Лабораторная работа № 4: БИТОВЫЕ ОПЕРАЦИИ.

(4 часа)

Цель лабораторной работы – познакомиться с возможностями языка C++ воспроизводить возможности машинных команд на уровне битов. Освоить основные операции с битами, понимать их назначение и применять для реализации заданных условий.

План лабораторной работы

1. Познакомиться с основными битовымии операциями.
2. Показать, для каких задач нужны операции с битами.
3. Изучить основные приёмы установки битов в ноль или единицу.
4. Научиться применять битовые операции для ускорения арифметических действий.

1. Краткие теоретические сведения по теме лабораторной работы

Побитовые операторы манипулируют отдельными битами в пределах переменной.

В далёком прошлом компьютерной памяти было очень мало и ею сильно дорожили. Это было стимулом максимально разумно использовать каждый доступный бит. Например, в логическом типе данных bool есть всего лишь два возможных значения (true и false), которые могут быть представлены одним битом, но по факту занимают целый байт памяти! А это, в свою очередь, из-за того, что переменные используют уникальные адреса памяти, а они выделяются только в байтах. Переменная bool занимает 1 бит, а другие 7 тратятся впустую.

Используя побитовые операторы, можно создавать функции, которые позволят уместить 8 значений типа bool в переменной размером 1 байт, что значительно сэкономит потребление памяти. В прошлом такой трюк был очень популярен. Но сегодня, по крайней мере, в прикладном программировании, это не так.

В противоположность большинству языков, C поддерживает все существующие битовые операторы. Поскольку C создавался, чтобы заменить ассемблер, то была необходимость поддержки всех (или по крайней мере большинства) операций, которые может выполнить ассемблер. Битовые операции — это тестирование, установка или сдвиг битов в байте или слове, которые соответствуют стандартным типам языка C `char` и `int`. Битовые операторы не могут использоваться с `float`, `double`, `long double`, `void` и другими сложными типами. При работе с побитовыми операторами используйте целочисленные типы данных unsigned.

Битовые операторы наиболее часто применяются при разработке драйверов устройств, например программ для модемов, дисков и принтеров, поскольку битовые операторы могут использоваться для выключения некоторых битов, например четности. (*Бит четности* используется для подтверждения того, что остальные биты в байте не изменялись. Он, как правило, является старшим битом в байте.)

Поразрядные операции

Поразрядные операции выполняются над соответствующими разрядами целочисленных операндов:

& : поразрядная конъюнкция (операция И или поразрядное умножение). Возвращает 1, если оба из соответствующих разрядов обоих чисел равны 1

| : поразрядная дизъюнкция (операция ИЛИ или поразрядное сложение). Возвращает 1, если хотя бы один из соответствующих разрядов обоих чисел равен 1

^ : поразрядное исключающее ИЛИ. Возвращает 1, если **только один** из соответствующих разрядов обоих чисел равен 1

~ : поразрядное отрицание или инверсия. Инвертирует все разряды операнда. Если разряд равен 1, то он становится равен 0, а если он равен 0, то он получает значение 1.

Применение операций:

```
1  int a = 5 | 2;          // 101 | 010 = 111  - 7
2  int b = 6 & 2;          // 110 & 010 = 10  - 2
3  int c = 5 ^ 2;          // 101 ^ 010 = 111 - 7
4  int d = ~9;             // -10
```

Например, выражение 5 | 2 равно 7. Число 5 в двоичной записи равно 101, а число 2 - 10 или 010. Сложим соответствующие разряды обоих чисел. При сложении если хотя бы один разряд равен 1, то сумма обоих разрядов равна 1. Поэтому получаем:

```
1   0   1
0   1   0
-----------
1   1   1
```

В итоге получаем число 111, что в десятичной записи представляет число 7.

Возьмем другое выражение 6 & 2. Число 6 в двоичной записи равно 110, а число 2 - 10 или 010. Умножим соответствующие разряды обоих чисел. Произведение обоих разрядов равно 1, если оба этих разряда равны 1. Иначе произведение равно 0. Поэтому получаем:

```
1    1    0
0    1    0
-------------
0    1    0
```

Получаем число 010, что в десятичной системе равно 2.

