МИНОБРНАУКИ РОССИИ

САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ

ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

“ЛЭТИ” ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)

Кафедра ВТ

ОТЧЕТ

по лабораторной работе №1

по дисциплине “ОЭВМиС”

Тема: Исследование внутреннего представления различных форматов данных

Студент гр. 4363 Карачев П.А.

Преподаватель Павлов С.М.

Санкт-Петербург

2016

**Общие положения**

При программировании на языке С++ используются 11 стандартных типов данных. Среди них можно выделить 3 группы:

1. данные символьные и целого типа беззнаковые (с фиксированной запятой);
2. данные символьные и целого типа со знаком (с фиксированной запятой), значения которых хранятся в двоичном дополнительном коде;
3. данные вещественного типа (с плавающей запятой (точкой)).

Числовое значение данных первой группы занимает всю разрядную сетку (количество двоичных разрядов от 8 до 32), отведенных под конкретный тип. Знак числа данных второй группы занимает старший (левый) разряд, а остальную часть разрядной сетки (от 7 до 31 двоичных разрядов) занимает числовое значение данных, отведенных под конкретный тип. Формат хранения данных третьей группы описывается IEEE - стандартом в виде значения мантиссы (M) со знаком (S) и значения порядка (P). Число бит для хранения мантиссы и порядка зависит от типа данных с плавающей запятой.

float

|  |  |  |
| --- | --- | --- |
| S | P | M |

31 30 23 22 0

double

|  |  |  |
| --- | --- | --- |
| S | P | M |

63 62 52 51 0

long double

|  |  |  |
| --- | --- | --- |
| S | P | M |

79 78 64 63 0

Вещественное число в памяти хранится с нормализованной мантиссой, значение которой в десятичном эквиваленте лежит в диапазоне от 1 до 2. Причём 2 не входит в границу диапазона. Если в процессе выполнения какой-либо операции над данными с плавающей запятой значение мантиссы выходит из указанного диапазона, то в конце операции выполняется нормализация результата путем приведения значения мантиссы к указанному диапазону с соответствующим изменением значения порядка. При этом значение старшего бита мантиссы должно оказаться равным единицы. Если значение порядка превышает допустимое значение, то вырабатывается признак переполнения разрядной сетки. Если значение мантиссы равно нулю или в процессе выполнения операции значение порядка становится меньше допустимой величины, то в результате выполнения операции сформируется так называемый «машинный ноль», то есть код, у которого значение всех бит равно нулю. Но если мантисса всегда нормализована, то старший её бит, то есть единицу, можно и не хранить в памяти. Стандартом предложено это бит не хранить в памяти и тем самым увеличить точность представления вещественных чисел в 2 раза. Эта единица присутствует неявно, то есть скрыта от глаз наблюдателя и называется неявной единицей (implicit one). Отбрасывание старшей цифры мантиссы выполняется для форматов float и double, но не выполняется для long double.

Порядок числа в соответствии с указанным форматом хранится «сдвинутым», то есть к его действительному значению добавляется в зависимости от формата такое число, чтобы порядок Р был всегда неотрицательным. Для формата float прибавляется 127, для чисел формата double прибавляется 1023, а для формата long double добавляется 16383. Всегда неотрицательный порядок упрощает выполнение операции сравнения порядков и арифметических операций над ними, а также избавляет от необходимости выделять один бит для хранения знака порядка.

Например, число 15.375 (1111.011 в двоичной системе счисления) в формате float IEEE-стандарта получается следующим образом:

1.921875 (1.111011 в двоичной системе счисления) - это значение нормализованной мантиссы;

3 (11- в двоичной системе счисления) это степень несмещённого двоичного порядка (15.375=1.921875\*8);

0-это знак числа.

Учитывая отбрасывание неявной единицы и сдвиг порядка, получаем внутреннее представление числа:

S=0;

P=3+127=130 (10000010 в двоичной системе счисления);

M=11101100000000000000000.

Таким образом, внутреннее представление числа 15.375 в формате float будет:

|  |  |  |
| --- | --- | --- |
| 0 | 10000010 | 11101100000000000000000 |

31 30 … 23 22 … 0

Это же число 15.375 в формате double записывается так:

S=0;

P=3+1023=1026 (10000000010 в двоичной системе счисления);

M=1110110000000000000000000000000000000000000000000000.

|  |  |  |
| --- | --- | --- |
| 0 | 10000000010 | 11101100000000000...00000000000000000000000 |

63 62 … 52 51 … 0

Это же число в формате long double записывается следующим образом:

S=0;

P=3+16383=16386 (100000000000010 в двоичной системе счисления);

M=11110110000000000000000…0 (единица не отбрасывается).

|  |  |  |
| --- | --- | --- |
| 0 | 100000000000010 | 111101100000000000…00000000000000000000000000000000 |

79 78 … 64 63 … 0

Приведём ещё несколько примеров, но уже без подробных комментариев.

