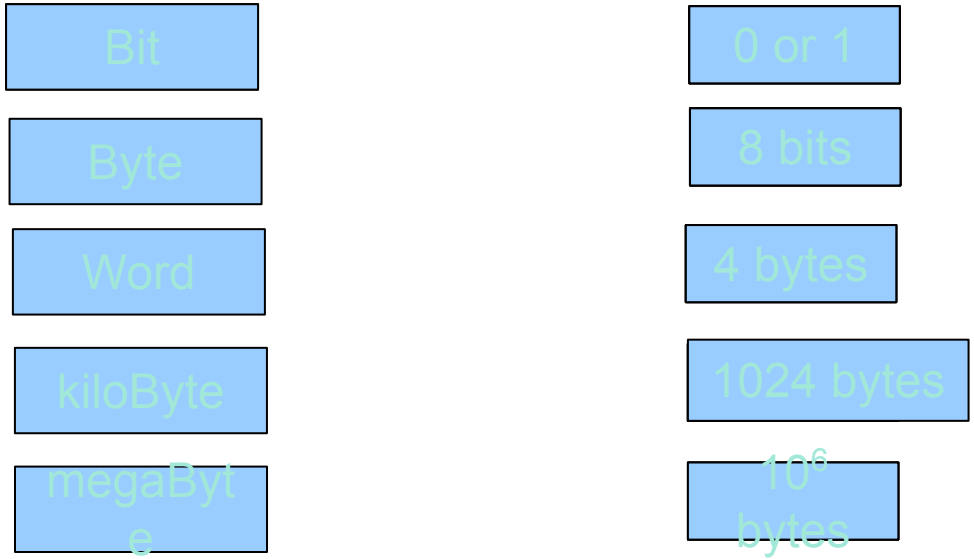
The Language of Bits

What does a Computer Understand ?

* Computers do not understand natural human languages, nor programming languages
* They only understand the language of bits



