# Обозначения

Если у числа имеется нижний индекс, например, 1012 или 1410 то он указывает систему счисления, в которой представлено число.

В коде на языке C все числа считаются компьютером десятичными, однако можно задавать и шестнадцатеричные числа, для этого необходимо перед числом ставить символ 0x. Например 0x10, 0xFC. Это нужно знать, так как часто удобнее пользоваться именно шестнадцатеричными числами.

В примерах кода будем опускать все несущественные в данный момент детали. Будут приводится только те строки, которые реально относятся к теме. Отдельно они работать не будут. Подразумевается, что эти строки должны находится внутри каких-либо функций и все переменные заранее объявлены.

# Битовые операции

Одно из базовых действий, которое приходится выполнять при программировании микроконтроллера является работа с регистрами. Чтобы работать с конкретными битами регистра удобно пользоваться битовыми операциями. Рассмотрим основные операции, которые существуют в языке C и которые пригодятся нам для редактирования битов регистра.

1. Битовое «И» (оператор в языке C – «&»). Для этой операции нужно два числа. Компьютер будет применять операцию логического «И» по отдельности для каждой пары битов чисел и сохранять результат. При написании кода применение данной операции будет выглядеть так:

x = 4 & 13;

Эта строка указывает компьютеру, что нужно провести операцию битового «И» с числами 4 и 13 и поместить результат выполнения в переменную x (которая объявлена заранее). Для компьютера числа 4 и 13 выглядят как 1002 и 11012. Для выполнения операции числа должны быть одной длинны (в двоичной записи), поэтому наименьшее из чисел дополняется слева нулями до нужной длины. Покажем примерный ход выполнения компьютером данной работы:

Шаг 1 Шаг 2 Шаг 3 Шаг 4

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 |  | 0 | 1 | 0 | 0 |  | 0 | 1 | 0 | 0 |  | 0 | 1 | 0 | 0 |
|  |  |  | && |  |  |  | && |  |  |  | && |  |  |  | && |  |  |  |
| 1 | 1 | 0 | 1 |  | 1 | 1 | 0 | 1 |  | 1 | 1 | 0 | 1 |  | 1 | 1 | 0 | 1 |
|  |  |  | = |  |  |  | = |  |  |  | = |  |  |  | = |  |  |  |
| 0 | 0 | 0 | 0 |  | 0 | 0 | 0 | 0 |  | 0 | 1 | 0 | 0 |  | 0 | 1 | 0 | 0 |

Здесь в первой строке всегда число 4, во второй – 13, а в третьей результат проделанной работы (значение переменной x). Нужно помнить, что операция логического «И» даёт единицу только тогда, когда оба числа, над которыми совершается операция, равны единице. В остальных случаях всегда будет нуль. Конкретное применение этой и других операций будет рассмотрено в разделе «работа с регистрами».

1. Битовое «ИЛИ» (оператор в языке C – «|»). Для этой операции нужно два числа. Компьютер будет применять операцию логического «ИЛИ» по отдельности для каждой пары битов чисел и сохранять результат. При написании кода применение данной операции будет выглядеть так:

x = 4 | 13;

Эта строка указывает компьютеру, что нужно провести операцию битового «ИЛИ» с числами 4 и 13 и поместить результат выполнения в переменную x (которая объявлена заранее). Здесь справедливы все те же принципы, что и в случае с битовым «И». Покажем примерный ход выполнения компьютером данной работы:

Шаг 1 Шаг 2 Шаг 3 Шаг 4

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 1 | 0 | 0 |  | 0 | 1 | 0 | 0 |  | 0 | 1 | 0 | 0 |  | 0 | 1 | 0 | 0 |
|  |  |  | || |  |  |  | || |  |  |  | || |  |  |  | || |  |  |  |
| 1 | 1 | 0 | 1 |  | 1 | 1 | 0 | 1 |  | 1 | 1 | 0 | 1 |  | 1 | 1 | 0 | 1 |
|  |  |  | = |  |  |  | = |  |  |  | = |  |  |  | = |  |  |  |
| 0 | 0 | 0 | 1 |  | 0 | 0 | 0 | 1 |  | 0 | 1 | 0 | 1 |  | 1 | 1 | 0 | 1 |

Здесь в первой строке всегда число 4, во второй – 13, а в третьей результат проделанной работы (значение переменной x). Нужно помнить, что операция логического «ИЛИ» даёт нуль только тогда, когда оба числа, над которыми совершается операция, равны нулю. В остальных случаях всегда будет единица.

1. Битовая инверсия (оператор в языке C – «~»). Эта операция совершается только над одним числом. Заменяет все единицы в двоичной записи числа на нули, а нули – на единицы. В коде будет выглядеть так:

x = ~4;

Эта строка указывает компьютеру, что нужно инвертировать биты числа 4 ( 1002 ) и записать результат в переменную x. Нетрудно видеть, что в нашем случае в переменную x запишется число 0112, что равно, в свою очередь, числу 310.

1. Битовый сдвиг влево (оператор в языке C – «<<»). Совершается только над одним числом. Сдвигает все биты число на указанное количество позиций. В коде это выглядит так:

x = 1 << 2;

Эта строка говорит компьютеру, что ему нужно сдвинуть число 1 на две позиции. При сдвиге младшие биты, которые имеют номера меньше чем число, на которое сдвигается число – обнуляются. Таким образом в переменную x будет записано число 1002. Приведём ещё пример:

x = 5 << 6;

Так как число 5 в двоичной системе равно 1012 то после сдвига получится число 1010000002.

