CS61 Lecture 6 — September 19

Signed Integer Representation and Overflow

What is -1? 1 + (-1) = 0.

What is -1 if 1 is uint8_t?

1: 00000001

255: 11111111

~00000000 → 00000000

What is -1 if 1 is uint32_t?

$2^{32}-1$ → representable as a 32 bit int

→ $(2^{32}-1)+1 \mod 2^{32} \equiv 0$

TWO'S COMPLEMENT REPRESENTATION

a has the same representation as unsigned arithmatic (modulo int size)

```
int i = -1;
unsigned u = 4294967295;
memcmp(&i, &u, sizeof(int)) == 0 !
```

* advantage to this method: addition is same signed or unsigned

```
uint8_t a = 0;        int8_t ai = 0;
uint8_t b = 255;      int8_t bi = -1;
```

a < b? true ai < bi? false

ai < (uint8_t) bi ? true

unplanned results

What if you compare 2 numbers that are not same type?

#1: increases memory of smaller type to that of larger type

#2: gives warning if one unsigned and another signed, treats both as unsigned

How can you tell if bit pattern is + or -?

Highest order bit.

00000001 = positive unsigned

11111111 = negative unsigned

* values w/ most significant bit 1 are negative

Largest/smallest positive 8-bit number?

INT8.MAX = 01111111 = 127

INT8.MIN = 10000000 = -128

difference means that

- INT8.MIN = INT8.MIN

Bitwise operators

~	complement	flip all bits
&	and	
|	or	
<<	left shift	moves bits left n positions
>>	right shift	move bits right n positions

a = 00000000

~a = 11111111

~a + 1 = negative a

b = 00000010

~b = 11111101

~b + 1 = 11111110

a << i ≡ a × 2^i

a = 00000011 = 3

a << 2 = 00001100

≡ 12 = 3 × 2^2

* shifting

unsigned: adds 0s

signed: implementation defined

Overflow

when a resulting value doesn't fit in a type

- unsigned numbers: extra bit gets thrown away
- in c++ abstract machine, overflow on signed numbers is undefined

SIGNED

INT8.MAX + 1 = INT8.MIN in computer memory

↳ compiler run fine, sanitizer throws error for INT8.MAX + 1 > INT8.MAX

UNSIGNED