Для более компактной формы записи используем шестнадцатеричную систему счисления:

-16.5 float: C1 84 00 00 h

double: C0 30 80 00 00 00 00 00 h

long double: C0 03 84 00 00 00 00 00 00 00 h

- 0.0625 float: BD 80 00 00 h

double: BF B0 00 00 00 00 00 00 h

long double: BF FE A0 00 00 00 00 00 00 00 h

**Вариант 3**

**Цель работы:** Установить в заданное пользователем состояние определённое количество рядом стоящих бит, номер старшего бита, как и всё остальное, вводится с клавиатуры.

Типы данных для работы: *unsigned* *long integer, double*.

|  |  |  |
| --- | --- | --- |
| **тип** | **байт** | **диапазон принимаемых значений** |
| *Unsigned long int* | 4 | 0 / 4 294 967 295 |
| *double* | 8 | -9 223 372 036 854 775 808 .0 / 9 223 372 036 854 775 807.0 |

**Алгоритм выполняемого задания.**

Возьмем работу алгоритма для unsigned integer, для *double* будет такой же, но с размером в 2 раза большим.

**Входное число** типа ***unsigned long integer*  = 4 000 000 000**:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Входное число | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | | | | *user\_range = 11* | | | | | | | | | | | | *user\_lsb=15* | | | | | | | | | | | | | | | |
| *size\_ui = 4 \* 8 = 32 Бита* | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Пользователь ввел *bits\_state = 1, user\_lsb = 15, user\_range=11*

Шаг 1.

Создаем число типа *unsigned long int* хранящее битовую маску, заполненную 1:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Маска. Шаг 1. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|  | | | | *user\_range = 11* | | | | | | | | | | | | *user\_lsb =15* | | | | | | | | | | | | | | | |
| *size\_ui = 4 \* 8 = 32 Бита* | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Шаг 2.

Заполним 0 левую границу. Для этого сдвигаем маску влево на:

mask <<= (unsigned\_size\_of\_bits - user\_lsb - 1) = 32 – 15 -1 =16

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Маска. Шаг 2(а) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
|  | | | | *user\_range = 11* | | | | | | | | | | | | *user\_lsb =15* | | | | | | | | | | | | | | | |
| *size\_ui = 4 \* 8 = 32 Бита* | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

А затем сдвигаем обратно вправо на:

mask >>= (unsigned\_size\_of\_bits - user\_lsb - 1) = 32 – 15 -1 =16

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Маска. Шаг 2(б) | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|  | | | | *user\_range = 11* | | | | | | | | | | | | *user\_lsb =15* | | | | | | | | | | | | | | | |
| *size\_ui = 4 \* 8 = 32 Бита* | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Шаг 3.

Заполним 0 правую границу. Для этого сдвигаем маску вправо на:

mask >>= (user\_lsb + 1 - user\_range) = 15 + 1 – 11 = 5

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Маска. Шаг 3(а). | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
|  | | | | *user\_range = 11* | | | | | | | | | | | | *user\_lsb =15* | | | | | | | | | | | | | | | |

А затем сдвигаем влево обратно на:

mask <<= (user\_lsb + 1 - user\_range) = 15 + 1 – 11 = 5

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Маска. Шаг 3(б). | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
|  | | | | *user\_range = 11* | | | | | | | | | | | | *user\_lsb =15* | | | | | | | | | | | | | | | |

Шаг 4.

Для установки всех битов группы в состояние *bits\_state = 1* необходимо полученную маску наложить на исходное число путем поразрядного логического "ИЛИ":

\*ui\_in\_number = (bits\_state) ? (\*ui\_in\_number) | mask :

(\*ui\_in\_number) & (~mask);

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Входное число **= 4 000 000 000** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| Маска. | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
| Выходное число **= 4 000 055 264** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 |
|  | | | | *user\_range = 11* | | | | | | | | | | | | *user\_lsb =15* | | | | | | | | | | | | | | | |

**Сокращенный пример кода для unsigned long int:**

int bits\_state = 0;

unsigned short user\_lsb = 0;

unsigned short user\_range = 0;

unsigned short unsigned\_size\_of\_bits = sizeof(unsigned) \* CHAR\_BIT;

printf("\nEnter in the format: \n"

