Отчет по 3 лабораторной работе.

Команда: Холодов А.В. Иванов А.А. ПИН-21

Задание Л3.з1. Измените функции $print16()$ и $print32()$ заданий Л1.з3–Л1.з4 так, чтобы все данные, выводимые одним вызовом $print16()$ либо $print32()$, занимали одну строку; а ширина каждого из представлений была бы постоянной: для шестнадцатеричного (а) и двоичного (б) представлений необходимо выводить ведущие нули (но не более, чем фактически присутствует: так, двоичное представление 16-битного числа должно содержать 16 бит); для десятичных (в), (е), (ж), (з) дополнять пробелами. Каждое из дублирующихся представлений — шестнадцатеричное (а) и (г), двоичное (б) и (д) — выводить в одном экземпляре.

Реализация:

```
void print16(void* p)
{
    std::bitset<16> signed_bitset(*reinterpret_cast<short *>(p));
    //std::bitset<16> unsigned_bitset(*reinterpret_cast<unsigned short *>(p));
    cout<<setfill(' ');
    cout<<setw(16) << "binary" << "|"<<
          setw(4) << "hex" << "|"<<
          setw(11) << "un decimal" << "|"<<
          setw(11) << "decimal" << endl;
    cout<<setfill(' ');
    cout<<setw(16) << signed_bitset << "|"<<
          setw(4) << hex<<(*(reinterpret_cast<unsigned short *>(p)))<< "|"<<
          setw(11) << dec<<*(reinterpret_cast<unsigned short *>(p))<< "|"<<
          setw(11) << dec<<*(reinterpret_cast<short *>(p))<< endl;

}

void print32(void* p)
{
    std::bitset<32> signed_bitset(*reinterpret_cast<int *>(p));
    //std::bitset<32> unsigned_bitset(*reinterpret_cast<unsigned int *>(p));
    cout<<setfill(' ');
    cout<<setw(32) << "binary" << "|"<<
          setw(8) << "hex" << "|"<<
          setw(11) << "un decimal" << "|"<<
          setw(11) << "decimal" << "|"<<
          setw(9) << "float" << "|"<<
          setw(9) << "exponent" << endl;
    cout<<setfill(' ');
    cout<<setw(32) << signed_bitset << "|"<<
          setw(8) << hex << (*(reinterpret_cast<unsigned int *>(p)))<< "|"<<
          setw(11) << dec << *(reinterpret_cast<unsigned int *>(p))<< "|"<<
          setw(11) << dec << *(reinterpret_cast<int *>(p)) << "|"<<
          setw(9) << dec << setprecision(4) << fixed << (*reinterpret_cast<float *>(p)) << "|"<<
          setw(9) << dec << setprecision(4) << scientific << *(reinterpret_cast<float *>(p)) << endl;
}
```

Задание Л3.з2. Разработайте программу на языке C++, которая расширяет значение целочисленной переменной из 16 бит до 32 бит, рассматривая числа как:

- знаковые (signed);
- беззнаковые (unsigned).

Проверьте её работу на значениях m и n (таблица Л3.1). Каждое из двух значений — как m, так и n — должно расширяться двумя способами — как знаковым, так и беззнаковым.

Реализация:

```
void exercise_2()
{
    cout<<"Exercise 2"<<endl;
    short m=986;
    cout<<"For 986:"<<endl;
    expansion_from_16_to_32(&m);
    unsigned n = -126;
    cout<<"For -126:"<<endl;
    expansion_from_16_to_32(&n);
}
```

