Roadmap

Last two weeks: pointers and memory in C

This week: representing data

Characters, integers, floats

Next week: representing code

Goals for Today

Manipulate bits as individual units

- Bitwise operators, masks

Interpret bits as unsigned integers

- Binary polynomial, arithmetic

Represent signed numbers

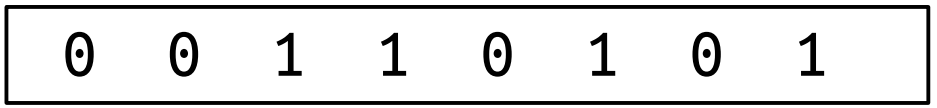
Definitions

Bit: binary digit; a 0 or 1

Byte: 8 bits, 2^8 possible bit patterns

Smallest addressable unit

`char`: one byte value



Bitwise Operators

```
unsigned char a, b;
```

a	0	0	1	1	0	1	0	1
b	0	1	0	1	0	0	1	1

Bitwise Operators

```
unsigned char a, b;
```

	a	0	0	1	1	0	1	0	1
	b	0	1	0	1	0	0	1	1
AND	a & b	0	0	0	1	0	0	0	1

Bitwise Operators

```
unsigned char a, b;
```

	a	0	0	1	1	0	1	0	1
	b	0	1	0	1	0	0	1	1
AND	a & b	0	0	0	1	0	0	0	1
OR	a | b	0	1	1	1	0	1	1	1

Bitwise Operators

```
unsigned char a, b;
```

	a	0	0	1	1	0	1	0	1
	b	0	1	0	1	0	0	1	1
AND	a & b	0	0	0	1	0	0	0	1
OR	a | b	0	1	1	1	0	1	1	1
XOR	a ^ b	0	1	1	0	0	1	1	0

Bitwise Operators

```
unsigned char a, b;
```

	a	0 0 1 1 0 1 0 1
	b	0 1 0 1 0 0 1 1
AND	a & b	0 0 0 1 0 0 0 1
OR	a | b	0 1 1 1 0 1 1 1
XOR	a ^ b	0 1 1 0 0 1 1 0
	a	0 0 1 1 0 1 0 1
NOT	~a	1 1 0 0 1 0 1 0

Bitwise Operators

```
unsigned char a, b;
```

	a	0	0	1	1	0	1	0	1
	b	0	1	0	1	0	0	1	1
AND	a & b	0	0	0	1	0	0	0	1
OR	a | b	0	1	1	1	0	1	1	1
XOR	a ^ b	0	1	1	0	0	1	1	0

	a	0	0	1	1	0	1	0	1
NOT	~a	1	1	0	0	1	0	1	0
Left shift	a << 2	1	1	0	1	0	1		
Right shift	a >> 3				0	0	1	1	0

Bitwise Operators

```
unsigned char a, b;
```

	a	0 0 1 1 0 1 0 1
	b	0 1 0 1 0 0 1 1
AND	a & b	0 0 0 1 0 0 0 1
OR	a | b	0 1 1 1 0 1 1 1
XOR	a ^ b	0 1 1 0 0 1 1 0
	a	0 0 1 1 0 1 0 1
NOT	~a	1 1 0 0 1 0 1 0
Left shift	a << 2	1 1 0 1 0 1 0 0
Right shift	a >> 3	0 0 0 0 0 1 1 0

Code: Setting/testing Bits

So Far

Manipulate bits as individual units

- Bitwise operators, masks

Interpret bits as unsigned integers

- Binary polynomial, arithmetic

Represent signed numbers

Binary Polynomial

In decimal: $5 \quad 6 \quad 7$

$10^2 \quad 10^1 \quad 10^0$

$5 \cdot 10^2 + 6 \cdot 10^1 + 7 \cdot 10^0 = 567$

Binary Polynomial

In decimal: $5 \quad 6 \quad 7$

$10^2 \quad 10^1 \quad 10^0$

$5 \cdot 10^2 + 6 \cdot 10^1 + 7 \cdot 10^0 = 567$

In binary: $0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$

$2^7 \quad 2^6 \quad 2^5 \quad 2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0$

Binary Polynomial

In decimal: $5 \quad 6 \quad 7$

$10^2 \quad 10^1 \quad 10^0$

$5 \cdot 10^2 + 6 \cdot 10^1 + 7 \cdot 10^0 = 567$

In binary: $0 \quad 1 \quad 1 \quad 0 \quad 1 \quad 0 \quad 1 \quad 1$

$2^7 \quad 2^6 \quad 2^5 \quad 2^4 \quad 2^3 \quad 2^2 \quad 2^1 \quad 2^0$

$0 \cdot 2^7 + 1 \cdot 2^6 + 1 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 1 \cdot 2^0$

$= 1 \cdot 64 + 1 \cdot 32 + 1 \cdot 8 + 1 \cdot 2 + 1 \cdot 1 = 107$

Hexadecimal

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

16 = 0x10, 17 = 0x11

Hexadecimal

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

16 = 0x10, 17 = 0x11

107 = 0110 1011

Hexadecimal

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

16 = 0x10, 17 = 0x11

107 = 0110 1011 = 0x6b

6 b

Hexadecimal

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

16 = 0x10, 17 = 0x11

107 = 0110 1011 = 0x6b
6 b

255 = 1111 1111 = 0xff
f f

Representing Characters

Dec	Hex	Char
0	0x0	‘\0’
	...	
32	0x20	‘ ’
33	0x21	‘!’
	...	

Dec	Hex	Char
48	0x30	‘0’
49	0x31	‘1’
	...	
57	0x39	‘9’
	...	
65	0x41	‘A’
66	0x42	‘B’
	...	
90	0x5a	‘Z’

Dec	Hex	Char
	...	
97	0x61	‘a’
98	0x62	‘b’
	...	
122	0x7a	‘z’

ASCII Table

Binary Addition

```
    1
  107
+  58
-----
  165
```

Binary Addition

```
    1                    
  107          0110 1011
+  58        + 0011 1010
-----       ------------
  165
```

Binary Addition

```
    1                    
  107          0110 1011
+  58        + 0011 1010
-----       ------------
  165                  1
```

Binary Addition

```
    1                1
   107          0110 1011
 +  58        + 0011 1010
-----        -----------
   165             01
```

Binary Addition

```
   1          1  1
  107       0110 1011
+  58     + 0011 1010
-----     ----------
  165          0101
```

Binary Addition

```
   1          1111  1
 107          0110 1011
+ 58        + 0011 1010
-----       ---------
 165          1010 0101
```

Binary Addition

```
    1          1111  1
  107          0110 1011
+  58        + 0011 1010
-----        -----------
  165          1010 0101
```

