Bowdoin

Representation

CSCI 2330

Office Hours

The Teaching Assistants (TAs) for the semester and their hours are:

- Pauline Unietis <pmunieti@bowdoin.edu>: Monday 6:30-8:30pm
- Joshua Lin <jlin@bowdoin.edu>: Tuesday 7-9pm
- Rose Xi <yxi@bowdoin.edu>: Thursday 7-9pm
- Steven Xu <hxu@bowdoin.edu>: Thursday 7-9pm

All TA Hours are in 224 Searles

My Office Hours are

Tuesday/Thursday 10-11 in H-L 112

Data Representation

- Bases: 2, 10, and 16
- Logical and Bitwise Operators
- Integers: Signs and …not
- Floating Point numbers
- Textual Data (ASCII)

Number Systems

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Binary to Decimal (base 10)

1	0	1	0	1	0	1	0	binary
2^7	2^6	2^5	2^4	2^3	2^2	2^1	2^0	
128	64	32	16	8	4	2	1	decimal

128 + 32 + 8 + 2 = 170

Binary to Hex (base 16)

$8 + 2 = A_{16}$ **$8 + 2 = A_{16}$** **AA_{16}**

Example

Hex	Decimal	Binary
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Write 42 (base 10) as…

- hexadecimal (base 16) **0x2a**
- binary (base 2) **00101010b**

Bowdoin

Exercises 1-4

Boolean Algebra

Logical vs. Bitwise Operators (C)

Operation	Logical	Bitwise
AND	&&	&
OR	||	|
NOT	!	~

1 && 0 = FALSE

10 && 4 = TRUE

1 & 0 = 0

1010 & 0010 = 0

The Big Three

Bitwise AND “&”

x	y	q
0	0	0
0	1	0
1	0	0
1	1	1

Bitwise OR “|”

x	y	q
0	0	0
0	1	1
1	0	1
1	1	1

Bitwise NOT “~”

x	q
0	1
1	0

Shifty Operators

Shift Left “<<”

Number	<<	q
00000101	<< 2	00010100
00000000	<< 6	00000000
10000000	<< 1	00000000
00010000	<< 3	10000000

Shift Right “>>”

Number	>>	q
00000101	>> 2	00000001
00000000	>> 6	00000000
10000000	>> 1	11000000
00010000	>> 3	00000010