Тут нужно быть внимательным и понимать, что результат выполнения записывается в некоторую переменную x и это накладывает ограничения на те числа, которые можно записать в неё без искажений. Например, если переменная x такова, что может хранить только число, состоящее из восьми бит, то в результате выполнения последнего примера в неё запишется число 010000002, что не равно тому числу, которое должно было получится при выполнении операции сдвига.

Также существует операция сдвига вправо, которая работает точно так же, но в другую сторону. Она используется реже.

1. Битовое исключающее или (оператор в языке С – «^»). Результат выполнения этой операции будет равен единице, если биты, над которыми производится операция имеют разные значения. Если биты содержат одинаковые значения – результат выполнения операции будет равен нулю.

# Работа с регистрами

Типичные задачи при работе с регистрами это:

* Считать значение конкретного бита(ов) регистра
* Записать единицу в конкретный бит(ы) регистра
* Записать нуль в конкретный бит(ы) регистра

Условимся, что в дальнейших примерах в переменной с именем reg будет значение регистра……. В переменной res – результат выполнения операции

## Чтение бита

Для чтения одного бита есть простой типовой подход. Совершают операцию битового «И» над читаемым регистром и единицей, сдвинутой на номер читаемого бита. Так, на языке C код чтения бита с номером 14 будет выглядеть так:

res = reg & (1 << 14);

Если после выполнения операции в переменная res будет содержать значение 1, то это означает, что в регистре в бите с индексом 14 записана единица, иначе – нуль.

Для чтения нескольких битов подряд (величина интересующего нас поля регистра может быть и больше одного бита) необходимо проделать ту же операцию, но с другой маской. Под маской тут понимается такое число, которое позволяет получить нужное нам значение в результате операции битового «И» над маской и регистром. В прошлом примере маской являлось число (1 << 14). Если нам надо считать два бита с 14 позиции (биты с индексами 14 и 15), то воспользуемся следующим кодом:

res = reg & (3 << 14);

Здесь маска (3 << 14), ведь 3 = 112. Так же возможен, например, такой вариант:

res = reg & ( (1 << 14) | (1<<15) );

Здесь производится всё то же самое, только маску получаем немного другим способом (сдвигая единицы в нужные разряды и делая над ними битовое «ИЛИ»)

При больших диапазонах считываемых значений удобнее пользоваться числами в шестнадцатеричной системе. Например, чтобы считать 6 бит подряд, начиная с 14 позиции:

res = reg & (0x3F << 14);

Нетрудно сообразить, что 0x3F равно 1111112

## Записать единицу в бит

Для записи единицы в бит удобно пользоваться операцией битового «ИЛИ». Рассмотрим на примере следующего кода:

reg = reg | (1 << 14);

Здесь мы записываем единицу в бит с индексом 14. Этот же код можно переписать в более компактном виде (более предпочтителен):

reg |= (1 << 14);

То есть выражение «reg = reg |» эквивалентно выражению «reg |=»

Обратим внимание, что операция битового «ИЛИ» когда в качестве маски нуль не изменяет числа. То есть при записи единицы в конкретный бит мы точно уверены, что остальные биты регистра не поменяются, что важно.

## Записать нуль в бит

Чтобы занулить бит необходимо применить операцию битового «И» с таким числом, которое содержит единицы во всех разрядах, кроме того разряда, который мы хотим обнулить в регистре (так мы сохраним остальные значения регистра без изменений). Чтобы получить такое число нужно сначала произвести сдвиг единицы в нужный разряд, а затем инвертировать полученное число. Код на языке C будет выглядеть так:

reg &= ~(1 << 14);

В данном примере сначала единица сдвигается на 14 разрядов. Получается число, которое содержит во всех позициях нули, а в 14-й – единицу. Затем компьютер инвертирует его, и получает число, которое содержит во всех своих разрядах единицы, а в 14-м – нуль. Битовое «И» с единицей не меняет старое значение регистра, битовое «И» с нулём – обнуляет значение регистра.

## Запись числа в регистр

Когда редактируемое поле регистра состоит из нескольких бит, удобно сразу записывать значение целиком, а не выставлять каждый бит по-отдельности. Например, если нам необходимо записать в поле из четырёх бит, занимающее 10-13 биты регистра, число 10112 то сделать это можно таким образом:

reg |= (1 << 10);

reg |= (1 << 11);

reg &= ~(1 << 12);

reg |= (1 << 13);

А можно по-другому:

reg &= ~(0xF << 10);

reg |= (0xB << 10);

Здесь в первой строке мы сначала обнуляем предыдущее значение регистра, чтобы оно не повлияло на то значение, которое мы хотим записать. Во второй строке сразу записываем желаемое значение в нужное поле.

## Поменять значение бита

Иногда бывает нужно просто сменить значение бита на противоположное. Чтобы не тратить на это лишние операции (можно изначально прочитать бит, и в зависимости от считанного значения установить новое значение) можно пользоваться следующим приемом:

reg ^= (1 << 15);

Здесь используется операция исключающего ИЛИ. Мы меняем значение в 15-м бите регистра reg. Если подробнее рассмотреть, что происходит, то картина следующая: если в бите изначально содержалась единица, то исключающее или с другой единицей даст нуль, а если изначально в бите был нуль, то исключающее или с единицей выставит в бите единицу. А значит в любом случае значение меняется правильно.