"the state [0.1], the highest bit number [0-31] , close near the number of bits [0-32]\n");

scanf("%d %hu %hu", &bits\_state, &user\_lsb, &user\_range);

while (user\_range > 32 || user\_lsb > 31 || user\_lsb < 0 || user\_range < 0 || user\_range > user\_lsb + 1){

printf("Invalid input, enter the correct values ??coinciding with unsigned sizes (4 bytes):\n");

scanf("%d %hu %hu", &bits\_state, &user\_lsb, &user\_range);

}

if(!(user\_lsb || user\_range)) return;

unsigned mask = ~0;

mask <<= (unsigned\_size\_of\_bits - user\_lsb - 1);

mask >>= (unsigned\_size\_of\_bits - user\_lsb - 1);

mask >>= (user\_lsb + 1 - user\_range);

mask <<= (user\_lsb + 1 - user\_range);

\*ui\_in\_number = (bits\_state) ? (\*ui\_in\_number) | mask :

(\*ui\_in\_number) & (~mask);

**Сокращенный пример кода для double:**

int bits\_state = 0;

unsigned short user\_lsb = 0;

unsigned short user\_range = 0;

unsigned short double\_size\_of\_bits = sizeof(long) \* CHAR\_BIT;

printf("\nEnter in the format: \n"

"the state [0.1], the highest bit number [0-63], close near the number of bits [0-64]\n");

scanf("%d %hu %hu", &bits\_state, &user\_lsb, &user\_range);

while (user\_range > 64 || user\_lsb > 63 || user\_lsb < 0 || user\_range < 0 || user\_range > user\_lsb + 1){

printf("Invalid input, enter the correct values ??coinciding with unsigned sizes (4 bytes):\n");

scanf("%d %hu %hu", &bits\_state, &user\_lsb, &user\_range);

}

if(!(user\_lsb || user\_range)) return;

unsigned long mask = ~0;

mask <<= (double\_size\_of\_bits - user\_lsb - 1);

mask >>= (double\_size\_of\_bits - user\_lsb - 1);

mask >>= (user\_lsb + 1 - user\_range);

mask <<= (user\_lsb + 1 - user\_range);

\*d\_in\_number = (bits\_state) ? (\*d\_in\_number) | mask :

(\*d\_in\_number) & (~mask);

**Пример работы программы:**

***Для unsigned long int:***

Введите число: 4 000 000 000

00000000 00000000 00000111 10111100

Введите состояние: 1

Введите старший бит: 15

Введите количество рядом стоящих бит 11

Новое число: 4 000 055 264

11101110 01101011 11111111 11100000

***Для double:***

Введите число: 48.1

01000000 01001000 00001100 11001100 11001100 11001100 11001100 11001101

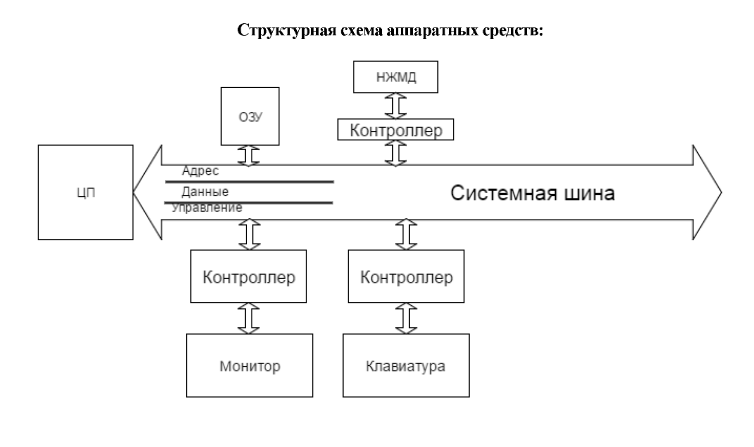
Введите состояние: 1

Введите старший бит: 18

Введите количество рядом стоящих бит 11

Новое число: 48.10000000148975

01000000 01001000 00001100 11001100 11001100 11001111 11111111 11001101



АЛУ-арифметико-логическое устройство

УУ-устройство управления

ОЗУ-оперативное запоминающее устройство

ПЗУ-постоянное запоминающее устройство

**Вывод**

Было исследовал внутреннее представление данных unsigned long integer и double на машине c ЦП Intel® Core™ i3-6100U CPU @ 2.30GHz построенного на архитектуре x64\_32 в OC Windows 10.