128 + 32 + 4 + 1 = 165

Overflow

```
   255        1111 1111
 +   1      + 0000 0001
-------    -------------
```

Overflow

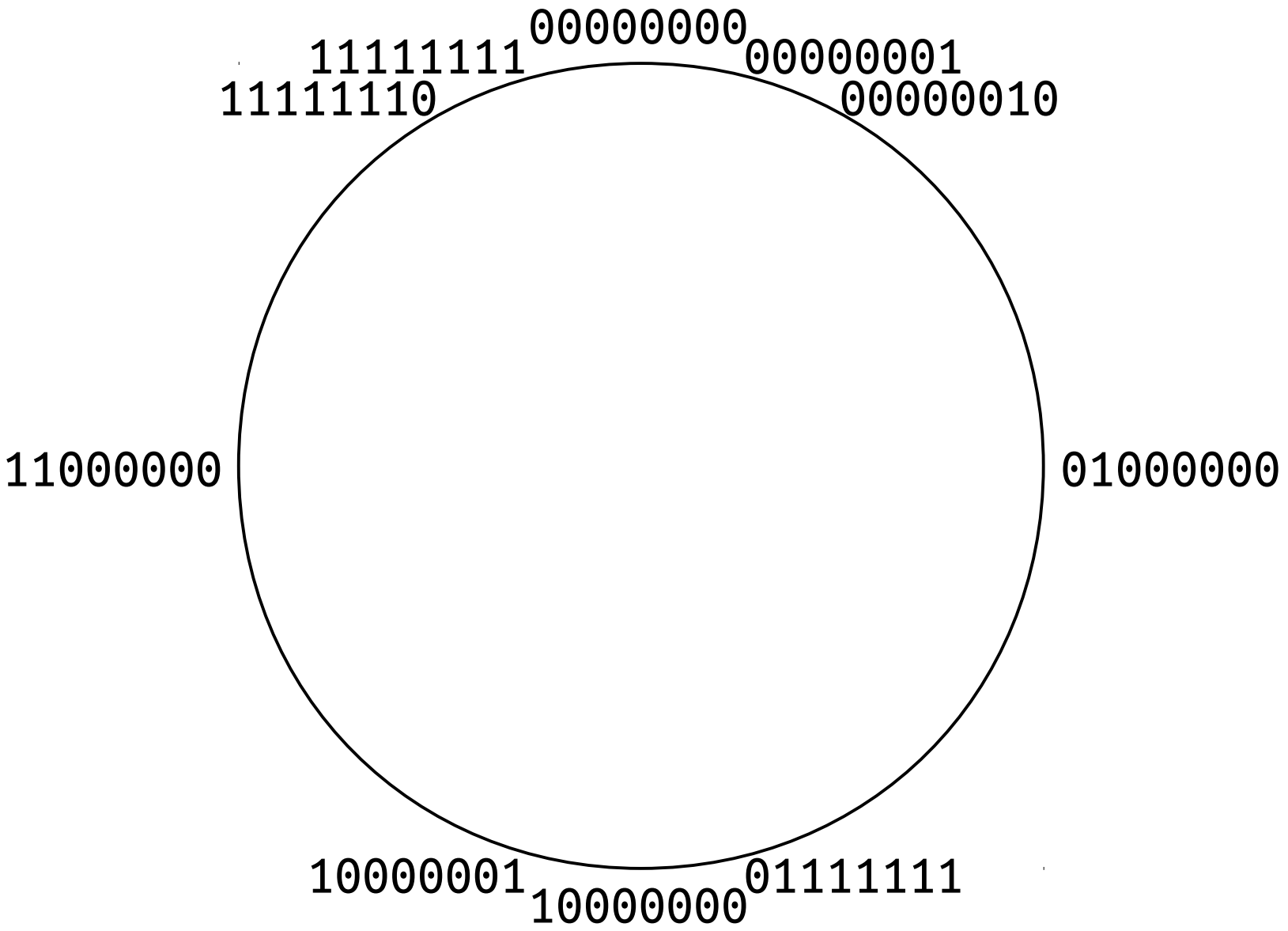
```
      1 1111 111
  255    1111 1111
+   1  + 0000 0001
-----  ----------
      1 0000 0000
```

Overflow

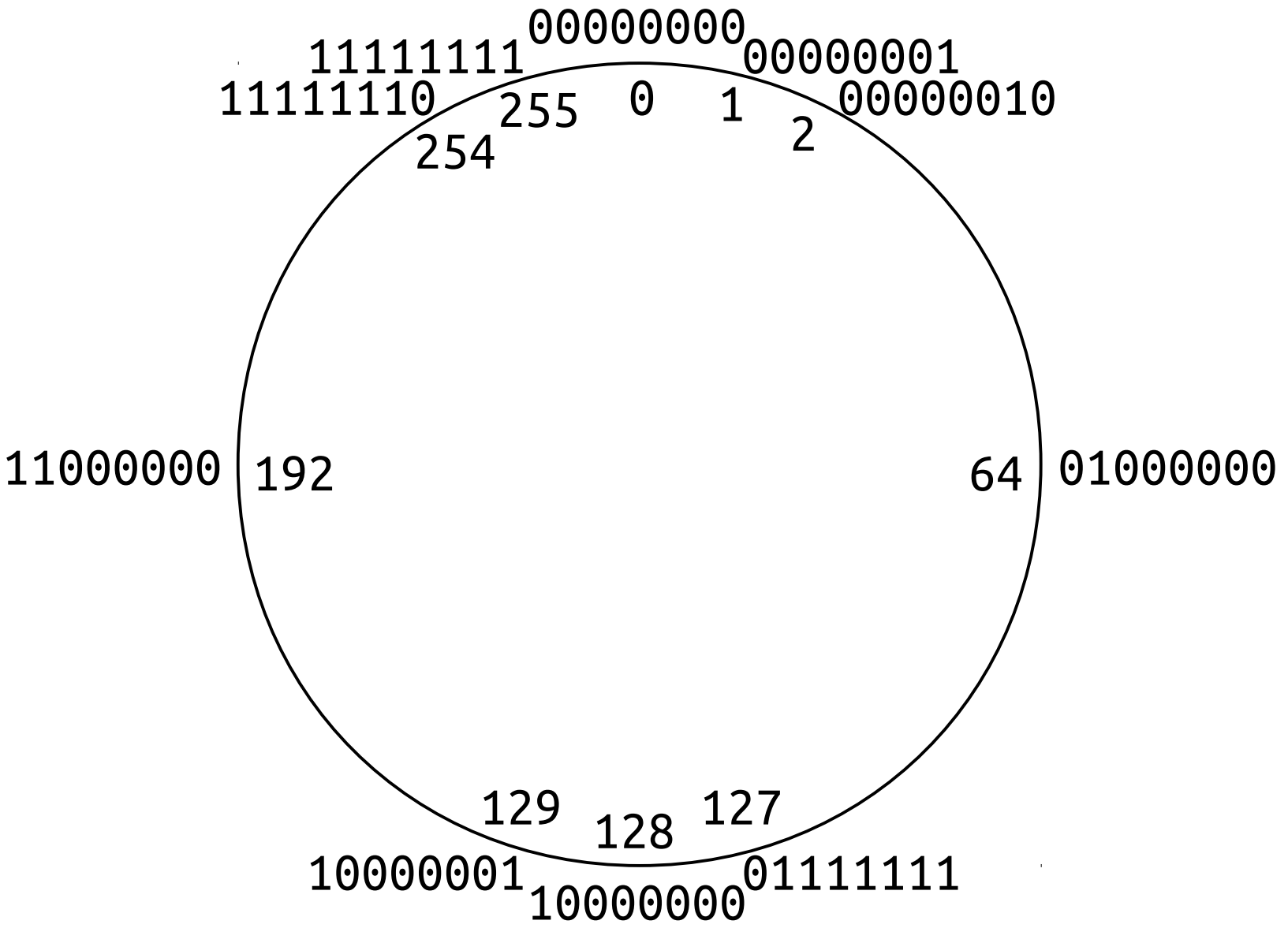
```
      255          1 1111 111
    +   1        + 0000 0001
-----------    -------------
        0          0000 0000
```

