Software Básico

Manipulação de Bits

Prof Bruno Silvestre

Operações Lógicas em C

- Operadores lógicos: or (||), and (&&) e not (!)
  - Tratam qualquer argumento diferente de zero como **true** e igual a zero como **false**
  - Operação resulta em 1 (true) ou 0 (false)

Expressão	Resultado
`!0x41`	`0x00`
`!0x00`	`0x01`
`0xaa && 0x55`	`0x01`
`0xaa || 0x55`	`0x01`

Manipulação de Bits em C

- Várias linguagem oferecem operadores para que possamos realizar operações como AND ou OR em nível de bits
- Em C, temos os operador

Operação	Operador em C
AND	&
OR	|
NOT	~
XOR	^
Left Shift	<<
Right Shift	>>

Operador AND (&)

- Operador binário: v = a & b;

  v = 0x6999 & 0xCD3A ;

Tabela Verdade

&	0	1
0	0	0
1	0	1

```
01101001 10011001
11001101 00111010  &
01001001 00011000
```

Operador AND (&)

- Operador binário: v = a & b;

  v = 0xA59B & 0x48D3 ;

Tabela Verdade

&	0	1
0	0	0
1	0	1

```
???????? ????????
???????? ???????? &
???????? ????????
```

Operador OR (|)

- Operador binário: v = a | b;

  v = 0x6999 | 0xCD3A;

Tabela Verdade

|	0	1
0	0	1
1	1	1

```
01101001 10011001
11001101 00111010 |
11101101 10111011
```

Operador OR (|)

- Operador binário: v = a | b;

  v = 0x72F9 | 0xE32B;

Tabela Verdade

|	0	1
0	0	1
1	1	1

```
????????  ????????
????????  ????????   |
????????  ????????
```

Operador XOR (^)

- Operador binário: v = a ^ b;

  v = 0x6999 ^ 0xCD3A ;

Tabela Verdade

^	0	1
0	0	1
1	1	0

```
01101001  10011001
11001101  00111010   ^
10100100  10100011
```

Operador XOR (^)

- Operador binário: v = a ^ b;

  v = 0x9BC6 ^ 0xBABA ;

Tabela Verdade

^	0	1
0	0	1
1	1	0

```
???????? ????????
???????? ???????? ^
???????? ????????
```

Operador NOT (~)

- Operador unário: v = ~a ;

  v = ~ 0xCD3A ;

```
11001101 00111010   ~
00110010 11000101
```

Operador NOT (~)

- Operador unário: v = ~a ;

  v = ~ 0xCAFE ;

```
????????  ????????   ~
????????  ????????
```

Operador Left Shift (<<)

- Desloca os bits de “a”, “b” posições para a esquerda
- Descarta os “b” bits mais à esquerda
- Completa os “b” bits mais à direita com zeros
- Operador binário: v = a << b;

  v = 0xCD3A << 1;

```
1100110100111010
               1     <<
1001101001110100
```

Operador Left Shift (<<)

- Desloca os bits de “a”, “b” posições para a esquerda
- Descarta os “b” bits mais à esquerda
- Completa os “b” bits mais à direita com zeros
- Operador binário: v = 0x52F2 << 3

~~010~~10010 11110010
3 <<
10010111 10010000

Operador Left Shift (<<)

- Desloca os bits de “a”, “b” posições para a esquerda
- Descarta os “b” bits mais à esquerda
- Completa os “b” bits mais à direita com zeros
- Operador binário: v = 0x4D2F << 5

???????? ????????
? <<
???????? ????????

Operador Right Shift (>>)

- Pode possuir dois sentidos
  - Shift aritmético
    - Lado esquerdo tem o bit de sinal propagado
  - Shift lógico
    - Lado esquerdo é preenchido com zero

Right Shift Aritmético

- Em C, o shift para direita aritmético só é aplicado aos tipos **com** sinal
  - int8_t, int16_t, int32_t, int64_t
  - long, int, short, char
  - signed long, signed int, signed short, signed char

```
short a = 0xCD3A;
short v = a  >> 5 ;
```

Right Shift Aritmético

- Em C, o shift para direita aritmético só é aplicado aos tipos **com** sinal
- Os “b” bits da direita são descartados

```
short a = 0xCD3A;
short v = a  >> 5 ;
```

(a) 11001101 001~~11010~~

5 >>

(v) ~~_____~~110 01101001

Right Shift Aritmético

- Em C, o shift para direita aritmético só é aplicado aos tipos **com** sinal
- Os “b” bits da direita são descartados
- O bit do sinal (mais à esquerda) é replicado “b” vezes

```
short a = 0xCD3A;
short v = a  >> 5 ;
```

(a) 11001101 001~~11010~~

5 >>

(v) 11111110 01101001

Right Shift Aritmético

- Em C, o shift para direita aritmético só é aplicado aos tipos **com** sinal
- Os “b” bits da direita são descartados
- O bit do sinal (mais à esquerda) é replicado “b” vezes

```
short a = 0x4D3A;
short v = a  >> 3 ;
```

