Билет 12: Данные. Представление данных в памяти ЭВМ: целые числа (3 способа представления чисел со знаком). Двоично-десятичные числа. Арифметика с насыщением. Арифметика с фиксированной точкой.

1. Данные

Данные — поддающееся многократной интерпретации представление информации в формализованном виде, пригодном для передачи, связи, или обработки (ISO/IEC 2382:2015).

2. Представление целых чисел в памяти ЭВМ

В памяти компьютера данные представимы в виде нулей и единиц. Для представления чисел в ЭВМ обычно используется битовый набор.

Битовый набор — последовательность из нулей и единиц фиксированной длины.

Разряд — позиция в битовом наборе

Беззнаковые числа

Самым удобным, экономным и используемым способом представления беззнаковых чисел оказался следующий: битовый набор, соответствующий числу, является k-разрядной записью этого числа в двоичной системе счисления. Простыми словами, переводим число из десятеричной системы счисления в двоичную и получаем вид числа, в котором оно представимо в памяти ЭВМ.

$$9_{10} \rightarrow 1001_2 \rightarrow 00001001_2$$

Изначально мы имеем наше число в десятичной системе счисления, потом преобразуем его в двоичную запись, после идет его запись в памяти ЭВМ (минимально адресуемая ячейка памяти - 8 бит)

Числа со знаком

Положительные числа

Представить число со знаком уже становится проблематично. Числа представляются в виде набора нулей и единиц, но при этом особого символа под знак числа у нас нет. Потому требовалось ввести стандарт записи чисел со знаком.

Положительные числа во всех методах представляются одинаково. Старший бит всегда равен 0. Дальше располагается двоичная запись абсолютной величины числа. Т.е это та же беззнаковая запись, но старший бит всегда равен 0.

Поскольку теперь мы теряем 1 бит на знак, мы можем записать максимум 2^{N-1} значений, а максимально число будет равно 2^{N-1}-1, где N - размер регистра.

Прямой код

Первое решение проблемы записи целых чисел со знаком был представлен прямой код - способ записи, в которой старший бить отводится под знак числа. Равен 0 при положительном числе и 1, если число отрицательное. Дальше идет запись числа в двоичной системе счисления.

Проблемы прямого кода.

- Всего диапазон целых чисел составляет 2^{N-1} значений
- Появляется проблема "-0" (10000000)
- Теперь нужен некоторый вычитатель, который позволит отнимать числа. Если представить *x - y* как *x + (-y)*, то у нас ничего не выйдет. Так как в результате выйдет бред.

Обратный код

Спустя время появляется новый способ представления чисел, который решает часть проблем. Получается он следующим образом: берется прямой код числа и инвертируются все биты абсолютной части, оставляя бит знака неизменным.

Обратный код помог решить проблему вычитания чисел. Теперь можно заменить операцию вычитания на операцию сложения, получив при этом нормальный результат. Важно понять, что компьютер при *сложении* и *вычитании* по прежнему продолжает воспринимать числа записанные в обратном коде, как целые числа. Так например операция *2 - 3* представляется как 2[00000010] + (-3)[11111100], где (-3) распознается как 252[11111100]. В результате получится число -1[11111110]

Оставшиеся проблемы обратного кода

- Отрицательный ноль все еще остается
- Существуют случаи, когда арифметика ошибочна и отличается от правильного ответа на единицу. Например -3-2=-6 / 2-1=0

Дополнительный код

И наконец появляется следом новый способ представления чисел - дополнительный код. Получается он при помощи добавления единицы к младшему разряда числа в обратном коде.

Решенные проблемы

- Отрицательный ноль уходит. Поскольку 111111111 + 1 = 1|00000000 превращается в обычный 0
- Отрицательные числа теперь записываются в порядке убывания. При побитовом сравнении число -1 больше -2, число -2 больше -3 и т.д
- Появляется дополнительное место для 1 отрицательно числа, поскольку мы избавились от -0
- Теперь у нас решается проблема в арифметике с недостающей единицей. 2-1=1

Теперь разберемся подробнее как работает дополнительный код.