Вывод:

```
Exercise 2
For 986:
         binary| hex| un decimal|    decimal
0000001111011010| 3da|        986|        986
                         binary|     hex| un decimal|    decimal|    float| exponent
00000000000000000000001111011010|     3da|        986|        986|  0.0000|1.3817e-42

         binary| hex| un decimal|    decimal
0000001111011010| 3da|        986|        986
                         binary|     hex| un decimal|    decimal|    float| exponent
00000000000000000000001111011010|     3da|        986|        986|  0.0000|1.3817e-42
For -126:
         binary| hex| un decimal|    decimal
1111111110000010|ff82|      65410|       -126
                         binary|     hex| un decimal|    decimal|    float| exponent
11111111111111111111111110000010|ffffff82| 4294967170|       -126|      nan|      nan

         binary| hex| un decimal|    decimal
1111111110000010|ff82|      65410|       -126
                         binary|     hex| un decimal|    decimal|    float| exponent
00000000000000001111111110000010|    ff82|      65410|      65410|  0.0000|9.1659e-41
```

Задание Л3.з3. Разработайте программу на языке C/C++, которая выполняет над 16-битной целочисленной переменной:

- знаковое умножение на 2;

$$m = 986,\ n = -126$$

Реализация:

```
void exercise_3()
{
    short m = 986;
    short n = -126;
    unsigned short m_uns = 986;
    unsigned short n_uns = -126;
    cout << "For m:" << endl;
    cout << "Initial m:" << endl;
    print16(&m);

    cout << "Signed m*2:" << endl;
    short m2 = m*2;
    print16(&m2);

    cout << "Unsigned m*2:" << endl;
    unsigned short m3 = m_uns*2;
    print16(&m3);

    cout << "Signed m/2:" << endl;
    short m4 = m/2;
    print16(&m4);

    cout << "Unsigned m/2:" << endl;
    unsigned short m5 = m_uns/2;
    print16(&m5);

    cout << "m mod 16:" << endl;
    unsigned short m6 = m_uns%16;
    print16(&m6);

    cout << "Rounding down:" << endl;
    unsigned short m7 = m_uns-m6;
    print16(&m7);
```

```
    // n

    cout << endl << "For n:" << endl;

    cout << "Initial n:" << endl;
    print16(&n);

    cout << "Signed m*2:" << endl;
    short n2 = n*2;
    print16(&n2);

    cout << "Unsigned m*2:" << endl;
    unsigned short n3 = n_uns*2;
    print16(&n3);

    cout << "Signed m/2:" << endl;
    short n4 = n/2;
    print16(&n4);

    cout << "Unsigned m/2:" << endl;
    unsigned short n5 = n_uns/2;
    print16(&n5);

    cout << "m mod 16:" << endl;
    unsigned short n6 = n_uns%16;
    print16(&n6);

    cout << "Rounding down:" << endl;
    unsigned short n7 = n_uns-m6;
    print16(&n7);
}
```

Вывод:

```
For m:
Initial m:
          binary| hex| un decimal|     decimal
0000001111011010| 3da|         986|         986
Signed m*2:
          binary| hex| un decimal|     decimal
0000011110110100| 7b4|        1972|        1972
Unsigned m*2:
          binary| hex| un decimal|     decimal
0000011110110100| 7b4|        1972|        1972
Signed m/2:
          binary| hex| un decimal|     decimal
0000000111101101| 1ed|         493|         493
Unsigned m/2:
          binary| hex| un decimal|     decimal
0000000111101101| 1ed|         493|         493
m mod 16:
          binary| hex| un decimal|     decimal
0000000000001010|   a|          10|          10
Rounding down:
          binary| hex| un decimal|     decimal
0000001111010000| 3d0|         976|         976

For n:
Initial n:
          binary| hex| un decimal|     decimal
1111111110000010|ff82|       65410|        -126
Signed m*2:
          binary| hex| un decimal|     decimal
1111111100000100|ff04|       65284|        -252
Unsigned m*2:
          binary| hex| un decimal|     decimal
1111111100000100|ff04|       65284|        -252
Signed m/2:
          binary| hex| un decimal|     decimal
1111111111000001|ffc1|       65473|         -63
Unsigned m/2:
          binary| hex| un decimal|     decimal
0111111111000001|7fc1|       32705|       32705
m mod 16:
          binary| hex| un decimal|     decimal
0000000000000010|   2|           2|           2
Rounding down:
          binary| hex| un decimal|     decimal
1111111101111000|ff78|       65400|        -136
```

Задание Л3.з4. Разработайте программу на языке C/C++, которая выполняет над 16-битной целочисленной переменной x:

- знаковый сдвиг влево на 1 бит;
- беззнаковый сдвиг влево на 1 бит;
- знаковый сдвиг вправо на 1 бит;
- беззнаковый сдвиг вправо на 1 бит;
- рассчитывает $x \ \& \ 15$;
- рассчитывает $x \ \& \ -16$.