Bouns

Bitwise XOR

x	y	q
0	0	0
0	1	1
1	0	1
1	1	**0**

Bitwise NAND

x	y	q
0	0	1
0	1	1
1	0	1
1	1	0

Integers

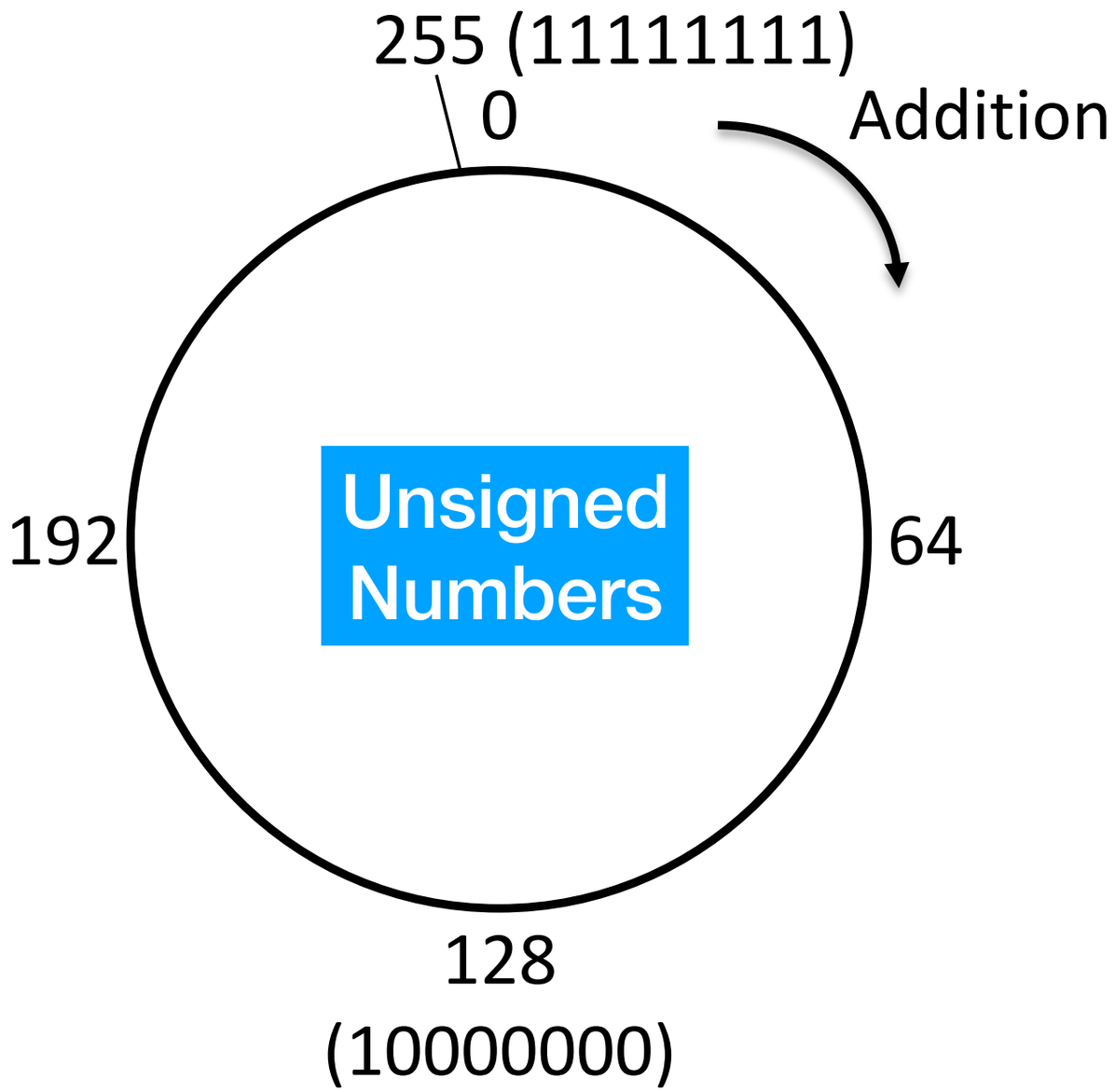
Binary Addition

```
    1
  0110      6
+0100     + 4
------    ----
 1010      10
```