2

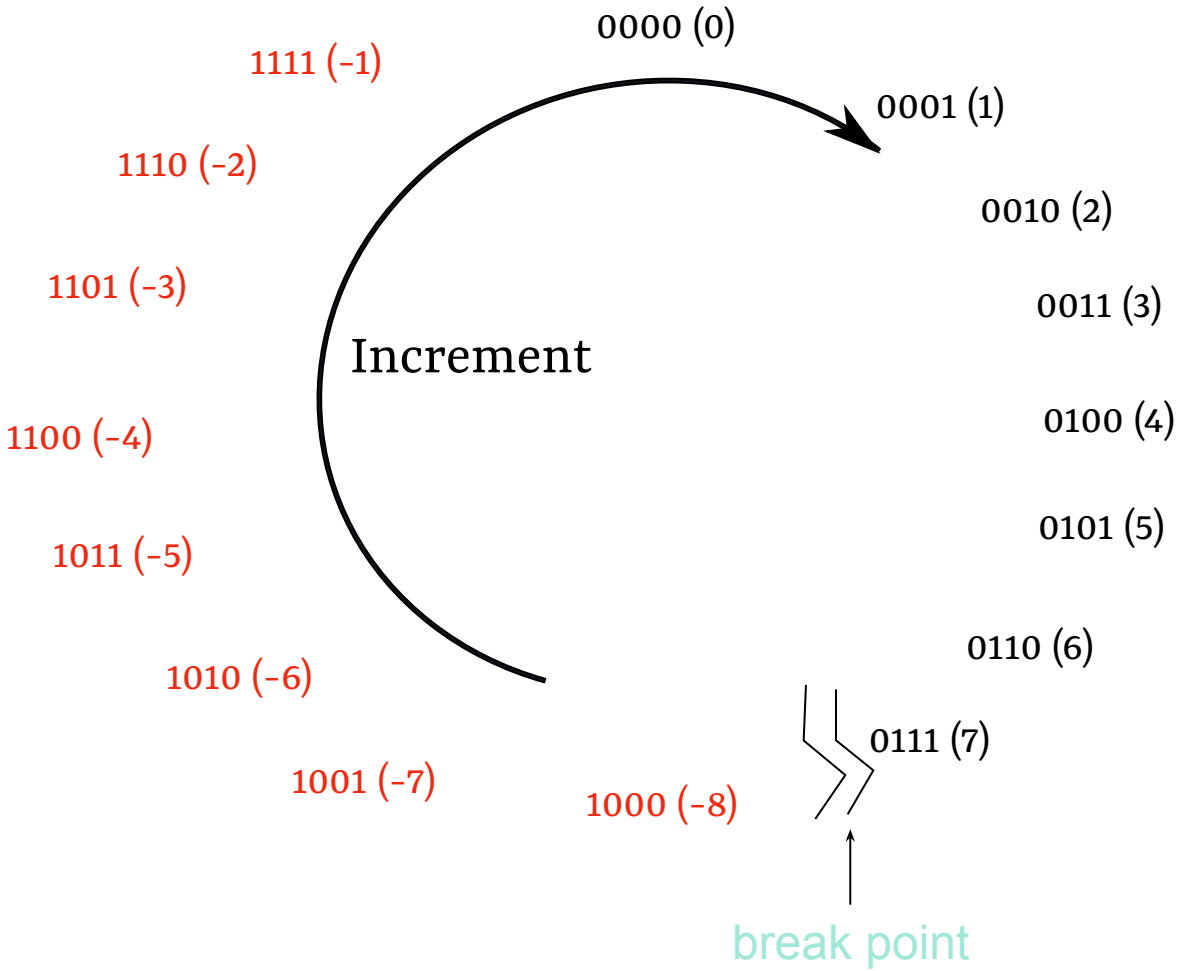
Representing Integers

Decimal Value	Target Notation	Value
5	4-bit unsigned	0101
10	4-bit unsigned	1010
5	4-bit signed	0101
10	4-bit signed	can't be represented!
-5	4-bit signed	1011
-10	4-bit signed	can't be represented!

Given n bits,

- unsigned representation allows values from 0 to 2^(n)-1
- signed representation allows values from -2^(n-1) to 2^(n-1)-1

Number Circle with Negative Numbers



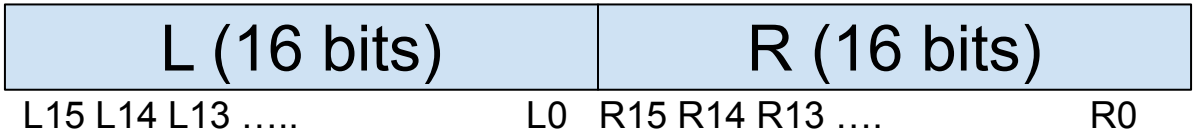
MSB and LSB

* MSB (Most Significant Bit) → The leftmost bit of a binary number. E.g., MSB of 1110 is 1
* LSB (Least Significant Bit) → The rightmost bit of a binary number. E.g., LSB of 1110 is 0

Floating-Point Numbers

* What is a floating-point number ?
  * 2.356
  * 1.3e-10
  * -2.3e+5
* What is a fixed-point number ?
  * Number of digits after the decimal point is fixed
  * 3.29, -1.83

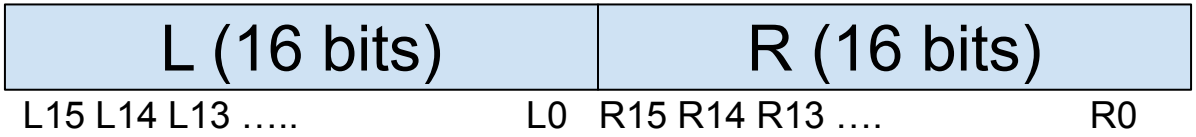
Encoding Floats: An Attempt



Required number, A = L15 * 2^(15) + L14 * 2^(14) + … + L0 * 2^(0) + R15 * 2^(-1) + R14 * 2^(-2) + … + R0 * 2^(-16)

- Range?
- Precision?