Пусть B - основание системы счисления, N - число разрядов. $P = B^N$. Дальше вычисления делаем по модулю P:

Берем число $x \in [0, P)$. Тогда $[x]_p = [x + nP]_p = \{x + Pk \mid k \in Z\}$ - класс эквивалентности.

Дальше берется число $s \in [0, P - 1)$ и принимается $[-s] = [P - s]_p$

Разберем на примере x-y = x + [-y].

[-y] = [P - y]

Теперь мы прибавим и отнимем единичку = [(P - 1 - y) + 1]

Благодаря этому мы получаем идущие подряд единицы во всех разрядах из записи P - 1. От них мы отнимем наше число *y* , т.е инвертируем его биты. После этого добавим единицу и получим дополнительный код.

Переполнение

Как следствие, в дополнительном коде, при сложении больших положительных чисел или совсем маленьких отрицательных чисел, могут возникнуть ситуации, которые могут сначала удивить. Например, для нашего случая, когда k=4: 7[0111] + 3[0011] мы получим число -6[1010]. Эта ситуация, когда при сложении чисел с одним знаком, мы получаем число с другим знаком, называется переполнением. Мы просто вышли на грани возможных чисел, которые могут поместиться в наш битовый набор.

3. Двоично-десятичные числа

Двоично-десятичный код (binary-coded decimal, BCD, 8421-BCD) — форма записи рациональных чисел, когда каждый десятичный разряд числа записывается в виде его четырёхбитного двоичного кода.

> ## Пример:
>
> Число 311_{10} будет записано в двоичной системе счисления в двоичном коде как $1\ 0011\ 0111_2$, а в двоично-десятичном коде как $0011\ 0001\ 0001_{BCD}$
> Поскольку $3_{10} = 0011_2$, $1_{10} = 0001_2$.
> При помощи четырех бит можно закодировать шестнадцать цифр. Из них используются 10. Остальные 6 комбинаций в двоично-десятичном коде являются запрещенными.

Преимущества	Недостатки
Не теряется точность для дробных чисел (как с фиксированной, так и с плавающей запятой) при переводе в десятичную систему счисления и наоборот.	Требует больше памяти
Упрощено умножение/деление на 10, а также округление	Усложнены арифметические операции. Поскольку в 8421-BCD используются только 10 возможных комбинаций 4-битового поля вместо 16, существуют запрещенные комбинации битов: 1010 (10) 1011 (11) 1100 (12) 1101 (13) 1110 (14) 1111 (15)
Просты для понимания. Является записью десятичных чисел в двоичной системе	

4. Арифметика с насыщением

Циклическая арифметика - если результат операции выходит за пределы допустимого диапазона, то "лишние" старшие биты результата отбрасываются.

Арифметика с насыщением - это разновидность арифметики, в которой все операции ограничены фиксированным диапазоном между минимальным и максимальным значениями. Если результат оказался вне допустимого диапазона, то он считается равным граничному значению диапазона.
«Насыщенные» арифметические операции доступны на многих современных платформах, и в частности были одним из расширений, сделанных на технологии Intel MMX специально для приложений обработки сигнала.