No warning or indication

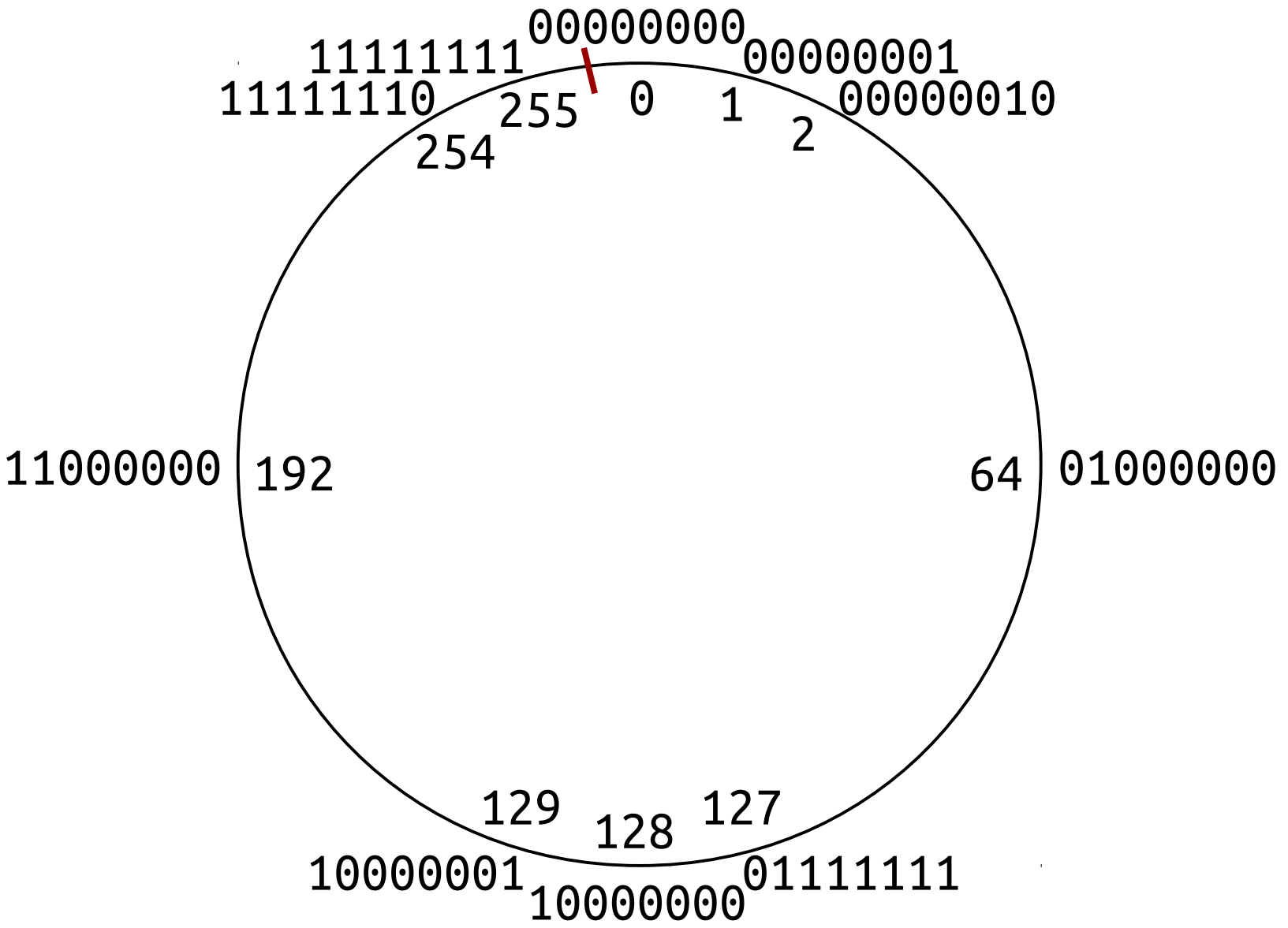
Unsigned Char



Unsigned Char



Unsigned Char



So Far

Manipulate bits as individual units

- Bitwise operators, masks

Interpret bits as unsigned integers

- Binary polynomial, arithmetic

Represent signed numbers

Signed Numbers

Idea: use half of circle for negatives

Signed Numbers

Idea: use half of circle for negatives