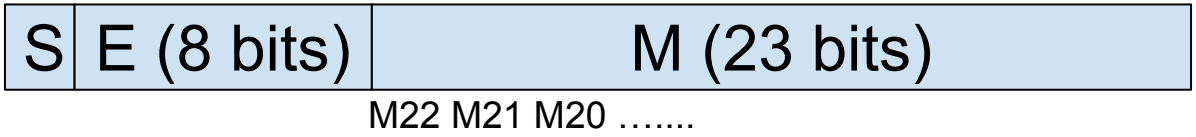
Encoding Floats: An Attempt



Required number, A = L15 * 2^(15) + L14 * 2^(14) + … + L0 * 2^(0) + R15 * 2^(-1) + R14 * 2^(-2) + … + R0 * 2^(-16)

- Range? 0 to ~2^(16)
- Precision? 2^(-16)

Encoding Floats: Attempt II



Exponent = E - 2^(8-1) = E - 127

Mantissa = M22 * 2^(-1) + M21 * 2^(-2) + ... + M0 * 2^(-23)

Required number, A = (-1)^S * (1 + Mantissa) * 2^(Exponent)

- Range? (approx) -2^(128) to +2^(128) (E=0 and E=255 are used for special purposes)
- Precision? ~2^(-126)

Known as the IEEE 754 Standard Format for Floating Point Numbers

IEEE 754 Format: Example

IEEE 754 format of 4.5 = ?

IEEE 754 Format: Example

IEEE 754 Format of 4.5 = 0x40900000

Can you try for

- -0.25
- 1.6

IEEE 754 Format: Example

Decimal	IEEE 754 Format
4.5	0x4090000
-0.25	0xbe800000
1.6	0x3fcccccd

Floating Point Arithmetic is approximate!

How do we represent 0.0?

S = 0

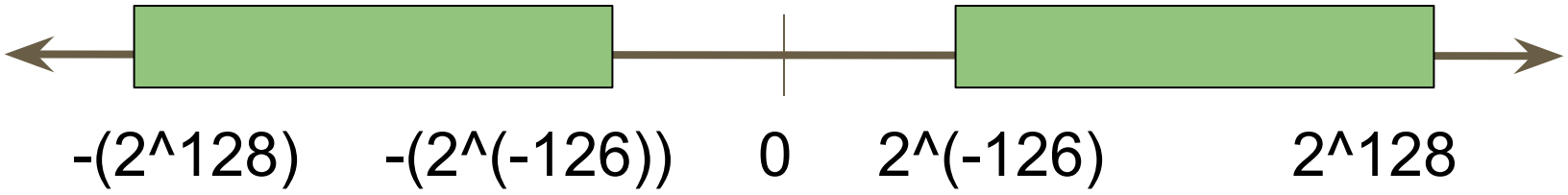
M = 0

E = 1 (remember we agreed on not using E = 0)

(-1) ^ S * (1 + M) * 2^(E-127) = 2^(-126)

- Not perfectly zero!

The Number Line



Need for some special numbers to increase the range

Special Floating Point Numbers

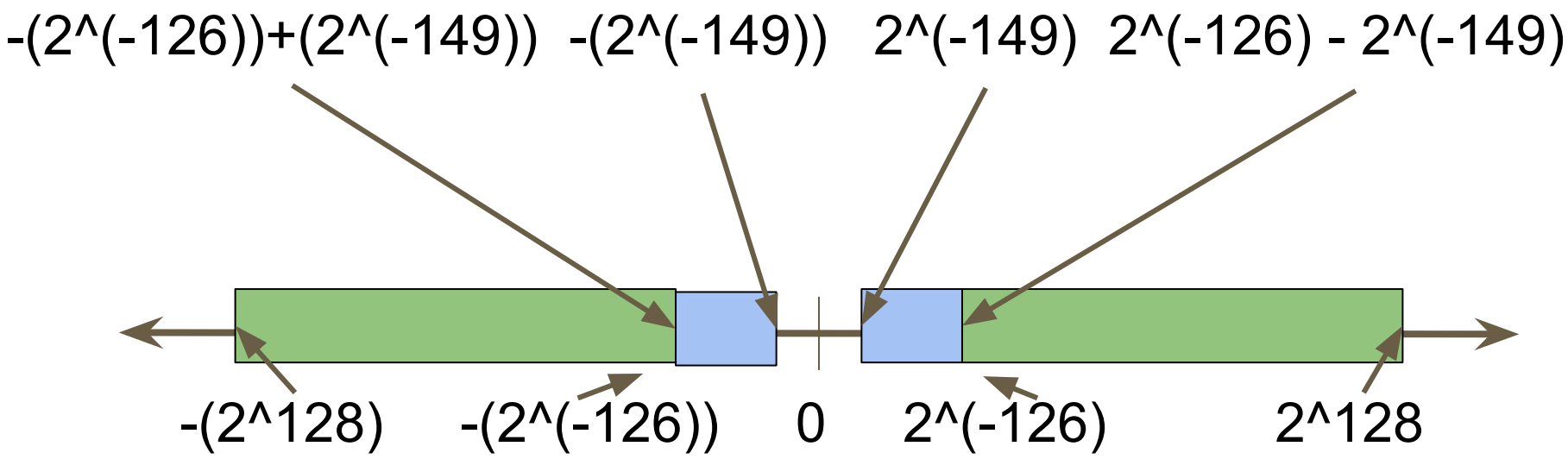
E	M	Value
255	0	∞ if $S = 0$
255	0	$-\infty$ if $S = 1$
255	$\neq 0$	NAN(Not a number)
0	0	0
0	$\neq 0$	Denormal number