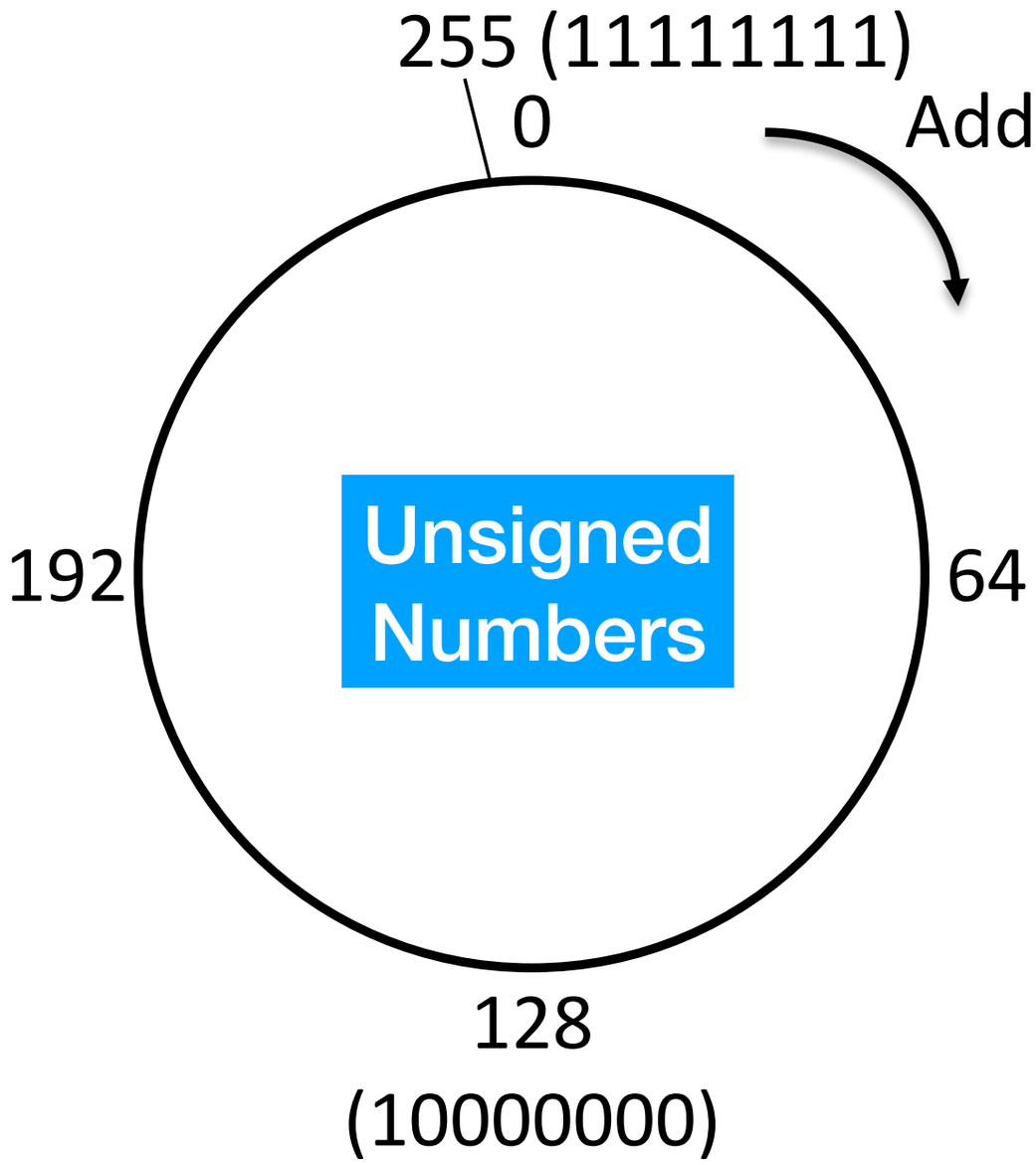
```
  1010      10
+ 1001    + 9
------    ----
 10011     19  or 3?
```