Таким образом, битовые операторы И, ИЛИ, НЕ используют ту же таблицу истинности, что и их логические эквиваленты, за тем исключением, что они работают побитно.

Исключающее ИЛИ имеет следующую таблицу истинности:

p	q	p^q
0	0	0
0	1	1
1	0	1
1	1	0

Для чего применяются битовые операции

Битовое И чаще всего используется для выключения битов: любой бит, установленный в 0, вызывает установку соответствующего бита в другом операнде также в 0. Например, следующий фрагмент программы читает символы, вводимые с консоли, и сбрасывает бит четности в 0:

```
char ch, ch1;
    cin >> ch;
    ch1 = ch & 127;
    cout << ch1<<endl;
```

В последовательной передаче данных часто используется формат 7 бит данных, *бит чётности*, один или два стоповых бита. Такой формат аккуратно размещает все 7-битные ASCII символы в удобный 8-битный байт.

В следующем примере показано, как работает данный фрагмент программы с битами. В нём предполагается, что ch имеет символ 'A' и имеет бит четности:

Бит чётности

```
11000001          ch содержит 'A' с битом четности
01111111          127 в двоичном представлении выполнение битового И
&  ---------------
01000001          'A' без бита четности
```

В результате работы программы чётность, отображаемая восьмым битом, устанавливается в 0 с помощью битового И, поскольку биты с номерами от 1 до 7 установлены в 1, а бит с номером 8 — в 0. Выражение

ch & 127 означает, что выполняется битовая операция И между битами переменной ch и битами числа 127. В результате получим ch со сброшенным старшим битом.

Битовое ИЛИ может использоваться для установки битов: любой бит, установленный в любом операнде, вызывает установку соответствующего бита в другом операнде. Например, в результате операции 128 | 3 получаем:

```
  10000000          128 в двоичном представлении
  00000011          3 в двоичном представлении
| --------------    битовое ИЛИ
  10000011          результат
```

Исключающее ИЛИ (XOR) устанавливает бит, если соответствующие биты в операндах отличаются. Например, в результате операции 127 ^ 120 получаем:

```
  01111111          127 в двоичном представлении
  01111000          120 к двоичном представлении битовое исключающее
^ --------------    ИЛИ
  00000111          результат
```

Оператор ~ инвертирует состояние каждого бита указанной переменной, то есть 1 устанавливается в 0, а 0 — в 1.

Битовые операторы часто используются в процедурах шифрования. Если есть желание сделать дисковый файл нечитабельным, можно выполнить над ним битовую операцию. Одним из простейших методов является использование битового дополнения для инверсии каждого бита в байте, как показано ниже:

```
Исходный байт              00101100
После первого дополнения   11010011
После второго дополнения   00101100
```

Следует обратить внимание, что в результате выполнения двух битовых дополнений получаем исходное число. Следовательно, первое дополнение будет создавать кодированную версию байта, а второе будет декодировать.

Операции сдвига

Операции битового сдвига могут быть полезны при декодировании информации от внешних устройств и для чтения информации о статусе. Операторы битового сдвига могут также использоваться для выполнения быстрого умножения и деления целых чисел.

Правила работы операторов сдвига.

Оператор **>>** *- сдвиг вправо* – сдвигает вправо биты выражения ***a*** на количество битов, указанных в выражении ***b***: **`a >> b.`**

Для заполнения позиций слева используется бит знака значения ***a.*** Цифры, сдвинутые за пределы диапазона, удаляются. Тип данных, возвращаемых данным оператором, определяется типом данных выражения ***a***.

