lecture 1

- two's complement
- floating point numbers
- hexadecimal

Car odometer (fixed number of digits)

$$
\begin{array}{r}
0\;0\;0\;0\;9\;9\;9 \\
+\quad 0\;0\;0\;0\;0\;0\;1 \\
\hline
0\;0\;0\;1\;0\;0\;0
\end{array}
$$

$$
\begin{array}{r}
999999 \\
+\ 000001 \\
\hline
000000
\end{array}
$$

$$\Rightarrow \quad 999999 \equiv -1$$

$$
\begin{array}{r}
328769 \\
+ \quad ? \\
\hline
000000
\end{array}
$$

$$\begin{array}{r} 328769 \\ +\ 671231 \\ \hline 000000 \end{array}$$

$$\equiv -328769$$

If you know what "modular arithmetic" is (MATH 240), then you recognize this: addition of integers mod 10^6.

Q: How to represent negative numbers in binary ?

A: Given an 8 bit binary number m, define -m so that m + (-m) = 0.

$$
\begin{array}{lr l}
 & 00011010 & m \\
+ & ? & -m \\
\hline
 & 00000000 &
\end{array}
$$

Two's complement representation of integers

Example: How to represent -26 ?

Use a trick!

$$
\begin{array}{rcccccccc}
 & 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 \\
+ & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 1 \\
\hline
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1
\end{array}
$$

$n = 26$

$\leftarrow$ invert bits

```
   00011010            m = 26
+  11100101   ← invert bits
   --------
   11111111
+         1   ← add 1
   --------
   00000000



   11100101   ← inverted bits
+         1   ← add 1
   --------
   11100110   ← -26
```

Another example: What is -0 ?

$$
\begin{array}{rcccccccc}
 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
+ & & & & & & & & 1 \\
\hline
 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
$$

m = 0

invert bits

add 1

$$
\left.
\begin{array}{rcccccccc}
 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\
+ & & & & & & & & 1 \\
\hline
 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}
\right\}
$$

We have verified that -0 = 0.

What about m = 128 ? What is -128 ?

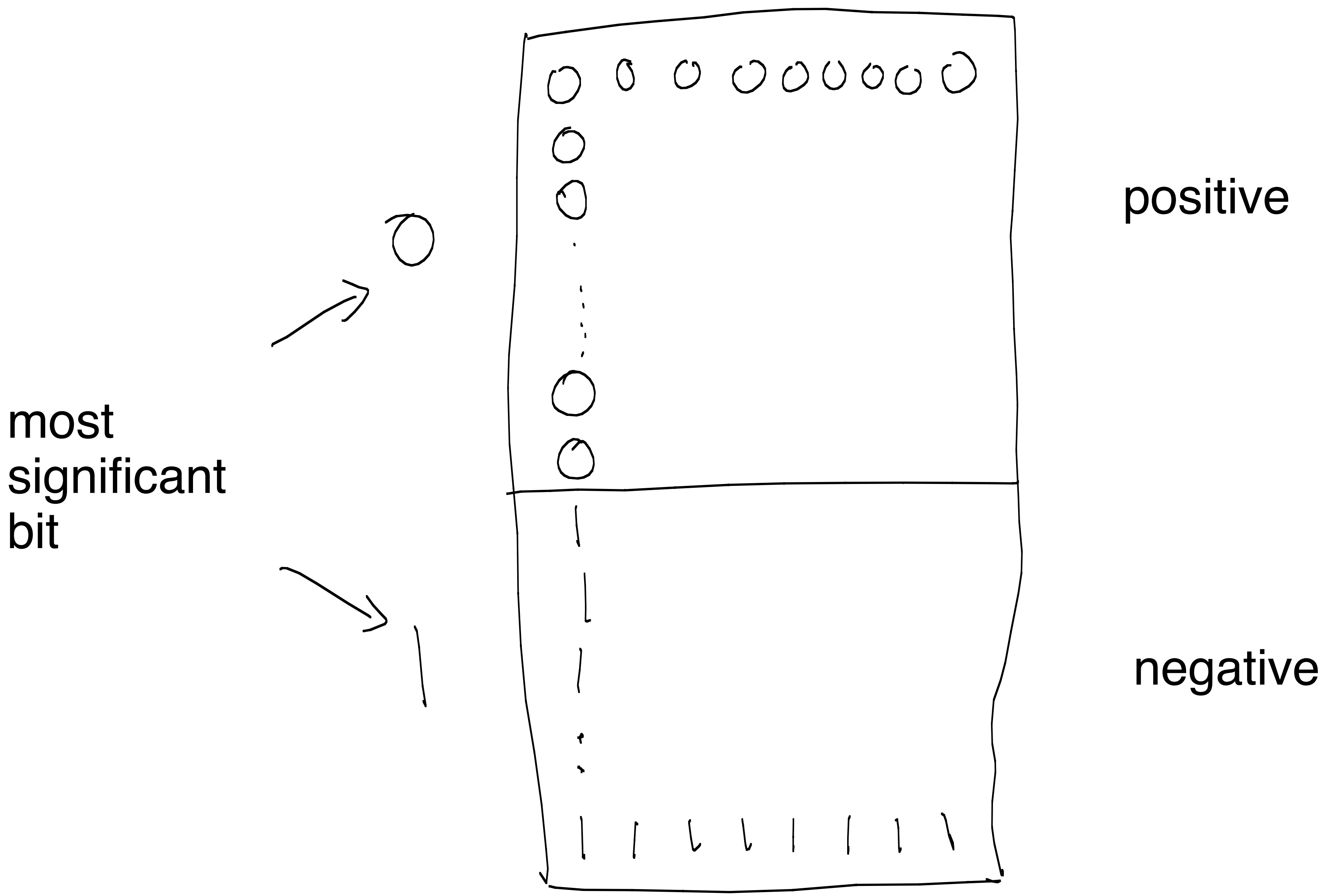
	1 0 0 0 0 0 0 0	m = 128
	0 1 1 1 1 1 1 1	invert bits
+	1	add 1
	0 0 0 0 0 0 0 0	