First attempt: sign and magnitude

MSB is sign bit, rest of bits store absolute value

E.g. `-1 = 1000 0001, -20 = 1001 0100`

Signed Numbers

Idea: use half of circle for negatives

First attempt: sign and magnitude

MSB is sign bit, rest of bits store absolute value

E.g. `-1 = 1000 0001, -20 = 1001 0100`

Problems

Two zeros: `0000 0000` and `1000 0000`

Arithmetic requires special cases

Signed Numbers

Idea: use half of circle for negatives

First attempt: sign and magnitude

MSB is sign bit, rest of bits store absolute value

E.g. `-1 = 1000 0001, -20 = 1001 0100`

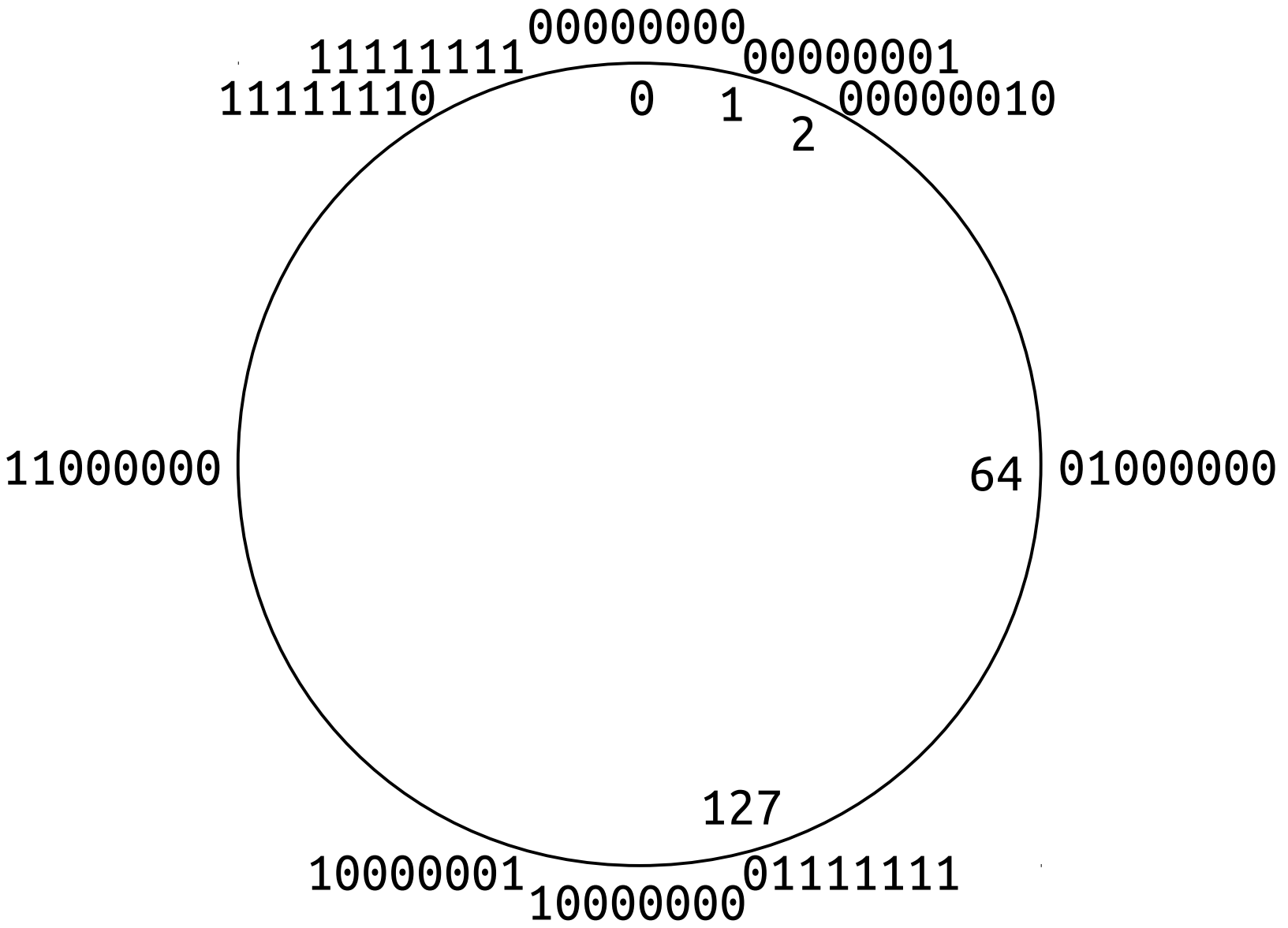
Problems

Two zeros: `0000 0000` and `1000 0000`

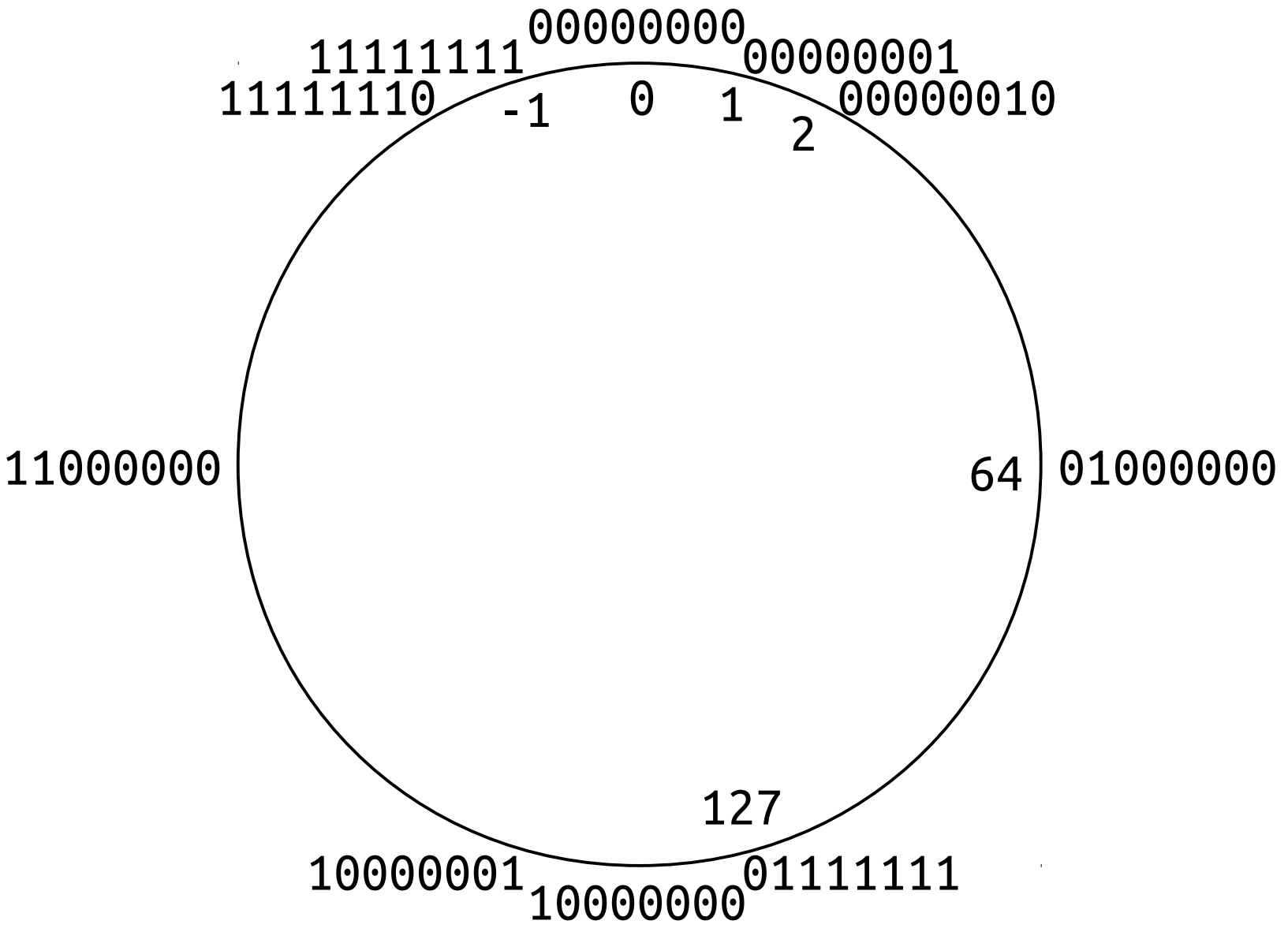
Arithmetic requires special cases

Solution: 2's complement

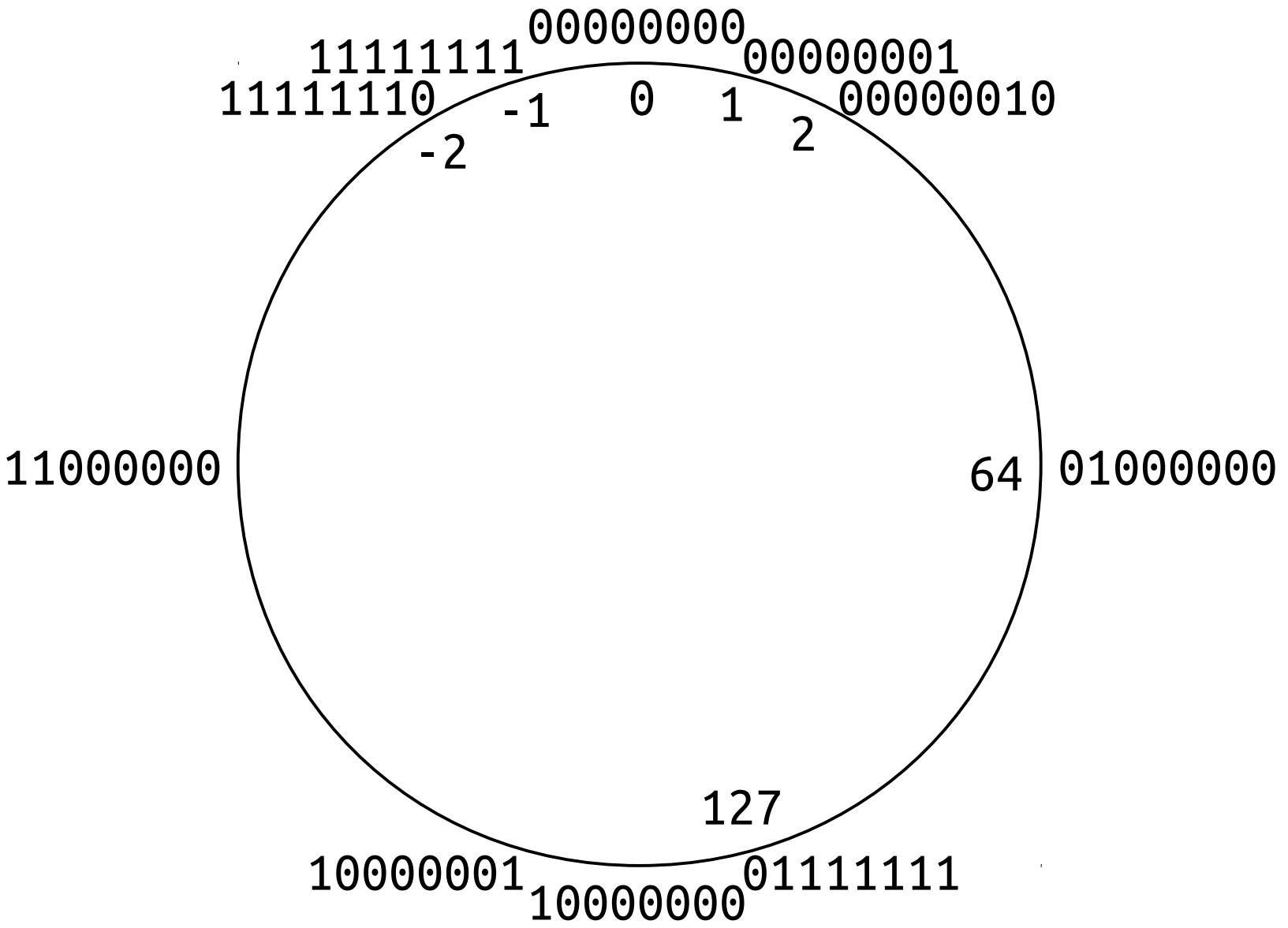