Например:

```
short int temp
temp = -14 >> 2
```

после вычисления этого кода переменная **`temp`** имеет значение -4, поскольку при сдвиге значения -14 (11110010 в двоичном выражении) на два бита в право получается значение -4 (11111100 в двоичном выражении):

код	**Знак**	**15**	**14**	**13**	**12**	**11**	**10**	**09**	**08**	**07**	**06**	**05**	**04**	**03**		**02**	**01**		
Прям.	**1**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**0**	**1**	**1**		**1**	**0**		
Обр.	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**		**0**	**1**	инверсия	
Доп.	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**0**	**0**		**1**	**0**	**+1**	
>>2	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**	**1**		**0**	**0**	**1**	**0**

Оператор **<<** *- сдвиг влево* – сдвигает влево биты выражения ***a*** на количество битов, указанных в выражении ***b***: **`a << b.`**

“Выталкиваемые наружу” биты пропадают, освобождающиеся биты заполняются нулями. Тип данных, возвращаемых данным оператором, определяется типом данных выражения ***a***.

Например:

```
short int temp
temp = -14 << 2
```

после вычисления этого кода переменная **`temp`** имеет значение -56, поскольку при сдвиге значения -14 (11110010 в двоичном выражении) на два бита влево получается значение -56 (10111000 в двоичном выражении).

Операторы битового сдвига могут также использоваться для выполнения быстрого умножения и деления целых чисел. Сдвиг влево равносилен умножению на 2, а сдвиг вправо - делению на 2 (четных чисел):

	Битовое представление x после выполнения каждого оператора	Значение x
char x;		
x = 7;	00000111	7
x = x << 1;	00001110	14
x = x << 3;	01110000	112
x = x << 2;	11000000	192
x = x >> 1;	01100000	96
x = x >> 2;	00011000	24

Каждый сдвиг влево приводит к умножению на 2. Обратим внимание, что после сдвига x << 2 информация теряется, поскольку биты сдвигаются за конец байта.

Каждый сдвиг вправо приводит к делению на 2. Обратим внимание, что деление не вернуло потерянные биты.

Пример1 задачи на установку необходимых битов.

Написать программу, которая позволит ввести два числа типа `unsigned int` с клавиатуры, найти и вывести на консоль их сумму, используя битовые операции сделать в ней, чтобы 2-й и 1-й биты были равны 0, 3-й бит - равен 1, а остальные сохранили свои значения, вывести результат.

Решение:

```
int main()        // главная функция программы
{
      unsigned int a, b, sum; /* описание типов переменных */

setlocale(LC_ALL, "rus"); // для вывода русского шрифта в консоль
printf("\nВведите a\n");
scanf_s("%u", &a); /* вводим a */
printf("\nВведите b\n");
scanf_s("%u", &b); /* вводим a */

sum = a + b; /* нашли сумму */
printf("\nСумма равна a и b =%u", sum); /* вывели сумму на
                                                монитор */
sum = sum & 0xfff9; /* установили 2 и 1 биты в 0
                           fff9, в двоичной системе
                           1111 1111 1111 1001 */
sum |= 0x8;        /* установили 3 бит в 1
                           8, в двоичной системе
                           0000 0000 0000 1000 */
                           /* выводим преобразованную сумму */
printf("\nПосле преобразования sum=%u\n", sum);

system("pause");
return 0; // вернулись в среду разработки
}
```

Исходя из свойств побитовых операций, нам необходимо было подобрать операцию с числом, которое не меняет ни один бит, кроме 2-го и 1-го (нумерация битов начинается с нуля). Для этого применяем побитовое И с числом где во всех разрядах стоят единицы, кроме второго и первого – это шестнадцатеричное число `0xfff9`, которое в двоичном формате будет записано как `1111 1111 1111 1001`.

Для того, чтобы в 3-м бите после требуемого преобразования стояла всегда единица, мы применили побитовое И с числом, где в третьем бите будет 1, а во всех остальных 0. Для этого годится число шестнадцатеричное число `0x8`, которое в двоичном формате будет записано как `0000 0000 0000 1000`.

Результат работы программы представлен следующим образом:

```
Введите a
20

Введите b
2

Сумма равна a и b =22
После преобразования sum=24
Для продолжения нажмите любую клавишу . . .
```

Покажем, что результат битовых операций правильный.