	0 1 1 1 1 1 1 1	
+	1	
	1 0 0 0 0 0 0 0	m = - 128

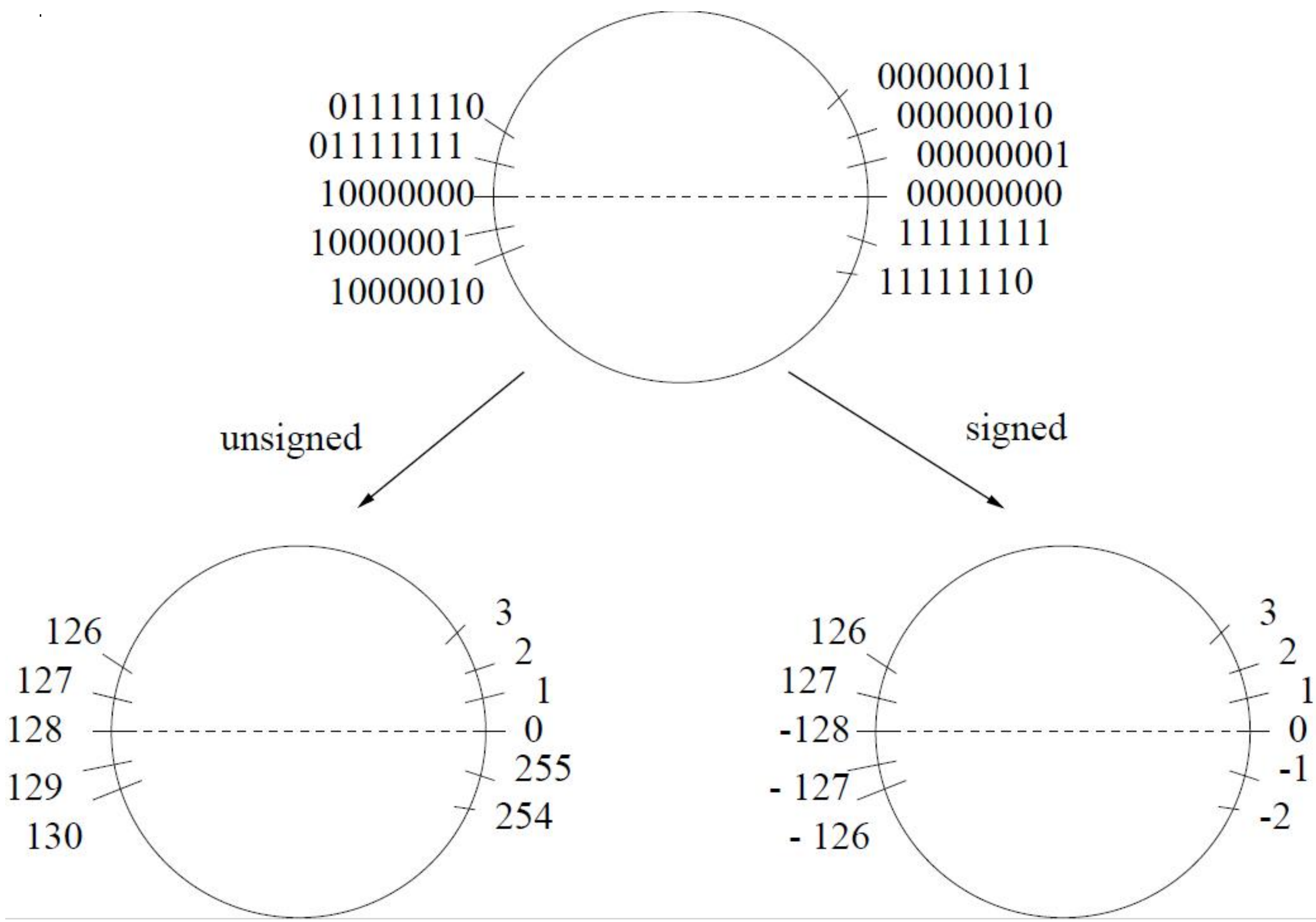
Thus, 128 is equivalent to -128.