Проверьте её работу на значениях m и n (таблица Л3.1). Исходное значение и результат выведите в представлениях (а)–(з) функцией $print16()$.

Сопоставьте результаты с заданием Л3.з3.

Реализация:

```
void exercise_4()
{
    short m=986;
    short n= -126;
    unsigned short m_=986;
    unsigned short n_= static_cast <int>(-126);
    cout<<"Initial data"<<endl;
    print16(&m);
    print16 (&n);
    print16(&m_);
    print16 (&n_);

    cout<<"Sign shift left by 1 bit"<<endl;
    short m1= m << 1;
    short n1= n << 1;
    print16(&m1);
    print16 (&n1);
    cout<<"Unsigned left shift by 1 bit"<<endl;
    unsigned short m2= m_ << 1;
    unsigned short n2= n_ << 1;
    print16(&m2);
    print16 (&n2);
    cout<<"Sign shift right by 1 bit"<<endl;
    short m3= m >> 1;
    short n3= n >> 1;
    print16 (&m3);
    print16 (&n3);
    cout<<"Unsigned right shift by 1 bit"<<endl;
    unsigned short m4= m_ >> 1;
    unsigned short n4= n_ >> 1;
    print16 (&m4);
    print16 (&n4);
    cout<<"X & 15"<<endl;
    short m5= m & 15;
    short n5= n & 15;
    print16(&m5);
    print16 (&n5);
    cout<<"X & -16"<<endl;
    short m6=m & (-16);
    short n6= n & (-16);
    print16(&m6);
    print16 (&n6);
}
```

Вывод:

```
Initial data
          binary| hex| un decimal|     decimal
0000001111011010| 3da|        986|         986
          binary| hex| un decimal|     decimal
1111111110000010|ff82|      65410|        -126
          binary| hex| un decimal|     decimal
0000001111011010| 3da|        986|         986
          binary| hex| un decimal|     decimal
1111111110000010|ff82|      65410|        -126
Sign shift left by 1 bit
          binary| hex| un decimal|     decimal
0000011110110100| 7b4|       1972|        1972
          binary| hex| un decimal|     decimal
1111111100000100|ff04|      65284|        -252
Unsigned left shift by 1 bit
          binary| hex| un decimal|     decimal
0000011110110100| 7b4|       1972|        1972
          binary| hex| un decimal|     decimal
1111111100000100|ff04|      65284|        -252
Sign shift right by 1 bit
          binary| hex| un decimal|     decimal
0000000111101101| 1ed|        493|         493
          binary| hex| un decimal|     decimal
1111111111000001|ffc1|      65473|         -63
Unsigned right shift by 1 bit
          binary| hex| un decimal|     decimal
0000000111101101| 1ed|        493|         493
          binary| hex| un decimal|     decimal
0111111111000001|7fc1|      32705|       32705
X & 15
          binary| hex| un decimal|     decimal
0000000000001010|   a|          10|          10
          binary| hex| un decimal|     decimal
0000000000000010|   2|           2|           2
X & -16
          binary| hex| un decimal|     decimal
0000001111010000| 3d0|        976|         976
          binary| hex| un decimal|     decimal
1111111110000000|ff80|      65408|        -128
```

Задание Л3.з5. Разработайте программу на языке C/C++, которая, используя только сложение, вычитание и побитовые операции, округляет целочисленное беззнаковое значение x до кратного значению D (таблица Л3.2) двумя способами:

а) вниз;

б) вверх.