$22_{10}=16_{16}=0000\ 0000\ 0001\ 0110_2$ – тип unsigned int

`0xfff9= 1111 1111 1111 1001` – по умолчанию эта константа имеет тип int и имеет, соответственно, 32 бита, но старшие 16 бит никак не повлияют на тип unsigned int и мы, можем старшие 16 бит не рассматривать.

Побитовое И даст следующий результат:

```
&  0000 0000 0001 0110
   1111 1111 1111 1001
   0000 0000 0001 0000 –1-й и 2-й биты выставлены в 0.
```

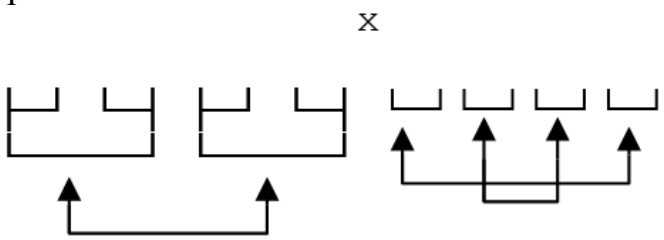
Побитовое ИЛИ полученного числа с числом `0x8` даст следующий результат:

```
|  0000 0000 0001 0000
   0000 0000 0000 1000
   0000 0000 0001 1000 –3-й бит выставлены в 1.
```

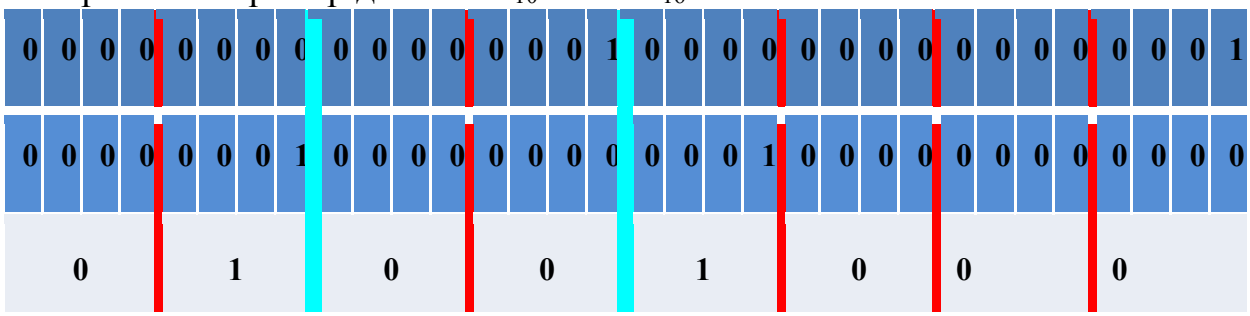
Полученное число $0000\ 0000\ 0001\ 1000_2 = 24_{10}$. Это соответствует результатам работы программы.

Пример2 задачи на применение битовых операций.

Написать программу, которая позволит ввести число x типа unsigned int с клавиатуры, напечатать его и, используя битовые операции, поменять в нем четверки и восьмерки бит по схеме

