CS1800 day 3

Admin:
- hw1 released today (due the following friday, as nearly all HWs are)
- tutoring groups

Content:
- Two's complement (system to represent negative binary numbers)
- Overflow
- Floating point (system to represent non-whole numbers) (if time)

Whats the difference between operating in base-b and operating in base-b on a computer?

Computers store all values with the same number of bits

why? quicker / easier

Assume: a computer is using a 3-bit representation of values. How does it compute & store the following?

$$(111)_2 + (001)_2 = (1000)_2$$

$$7 + 1 = 8$$

ONLY THESE THREE BITS CAN BE STORED

THIS IS DISCARDED

For today: assume we're working with values on a computer

- all values are N-digits
	(you'll be given this info in problem statement)

- discard the most significant (left-most) digits if needed
	(as shown in green on last slide)

LARGEST PLACE VALUE

MAGNITUDE

Number Systems:

$(110)_2$

Currently we're missing:

- negative values (e.g. -43)
- non-whole values (e.g. 321.12358)

Number systems:

- Unsigned Integers:

    can represent whole, non-negative numbers
    everything we've done so far are unsigned integers (we just didn't cover name until now)
        e.g. (110)_2 = 6

- Two's Complement:

    can represent whole (potentialy negative) numbers
    (will study today)

- Floating Point Values:

    non whole-numbers
    (will study today if time)

Sign bit*:

A not-so-great number system for negative values

3 BIT "SIGN BIT"

1 IF NEGATIVE

0 OTHERWISE

SIGN BIT	2	1
0	0	0

$(001)_2 = 1$

$(000)_2 = 0$

$(111)_2 = -3$

$(101)_2 = -1$

Sign Bit: Problems

No unique zero

$$(000)_2 = 0$$

$$(100)_2 = -0$$

Our operations yield incorrect results

Discard

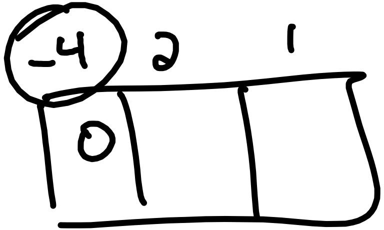
$$(111)_2 + (001)_2 = (\cancel{1}000)_2$$

$$-3 + 1 = 0$$

$$\begin{array}{r} {\scriptstyle 1\,1\;\;} \\ 111 \\ +\ 001 \\ \hline 1000 \end{array}$$

Two's complement: A better way to store negative numbers

Big idea: the most significant (biggest) place value is negative, all others are positive



Example: 3-bit two's complement

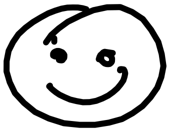
100	-4
101	-3
110	-2
111	-1
000	0
001	1
010	2
011	3

IF LEADING BIT IS ZERO 2'S COMP IS EQUIVALENT TO UNSIGNED

Two's complement, Problems Solved

Unique Zero

$$(000)_2 = 0$$

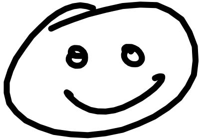


Our operations yield correct* results

Discard

$$(111)_2 + (001)_2 = (\cancel{1}000)_2$$

$$-1 + 1 = 0$$



* Assumes that correct result may be represented (more later)

In Class Activity:

If possible, convert each of the following values to the given number system. If not possible, justify why.

(Use guess-and-check as needed, a reliable decimal-to-2's-complement method coming shortly)

0	unsigned 2 bit integer
-2	unsigned 3 bit integer
0	3 bit 2's complement
-4	3 bit 2's complement
-4	4 bit 2's complement
5	4 bit 2's complement
10	4 bit 2's complement
-3	4 bit 2's complement

(++) What does the 2's complement idea look like in a base which isn't 2? Does it also have the properties we love so much in binary (unique zero, addition operations still work)?

0 unsigned 2 bit integer

$(00)_2$

2	1
0	0

-2 unsigned 3 bit integer

4	2	1

impossible! each place value is positive,
settting those bits to 1 only makes a value bigger

0 3 bit 2's complement

$(000)_2$

-4	2	1
0	0	0

-4 3 bit 2's complement

$(100)_2$

-4	2	1
1	0	0

-4 4 bit 2's complement

$(1100)_2$

-8	4	2	1
1	1	0	0

$-8+4+0+0=-4$

5 4 bit 2's complement

$(0101)_2$

-8	4	2	1
0	1	0	1

10 4 bit 2's complement

-8	4	2	1
0	1	1	1

$4+2+1=7$

BIGGEST WE CAN GET 10 IS IMPOSSIBLE

-3 4 bit 2's complement

-8	4	2	1
1	1	0	1

$-3=-8+x$ $x=5$

What values can we represent with N bits?