(a) 01001101 00111~~010~~

3 >>

(v) 000010 0110100111

Right Shift Lógico

- Em C, o shift para direita lógico só é aplicado aos tipos **sem** sinal
  - uint8_t, uint16_t, uint32_t, uint64_t
  - unsigned long, unsigned int, unsigned short, unsigned char

```
unsigned short a = 0xCD3A;
unsigned short v = a  >> 5 ;
```

Right Shift Lógico

- Em C, o shift para direita lógico só é aplicado aos tipos **sem** sinal
- Os “b” bits da direita são descartados

```
uint16_t a = 0xCD3A;
uint16_t v = a  >> 5 ;
```

(a) 11001101 001~~11010~~

5 >>

(v) ~~_____~~110 01101001

Right Shift Lógico

- Em C, o shift para direita lógico só é aplicado aos tipos **sem** sinal
- Os “b” bits da direita são descartados
- Os “b” bits da esquerda são preenchidos com 0 (zero)

```
uint16_t a = 0xCD3A;
uint16_t v = a  >> 5 ;
```

(a) 11001101 001~~11010~~

5 >>

(v) 00000110 01101001

Right Shift Lógico

- Em C, o shift para direita lógico só é aplicado aos tipos **sem** sinal
- Os “b” bits da direita são descartados
- Os “b” bits da esquerda são preenchidos com 0 (zero)

```
uint16_t a = 0x4D3A;
uint16_t v = a  >> 3 ;
```

(a) 01001101 00111~~010~~

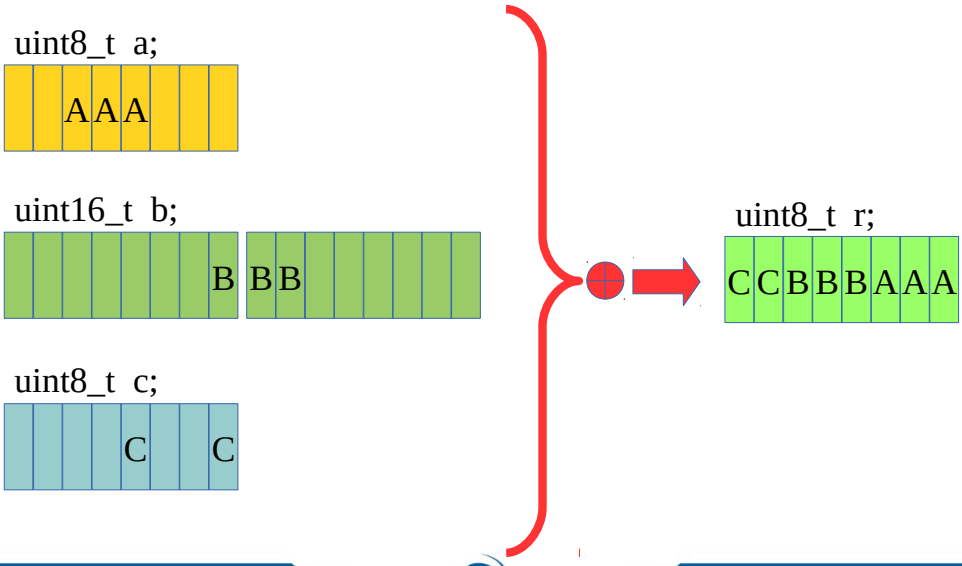
3 >>

(v) ~~000~~010 0110100111

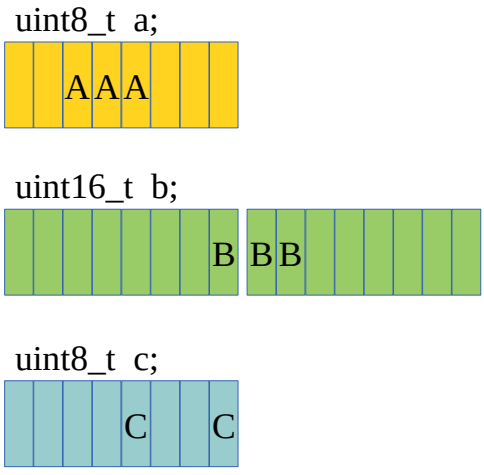
Exemplo de Uso: Cálculo de CRC

```
/* CRC function */
unsigned short cc2420_icrc1(unsigned short crc, unsigned char onech)
{
    int i;
    unsigned short ans = (crc ^ onech << 8);
    for (i = 0; i < 8; i++) {
        if (ans & 0x8000) {
            ans = (ans << 1) ^ 4129;
        }
        else {
            ans = ans << 1;
        }
    }
    return ans;
}
```

Como montar o valor de “r”?



Como montar o valor de “r”?




```
uint8_t tmp1 = a >> 3;
uint8_t tmp2 = b >> 3;
uint8_t tmp3 =
  (c & 1) | ((c & 8) >> 2);
tmp3 = tmp3 << 6;

uint8_t r =
   (tmp1 & 0x07) |
   (tmp2 & 0x38) |
   (tmp3 & 0xC0);
```