4.5.9 O IMPORTANTÍSSIMO TRABALHO COM BITS

O trabalho com bits é fundamental para a programação de um microcontrolador. Assim, compreender como podem ser realizadas operações com bits é primordial para uma programação eficiente.

Quando se programa em C, é comum utilizar números com notação decimal, hexadecimal ou binária e seu uso depende da conveniência. No compilador AVR-GCC, essas notações são dadas por:

DECIMAL - qualquer número no formato decimal (0, 1, 2, 3, 4, 5, 6, 7, 8, 9).
HEXADECIMAL – começa com 0x seguido pelo número hexadecimal (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F).
BINÁRIO – começa com 0b seguido por zeros ou uns (0, 1), o bit menos significativo (LSB) fica a direita e o mais significativo (MSB) a esquerda.

Na tab. 4.6, é apresentado a equivalência entre as notações. No apêndice F é dada uma tabela para todos os números entre 0 e 255.

Tab. 4.6. Equivalência entre números decimais, hexadecimais e binários.

Decimal	Hexadecimal		Binário			
			Bit 3 (MSB)	Bit 2	Bit 1	Bit 0 (LSB)
0	0x0	0b	0	0	0	0
1	0x1	0b	0	0	0	1
2	0x2	0b	0	0	1	0
3	0x3	0b	0	0	1	1
4	0x4	0b	0	1	0	0
5	0x5	0b	0	1	0	1
6	0x6	0b	0	1	1	0
7	0x7	0b	0	1	1	1
8	0x8	0b	1	0	0	0
9	0x9	0b	1	0	0	1
10	0xA	0b	1	0	1	0
11	0xB	0b	1	0	1	1
12	0xC	0b	1	1	0	0
13	0xD	0b	1	1	0	1
14	0xE	0b	1	1	1	0
15	0xF	0b	1	1	1	1

Como cada número hexadecimal corresponde a 4 bits, é fácil converter um número hexadecimal para binário e vice-versa, por exemplo:

0x	**A**				**8**			
0b	1	0	1	0	1	0	0	0
0x	**F**				**0**			
0b	1	1	1	1	0	0	0	0
0x	**C**				**5**			
0b	1	1	0	0	0	1	0	1

Como os registradores de entrada e saída (memória de dados) do microcontrolador ATmega são de 8 bits, os números com resolução de 8 bits são muito utilizados. Por exemplo, quando se quer escrever em um ou mais pinos se emprega um registrador PORTx, onde cada bit corresponde a um pino de I/O do microcontrolador (PXn). Considerando-se o registrador PORTD, e que todos os pinos foram configurados para serem saídas, se PORTD = 0b10101000 ou PORTD = 0xA8, isso significa que os pinos PD3, PD5 e PD7 vão ser colocados em nível lógico alto (VCC) e os demais em nível lógico zero (0 V):

	MSB							LSB
	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PORTD	**PD7**	**PD6**	**PD5**	**PD4**	**PD3**	**PD2**	**PD1**	**PD0**
0b	1	0	1	0	1	0	0	0
0x	A				8			

Quando se quer escrever em um ou mais bits conhecidos, a notação binária é a mais adequada. Quando o número é conhecido, a notação hexadecimal é mais rápida de ser digitada por ser mais compacta.

MASCÁRA DE BITS E MACROS

Como os pinos do ATmega são tratados como bits individuais, o emprego de operações lógicas bit a bit são usuais e a linguagem C oferece suporte total a essas operações (tab. 4.4):

| OU lógico bit a bit (usado para ativar bits , colocar em 1)

& E lógico bit a bit (usado para limpar bits, colocar em 0)

^ OU EXCLUSIVO bit a bit (usado para trocar o estado dos bits)

~ complemento de 1 (1 vira 0, 0 vira 1)

Nr **>>** x O número é deslocado x bits para a direita

Nr **<<** x O número é deslocado x bits para a esquerda

Como o AVR-GCC não tem acesso direto a bits como o *assembly* (instruções SBI e CBI, detalhes no apêndice A), é necessário utilizar máscara de bits em operações com as variáveis. Isso é empregado quando se deseja alterar um ou mais bits de um registrador do ATmega ou de uma variável do C sem mudar os demais bits.

Exemplos para a **Variável_8bits = 0b10101000:**

1. Ativar o bit 0:

```
Variavel_8bits = Variavel_8Bits | 0b00000001;

                   0b1010100 0
                 | 0b0000000 1  (máscara)
Variavel_8bits   = 0b1010100 1
```

2. Limpar os bits 3 e 5:

```
Variavel_8bits = Variavel_8Bits & 0b11010111;

                   0b10 1 0 1000
                 & 0b11 0 1 0 111  (máscara)
Variavel_8bits   = 0b10 0 0 0 000
```

Para o trabalho exclusivo com bits individuais, declaram-se macros para que o compilador as utilize como se fossem funções. Essas macros trabalham com máscara e rotação de bits. Assim, o programador necessita criá-las e colocá-las nos seus programas.