$D = 64$

Реализация:

```
void exercise_5()
{
    int x;
    cout << "Enter the number "<<endl;
    cin >> x;
    int D = 64;
    cout << "Rounding down: " << x - (x&(D-1)) << endl;
    cout << "Rounding up: "     << x + D - (x&(D-1)) << endl;

}
```

Вывод:

```
Enter the number
654
Rounding down: 640
Rounding up: 704
```

Задание Л3.з6. Разработайте программу на языке C/C++, которая выполняет для 32-битной переменной целочисленный инкремент (то есть целочисленная интерпретация соответствующего 32-битного участка памяти должна увеличиться на 1) и целочисленный декремент (аналогично, целочисленная интерпретация должна уменьшиться на 1).

Проверьте её работу на целочисленных значениях m и n (таблица Л3.1), значениях с плавающей запятой из таблицы Л3.3, а также на целочисленных значениях:

Варианты значений с плавающей запятой

Таблица Л3.3

$(№ - 1)\%3 + 1$	**Вариант**
1	$a = 0, b = 1, c = 12345678, d = 123456789$
2	$a = 0, b = 1, c = 12233445, d = 122334455$

– 0;
– максимальное целое 32-битное значение без знака;
– минимальное целое 32-битное значение со знаком;
– максимальное целое 32-битное значение со знаком.

Исходное значение и результат выведите в представлениях (а)–(з) функцией $print32()$.

Реализация(инкремент):

```
void exercise_6_increment()
{
    int m = 564;
    int n = -322;

    double a = 0;
    double b = 1;
    double c = 12233445;
    double d = 122334455;

    int zero = 0;
    unsigned int umax = UINT_MAX;
    int max = INT_MAX;
    int min = INT_MIN;

    cout << "Original numbers" << endl;
    print32(&m);
    print32(&n);
    print32(&a);
    print32(&b);
    print32(&c);
    print32(&d);
    print32(&zero);
    print32(&umax);
    print32(&max);
    print32(&min);

    cout << "Increment" << endl;
    print32(&(++m));
    print32(&(++n));
    print32(&(++a));
    print32(&(++b));
    print32(&(++c));
    print32(&(++d));
    print32(&(++zero));
    print32(&(++umax));
    print32(&(++max));
    print32(&(++min));
}
```