> ## Пример:
>
> Пускай диапазон значений будет от -100 до +100, тогда арифметические операции с насыщением дают следующие значения:
>
> $$60 + 30 \rightarrow 90$$
> $$60 + 43 \rightarrow 100$$
> $$(60 + 43) - (75 + 75) \rightarrow 0$$
> $$10 \cdot 11 \rightarrow 100$$
> $$99 \cdot 99 \rightarrow 100$$
> $$30 \cdot (5 - 1) \rightarrow 100$$
> $$(30 \cdot 5) - 30 \rightarrow 70$$
> $$100 - 30 \rightarrow 70$$
> $$5 - 200 \rightarrow -100$$

Арифметика с насыщением позволяет создавать эффективные алгоритмы для многих задач, особенно в цифровой обработке сигналов. Например, регулировка уровня громкости звукового сигнала могут привести к переполнению, и использование арифметики с насыщением вызывает значительно меньшие искажения в звуке. Кроме того, такой тип арифметики используется в формате IEEE-754 (стандарт чисел с плавающей точкой), в котором переполненные значения превращаются в "бесконечность" или "минус бесконечность", и любые другие операции с этими результатами продолжают возвращать то же самое значение.

5. Арифметика с фиксированной точкой.

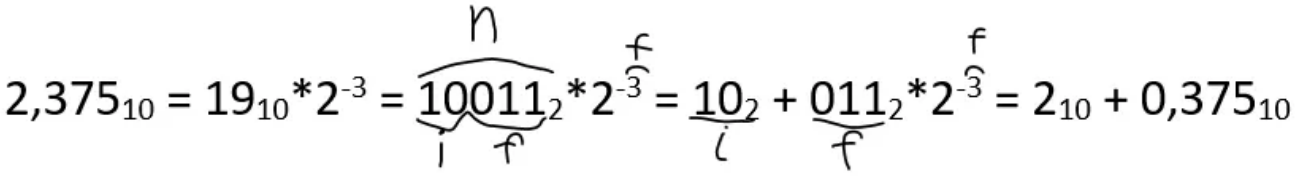
Число с фиксированной точкой/запятой - формат представления вещественного числа в памяти ЭВМ в виде целого числа.
Т.е это метод представления нецелых чисел путем хранения фиксированного количества цифр их дробной части. При этом само число x и его целочисленное представление x'(мантисса) связаны формулой $x = x` \bullet z$
z - цена младшего разряда.

В представлении с фиксированной точкой дробь часто выражается в той же системе счисления, что и целая часть, но с использованием отрицательных степеней b. Наиболее распространенными вариантами является десятичная и двоичная система счисления. Таким образом, если хранятся n дробных разрядов, значение всегда будет кратным b^{-n}. Представление с фиксированной запятой также можно использовать для пропуска младших разрядов целочисленных значений, например, при представлении больших значений в рублях как кратных 1000 рублей.

Если не требуется, чтобы какие-либо конкретные дробные числа входили в разрядную сетку, программисты обычно выбирают $z=2^{-f}$ (-f - экспонента) — это позволяет использовать в операциях умножения и деления битовые сдвиги. Про такую арифметику говорят: «f битов на дробную часть, i = n − f — на целую» и обозначают как «i,f», «i.f» или «Qi.f».

> ## Пример:
>
> Арифметика 8,24 отводит 8 бит на целую часть и 24 бит на дробную. Она способна хранить числа от -128 до 128 - z ($127 + 0.11111.....111_2$)
>
> С ценой младшего разряда $z = 2^{-24} = 5{,}96 \bullet 10^{-8}$

Картинка для понимания



экспонента f = 3	$z = 2^{-3}$	Q2,3
мантисса x` = 19	n = 5	i = 2

То есть представление дробного числа с фиксированной точкой — это, по сути, целое число, которое должно быть неявно умножено на фиксированный коэффициент масштабирования (в английской литературе scaling factor, далее использован термин коэффициент масштабирования, чем является z). Например, значение 1,23 можно сохранить в переменной как целочисленное значение 1230 с неявным коэффициентом масштабирования 1/1000 (это означает, что последние 3 десятичных разряда неявно считаются десятичной дробью).

Это представление позволяет стандартным целочисленным арифметическим устройствам выполнять вычисления с рациональными числами.

Отрицательные значения обычно представляются в двоичном формате с фиксированной точкой как целое число со знаком в дополнительном коде с неявным коэффициентом масштабирования, как указано выше.

