CS 107 lecture 1

C
Assembly
C++ C and objects, object-oriented
Concurrent Programming
Scheme
Python

} both similar when compiled

CS107 lecture 2

C/C++ representation size

bool	1 byte
char	2 bytes
short	4 bytes
int	4 bytes
long	
float	4 bytes
double	8 bytes

} finite amount of memory to represent an "infinite" amount of precision

char

binary digit ⇒ bit

0/1	0/1	0/1	0/1	0/1	0/1	0/1	0/1

$2^8 = 256$

independent

power series expansion

$|A| = 65 = \underbrace{64+1}_{\text{sum of powers of 2}} = 2^6 + 2^0 \leftarrow \underbrace{01000001}_{\text{transistors in memory}}$

short

0	0	0	0	0	0	1	0	0	0	0	0	0	1	1	1

2^9 2^0

512 1 $= 519$

0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

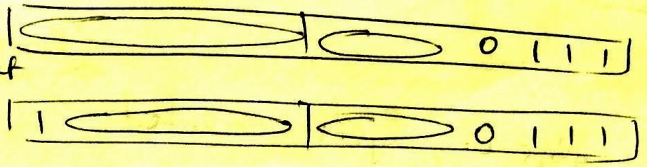
$2^{15} - 1$

negative ~~#~~ representation

7 :

one's complement

??? -7 :



} not represented on this way to enable addition and subtraction to ~~f~~follow simple rules.

ex:

```
  7 | (8)  (4)  0 111
 -7 |1(7)  (4)  0 111
-----------------------
-14 |1   (0)    1 110
```

↗ false

1 in front of the same digit pattern is not suitable

```
   7 | 0111    like decimal
+  1 | 0001      addition
-----------
   8 | 1000
```

Another idea:

2's complement

```
 15 | 0000 0000 0000 1111
-15 | 1111 1111 1111 0000  ↙1
------------------------------
  0 | 0000 0000 0000 0001  ← almost there
```

need to add then all digits become 0 and the overflow by 1 is disregarded due to 2 Byte limitation.

negative # : 1) invert the positive #

2) add 1 to it

symmetry :

```
                  15     0|000 0000 0000 1111
invert + 1    ↓
                 -15     1|111 1111 1111 0001
invert + 1    ↓
                  15     0|000 0000 0000 1111
invert + 1    ↓
                 -15     1|111 1111 1111 0001
```

this digit indicates positive or negative

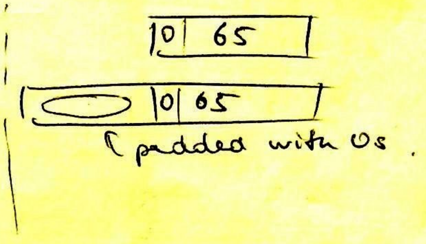
representation $(2^{15}-1)$ to $-(2^{15})$

lecture 2 CS107

ex

```
char ch = 'A';
short s = ch;
cout << s << endl;
```

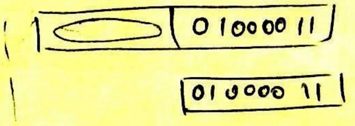
65



ex

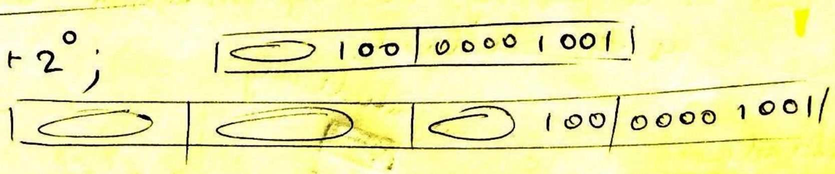
```
short s = 67;
char ch = s;
cout << ch << endl;
```

C



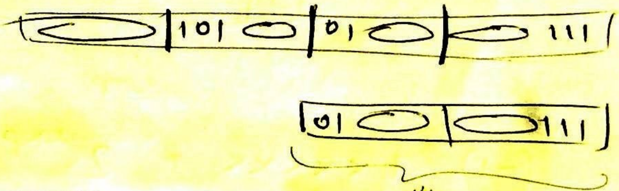
ex

```
short s = 2^10 + 2^3 + 2^0;
int i = s;
```