Unsigned Integers

2^{N-1} 8 4 2 1

$2^{N-1} \ldots 2^1 \quad 2^0$

1	1	1	1	1

SMALLEST VALUE

0

LARGEST VALUE

$2^N - 1$

Two's Complement

$-2^{N-1} \ldots 2^1 \quad 2^0$

0	1	1	1	1

SMALLEST VALUE

-2^{N-1}

LARGEST VALUE

$2^{N-1} - 1$

WHAT IS $(1)_2 + \underbrace{(11111)_2}_{2^5} = x + 1$

$= x + 1$

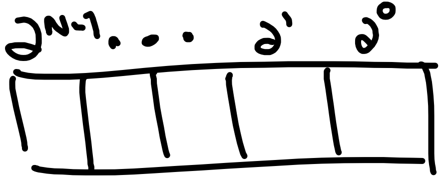
$x = 2^5 - 1$

$$
\begin{array}{rcccccc}
 & & 1 & 1 & 1 & 1 & \\
 & & 1 & 1 & 1 & 1 & 1 \\
+ & & 0 & 0 & 0 & 0 & 1 \\
\hline
 & 1 & 0 & 0 & 0 & 0 & 0 \\
 & 32 & 16 & 8 & 4 & 2 & 1
\end{array}
$$

2^5

What values can we represent with N bits? (representability)

Unsigned Integers



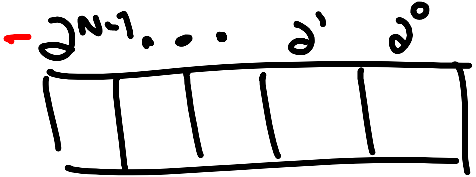
SMALLEST VALUE

0

LARGEST VALUE

$2^N - 1$

Two's Complement



SMALLEST VALUE

-2^{N-1}

LARGEST VALUE

$2^{N-1} - 1$

We can represent all whole values from smallest to largest (including smallest & largest)
(we won't justify this)

OVERFLOW

Overflow: the outcome of an operation can't be represented in the given number system

example from earlier in lesson: $(111)_2 + (001)_2 = (\cancel{1}000)_2$

7 + 1 = 8 as 3 bit values $-1 + 1 = 0$ DISCARD

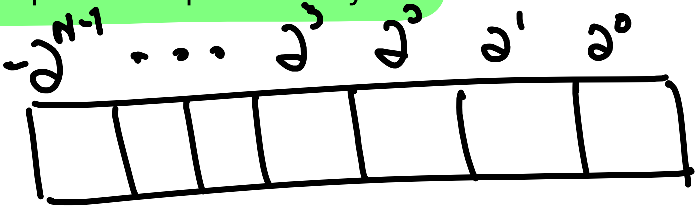
overflow since 8 can't be represented as a 3-bit value

Common misconception:

There are times when we discard a bit but result is correct (no overflow occurs)

punchline: bit discard not relevant when determining overflow

Decimal to N-bit two's complement: preliminary

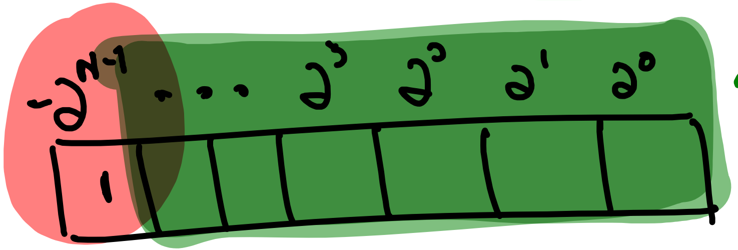


1. Validate that value can be represented as N-bit two's complement (see "representability")

2. If value is positive, its the same as N bit unsigned integer
   methods:
   - subtract largest power of two
   - Euclid's Division Algorithm

3. If value is negative: see "x" method on next slide

Decimal to N-bit two's complement: "x" method for negative representable values



DEFINE X AS VALUE OF LAST N-1 BITS (UNSIGNED)

A. Solve for X
B. Represent X as N-1 bit unsigned int
C. Append a leading 1 to indicate the -2^{N-1}

$$\text{VALUE} = -2^{N-1} + X$$

SO THAT

$$X = \text{VALUE} + 2^{N-1}$$

"X" method example

Represent -2 in 4 bit 2's complement

$(1110)_2$

-8	4	2	1
1	1	1	0

$-8 + 4 + 2 + 0 = -2$

$-8 + X = -2$

$X = 6$

SMALLEST $= -2^{N-1}$

$= -2^3 = -8$

BIGGEST $= 2^{N-1} - 1$

$= 2^3 - 1 = 7$

$6 = 4 + 2$

In Class Activity 2

If possible, express each of the following as a 6 bit two's complement value. Use the "x" method where possible.

-5
5
~~32~~

$5 = 4 + 1$

$(000101)_2$

$\text{BIGGEST} = 2^{N-1} - 1$
$= 2^5 - 1 = 31$

$\text{SMALLEST} = -2^{N-1}$
$= -2^5$
$= -32$

-5 IN 6 BIT 2'S COMP

-32	16	8	4	2	1
1	1	1	0	1	1

$(111011)_2$

$-32 + \boxed{X} = 5$

$X = 27$

$= 16 + 11$

$= 16 + 8 + 3$

$= 16 + 8 + 2 + 1$

(floating point if time)

Floating Point: Representing non-whole values

To express 12.345, rewrite it as:

$$12.345 = \underbrace{12345}_{\text{significand}} \times \underbrace{10}_{\text{base}}{}^{\overbrace{-3}^{\text{exponent}}}$$

ALSO KNOWN AS MANTISSA

big idea: the signifcand and exponent will always be whole values and we can store those!

A few notes about the "base"

- isn't the same base the number system for significand & exponent number system
    - (you can use base 10, as shown, and still store significand & exponent in binary)
- no need to store floating point base per individual value

IF TIME: NUMPY DOCUMENTATION

img credit: wikipedia

R

			172	200
			180	149
8	2			
7	14			

G

			172	200
			180	149
8	2			
7	14			

B

			172	200
			180	149
8	2			
7	14			

-8	4	2	1
1	1	1	1

$$-8+4+2+1=-1$$

$$(11111111111)_2=-1$$