АКОС: представление данных в компьютере

Целые числа

- Регистры процессора хранят числа
- Но как именно эти числа представлены?

Беззнаковые типы

- Представляют из себя N-битные положительные целые числа на отрезке $[0, 2^N - 1]$
- Переполнение точно определено стандартом C (как сложение в Z_{2^N})
- $1111 + 0001 = 10000 = 0$

Endianess

- Если N = 64, то 64 / 8 = 8 байт нужно, чтобы представить число в памяти
- Если N = 32, то 32 / 8 = 4 байта
- В какой последовательности хранить биты?

Endianess

- Little-endian: первые байты хранят младшие биты числа
- Big-ending: первые байты хранят старшие биты

Endianess

10000011010000001111111010111111

Little endian 01111111111111101010000010000011

Big endian 10000011010000001111111010111111

Выравнивание

- Числа быстрее считываются процессором, если они лежат по адресам, кратным их размеру
- Например: sizeof(int) = 4 $\Longrightarrow$ выравнивание по границе 4 байт
- char – 1 байт
- short – 2 байта
- int – 4 байта
- long long – 8 байт

Выравнивание структур

- Члены структур располагаются рядом
- Но если им не хватает выравнивания, компилятор «добивает» структуру pad'ами
- Выравнивание структуры — максимальное выравнивание среди всех выравниваний её членов

Знаковые числа

One's complement

- $-A = BitwiseNot(A)$
- Диапазон: $[-2^N + 1, 2^N - 1]$

One's complement

- $-1 = 1110$
- $+1 = 0001$
- $1110 + 0001 = 1111 = -0$

One's complement

- $-1 = 1110$
- $+2 = 0010$
- $1110 + 0010 = 10000 = 0$
- Упс...

One's complement: end-around-carry

- Бит переноса отправляется назад, чтобы всё исправить
- $1110 + 0010 = 10000 = 0 + 1 = 1$

One's complement: недостатки

- Два представления для 0: $0000 = +0$ и $1111 = -0$.
- End-around-carry
- Зато сложение и вычитание одинаковое для знаковых и беззнаковых (почти)!

Two's complement

- Определение отрицательных чисел: $A + (-A) = 0$
- Давайте каждому положительному числу сопоставим отрицательное
- $-A = BitwiseNot(A) + 1$
- Одно представление нуля: $-0 = BitwiseNot(A) + 1 = 1111 + 1 = 0000 = +0$
- Диапазон чуть больше, чем у one's complement: $[-2^N, 2^N - 1]$
- Используется в современных процессорах

Two's complement: недостатки

- Операции сравнения теперь сложные
- Умножение требует sign extension: 0010 = 00000010, 1000 = 11111000
- «Перекос» диапазона представимых чисел
- `abs(INT_MIN)` = ???

Действительные числа

Числа с фиксированной точкой

- N бит на целую часть, M бит на дробную
- Всегда одинаковая точность
- Операции легко реализуются

Числа с плавающей точкой

- IEEE 754
- Стандарт 1985 года

Числа с плавающей точкой

- Представление: $(-1)^S \times M \times 2^E$
- S – бит знака, M – мантисса, E – экспонента
- float (single): |S| = 1, |M| = 23, |E| = 8
- double: |S| = 1, |M| = 52, |E| = 11

Нормализованные значения

- $E = 0$ и $E = 2^{|E|} - 1$
- Экспонента хранится со смещением: $E_{real} = E - 2^{|E|-1}$
- Мантисса имеет «виртуальную 1»: $M_{real} = 1.mmmmmmm$

Денормализованные значения

- $E = 0$
- $E_{real} = 1 - 2^{|E|-1}$
- Это самые близкие к нулю числа и сам ноль (0.0 и +0.0)

Специальные значения

- $E = 2^{|E|} - 1$
- Если $M = 0$, то число представляет собой бесконечное значение
- Если $M = 0$, то число – NaN (not a number)
    - Используются при операциях с неопределённым значением: например, sqrt(X), log(X), X < 0

Проблемы IEEE754

- При вычислениях накапливается ошибка
- Сложение и умножение неассоциативно
- Умножение недистрибутивно
- NaN != NaN (???)
- 0.0 и +0.0

http://steve.hollasch.net/cgindex/coding/ieeefloat.html

Decimals

- Представляются в виде двух чисел: N – знаменатель, M – числитель
- Все операции реализуются через приведение к общему знаменателю
- N и M обычно используют длинную арифметику, поэтому в теории точность ограничена только оперативной памятью
- Используются в финансах

Кодировки

- Умеем оперировать числами, но как перевсти числа в текст?
- Кодировки — «карты» сопоставляющие наборы байт каким-то образом в символы

Кодировки: немного терминологии

- Character — что-то, что мы хотим представить
- Character set — какое-то множество символов
- Coded character set (CCS) — отображение символов в уникальные номера
- Code point — уникальный номер какого-то символа

ASCII

- American Standard Code for Information Interchange, 1963 год
- 7-ми битная кодировка, то есть кодирует 128 различных символов
- Control characters: с 0 по 31 включительно, непечатные символы, мета-информация для терминалов

Unicode

- Codespace: 0 до 0x10FFFF (~1.1 млн. code points)
- Code point'ы обозначаются как U+<число>
- `א` = U+2135
- `r` = U+0072
- Unicode — не кодировка: он не определяет как набор байт трактовать как characters

http://www.unicode.org/charts/

UTF-32

- Использует всегда 32 бита (4 байта) для кодировки
- Используется во внутреннем представлении строк в некоторых языках (например, Python)
- Позволяет обращаться к произвольному code point'у строки за O(1)
- BOM определяет little vs big endian

UTF-8

- Unicode Transformation Format
- Определяет способ как будут преобразовываться code point'ы
- Переменная длина: от 1 байта (ASCII) до 4 байт

UTF-8

U+0000...U+007F	→	0xxxxxxx
U+0080...U+07FF	→	110xxxxx 10xxxxxx
U+0800...U+FFFF	→	1110xxxx 10xxxxxx 10xxxxxx
U+10000...U+10FFFF	→	11110xxx 10xxxxxx 10xxxxxx 10xxxxxx

UTF-8: overlong encoding

- `00100000` = U+0020
- `11000000 10100000` = U+0020!
- overlong form или overlong encoding
- С точки зрения стандарта является некорректным представлением

Thanks!