binary	"unsigned"	"signed"
0 0 0 0 0 0 0 0	0	0
0 0 0 0 0 0 0 1	1	1
0 0 0 0 0 0 1 0	2	2
0 0 0 0 0 0 1 1	3	3
0 0 0 0 0 1 0 0	4	4
⋮	⋮	⋮
0 1 1 1 1 1 1 1	127	127
1 0 0 0 0 0 0 0	128	−128
⋮	⋮	⋮
1 1 1 1 1 1 1 0	254	−2
1 1 1 1 1 1 1 1	255	−1

signed integers



8 bit integers (unsigned vs. signed)



n bits defines 2^n integers

unsigned

$0, 1, \ldots \ldots, 2^n - 1$

signed

$-2^{n-1}, \ldots, 0, \ldots, 2^{n-1} - 1$

Take $n = 32$.

The largest signed integer is $2^{31} - 1$.

$2^{10} = 1024 \sim 10^3$ = one thousand.

$2^{20} \sim 10^6$ = one million

$2^{30} \sim 10^9$ = one billion

$2^{31} \sim 2{,}000{,}000{,}000$ = two billion

Java Example

```
int  j = 4000000000;   // 4 billion  >  2^31
```

This gives a compiler error. "The literal of type int is out of range."

```
int  j = 2000000000;  //   2 billion <  2^31

System.out.println( 2 * j );
```

// This prints out -294967296.
// To understand why these particular digits are printed, you
// would need to convert 4000000000 to binary, which I don't
// recommend.)

lecture 1

- two's complement
- floating point numbers
- hexadecimal

Floating Point

"decimal point"

$$26.375 = 2 \times 10^{1} + 6 \times 10^{0} + 3 \times 10^{-1} + 7 \times 10^{-2} + 5 \times 10^{-3}$$

"binary point"

$$(11010.011)_2 = 2^{4} + 2^{3} + 2^{1} + 2^{-2} + 2^{-3}$$

$$= 16 + 8 + 2 + 0.25 + 0.125$$

$$= 26.375$$

Convert from binary to decimal

We must use both positive and negative powers of 2.

n	2^n
-1	.5
-2	.25
-3	.125
-4	.0625
-5	.03125
-6	.015625
-7	.0078125
$\vdots$	etc.