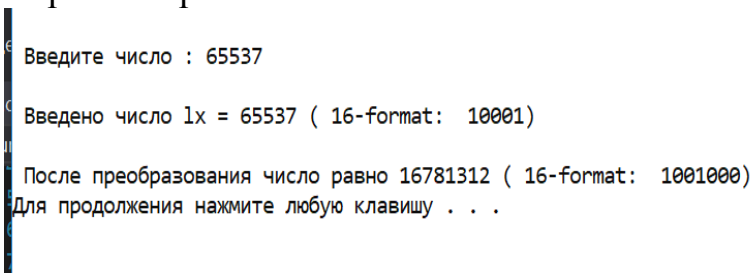


Контрольный пример для $65537_{10} = 10001_{16}$




```
int main()
{unsigned int lx, l41, l42, l43, l44, l83;
setlocale(LC_ALL, "rus"); // для вывода русского шрифта в консоль
printf("\n Введите число : "); /* поясняющая надпись */
scanf_s("%ld", &lx); /* вводим число */
printf("\n Введено число lx = %ld ( 16-format:  %X) \n", lx,lx); /*
вывели число */
l41 = lx & 0xf; /* нашли первую четверку */
l42 = lx & 0xf0; /* нашли вторую четверку */
l43 = lx & 0xf00; /* нашли третью четверку */
l44 = lx & 0xf000; /* нашли четвертую четверку */
l83 = lx & 0xff0000; /* нашли третью восьмерку */
/* поставили четвертую восьмерку на место третьей
и обнулили младшие шестнадцать и старшие 8 бит */
lx = (lx >> 8) & 0xff0000;
/* поставили все на место */
lx += (l41 << 12) + (l42 << 4) + (l43 >> 4) + (l44 >> 12) + (l83 <<
8);
/* вывели полученное значение на монитор */
printf("\n После преобразования число равно %ld ( 16-format:  %X)\n",
lx, lx);
system("pause");  return 0; }
```

Скриншот решения:



Получили 16-е число и, используя позиционную формулу, перевели его в 10-е:
$\mathbf{1001000_{16} = 16^6 + 16^3 = 16781312_{10}}$

Пример 3 задачи на применение битовых операций.
Дано число k, $0 \le k \le 31$. Не используя арифметические операторы сложения, умножения, вычитания, деления, взятия остатка, вычислить 2^k, применяя побитовые операторы &, |, ~, ^, <<, >>.

Контрольный пример:

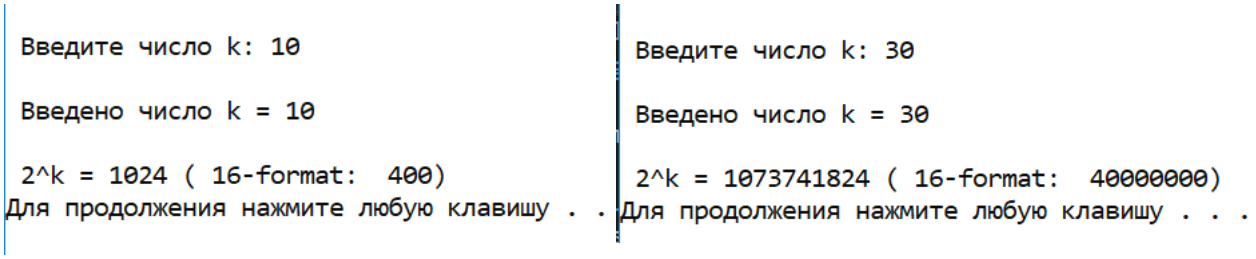
$2^0 = 1_{10} = 00000001_2$
$2^1 = 2_{10} = 00000010_2$
$2^2 = 4_{10} = 00000100_2$
…..

Очевидно, что степень k позволяет сместить 1 влево на k позиций

Решенние:

```
int main()
{
unsigned int k, n=1,m; /* описание типов переменных */
setlocale(LC_ALL, "rus"); // для вывода русского шрифта в консоль
printf("\n Введите число k: "); /* поясняющая надпись */
scanf_s("%ld", &k); /* вводим число */
printf("\n Введено число k = %ld  \n", k); /* вывели число */
m = n << k;
printf("\n 2^k = %ld ( 16-format:  %X)\n", m, m);
system("pause");  return 0; // вернулись в операционную систему
}
```

Скриншот решения:



3. Методика выполнения самостоятельной работы

1. Ознакомиться с условием задачи и примерами решения аналогичных задач из первых разделов лабораторной работы, составить контрольные примеры для простых числовых значений на все варианты ветвления.

2. Проверить полноту задачи: рассмотреть все возможные исходы решения в зависимости от исходных данных, предусмотреть случаи возможного зависания, зацикливания программы и запрограммировать корректную реакцию программы на эти ситуации.