Некоторая проблема Округления

Даже при самом тщательном округлении значения с фиксированной точкой, представленные с коэффициентом масштабирования S, могут иметь ошибку до ±0,5 в сохраненном целом числе, то есть ±0,5S в значении. Поэтому меньшие коэффициенты масштабирования обычно дают более точные результаты.

С другой стороны, меньший коэффициент масштабирования означает меньший диапазон значений, которые могут быть сохранены в данной программной переменной. Максимальное значение с фиксированной точкой, которое может быть сохранено в переменной, равно наибольшему целочисленному значению, которое может

быть сохранено в ней, умноженному на коэффициент масштабирования; и аналогично для минимального значения.

Сложение и вычитание

Даже при самом тщательном округлении значения с фиксированной точкой, представленные с коэффициентом масштабирования S, могут иметь ошибку до ±0,5 в сохраненном целом числе, то есть ±0,5S в значении. Поэтому меньшие коэффициенты масштабирования обычно дают более точные результаты.

С другой стороны, меньший коэффициент масштабирования означает меньший диапазон значений, которые могут быть сохранены в данной программной переменной. Максимальное значение с фиксированной точкой, которое может быть сохранено в переменной, равно наибольшему целочисленному значению, которое может быть сохранено в ней, умноженному на коэффициент масштабирования; и аналогично для минимального значения.

Пример сложения двух чисел:

```c
C
int32_t a = 0x1000L; // q15: a = 0.125
int32_t b = 0x20000L;  // q20: b = 0.125
int32_t z = 0;   // q25
z = (a << 5) + b;  // q20: (a * 2 ^ (20 - 15) + b);  z = 0x40000L (0.25 в q20)
z <<= 5;  // q25: c = 0x800000L (0.25 в q25)
```

Умножение

Чтобы умножить два числа с фиксированной точкой, достаточно умножить два основных целых числа и предположить, что коэффициент масштабирования результата является произведением их коэффициентов масштабирования. Результат будет точным, без округления, при условии, что он не переполняет принимающую переменную.

Например, умножение чисел 123 в масштабе 1/1000 (0,123) и 25 в масштабе 1/10 (2,5) дает целое число 123•25 = 3075 в масштабе (1/1000) • (1/10) = 1/10000, то есть 3075/10000 = 0,3075. В качестве другого примера, умножение первого числа на 155, неявно масштабированное на 1/32 (155/32 = 4,84375), дает целое число 123 • 155 = 19065 с неявным коэффициентом масштабирования (1/1000) • (1/32) = 1/32000, то есть 19065/32000 = 0,59578125.

> ## Пример умножения двух чисел с последующим сложением с 3 числом:
>
> ```c
> int32_t a = 0x8000L; // q16: a = 0.5
> int32_t b  = 0x100000L; // q21: b = 0.5
> int32_t c = 0xC0000L; // q20: c = 0.75
> int64_t d ; // Временная переменная с увеличенным числом разрядов, чтобы хватило на результат.
> d  = (int64_t)a * (int64_t)b; // q37 = q16 * q21;  d = 0x800000000L (0.25 in q37)
> d >>= 17; // q37 / 2 ^ 17 = q20
> c += (int32_t)d; // q20: c = 0x100000 (1 in q20)
> ```

Деление

Чтобы разделить два числа с фиксированной точкой, нужно взять целое частное их основных целых чисел и предположить, что коэффициент масштабирования является частным их коэффициентов масштабирования. Как правило, первое деление требует округления, поэтому результат не является точным.

Например, деление 3456 в масштабе 1/100 (34,56) и 1234 в масштабе 1/1000 (1,234) дает целое число 3456÷1234 = 3 (округленное) с коэффициентом масштабирования (1/100)/(1/1000) = 10, то есть 30. В качестве другого примера, деление первого числа на 155 в масштабе 1/32 (155/32 = 4,84375) дает целое число 3456÷155 = 22 (округленное) с неявным коэффициентом масштабирования (1/ 100)/(1/32) = 32/100 = 8/25, то есть 22×32/100 = 7,04.