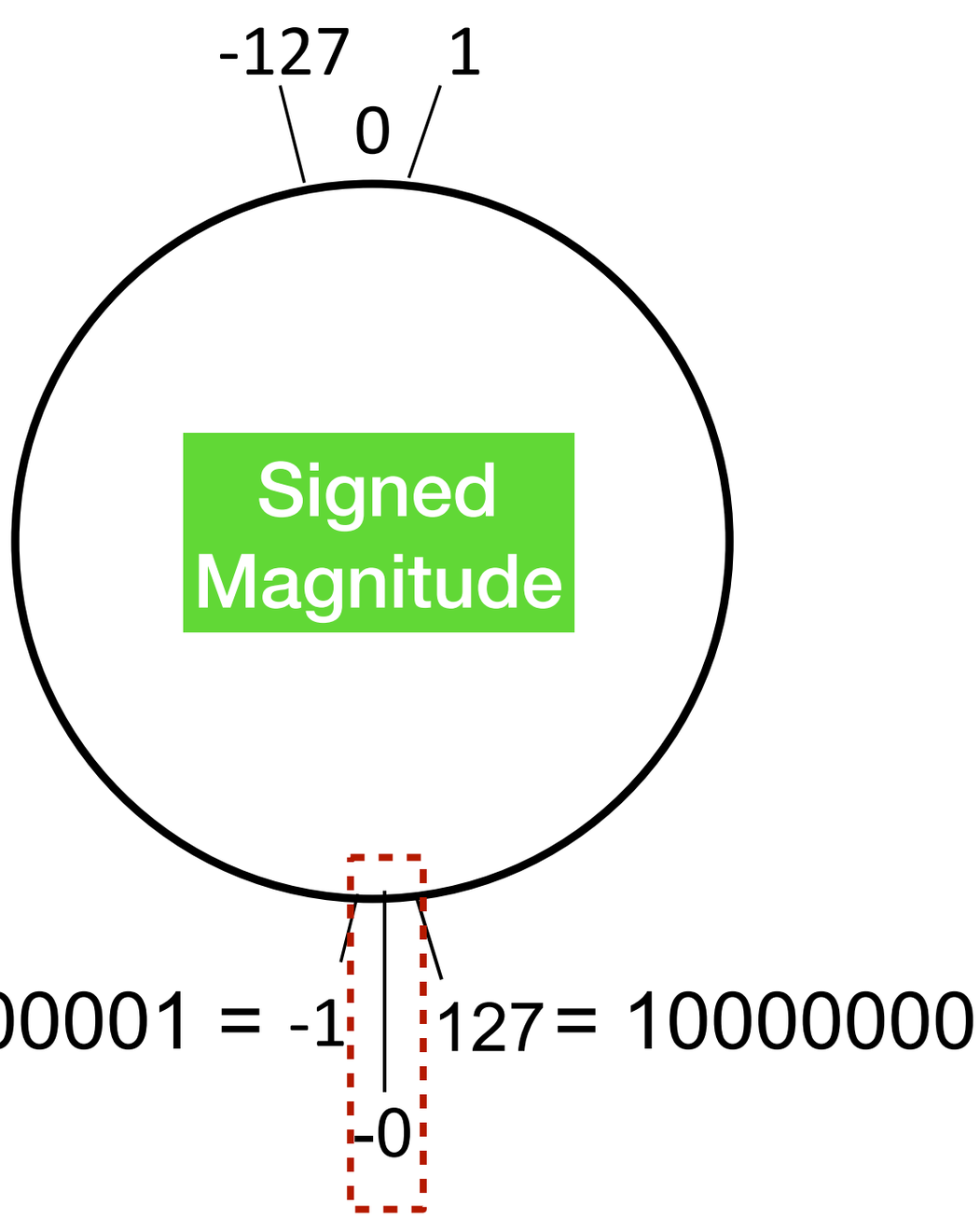
Overflow!