2’s Complement



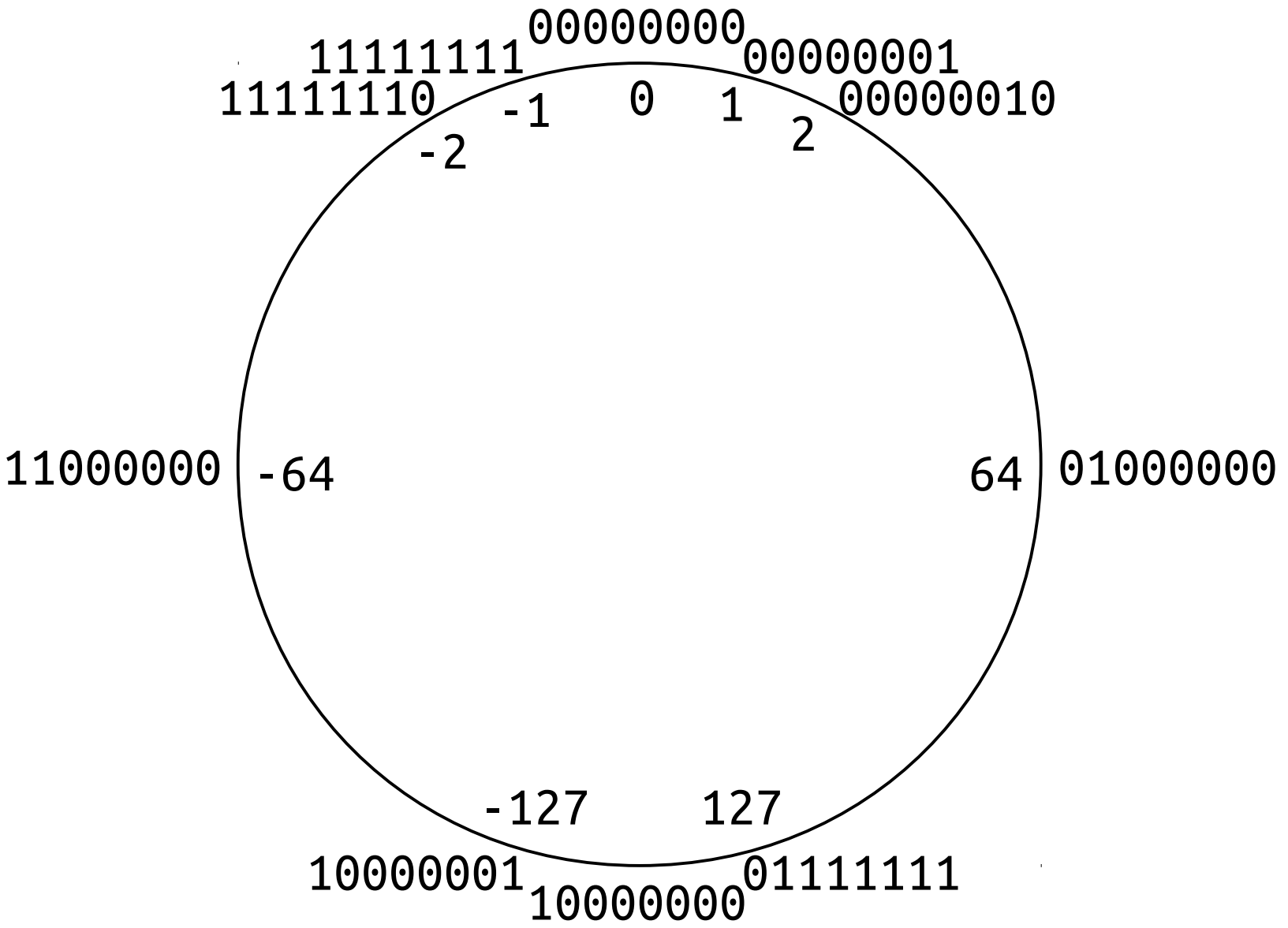
2’s Complement



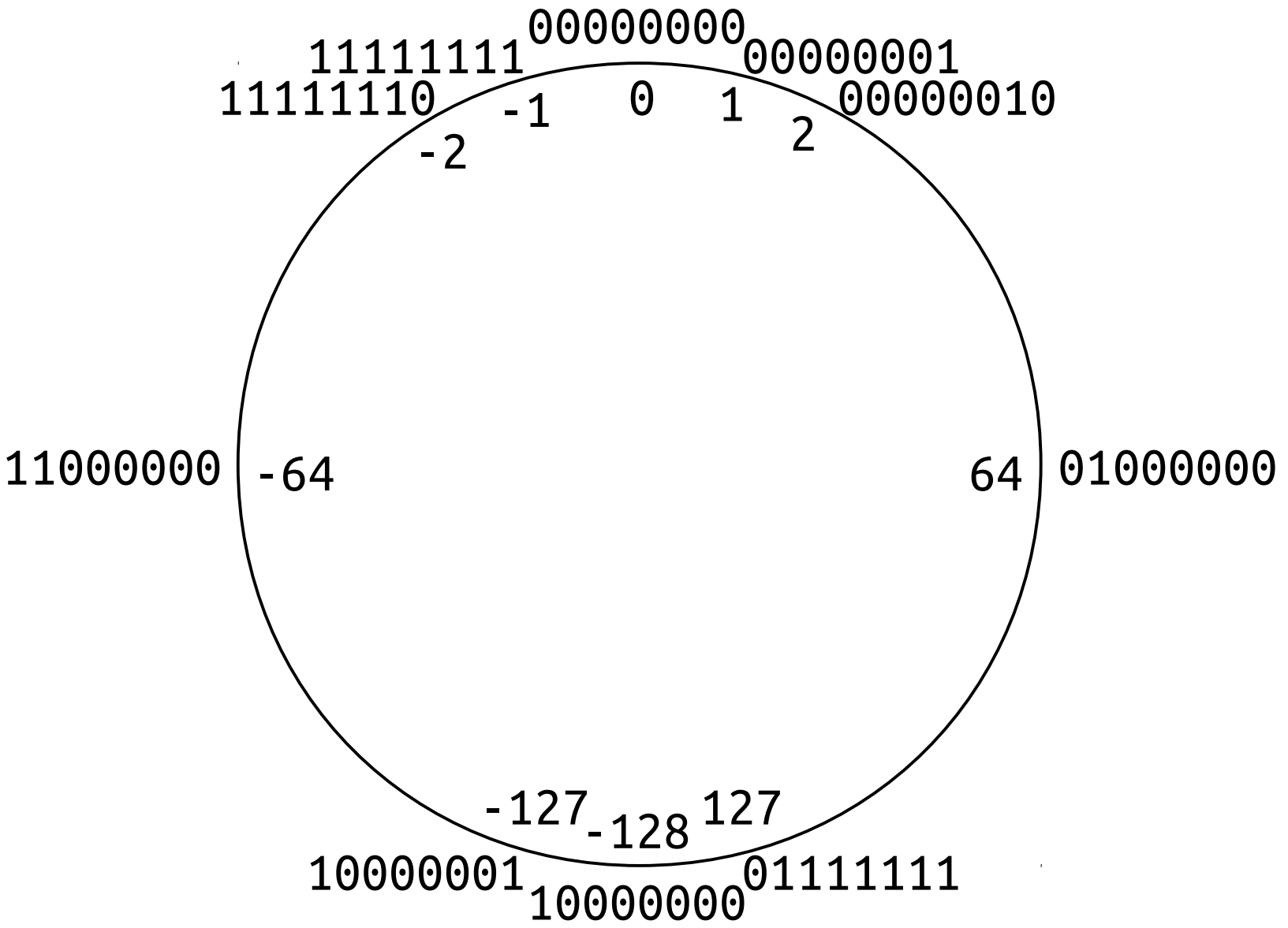
2’s Complement



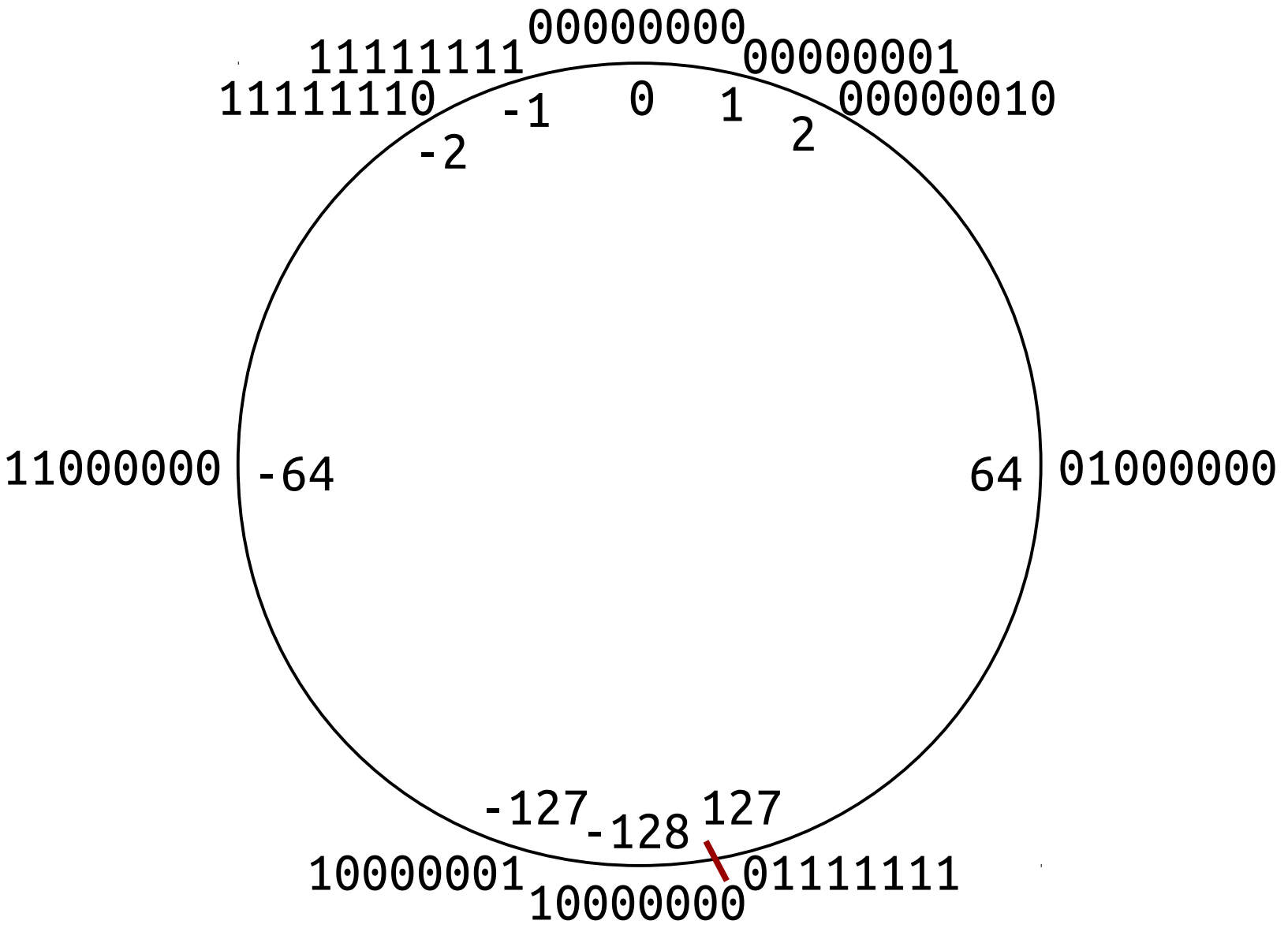
2’s Complement



2’s Complement



2’s Complement



Addition with Signed

```
     5        0000 0101
+   -2      + 1111 1110
-------    -------------
     3
```

Addition with Signed

```