Если результат не является точным, ошибку, вызванную округлением, можно уменьшить или даже устранить, преобразовав делимое в меньший коэффициент масштабирования. Например, если r = 1,23 представлено как 123 в масштабе 1/100, а s = 6,25 представлено как 6250 в масштабе 1/1000, то простое деление целых чисел дает 123÷6250 = 0 (округлено) с коэффициентов масштабирования (1/100)/(1/1000) = 10. Если r сначала преобразовать в 1 230 000 с коэффициентом масштабирования 1/1000000, результатом будет 1 230 000÷6250 = 197 (округлено) с коэффициентом масштабирования 1/1000 (0,197). Точное значение 1,23/6,25 равно 0,1968.

> ## Пример деления двух чисел без потерь:
>
> ```c
> int32_t a = 0x4000L; // q15: a = 0.5
> int32_t b = 0x80000L;  // q20: b = 0.5
> int32_t c = 0; // q25
> int64_t d; // Временная переменная с увеличенным числом разрядов.
> d = (int64_t)a << 30; // q45: d = 0x200000000000; (0.5 in q45)
> c = (int32_t)(d / (int64_t)b); // q25: c = 0x2000000; (1 in q25)
> ```

Преобразование коэффициента масштабирования

- Сохранение значения в переменной, которая имеет другой неявный коэффициент масштабирования
- Преобразование коэффициентов масштабирования двух значения для сложения или вычитания
- Восстановление исходного коэффициента масштабирования значения после умножения или деления его на другое
- Повышение точности результата деления
- Гарантия, что коэффициент масштабирования произведения или частного представляет собой простую степень, такую как 10^n или 2^n
- Сохранение результата операции в переменную без переполнения

Чтобы преобразовать число с коэффициентом масштабирования *R* в значение с коэффициентом масштабирования *S*, основное целое число должно быть умножено на отношение *R/S* . Таким образом, например, чтобы преобразовать значение 1,23 = 123/100$с коэффициентом масштабирования *R* = 1/100 в значение с коэффициентом масштабирования *S* = 1/1000, целое число 123 должно быть умножено на (1/100)/(1/1000) = 10, что дает представление 1230/1000.

Если *S* не делит *R* (в частности, если новый коэффициент масштабирования *S* больше исходного *R*), новое целое число, возможно, придется округлить.

Например, если общий коэффициент масштабирования равен 1/100, умножение 1,23 на 0,25 влечет за собой умножение 123 на 25, чтобы получить 3075 с промежуточным коэффициентом масштабирования 1/10000. Чтобы вернуться к исходному коэффициенту масштабирования 1/100, целое число 3075 затем необходимо умножить на 1/100, чтобы получить либо 31 (0,31), либо 30 (0,30), в зависимости от политики округления.

Преобразование с числами с плавающей запятой

Чтобы преобразовать число из числа с плавающей запятой в число с фиксированной запятой, можно умножить его на коэффициент масштабирования *S*, а затем округлить результат до ближайшего целого числа. Необходимо позаботиться о том, чтобы результат помещался в целевую переменную или регистр. В зависимости от коэффициента масштабирования и размера хранилища, а также от диапазона входных чисел преобразование может не повлечь за собой никакого округления.

Чтобы преобразовать число с фиксированной запятой в число с плавающей запятой, можно преобразовать целое число в число с плавающей запятой, а затем разделить его на коэффициент масштабирования *S*. Это преобразование может повлечь за собой округление, если абсолютное значение целого числа больше 2^{24} (для двоичного числа одинарной точности с плавающей запятой IEEE) или 2^{53} (для двойной точности). Переполнение или недополнение может произойти, если основное целое число соответственно очень большое или очень маленькое.