Discussion 2
CS2112

September 6th/September 7th, 2022

Demo

Why Base-10 Works

$$147 = (100 * 1) + (10 * 4) + 7$$

$$= (10^2 * 1) + (10^1 * 4) + (10^0 * 7)$$

$$abc \ldots xyz = (10^n * a) + (10^{n-1} * b) \ldots$$

$$+ (10^1 * y) + 10^0 * z$$

Why Base 2 Works: Binary Integers

$$147_{10} = (128 * 1) + (16 * 1) + (2 * 1) + 1$$

$$= (2^7 * 1) + (2^4 * 1) + (2^1 * 1) + (2^0 * 1)$$

$$= 10001011_2$$

Why Base 2 Works: Binary Integers

$$147_{10} = (128 * 1) + (16 * 1) + (2 * 1) + 1$$

$$= (2^7 * 1) + (2^4 * 1) + (2^1 * 1) + (2^0 * 1)$$

$$= 10001011_2$$

Most Significant Bit (MSB)

Least Significant Bit (LSB)

Each digit = 0 or 1 (bit = binary digit)

# of bits in storage - 8 bit number

Exercise

Convert 7 and 11 to binary

Bitwise Operators

Operations on each bit of a number

Bitwise Operators: Complement

Flips each bit of the number

```
 x = 0010

~x = 1101
```

Bitwise Operators: and, or, xor

```
x = 1010, y = 1100

And: x & y = ??

Or:  x | y = ??

Xor: x ^ y = ??
```

Bitwise Operators: and, or, xor

```
x = 1010, y = 1100

And: x & y = 1000

Or:  x | y = 1110

Xor: x ^ y = 0110
```

Two's Complement

Formal Definition:

A signed n-bit representation where the most significant bit stands for -2^{n-1} instead of 2^{n-1}. This ensures that x and -x always sum to 0 with normal binary addition.

$$b_{n-1}b_{n-2}...b_0$$

$$-b_{n-1}(2^{n-1}) + \sum_{i=0}^{n-2} b_i(2^i)$$

Two's Complement Example

$11000001_2 = -128 + 64 + 1$

$= -63$

$11111111_2 = -128 + 64 + 32 + 16 + 8 + 4 + 2 + 1$

$= -1$

Two’s Complement Range

	0 .. 127	128 .. 255	Unsigned representation
-128 .. -1	0 .. 127		Signed representation

Two's Complement Conversion

Converting to two's complement representation of -*x* can be done much more easily by flipping the bits of *x*, and adding 1

$$-1_{10} = -(00000001_2) \rightarrow 11111110_2 + 1_2 = 11111111_2$$

$$11000001_2 \rightarrow -(00111110_2 + 1_2) = -(00111111_2) = -63_{10}$$

Exercise

Convert 3 and -3 to two's complement

(Assume that the numbers are represented using 4 bits)

Verify that these two numbers sum to 0 in binary.

Bitwise Operators Deja Vu

Negation: flip bits using complement, then add 1.

```
 x      = 0010 = 2
~x      = 1101 = -3
~x + 1 = 1110 = -2
```

Bitwise Operators: shift left

- Moves binary representation to left
    - Digits on left get thrown out
    - Zeros are appended on the right
    - Equivalent to multiplying by 2^n

$1011\ 0011\ 1000_2 \ll 6 = 1110\ 0000\ 0000_2$

Bitwise Operators: shift right arithmetic

- Moves binary representation to right
  - Digits on right get thrown out
  - MSB is appended on the left
  - Equivalent to dividing by 2^n

$1011\ 0011\ 1000_2 >> 6 = 1111\ 1110\ 1100_2$

Bitwise Operators: shift right logical

- Moves binary representation to right
  - Digits on right get thrown out
  - **Zero** is appended on the left
  - ~~Equivalent to dividing by 2^n~~

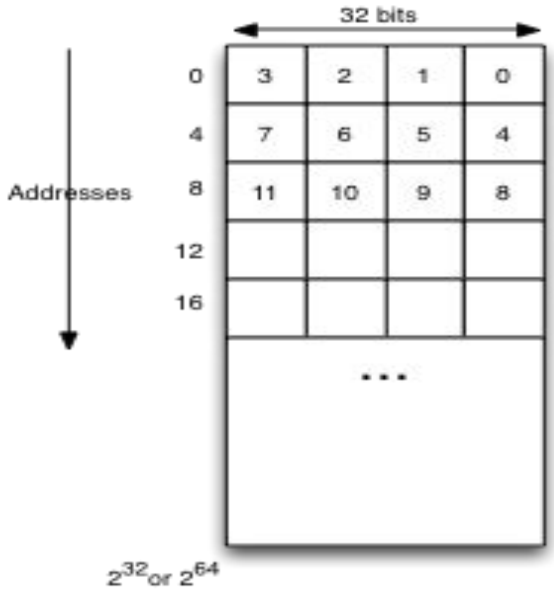
$1011\ 0011\ 1000_2 >>> 6 = 0000\ 0010\ 1100_2$