Integers

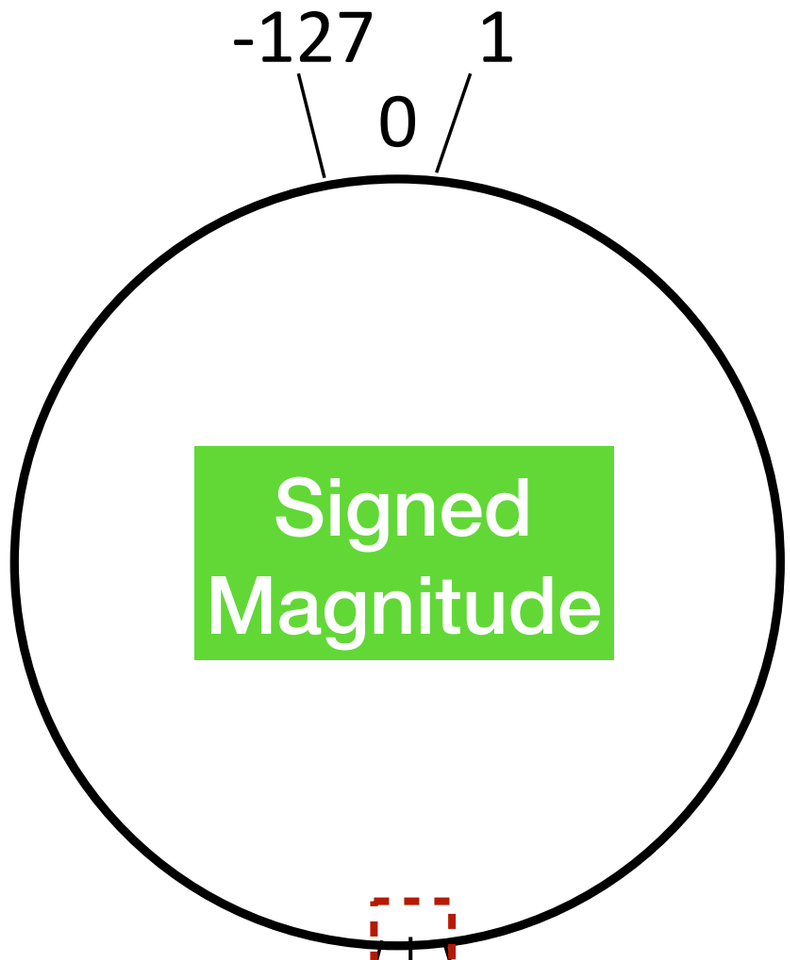

255 (11111111)
0
Addition
192
64
Unsigned Numbers
128
(10000000)

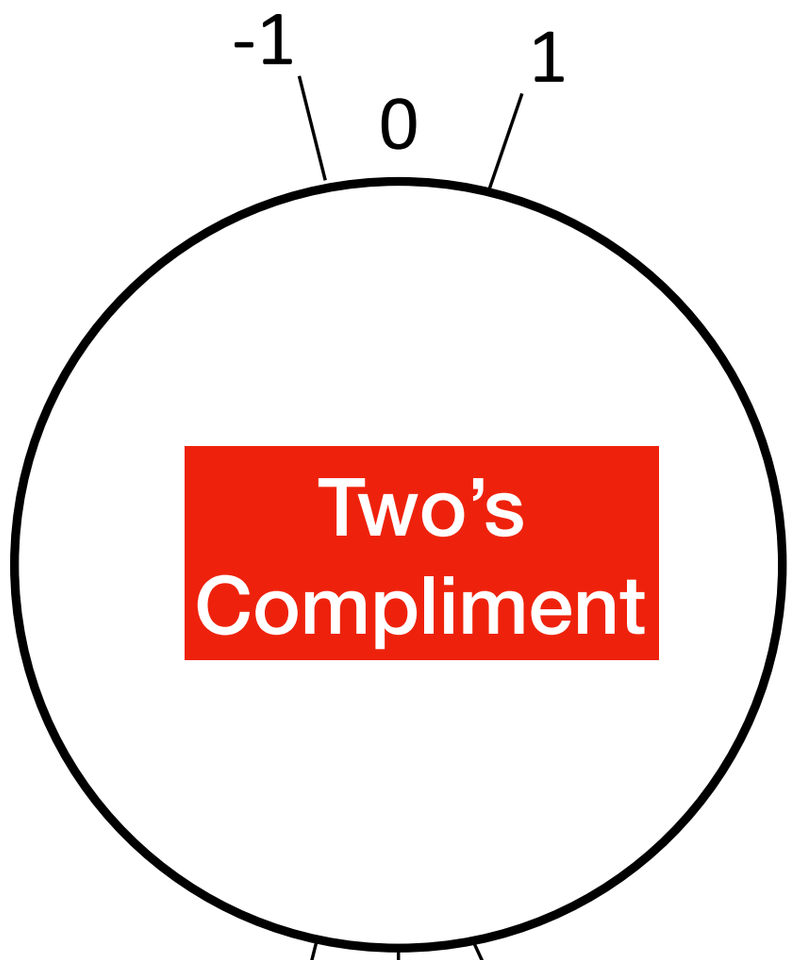
Integers


255 (11111111)
0
Addition
192
64
Unsigned Numbers
128
(10000000)


-127
0
1
Signed Magnitude
10000001 = -1
127 = 10000000
-0

Integers





10000001 = -1 127 = 10000000 10000001 = -127 127 = 01111111

-0

-128 = 10000000

Integers - Sign

Binary	Hex	Unsigned	Sign Magnitude	1's Comp.	2's Comp
0000	0x0	0	0	0	0
0001	0x1	1	1	1	1
0010	0x2	2	2	2	2
0011	0x3	3	3	3	3
0100	0x4	4	4	4	4
0101	0x5	5	5	5	5
0110	0x6	6	6	6	6
0111	0x7	7	7	7	7
1000	0x8	8	-0	-7	-8
1001	0x9	9	-1	-6	-7
1010	0xA	10	-2	-5	-6
1011	0xB	11	-3	-4	-5
1100	0xC	12	-4	-3	-4
1101	0xD	13	-5	-2	-3
1110	0xE	14	-6	-1	-2
1111	0xF	15	-7	-0	-1

Data Sizes

Data Type	Bytes
char	1
short	2
int	4
long	8
float	4
double	8

Byte Ordering

How do we keep a multi-byte things in memory?

