)`	Create mask for bits left of lowest 0 bit, exclusive. 111···111 becomes 000···000.	01101111 → 11110000
`x ^ (x - 1)`	Create mask for bits right of lowest 1 bit, inclusive. 000···000 becomes 111···111.	10111000 → 00001111
`~x & (x - 1)`	Create mask for bits right of lowest 1 bit, exclusive. 000···000 becomes 111···111.	10111000 → 00000111
`x ^ (x + 1)`	Create mask for bits right of lowest 0 bit, inclusive. 111···111 is unchanged.	01101111 → 00001111
`x & (~x - 1)`	Create mask for bits right of lowest 0 bit, exclusive. 111···111 is unchanged.	01101111 → 00000111

Bit Manipulation

Formula	Operation / Effect	Illustration
`x & (x - 1)`	Clear lowest 1 bit. If result is zero, then x is zero or 2^k. 000···000 is unchanged.	10111000 → 10110000
`x | (x + 1)`	Set lowest 0 bit. 111···111 is unchanged.	01101111 → 01111111
`x | (x - 1)`	Set all bits to right of lowest 1 bit. 000···000 becomes 111···111.	10111000 → 10111111
`x & (x + 1)`	Clear all bits to right of lowest 0 bit. If result is zero, then x is zero or $2^k - 1$. 111···111 becomes 000···000.	01101111 → 01100000
`x & -x`	Extract lowest 1 bit. 000···000 is unchanged.	10111000 → 00001000
`~x & (x + 1)`	Extract lowest 0 bit (as a 1 bit). 111···111 becomes 000···000.	01101111 → 00001000