Data Type Ranges

Data Type	Minimum	Maximum
`int`	-2^{31}	$2^{31} - 1$
`long`	-2^{63}	$2^{63} - 1$
`short`	-2^{15}	$2^{15} - 1$
`byte`	-2^{7}	$2^{7} - 1$
`char`	0	$2^{16} - 1$

Memory

- Computer memory is a grid with a bit stored in every cell
- Each address names a group of 8 bits (a **byte**)
- Computer memory can read the four bytes beginning at an address. These four bytes are called a **word**



Data Type Ranges

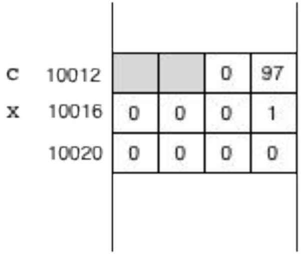
Data Type	Minimum	Maximum	Bits?	Bytes?
`int`	-2^{31}	$2^{31} - 1$	??	??
`long`	-2^{63}	$2^{63} - 1$	??	??
`short`	-2^{15}	$2^{15} - 1$	??	??
`byte`	-2^{7}	$2^{7} - 1$	??	??
`char`	0	$2^{16} - 1$	??	??

Data Type Ranges

Data Type	Minimum	Maximum	Bits?	Bytes?
`int`	-2^{31}	$2^{31} - 1$	32	4
`long`	-2^{63}	$2^{63} - 1$	64	8
`short`	-2^{15}	$2^{15} - 1$	16	2
`byte`	-2^{7}	$2^{7} - 1$	8	1
`char`	0	$2^{16} - 1$	16	2

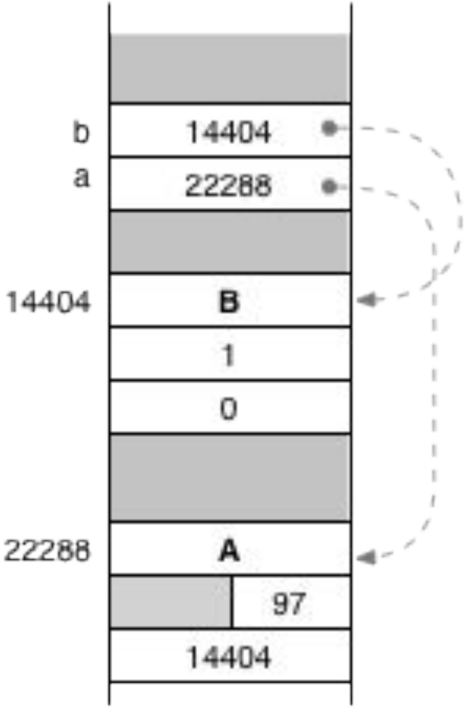
Variables

- Variables are assigned an integer number of words (even if it needs less space)
- char c = ‘a’; int i = 97
  long x = 1;



Objects

```
class A {
    char c;
    B y;
}
class B {
    long z;
}
...
B b = new B();
b.z = 1;
A a = new A();
a.c = 'a';
a.y = b;
```



Decimal Scientific Notation

- Consists of an integer *n* between 1 and 9
- A rational number *r* between 0 and 1
- A factor of 10 raised to an integer power *i*

$n.r \times 10^i$

7.234×10^{-5}

Floating Point Representation

How do we represent decimals?

- Real numbers can have infinite and non repeating representations, so they would take infinite memory to store (3.14159265358979323846264338327950288419716939937510582097494459230781…)

Floating Point Representation

How do we represent decimals?

- Real numbers can be approximated in Java using
  - Floats
  - Doubles

Floating Point Representation

How do we represent decimals?

- Real numbers can be approximated in Java using
    - Floats
        - ~7 digits of precision
        - Roughly -10^{38} to 10^{38}

Floating Point Representation

How do we represent decimals?

- Real numbers can be approximated in Java using
  - Doubles
    - ~17 digits of precision
    - Roughly -10^{308} to 10^{308}
  - Please, please use this over floats!

Floating Point Representation

Consists of the following components:

- sign (s): 0 or 1
- exponent (exp): integer
- mantissa (m): sequence of binary digits such that $1 \leq 1.m < 2$

$(-1)^s \times 2^{exp} \times (1.m)$

Example

Convert 0.5 to its 1 byte floating point representation.

Here we will use 1 bit for the sign, 3 bits for the exponent and 4 bits for the mantissa

- $(-1)^s \times 2^{exp} \times (1.m)$
- $0.5 = (-1)^0 \times 2^{-1} \times (1.0)$
- $0.5_{10} \rightarrow 01110000_2$

Floating Point Pitfalls!!

- Due to precision related errors check for closeness instead of equality
  - $f1 == f2$
  - $|f1 - f2| < \varepsilon$
- Floating point addition is not commutative
  - $1 + 10^{-40} - 1 \neq 1 - 1 + 10^{-40}$
  - Roundoff error accumulates!