0x1234567 (4 byte int)

Memory ->	0x0FF	0x100	0x101	0x102	0x103	0x104
Big Endian	…	0x01	0x23	0x45	0x67	…
Little Endian	…	0x67	0x45	0x23	0x01	…

Which is better?

Intel uses little-endian

Floating Point

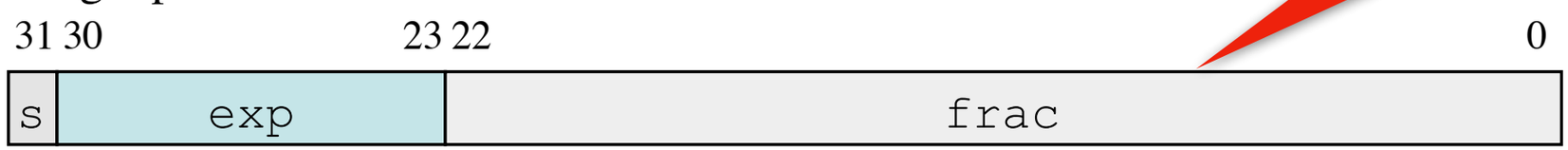
IEEE 754 Standard

4.5 = 4.5 * 10^1

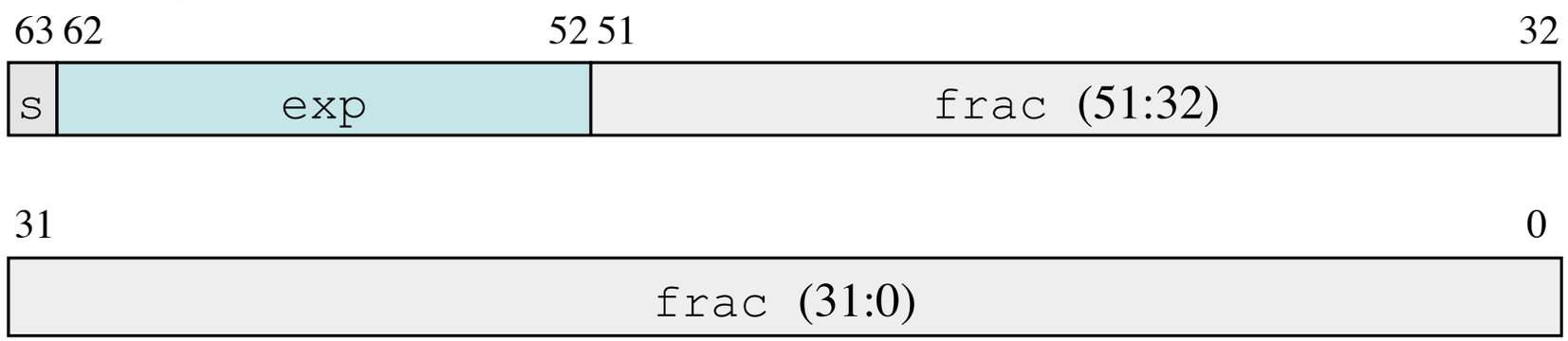
(normalized)

But this needs to be binary!