3. Записать словесный алгоритм или составить блок-схему алгоритма.

4. Записать код программы на C^{++}.

5. Запустить программу, провести синтаксическую отладку.

6. Проверить работоспособность программы путём сравнения результатов с контрольным примером на все возможные случаи исходных данных.

7. Завершить работу составлением Отчёта, где будут описаны все этапы выполнения самостоятельного задания и приведены тексты программ распечатки консольного вывода. Образец отчета приведен в Приложении.

4. Задания для самостоятельной работы

Во всех вариантах всех заданий для исходных чисел брать тип, если не указано иное, `unsigned int`.

Если не оговорено иное, нельзя использовать арифметические операторы сложения, умножения, вычитания, деления, взятия остатка. Вместо них используем побитовые операторы &, |, ~, ^, <<, >>.

Для более удобного тестирования программ заданий можно вводить и выводить результат в HEX формате (%x).

Для формирования алгоритма и проверки корректности решения обязательно составлять контрольные примеры или разобрать представленные в условии примеры.

Задание 1.

Вариант 1. Написать программу, которая позволит ввести целое число с клавиатуры и преобразовать его так, чтобы оно было числом, приведенном к числу кратному 16. При преобразовании использовать только битовые операции.

Вариант 2. Написать программу, которая позволит ввести целое число с клавиатуры и преобразовать его так, чтобы оно было числом, приведенном к числу кратному 32

Вариант 3. Написать программу, которая позволит ввести число типа `unsigned short int` с клавиатуры, напечатать его и, используя битовые операции, сделать в нем, чтобы четные биты были равны 0, а нечетные сохранили свои значения и вывести результат. Так, при ведении числа 1111 1110 1010 1100 на выходе должно получиться число 1010 1010 1010 1100.

Вариант 4. Написать программу, которая позволит ввести число типа `unsigned short int` с клавиатуры, напечатать его и, используя битовые операции, сделать в нем, чтобы нечетные биты были равны 0, а четные сохранили свои значения и вывести результат. Так, при ведении числа 1111 1110 1010 1100 на выходе должно получиться число 0101 0100 0000 0100.

Вариант 5. Написать программу, которая позволит ввести число типа `unsigned short int` с клавиатуры, напечатать его и, используя битовые операции, сделать в нем, чтобы 4, 5, 6 биты были равны 0, а остальные сохранили свои значения и вывести результат.

Вариант 6. Написать программу, которая позволит ввести число типа `unsigned short int` с клавиатуры, напечатать его и, используя битовые операции, сделать в нем, чтобы 1, 2, 3 биты были равны 0, а остальные сохранили свои значения и вывести результат.

Вариант 7. Написать программу, которая позволит ввести число типа `unsigned short int` с клавиатуры, напечатать его и, используя битовые операции, сделать в нем, чтобы 4 и 5 биты были равны 1, а остальные сохранили свои значения и вывести результат.

Указание: Какому числу будет кратен результат, если 3, 2, 1, 0 биты будут установлены в 0?

Вариант 8. Написать программу, которая позволит ввести число типа `unsigned short int` с клавиатуры, напечатать его и, используя битовые операции, сделать в нем, чтобы 0, 2 и 4 биты были равны 1, а остальные сохранили свои значения и вывести результат.

– Указание: Какому числу будет кратен результат, если 1 и 0 биты будут установлены в 0?

Вариант 9. Написать программу, которая позволит ввести два числа типа `unsigned int` с клавиатуры, найти и вывести на консоль остаток от деления первого числа на второе и, используя битовые операции, установить в нем в 0

те биты, которые соответственно в первом числе установлены в 1, вывести результат.

Вариант 10. Написать программу, которая позволит ввести число с клавиатуры, напечатать его и, используя битовые операции, проверить, делится оно на 16 и не делится на 256, вывести результат.

Указание: как выглядят в двоичном представлении числа, кратные 16 и 256?

