CS 107 Lecture 1

C
Assembly
C++
Concurrent Programming
Scheme
Python

(and objects, object-oriented)

↳ Both similar when compiled

CS 107 Lecture 2

C / C++

representation size

bool	1 Byte
char	2 bytes
short	4 bytes
int	4 bytes
long	4 bytes
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

$|A| = 65 = 64 + 1 = 2^6 + 2^0 \leftarrow 01000001$

sum of powers of 2 — translation 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: 0 1 1 1

one's complement

??? -7: 1 0 1 1 1

} not represented in this way to enable addition and subtraction to follow simple rules.

ex: 7: 8 4 0 1 1 1

-7: 1 7 4 0 1 1 1

-14: 1 0 1 1 1 0

↑ false

1 in front of the same digit pattern is not suitable

7: 0 1 1 1

+ 1: 0 0 0 1

8: 1 0 0 0

like decimal addition

Another idea:

2's complement

15: 0 0 0 0 0000 0000 0000 1111 ← 1

-15: 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0

0: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ← 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

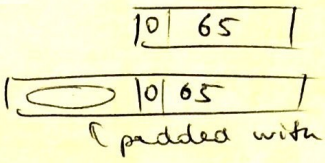
invert + 1 → 15: 0 000 0000 0000 1111

invert + 1 → -15: 1 111 1111 1111 0001

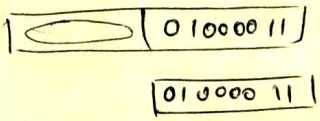
this digit indicates positive or negative

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

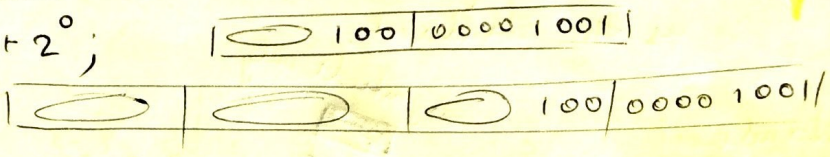
ex
```
char ch = 'A';
short s = ch;
cout << s << endl;
```
65



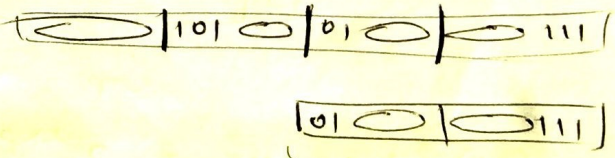
ex
```
short s = 67;
char ch = s;
cout << ch << endl;
```
C



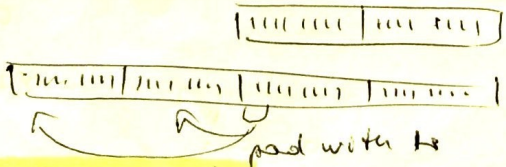
ex
```
short s = 2^10 + 2^3 + 2^0;
int i = s;
```



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



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



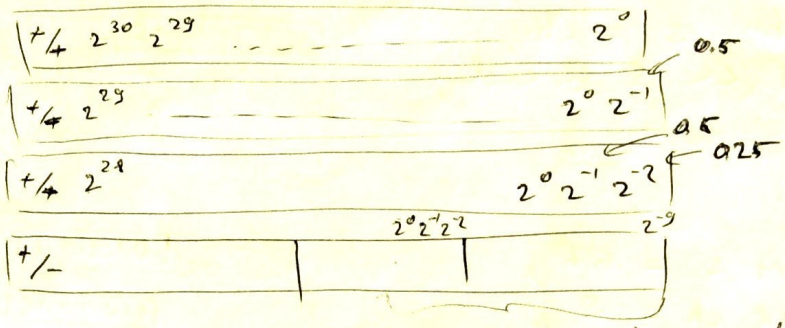
padding: with the sign digit to preserve the sign

floating # representation

32 Bits:

One idea:

reserve bits for fractional part

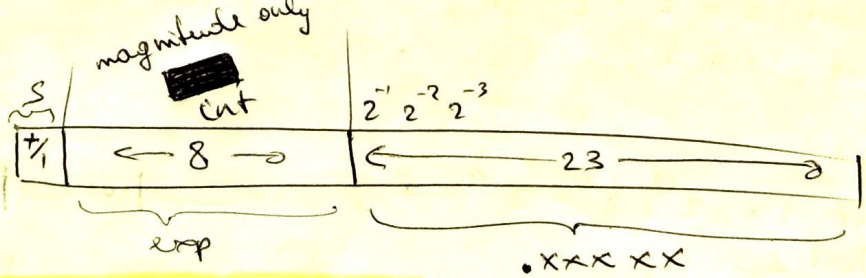


addition works fine! the same way

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

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

Standard:



$$(-1)^S * 1.xxxxx * 2^{exp-127}$$

power of 2 domain

$0 \leq exp \leq 255$

$-127 \leq exp-127 \leq 128$

ex:

7.0

7.0×2^0

3.5×2^1

1.75×2^2

$S = 0$

$exp = 2$

$.xxxxx = 75$

CS 107 Lecture 2

From int to float, value assignment

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

5 ⇓ 5.0 ⇓ $1.25 \cdot 2^2$

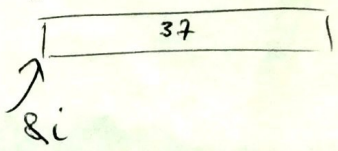
value

use the bit pattern of int to build a float

bit pattern

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

- `&i` — address of
- `(float *)` — pretend of type (float *)
- `*` — deref



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