Single precision



Double precision



Floating Point

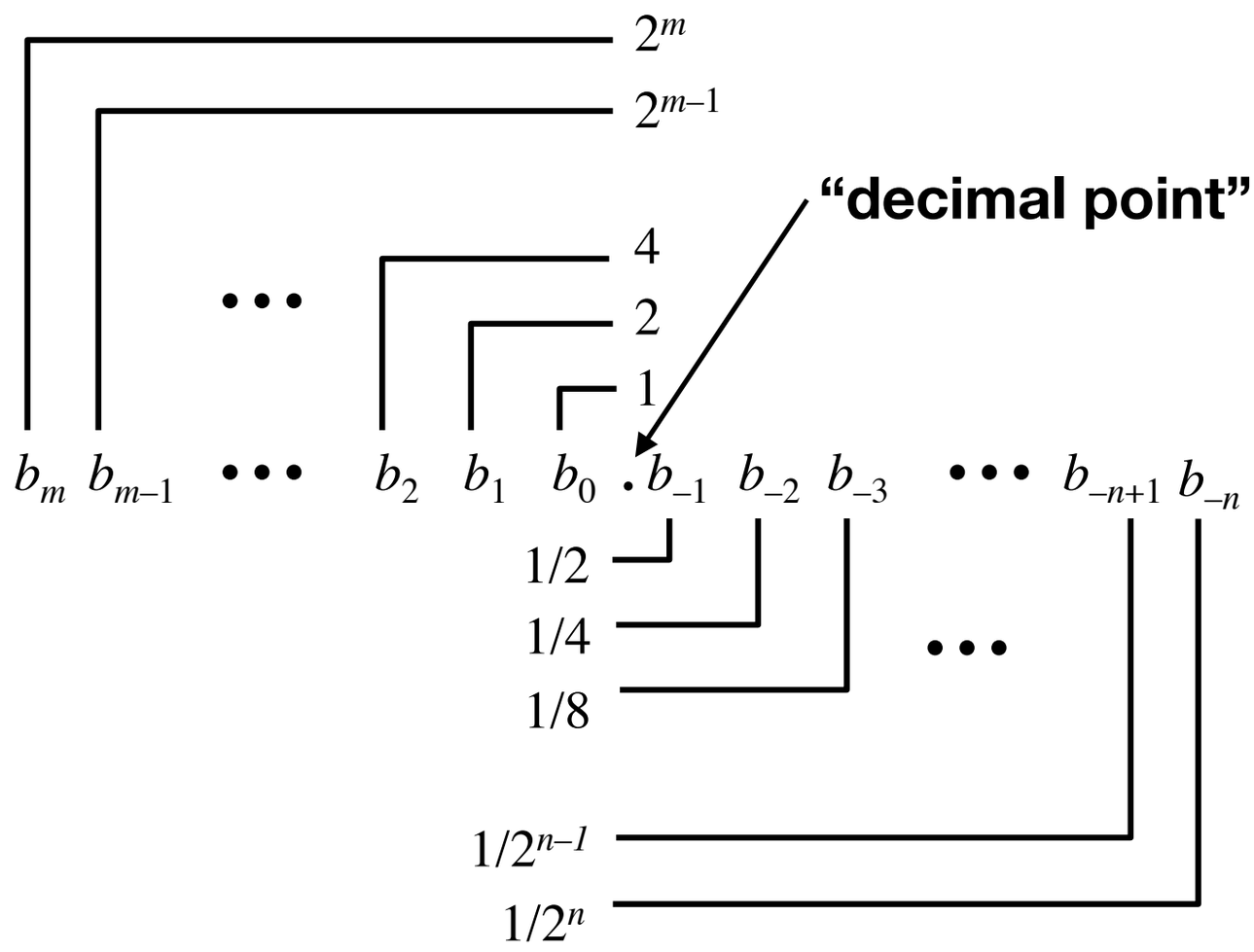
What does 4.5 look like in binary?

4 = 100

0.5 = ?