Вывод:

```
Original numbers
                          binary|     hex| un decimal|    decimal|    float| exponent
00000000000000000000001000110100|     234|        564|        564|   0.0000|7.9033e-43
                          binary|     hex| un decimal|    decimal|    float| exponent
11111111111111111111111010111110|fffffebe| 4294966974|       -322|      nan|      nan
                          binary|     hex| un decimal|    decimal|    float| exponent
00000000000000000000000000000000|       0|          0|          0|   0.0000|0.0000e+00
                          binary|     hex| un decimal|    decimal|    float| exponent
00000000000000000000000000000000|       0|          0|          0|   0.0000|0.0000e+00
                          binary|     hex| un decimal|    decimal|    float| exponent
10100000000000000000000000000000|a0000000| 2684354560|-1610612736|  -0.0000|-1.0842e-19
                          binary|     hex| un decimal|    decimal|    float| exponent
11011100000000000000000000000000|dc000000| 3690987520| -603979776|-144115188075855872.0000|-1.4412e+17
                          binary|     hex| un decimal|    decimal|    float| exponent
00000000000000000000000000000000|       0|          0|          0|   0.0000|0.0000e+00
                          binary|     hex| un decimal|    decimal|    float| exponent
11111111111111111111111111111111|ffffffff| 4294967295|         -1|      nan|      nan
                          binary|     hex| un decimal|    decimal|    float| exponent
01111111111111111111111111111111|7fffffff| 2147483647| 2147483647|      nan|      nan
                          binary|     hex| un decimal|    decimal|    float| exponent
10000000000000000000000000000000|80000000| 2147483648|-2147483648|  -0.0000|-0.0000e+00
Increment
                          binary|     hex| un decimal|    decimal|    float| exponent
00000000000000000000001000110101|     235|        565|        565|   0.0000|7.9173e-43
                          binary|     hex| un decimal|    decimal|    float| exponent
11111111111111111111111010111111|fffffebf| 4294966975|       -321|      nan|      nan
                          binary|     hex| un decimal|    decimal|    float| exponent
00000000000000000000000000000000|       0|          0|          0|   0.0000|0.0000e+00
                          binary|     hex| un decimal|    decimal|    float| exponent
00000000000000000000000000000000|       0|          0|          0|   0.0000|0.0000e+00
                          binary|     hex| un decimal|    decimal|    float| exponent
11000000000000000000000000000000|c0000000| 3221225472|-1073741824|  -2.0000|-2.0000e+00
                          binary|     hex| un decimal|    decimal|    float| exponent
11100000000000000000000000000000|e0000000| 3758096384| -536870912|-36893488147419103232.0000|-3.6893e+19
                          binary|     hex| un decimal|    decimal|    float| exponent
00000000000000000000000000000001|       1|          1|          1|   0.0000|1.4013e-45
                          binary|     hex| un decimal|    decimal|    float| exponent
00000000000000000000000000000000|       0|          0|          0|   0.0000|0.0000e+00
                          binary|     hex| un decimal|    decimal|    float| exponent
10000000000000000000000000000000|80000000| 2147483648|-2147483648|  -0.0000|-0.0000e+00
                          binary|     hex| un decimal|    decimal|    float| exponent
10000000000000000000000000000001|80000001| 2147483649|-2147483647|  -0.0000|-1.4013e-45
```

Реализация(декремент):

```
void exercise_6_decrement()
{
    int m = 564;
    int n = -322;

    double a = 0;
    double b = 1;
    double c = 12233445;
    double d = 122334455;

    int zero = 0;
    unsigned int umax = UINT_MAX;
    int max = INT_MAX;
    int min = INT_MIN;

    cout << "Original numbers" << endl;
    print32(&m);
    print32(&n);
    print32(&a);
    print32(&b);
    print32(&c);
    print32(&d);
    print32(&zero);
    print32(&umax);
    print32(&max);
    print32(&min);

    cout << "Decrement" << endl;
    print32(&(--m));
    print32(&(--n));
    print32(&(--a));
    print32(&(--b));
    print32(&(--c));
    print32(&(--d));
    print32(&(--zero));
    print32(&(--umax));
    print32(&(--max));
    print32(&(--min));
}
```

