Побитовые операции

Побитовые операции – мощный и изящный инструмент, полезный при низкоуровневом программировании аппаратных средств. Однако мощный инструмент требует правильного использования. Рассмотрим сущность побитовых операции, их свойства и применение при работе с портами ввода/вывода.

Определим **побитовые** операции, сравнив их с родственными **логическими** операциями (*И, ИЛИ*). При вычислении логической операции на вход подаются два булевых операнда, на выходе получаем булево значение, например:

1 И 0 = 0

1 ИЛИ 0 = 1

Каждая **одиночная** побитовая операция представляет собой **серию** логических операции над парами операндов. Пример:

```
  A = 10101
& B = 00111
-----------
    00101
```

Здесь выполнено побитовое *И* – серия логических *И* для пар битов пятиразрядных двоичных чисел A и B. (Знак «&» - амперсанд обозначает побитовое *И* в языке Си). Аналогичным образом, побитовое *ИЛИ* (знак вертикальная черта «|» в Си) представляет собой серию логических *ИЛИ*:

```
  A = 110011
| B = 100111
-----------
    110111
```

Рассмотрим важные в дальнейшем свойства логических *И*, *ИЛИ*. Рассмотрим выражение x && 1 при различных значениях x:

$$x \;\&\&\; 1 = \begin{cases} 0, \text{если } x = 0 \\ 1, \text{если } x = 1 \end{cases}$$

Проще говоря, логическое *И* с единицей не меняет x:

$1°)\; x \;\&\&\; 1 = x$

Рассмотрим теперь выражение x && 0:

$$x \;\&\&\; 0 = \begin{cases} 0, \text{если } x = 0 \\ 0, \text{если } x = 1 \end{cases}$$

То есть логическое *И* с нулем всегда дает 0, независимо от x.

$2°)\; x \;\&\&\; 0 = 0$

Аналогично анализируя результат выражения при различных значениях x, получим правила для логического *ИЛИ*:

$3°)\; x \;||\; 1 = 1$

$4°)\; x \;||\; 0 = x$

(докажите это самостоятельно).

Основываясь на изложенном, научимся решать задачи:

– Задача 1. Проверка отдельного бита числа.
– Задача 2. Установить нужный бит числа в 0, остальные оставить неизменными.
– Задача 3. Установить заданный бит числа в 1, остальные не трогать.

Необходимость проверки отдельного бита (задача 1) возникает при обработке нажатий кнопки, подключенной к одной из ножек порта ввода/вывода микроконтроллера. Допустим, нас интересует кнопка, подключенная к ножке PD2 (порт D, вторая ножка). Чтобы программно определить, что кнопка нажата, надо определить состояние второго бита регистра PIND. Рассчитаем результат выражения PIND & 0b00000100 при всевозможных значениях PIND (учтем свойства 1° и 2°):

$$\begin{array}{ll} & \mathtt{PIND} = b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 \\ \& & \mathtt{0\ 0\ 0\ 0\ 0\ 1\ 0\ 0} \\ \hline & \mathtt{0\ 0\ 0\ 0\ 0}\, b_2\, \mathtt{0\ 0} \end{array}$$

Второй бит результата равен второму биту PIND, остальные биты PIND не влияют на результат. Таким образом:

при нажатой кнопке: PIND & 0b00000100 = 0b00000000
при отпущенной кнопке: PIND & 0b00000100 = 0b00000100 ≠ 0

Учитывая это, код проверки положения кнопки можно написать так:

```
int year;
if (PIND & 0b00000100)
    year = 1824;
else
    year = 1799;
```

Очевидно, в случае нажатой кнопки в переменной year год рождения гениального поэта, а при отпущенной кнопке – великого писателя.

Изменять значение одного бита, не трогая другие (задачи 2, 3) нужно, если мы хотим зажечь (или погасить) один из светодиодов, подключенный к порту ввода/вывода. Пусть нас интересует диод, подключенный к ножке PB1. Чтобы его зажечь, надо записать 0 в 1-й бит регистра PORTB, оставив неизменными. Рассмотрим фрагмент кода в языке Си:

```
PORTB = PORTB & 0b11111101;
```

Рассмотрим, что окажется в PORTB после выполнение операции:

$$\begin{array}{ll} & \mathtt{PORTB} = b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 \\ \& & \mathtt{1\ 1\ 1\ 1\ 1\ 1\ 0\ 1} \\ \hline & b_7 b_6 b_5 b_4 b_3 b_2\, \mathtt{0}\ b_0 \end{array}$$

Результат операции PORTB & 0b11111101 совпадает с исходным значением PORTB за исключением 1-го бита (снова вследствие свойств 1° и 2°).

Если мы теперь хотим погасить диод на PB1, то нам надо записать в бит b_1 единицу, как и раньше не испортив (т.е. оставив неизменными) остальные биты. Заметим, что это нельзя сделать с помощью побитового *И* (самостоятельно рассмотрите результат выражения `PORTB & 0b00000010`, которое так и тянет использовать). Рассмотрим, как это достигается побитовым *ИЛИ*:

```
PORTB = PORTB | 0b00000010;
```

Рассмотрим результат побитовой операции (учтем свойства 3° и 4°):

$$\begin{array}{rl} & \text{PORTB} = b_7 b_6 b_5 b_4 b_3 b_2 b_1 b_0 \\ | & \quad\;\; 0\;0\;0\;0\;0\;0\;1\;0 \\ \hline & b_7 b_6 b_5 b_4 b_3 b_2 1\; b_0 \end{array}$$

Результат отличается от начального значения PORTB только первым битом.

Отметим важную особенность побитовых операций: если требуется установить бит в 0, то надо использовать побитовое *И*, если надо установить бит в 1, используется побитовое *ИЛИ*.