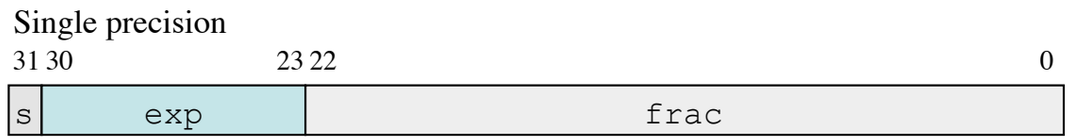
0.5 = 0.1

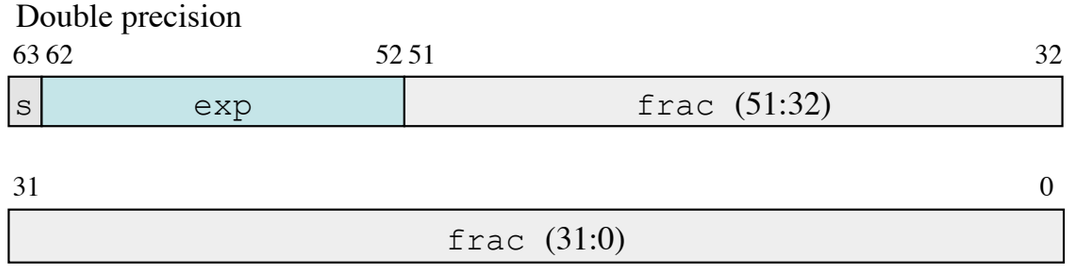
4.5 = 100.1



Floating Point

$12.25 = 1100.01 = 1.10001 * 2^3$ (normalized)





8-bit float (k=4, n=3, bias = 7)

Floating Point

8-bit float (k=4, n=3, bias = 7)

$4.5 = 100.1 = 1.001 * 2^2$ (normalized)

sign = 0 (positive)

exponent = 2 + bias = 2 + 7 = 9 ⟹ 1001

fraction = 1001 ⟹ drop leading 1 ⟹ 001

0 1001 001

Floating Point

8-bit float (k=4, n=3, bias = 7)

0 1001 001

sign = 0 (positive)

exponent = 1001 = 9 ⟹ 9-bais = 9-7 = 2

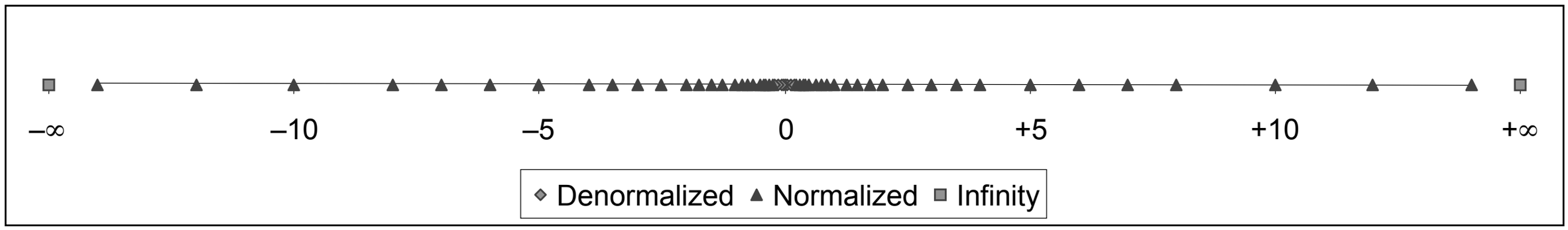
fraction = 001 ⟹ 1.001

$1.001 * 2^2 = 100.1 = 4.5$

~ OR ~

$1.001 = 1.125 * 2^2 = 1.125 * 4 = 4.5$

Floating Point

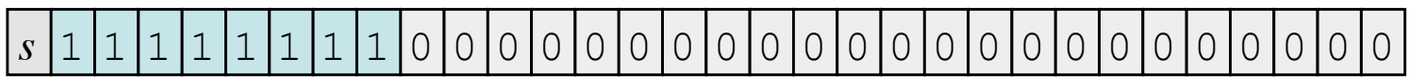


Not evenly distributed

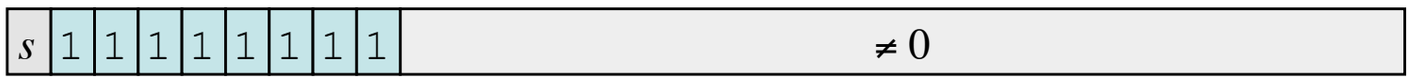
1. Normalized

2. Denormalized

3a. Infinity



3b. NaN



Text ASCII

ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	