Вывод:

```
Original numbers
                          binary|     hex| un decimal|    decimal|    float| exponent
00000000000000000000001000110100|     234|        564|        564|   0.0000|7.9033e-43
                          binary|     hex| un decimal|    decimal|    float| exponent
11111111111111111111111010111110|fffffebe| 4294966974|       -322|      nan|      nan
                          binary|     hex| un decimal|    decimal|    float| exponent
00000000000000000000000000000000|       0|          0|          0|   0.0000|0.0000e+00
                          binary|     hex| un decimal|    decimal|    float| exponent
00000000000000000000000000000000|       0|          0|          0|   0.0000|0.0000e+00
                          binary|     hex| un decimal|    decimal|    float| exponent
10100000000000000000000000000000|a0000000| 2684354560|-1610612736|  -0.0000|-1.0842e-19
                          binary|     hex| un decimal|    decimal|    float| exponent
11011100000000000000000000000000|dc000000| 3690987520| -603979776|-144115188075855872.0000|-1.4412e+17
                          binary|     hex| un decimal|    decimal|    float| exponent
00000000000000000000000000000000|       0|          0|          0|   0.0000|0.0000e+00
                          binary|     hex| un decimal|    decimal|    float| exponent
11111111111111111111111111111111|ffffffff| 4294967295|         -1|      nan|      nan
                          binary|     hex| un decimal|    decimal|    float| exponent
01111111111111111111111111111111|7fffffff| 2147483647| 2147483647|      nan|      nan
                          binary|     hex| un decimal|    decimal|    float| exponent
10000000000000000000000000000000|80000000| 2147483648|-2147483648|  -0.0000|-0.0000e+00
Decrement
                          binary|     hex| un decimal|    decimal|    float| exponent
00000000000000000000001000110011|     233|        563|        563|   0.0000|7.8893e-43
                          binary|     hex| un decimal|    decimal|    float| exponent
11111111111111111111111010111101|fffffebd| 4294966973|       -323|      nan|      nan
                          binary|     hex| un decimal|    decimal|    float| exponent
00000000000000000000000000000000|       0|          0|          0|   0.0000|0.0000e+00
                          binary|     hex| un decimal|    decimal|    float| exponent
00000000000000000000000000000000|       0|          0|          0|   0.0000|0.0000e+00
                          binary|     hex| un decimal|    decimal|    float| exponent
10000000000000000000000000000000|80000000| 2147483648|-2147483648|  -0.0000|-0.0000e+00
                          binary|     hex| un decimal|    decimal|    float| exponent
11011000000000000000000000000000|d8000000| 3623878656| -671088640|-562949953421312.0000|-5.6295e+14
                          binary|     hex| un decimal|    decimal|    float| exponent
11111111111111111111111111111111|ffffffff| 4294967295|         -1|      nan|      nan
                          binary|     hex| un decimal|    decimal|    float| exponent
11111111111111111111111111111110|fffffffe| 4294967294|         -2|      nan|      nan
                          binary|     hex| un decimal|    decimal|    float| exponent
01111111111111111111111111111110|7ffffffe| 2147483646| 2147483646|      nan|      nan
                          binary|     hex| un decimal|    decimal|    float| exponent
01111111111111111111111111111111|7fffffff| 2147483647| 2147483647|      nan|      nan
```

Л3.3. Вопросы

1. Что такое расширение чисел со знаком и без знака? Для чего нужны операции расширения?
2. Как выполняются логические операции, побитовые операции и сдвиги над строкой битов?
3. Как представляются в памяти компьютера числа с плавающей запятой?

1.

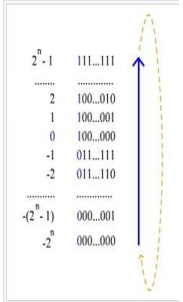
Код со сдвигом [править]

При использовании **кода со сдвигом** (англ. *Offset binary*) целочисленный отрезок от нуля до 2^n (n — количество бит) сдвигается влево на 2^{n-1}, а затем получившиеся на этом отрезке числа последовательно кодируются в порядке возрастания кодами от $000\ldots0$ до $111\ldots1$. Например, число -5 в восьмибитном типе данных, использующем код со сдвигом, превратится в $-5+128=123$, то есть будет выглядеть так: 01111011.

По сути, при таком кодировании:

- к кодируемому числу прибавляют 2^{n-1};
- переводят получившееся число в двоичную систему исчисления.

Можно получить диапазон значений $[-2^{n-1}; 2^{n-1}-1]$.



Код со сдвигом. Как видно, двоичное представление зациклено по модулю $1000..000_{(2)}$ (n нулей)

Достоинства представления чисел с помощью кода со сдвигом [править]

1. Не требуется усложнение архитектуры процессора.
2. Нет проблемы двух нулей.

Недостатки представления чисел с помощью кода со сдвигом [править]

1. При арифметических операциях нужно учитывать смещение, то есть проделывать на одно действие больше (например, после «обычного» сложения двух чисел у результата будет двойное смещение, одно из которых необходимо вычесть).
2. Ряд положительных и отрицательных чисел несимметричен.

Из-за необходимости усложнять арифметические операции код со сдвигом для представления целых чисел используется не часто, но зато применяется для хранения порядка вещественного числа.