* NAN + x= NAN 1/0 = ∞
* 0/0 = NAN -1/0 = -∞
* $\sin^{-1}(5)$ = NAN

Denormal Numbers

- E = 0, M≠ 0
- A = (-1)^S * (0 + M) * 2^(-126)
- Range?
  - From 2^(-149) to 2^(-126) - 2^(-149)

The Number Line



Denormal Numbers

- E = 0, M≠ 0
- A = (-1)^S * (0 + M) * 2^(-126)
- Range?
  - From 2^(-149) to 2^(-126) - 2^(-149)

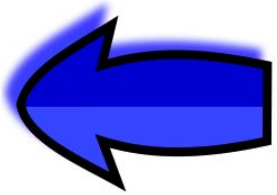
Double Precision Numbers

Field	Size(bits)
S	1
E	11
M	52

- Approximate range of doubles
  - $\pm 2^{1023} = \pm 10^{308}$
  - This is a lot !!!

Outline

* Boolean Algebra
* Positive Integers
* Negative Integers
* Floating Point Numbers
* Strings



ASCII Character Set

* ASCII – American Standard Code for Information Interchange
* It has 128 characters
* First 32 characters (control operations)
  * backspace (8)
  * line feed (10)
  * escape (27)
* Each character is encoded using 7 bits

ASCII Character Set

Character	Code	Character	Code	Character	Code
a	97	A	65	0	48
b	98	B	66	1	49
c	99	C	67	2	50
d	100	D	68	3	51
e	101	E	69	4	52
f	102	F	70	5	53
g	103	G	71	6	54
h	104	H	72	7	55
i	105	I	73	8	56
j	106	J	74	9	57
k	107	K	75	!	33
l	108	L	76	#	35
m	109	M	77	$	36
n	110	N	78	%	37
o	111	O	79	&	38
p	112	P	80	(	40
q	113	Q	81	)	41
r	114	R	82	*	42
s	115	S	83	+	43
t	116	T	84	,	44
u	117	U	85	.	46
v	118	V	86	;	59
w	119	W	87	=	61
x	120	X	88	?	63
y	121	Y	89	@	64
z	122	Z	90	^	94

Unicode Format

* UTF-8 (Universal character set Transformation Format)
  * UTF-8 encodes 1,112,064 characters defined in the Unicode character set. It uses 1-6 bytes for this purpose. E.g.अ आ क ख, ഊ □ ഴ ഡ
  * UTF-8 is compatible with ASCII. The first 128 characters in UTF-8 correspond to the ASCII characters. When using ASCII characters, UTF-8 requires just one byte. It has a leading 0.
  * Most of the languages that use variants of the Roman script such as French, German, and Spanish require 2 bytes in UTF-8. Greek, Russian (Cyrillic), Hebrew, and Arabic, also require 2 bytes.

UTF-16 and 32

* Unicode is a standard across all browsers and operating systems
* UTF-8 has been superseded by UTF-16, and UTF-32
* UTF-16 uses 2 byte or 4 byte encodings (Java and Windows)
* UTF-32 uses 4 bytes for every character (rarely used)