Вариант 11. Написать программу, которая позволит ввести целое число с клавиатуры и преобразовать его так, чтобы оно было числом, приведенном к числу кратному 16. При преобразовании использовать только битовые операции. Например, если введено число кратное16, то преобразования не требуется, если 17, то должно быть получено число 17·16=272, если 12, то число 12·4=48, если 10, то число 10·8=80.

Указание: Как в двоичном представлении выглядят числа, кратные 16?

Вариант 12. Написать программу, которая позволит ввести целое число с клавиатуры и преобразовать его так, чтобы оно было числом, приведенном к числу кратному 32, максимально близкому к исходному числу. При преобразовании использовать только битовые операции. Например, если введено число 64 или 128, то преобразования не требуется, если 17, то должно быть получено число 17·32= 544, если 70, то число 70·16= 1120, если 44, то число 44·8= 352, если 56, то число 56·4= 224.

Указание: Как в двоичном представлении выглядят числа, кратные 32?

Задание 2.

Вариант 1. Написать программу, которая позволит ввести два целых числа n и k и, используя битовые операции, сделать в числе n циклическую перестановку на k бит вправо, вывести результат. Циклическая перестановка числа 0011 0101 0011 1101 на один бит вправо даст число 1001 1010 1001 1110.

Вариант 2. Написать программу, которая позволит ввести два целых числа n и k и, используя битовые операции сделать в числе n циклическую перестановку нечетных бит на k бит влево, вывести результат. Циклическая перестановка нечетных бит числа 0111 0111 0011 1101 на один бит влево даст число 1101 1101 1011 0101.

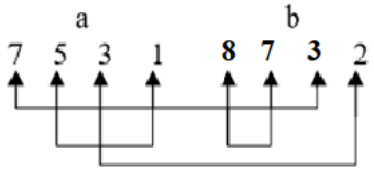
Вариант 3. Написать программу, которая позволит ввести целое число и, используя битовые операции, сделать в нем, чтобы четные биты стали равны

нечетным, а нечетные четным и вывести результат. Например, при вводе числа $B2_{16}$, на выходе получить число 71_{16}.

Вариант 4. Написать программу, которая позволит ввести целое число и, используя битовые операции, поменять в нем восьмерки бит, так, чтобы первая восьмерка стала равной второй, а вторая первой и вывести результат. Например, число $B3C5_{16}$ программа превратит в число $C5B3_{16}$.

Вариант 5. Написать программу, которая позволит ввести целое число и, используя битовые операции, сделать в нем, сделать в нем, чтобы нечетные биты стали равны четным, стоящим справа, вывести результат. Например, число $B3C5_{16}$ программа превратит в число $33CF_{16}$.

Вариант 6. Написать программу, которая позволит ввести два числа a и b типа `long` с клавиатуры, напечатать их на консоли, используя битовые операции поменять местами в них двойки бит по указанной схеме, вывести результат.

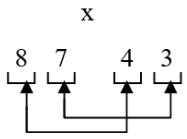


Вариант 7. Написать программу, которая позволит ввести число типа `unsigned int` с клавиатуры, напечатать его на консоли, используя битовые операции сделать в нем преобразование бит N[i]=N[i-1], N[i-1]=N[i], i=1, 3, 5, 7 и вести результат.

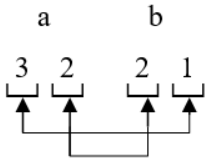
Вариант 8. Написать программу, которая позволит ввести два числа типа `unsigned int` с клавиатуры, напечатать их на консоли, используя битовые операции поменять местами в них четные биты второго числа на нечетные биты первого, вывести результат.

Вариант 9. Написать программу, которая позволит ввести два числа типа `unsigned int` с клавиатуры, напечатать их на консоли, используя битовые операции поменять местами в них нечетные биты первого числа на четные биты второго, вывести результат.

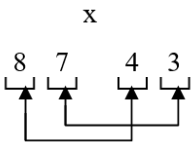
Вариант 10. Написать программу, которая позволит ввести два числа a и b с типа `unsigned int` с клавиатуры, напечатать их на консоли, используя битовые операции, поменять местами в них четверки бит по представленной схеме, вывести результат.