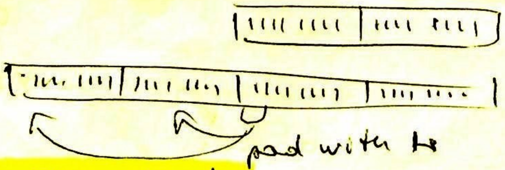
ex

```
int i = 2^23 + 2^21 + 2^14 + 7;
short s = i;
```



ex:

```
short s = -1;
int i = s;
```



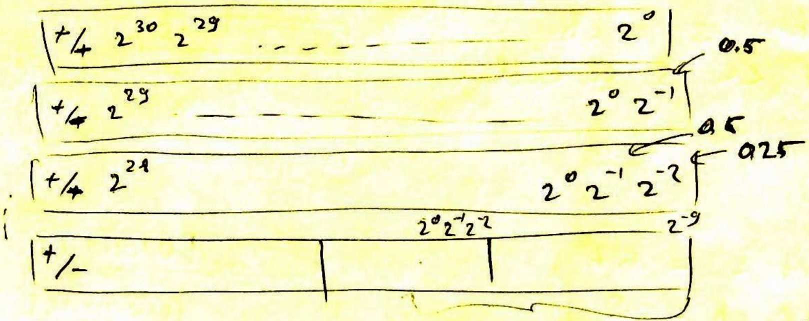
padding: with the sign digits to preserve the sign

floating pt representation

32 bits:

One idea:

reserve bits for fractional parts

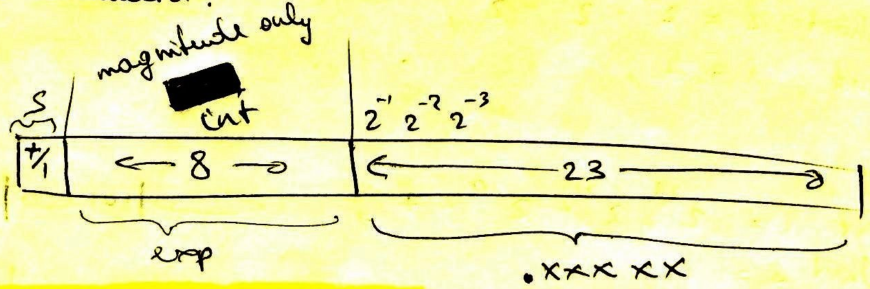


use fraction contribution to come as close as possible to the number

addition works fine!
the same way

This is a perfectly valid representation, but was not chosen as the standard.

Standard:



$$(-1)^S * 1.xxxxx * \underbrace{2^{exp-127}}_{\text{power of 2 domain}}$$

$0 \le exp \le 255$

$-127 \le exp - 127 \le 128$

ex:

7.0

7.0×2^0

3.5×2^1

$\boxed{1.75 \times 2^2}$

$S = 0$

$exp = 2$

$.xxxxx = 75$

CS 107 lecture2

From int to float, value assignment

```
int i = 5;
float f = i;
cout << f << endl;
```

5
↓
5.0
↓
$1.25 \cdot 2^2$

value

use the bit pattern of int to build a float

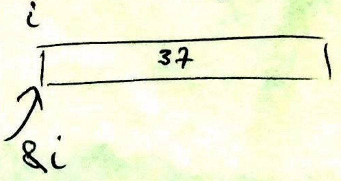
bit pattern

```
out i = 37;
float f = * (float *) &i
```

deref

pretend of type (float *)

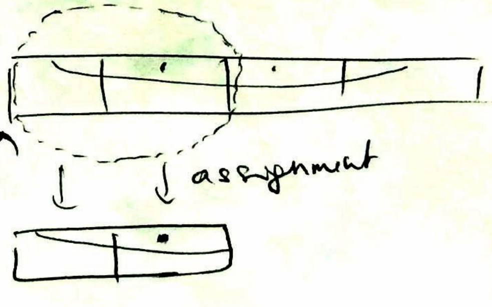
is of ty (int *)



use float bit pattern to build short, from 4 bytes to 2 Bytes

```
float f = 7.0;
short s = * (short *) &f;
```

pretend pointer does not point to 4 byte float but to a 2 byte short



s initialization