Sum up the contributing 1 bits as on previous slide.

How to convert from decimal to binary ?

$$26.375 = (\underline{\quad ? \quad} . \underline{\quad ? \quad})_2$$

To find the bits for the positive powers of 2, use the algorithm from last lecture ("repeated division") .

m	b_i
26	
13	0
6	1
3	0
1	1
0	1

$$\Rightarrow \quad 26 = (11010)_2$$

What about negative powers of 2 ?

In general, note that multiplying by 2 shifts bits to the left (or shifts binary point to the right)

Example:

$$(11010.011)_2 \quad \times 2$$

$$= (110100.11)_2$$

Similarly....dividing by 2 and not ignoring remainder shifts bits to the right (or shifts binary point to the left)

$$(11010.011)_2 / 2$$

$$= (1101.0011)_2$$

For the negative powers of 2, use "repeated multiplication"

$$.375$$

$$= .375 \times 2^{1} \times 2^{-1}$$

$$= .75 \times 2^{-1}$$

$$= 1.5 \times 2^{-2}$$

$$= 3.0 \times 2^{-3}$$

convert decimal to binary

$$= (11)_2 \times 2^{-3}$$

$$= (.011)_2$$

A more subtle example:

$$19.243 = (\ ? \ )_2$$

First, find the bits for the positive powers of 2 using "repeated division" (last lecture).

m	b_i
19	
9	1
4	1
2	0
1	0
0	1

$\therefore \ 19 = (10011)_2$

Then find the bits for the negative powers of 2 using repeated multiplication.

$$.243$$

$$= .243 \times 2^{1} \times 2^{-1}$$

$$= (0).486 \times 2^{-1}$$

$$= ?$$

Then find the bits for the negative powers of 2 using repeated multiplication.

$$.243$$

$$= (0)_2 \; .486 \times 2^{-1}$$

$$= (00)_2 \; .972 \times 2^{-2}$$

$$= (001)_2 \; .944 \times 2^{-3}$$

$$= (0011)_2 \; .888 \times 2^{-4}$$

Thus $(.243)_{10} = (.0011)_2 + \sum_{i=-5}^{-\infty} b_i 2^i$

Note the summation is over bits bi from -5, -6, ..., -infinity.

$$19.243 = (10011.0011\ldots)_2$$

We cannot get an exact representation using a finite number of bits for this example.

Can we say anything more general about what happens ?

$$.(05)_{10}$$

$$= (0)_2 \,.\, 1 \times 2^{-1}$$

$$= (00)_2 \,.\, 2 \times 2^{-2}$$

$$= (000)_2 \,.\, 4 \times 2^{-3}$$

$$= (0000)_2 \,.\, 8 \times 2^{-4}$$

$$= (00001)_2 \,.\, 6 \times 2^{-5}$$

$$= (000011)_2 \,.\, 2 \times 2^{-6}$$

$$= .00\,0011\,0011\,0011 \text{ etc.}$$

This will repeat over and over again.

When we convert a floating point decimal number with a finite number of digits into binary, we get:

- a finite number of non-zero bits to left of binary point

- an infinitely repeating sequence of bits to the right of the binary point

Why ?

[Note: sometimes the infinite number of repeating bits are all 0's, as in the case of 0.375 a few slides back.]

Recall previous example...

$$.243$$

$$= (0)_2 \; .486 \times 2^{-1}$$

$$= (00)_2 \; .972 \times 2^{-2}$$

$$= (001)_2 \; .944 \times 2^{-3}$$

$$= (0011)_2 \; 888 \times 2^{-4}$$

$$= \text{etc}$$

Eventually, the three digits to the right of the decimal point will enter a cycle that repeats forever. This will produce a bit string that repeats forever.

Hexadecimal

Writing down long strings of bits is awkward and error prone.

Hexadecimal simplifies the representation.

Hexadecimal (base 16)

0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
a	1010
b	1011
c	1100
d	1101
e	1110

Examples of hexadecimal

1) 0010 1111 1010 0011

2 f a 3

We write 0x2fa3 or 0X2FA3.

2) 101100

We write 0x2c (10 1100), not 0xb0 (1011 00)