      5        0000 0101
+    -2      + 1111 1110
-------      -----------
      3             11
```

Addition with Signed

```
                 1
      5    0000 0101
+    -2  + 1111 1110
-------  -----------
      3          011
```

Addition with Signed

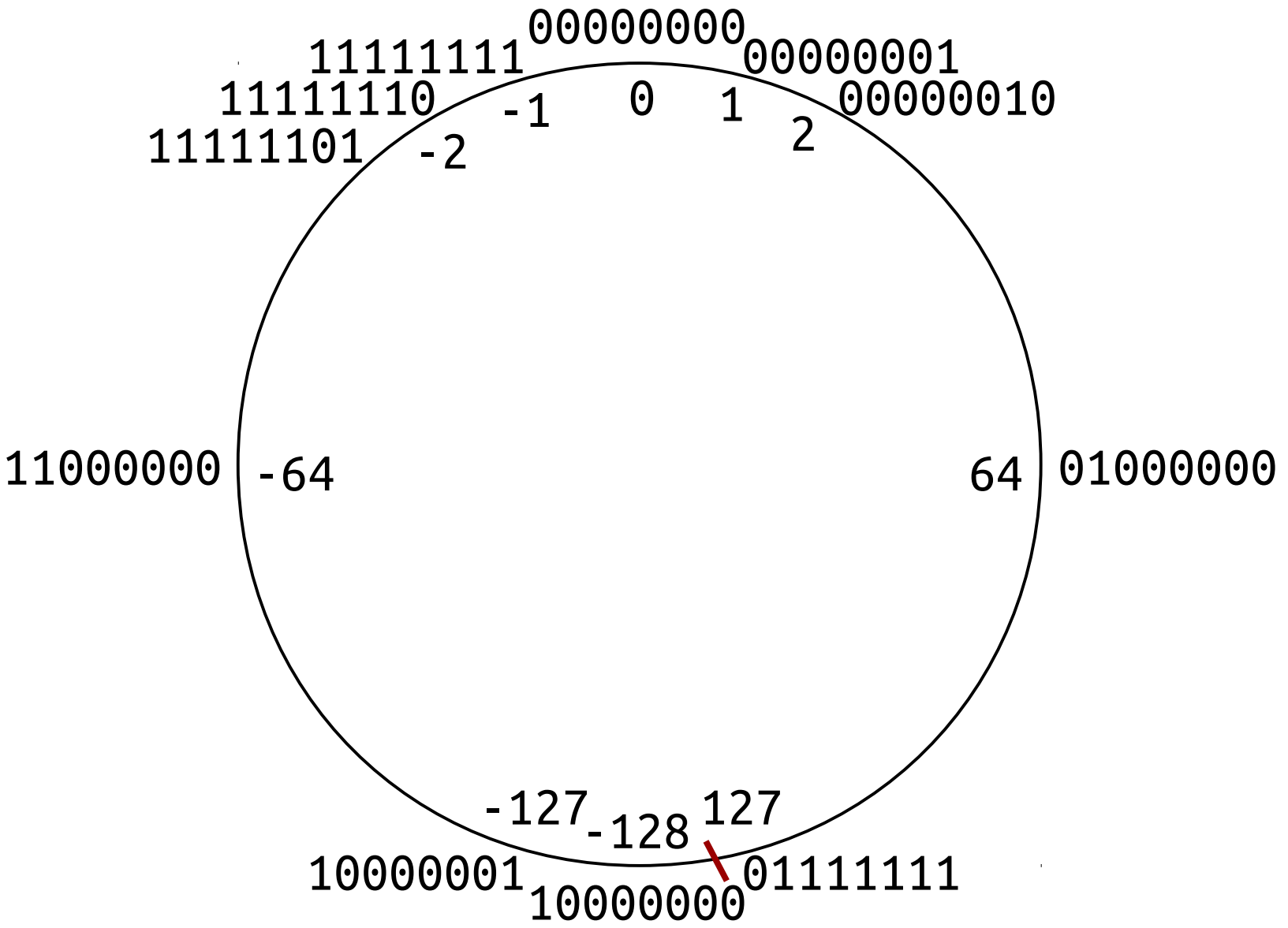
```
        1 1111 1
    5     0000 0101
+  -2   + 1111 1110
-------  ----------
    3   1 0000 0011
```