2.

Принцип работы [править]

Логические побитовые операции [править]

Битовые операторы И (AND, &), ИЛИ (OR, |), НЕ (NOT, $\sim$) и исключающее ИЛИ (XOR, $\textasciicircum$, $\oplus$) используют те же таблицы истинности, что и их логические эквиваленты.

Побитовое И [править]

Побитовое И используется для выключения битов. Любой бит, установленный в 0, вызывает установку соответствующего бита результата также в 0.

&	11001010 11100010
	11000010

Побитовое ИЛИ [править]

Побитовое ИЛИ используется для включения битов. Любой бит, установленный в 1, вызывает установку соответствующего бита результата также в 1.

|	11001010 11100010
	11101010

Побитовое НЕ [править]

Побитовое НЕ инвертирует состояние каждого бита исходной переменной.

~	11001010
	00110101

Побитовое исключающее ИЛИ [править]

Исключающее ИЛИ устанавливает значение бита результата в 1, если значения в соответствующих битах исходных переменных различны.

^	11001010 11100010
	00101000

Побитовые сдвиги [править]

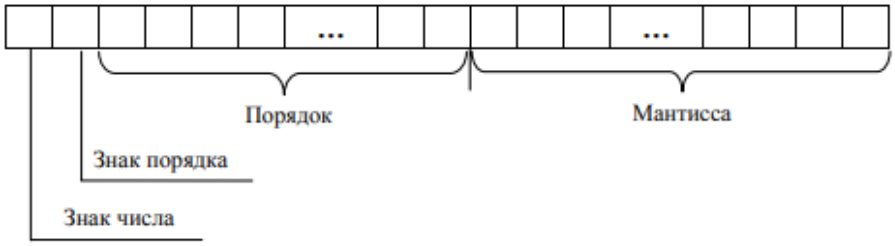
Операторы сдвига $<<$ и $>>$ сдвигают биты в переменной влево или вправо на указанное число. При этом на освободившиеся позиции устанавливаются нули (кроме сдвига вправо отрицательного числа, в этом случае на свободные позиции устанавливаются единицы, так как числа представляются в двоичном дополнительном коде и необходимо поддерживать знаковый бит).

Сдвиг влево может применяться для умножения числа на два, сдвиг вправо — для деления.

3.

4.2 Представление в виде набора битов

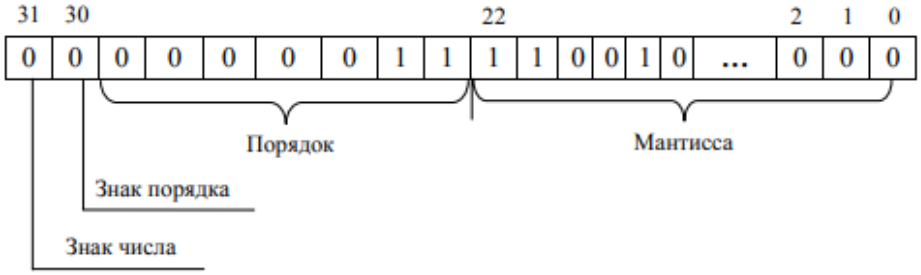
Числа с плавающей точкой представляются в виде битовых наборов, в которых отводяся разряды для мантиссы, порядка, знака числа и знака порядка:



Чем больше разрядов отводится под запись мантиссы, тем выше точность представления числа. Чем больше разрядов занимает порядок, тем шире диапазон от наименьшего отличного от нуля числа до наибольшего числа, представимого в машине при заданном формате.

Покажем на примерах, как записываются некоторые числа в нормализованном виде в четырехбайтовом формате с семью разрядами для записи порядка.

Число $6.25_{10} = 110.01_2 = 0.11001 \cdot 2^{11}$:



Число $-0.125_{10} = -0.001_2 = -0.1 \cdot 2^{-10}$ (отрицательный порядок записан в дополнительном коде):