As macros usuais são:

- **Ativação de bit, colocar em 1:**

```
#define set_bit(Y,bit_x) (Y|=(1<<bit_x))
```

onde `Y |= (1<<bit_x)` ou `Y = Y | (1<<bit_x).`

O programa ao encontrar a macro **set_bit(Y, bit_x)** irá substituí-la por **Y|=(1<<bit_x)**. Por exemplo, supondo que se deseje colocar o bit 5 do PORTD em 1 (pino PD5), a operação realizada é dada por **set_bit(PORTD,5)**. Dessa forma, o **Y** se transforma no PORTD e o **bit_x** é o número 5, resultando em:

PORTD = PORTD | (1<<5) ,

```
          0bxxxxxxxx     (PORTD, x pode ser 0 ou 1)
        | 0b00100000     (1<<5 é a máscara)
PORTD = 0bxx1xxxxx       (o bit 5 com certeza será 1)
```

A macro lê o valor PORTD e faz um OU bit a bit com o valor (1<<5) , 1 binário deslocado 5 bits para a esquerda, que é igual a 0b00100000, ou seja, o bit 5 é colocado em 1. Dessa forma independente do valor do PORTD, somente o seu bit 5 será colocado em 1 (OU lógico de x com 1 é igual a 1).

- **Limpeza de bit, colocar em 0:**

```
#define clr_bit(Y,bit_x) (Y&=~(1<<bit_x))
```

onde `Y &= ~(1<<bit_x)` ou `Y = Y & (~ (1<<bit_x)).`

O programa ao encontrar a macro **clr_bit(Y, bit_x)** irá substituí-la por **Y&=~(1<<bit_x)**. Por exemplo, supondo que se deseje zerar o bit 2 do PORTB (pino PB2), a operação realizada é dada por **clr_bit(PORTB,2)**. Assim, o **Y** se transforma no PORTB e o **bit_x** é o número 2, resultando em:

PORTB = PORTB & (~ (1<<2)) ,

```
          0bxxxxxxxx      (PORTB, x pode ser 0 ou 1)
        & 0b11111011      (~(1<<2) é a máscara)
PORTB = 0bxxxxx0xx        (o bit 2 com certeza será 0)
```

A macro lê o valor PORTB e faz um E bit a bit com o valor ~(1<<2) , 1 binário deslocado 2 bits para a esquerda com o valor invertido dos seus bits, o que é igual a 0b11111011, ou seja, o bit 2 é colocado em 0. Dessa forma, independente do valor do PORTB, somente o seu bit 2 será colocado em 0 (E lógico de x com 0 é igual a 0).

- **Troca o estado lógico de um bit, 0 para 1 ou 1 para 0:**

```
#define cpl_bit(Y,bit_x) (Y^=(1<<bit_x))
```

onde `Y ^ = (1<<bit_x)` ou `Y = Y ^ (1<<bit_x).`

O programa ao encontrar a macro **cpl_bit(Y, bit_x)** irá substituí-la por **Y^=(1<<bit_x)**. Por exemplo, supondo que se deseje alterar o estado do bit 3 do PORTC (pino PC3), a operação realizada é dada por **cpl_bit(PORTC,3)**. Logo, o **Y** se transforma no PORTC e o **bit_x** é o número 3, resultando em:

PORTC = PORTC ^ (1<<3) ,

```
          0bxxxx1xxx      (PORTC, x pode ser 0 ou 1)
        ^ 0b00001000      (1<<3 é a máscara)
PORTC = 0bxxxx0xxx        (o bit 3 será 0 se o bit a ser
                           complementado for 1 e 1 se ele for 0).
```

A macro lê o valor PORTC e faz um OU EXCLUSIVO bit a bit com o valor (1<<3), 1 binário deslocado 3 bits para a esquerda, que é igual a 0b00001000, ou seja o bit 3 é colocado em 1. Dessa forma, independente do valor do PORTC, somente o seu bit 3 terá o seu estado lógico trocado (OU EXCLUSIVO de 0 com 1 é igual a 1, de 1 com 1 é igual a 0).

- **Leitura de um bit:**

```
#define tst_bit(Y,bit_x) (Y&(1<<bit_x))
```

O programa ao encontrar a macro **tst_bit(Y, bit_x)** irá substituí-la por **Y & (1<<bit_x)**. Por exemplo, supondo que se deseje ler o estado do bit 4 do PORTD (pino PD4), a operação realizada é dada por **tst_bit(PIND,4)**. Portanto, o **Y** se transforma no PIND e o **bit_x** é o número 4 (a leitura dos pinos do PORTD é feita no registrador PIND, ver o capítulo 5), resultando em:

PIND & (1<<4) ,

```
          0bxxxTxxxx     (PIND, x pode ser 0 ou 1)
        & 0b00010000     (1<<4 é a máscara)
resultado = 0b000T0000   (o bit 4 terá o valor T, que será 0 ou 1).
```

A macro efetua apenas um operação E bit a bit de PIND com valor (1<<4), 1 binário deslocado 4 bits para a esquerda, que é igual a 0b00010000, ou seja o bit 4 é colocado em 1. Assim, somente o bit 3 do PIND mantém o seu valor. (E lógico de T com 1 é igual a T).

Na macro **tst_bit** não existe uma variável que recebe o valor da operação efetuada. Isso acontece porque a macro é utilizada com comandos de decisão tais como o **`if`** e o **`while`**. Por exemplo:

```
if(tst_bit(PIND,4))
{
    Ação a ser executada se o pino PD4 estiver em 1 lógico.
}
```

A macro retorna 0 para o comando de decisão se o bit testado for 0 e um número diferente se ele estiver em 1. Como os comandos de decisão em C executam a condição entre as chaves ({ }) se a sua condição de teste for verdadeira, ela será verdade para valores diferentes de zero.