Addition with Signed

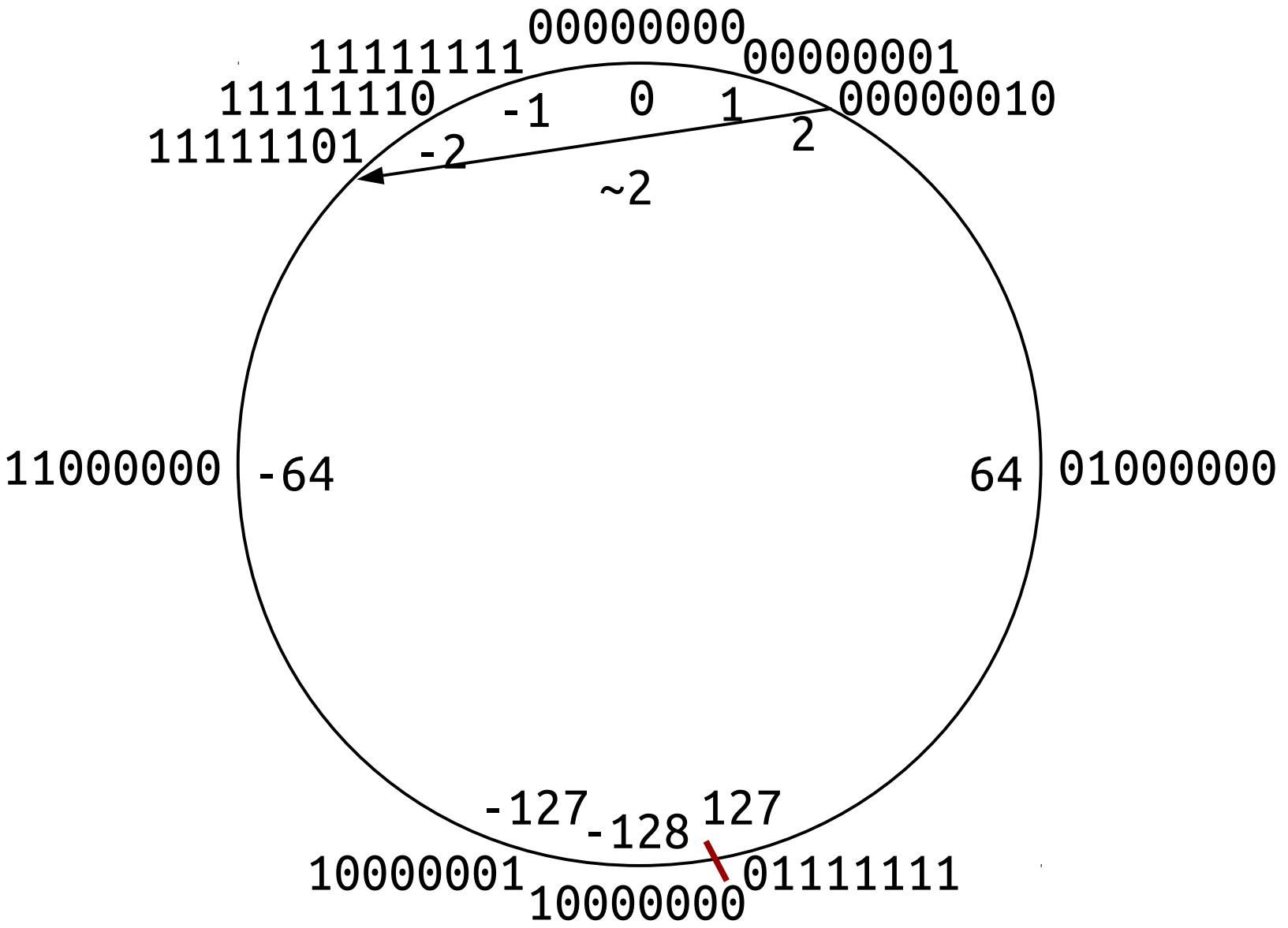
```
        1 1111 1
    5     0000 0101
+  -2   + 1111 1110
-------  ----------
    3     0000 0011
```

Same logic, no special case

2’s Complement



2’s Complement



Bit Pattern for -x

Reflection across number circle

Almost ~x

Formula: `-x = ~x + 1`

Important number to know: `-1 = 0xff`

Asymmetry

What is `-(-128)`?

Asymmetry

What is -(-128)?

```
-128 = 0x80 (1000 0000)
~(-128) = 0x7f (0111 1111)
~(-128) + 1 = 0x80 (1000 0000) = -128!
```

Signed vs. Unsigned

Same

- Arithmetic (+, -, *, /)
- Equality (==, !=)

Signed vs. Unsigned

Same

- Arithmetic (+, -, *, /)
- Equality (==, !=)

Different

- Discontinuity (where over/underflow occurs)
- Right shift (>>)
- Relational (>, <)
  - When comparing signed to unsigned, unsigned comparison used

Summary

Manipulate bits as individual units

- Bitwise operators, masks

Interpret bits as unsigned integers

- Binary polynomial, arithmetic

Represent signed numbers