Вариант 11. Написать программу, которая позволит ввести два числа a и b типа `unsigned int` с клавиатуры, напечатать их на консоли, используя битовые операции поменять местами в них восьмерки бит по представленной схеме, вывести результат.



Вариант 12. Написать программу, которая позволит ввести число x типа `unsigned int` с клавиатуры, напечатать его на консоли, используя битовые операции поменять в нем четверки бит по следующей схеме, вывести результат.



Задание 3.

Вариант 1. Даны два **неравных** целых неотрицательных числа: k и n. Вычислите 2^k+2^n.

Вариант 2. Дано целое число A и целое неотрицательное число k. Обнулите у числа A его последние k бит и выведите результат.

Вариант 3. Дано целое число A и целое неотрицательное число k. Выведите число, которое получается из числа A установкой значения k-го бита равному 1.

Вариант 4. Дано целое число A и целое неотрицательное число k. Выведите число, которое получается из числа A инвертированием k-го бита.

Вариант 5. Написать программу, которая позволит ввести число типа `unsigned long` с клавиатуры, напечатать его на консоли, используя битовые операции сделать в нем, чтобы биты кратные 3 стали равны соседним слева.

Вариант 6. Дано целое число A и натуральное число k. Выведите число, которое состоит только из k последних бит числа A.

Вариант 7. Дано натуральное число. Выведите его битовое представление. В результате должно получиться число состоящее из нулей и единиц, которое можно вывести с применением `cout` или `printf`. Разрешается использовать арифметическое сложение.

Вариант 8. Даны числа a и b. Используя только битовые операции и операции арифметического сложения и вычитания вычислите число

x=(36a+[$\frac{b}{16}$]) *mod* 32. Выведите результат на экран. Операция *mod* – вычисление остатка от целочисленного деления.

Вариант 9. Даны целые числа a и b. Не используя операции `*`, `/`, `%` вычислите их произведение, используя только битовые операции.

Вариант 10. Дано число, замените первую справа единицу его двоичной записи на ноль. Разрешается использовать битовые и арифметические операции. Запрещается использовать ветвления и циклы.

Вариант 11. Дано число, замените первый справа ноль его двоичной записи на единицу. Разрешается использовать битовые и арифметические операции. Запрещается использовать ветвления и циклы.

Вариант 12. Шифрование с перемешиванием: дано число, переставьте его соседние биты (то есть поменяйте местами биты с номерами 0 и 1, 2 и 3, 4 и 5 и т.д.). Общее число бит в числе не превосходит 32. Разрешается использовать битовые операции. Запрещается использовать арифметические операции, ветвления, циклы. Составьте программу дешифровки зашифрованного числа.

* * *

4. Контрольные вопросы

1. Для каких целей используются битовые операции?
2. Каков приоритет битовых операци?
3. Какие битовые операции не допускают совмещения с оператором присваивания?
4. С каким числом необходимо произвести логическое умножение, чтобы сохранить в нём неизменными 1-й, 5-й и 6-й биты, а остальные обнулить?
5. С каким числом необходимо произвести логическое сложение, чтобы выставить в нём единцы в 3-ем, 4-м и 7-м битах, а остальные оставить неизменными?

5. Домашнее задание

Для решения задач ДЗ обязательно составить контрольные примеры, позволяющие отработать алгоритм решения и проверить корректность полученного результата.

1. Написать программу с применением битовых операций, которая для трёх целых неотрицательных переменных X, Y, Z выполняет следующие

операции: переменной X присваивает значения младших 5 битов, а переменной Y – следующие 6 битов переменной Z.

2. Написать программу с применением битовых операций, которая для трёх переменных X, Y, Z, являющихся двухбайтными беззнаковыми величинами (X<64, Y<128), выполняет следующие операции: в младшие 6 битов переменной Z помещается число X, в следующие 7 битов Z размещается число Y, старшие биты Z сохраняются без изменения.