C 言語における論理演算

S.Matoike

2023 年 1 月 21 日

ソースコード 1　C 言語で論理演算

```c
#include<stdio.h>
#define CHAR_BIT 8
static char frm[15] = "␣=␣(␣0x%08X␣)";

void printb(unsigned int v) {
  unsigned int mask = (int)1 << (sizeof(v) * CHAR_BIT - 1);
  int n=0;
  do{
     if(n++%4==0)   putchar('␣');
     putchar(mask & v ? '1' : '0');
  }while(mask >>= 1);
}

void binhex(unsigned int v1) {
    printb(v1);     printf(frm, v1);    printf("\n");
}

void binhex1(char* s1,unsigned v1) {
    printf("%s", s1);   binhex( v1 );
}

void binhex2(char* s1,unsigned v1,char* s2,unsigned v2) {
    binhex1( s1, v1 );   binhex1( s2, v2 );
}

void binhex3(char* s1,unsigned v1,char* s2,unsigned v2,char* s3,unsigned v3) {
    binhex2( s1, v1, s2, v2 );  binhex1( s3, v3 );
}

void main(void) {
    sprintf( frm, "%s%c%s", "␣=␣(␣0x%0", CHAR_BIT+'0' ,"X␣)" );
    unsigned int a, b, c;
    a = 0xA5A5A5A5;     b = 0x000000FF;
    /* NOT */
    printf("===␣NOT:(~a)␣===\n");
    binhex2( "␣a␣=", a, "~a␣=", ~a);                          printf("\n");
    /* AND */
    printf("===␣AND:(a␣&␣b)␣===\n");
    binhex3("a␣␣␣␣␣=", a, "b␣␣␣␣␣=", b, "a␣&␣b␣=", a & b);   printf("\n");
    /* OR */
    printf("===␣OR:(a␣|␣b)␣===\n");
    binhex3("a␣␣␣␣␣=", a, "b␣␣␣␣␣=", b, "a␣|␣b␣=", a | b);   printf("\n");
    /* XOR */
    printf("===␣XOR:(a␣^␣b␣)␣===\n");
    b = 0xFFFFFFFF;
    binhex3("a␣␣␣␣␣=", a, "b␣␣␣␣␣=", b, "a␣^␣b␣=", a ^ b);   printf("\n");
    b = 0x00000000;
    binhex3("a␣␣␣␣␣=", a, "b␣␣␣␣␣=", b, "a␣^␣b␣=", a ^ b);   printf("\n");
    /* 応用問題 */
    printf("===␣c␣=␣((␣a␣&␣3)␣<<␣8␣)␣|␣(b␣&␣255␣)␣===\n");
    printf("␣a␣を入力：␣");  scanf("%u", &a);
    printf("␣b␣を入力：␣");  scanf("%u", &b);                 printf("\n");
    printf("␣a␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣=");   binhex( a );
    printf("␣␣␣␣␣3␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣=");   binhex( 3 );
    printf("␣a␣&␣3␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣=");   binhex( a&3 );
    printf("(a␣&␣3)␣<<␣8␣␣␣␣␣␣␣␣␣=");   binhex( (a&3)<<8 ); printf("\n");
    printf("␣b␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣=");   binhex( b );
    printf("␣␣␣␣␣255␣␣␣␣␣␣␣␣␣␣␣␣␣=");   binhex( 255 );
    printf("␣b␣&␣255␣␣␣␣␣␣␣␣␣␣␣␣␣=");   binhex( b&255 );    printf("\n");
    c = ( (a & 3) << 8 ) | ( b & 255 );
    printf("␣c␣=␣(a&3)<<8+(b&255)␣=");   binhex( c );
}
```

ソースコード 2　プログラムの実行結果

```
 === NOT:(~a) ===
  a = 1010 0101 1010 0101 1010 0101 1010 0101 = ( 0xA5A5A5A5 )
 ~a = 0101 1010 0101 1010 0101 1010 0101 1010 = ( 0x5A5A5A5A )

 === AND:(a & b) ===
 a     = 1010 0101 1010 0101 1010 0101 1010 0101 = ( 0xA5A5A5A5 )
 b     = 0000 0000 0000 0000 0000 0000 1111 1111 = ( 0x000000FF )
 a & b = 0000 0000 0000 0000 0000 0000 1010 0101 = ( 0x000000A5 )

 === OR:(a | b) ===
 a     = 1010 0101 1010 0101 1010 0101 1010 0101 = ( 0xA5A5A5A5 )
 b     = 0000 0000 0000 0000 0000 0000 1111 1111 = ( 0x000000FF )
 a | b = 1010 0101 1010 0101 1010 0101 1111 1111 = ( 0xA5A5A5FF )

 === XOR:(a ^ b ) ===
 a     = 1010 0101 1010 0101 1010 0101 1010 0101 = ( 0xA5A5A5A5 )
 b     = 1111 1111 1111 1111 1111 1111 1111 1111 = ( 0xFFFFFFFF )
 a ^ b = 0101 1010 0101 1010 0101 1010 0101 1010 = ( 0x5A5A5A5A )

 a     = 1010 0101 1010 0101 1010 0101 1010 0101 = ( 0xA5A5A5A5 )
 b     = 0000 0000 0000 0000 0000 0000 0000 0000 = ( 0x00000000 )
 a ^ b = 1010 0101 1010 0101 1010 0101 1010 0101 = ( 0xA5A5A5A5 )

 === c = (( a & 3) << 8 ) | (b & 255 ) ===
 a を入力して：   90
 b を入力して： 1355

a                 = 0000 0000 0000 0000 0000 0000 0101 1010 = ( 0x0000005A )
     3            = 0000 0000 0000 0000 0000 0000 0000 0011 = ( 0x00000003 )
a &  3            = 0000 0000 0000 0000 0000 0000 0000 0010 = ( 0x00000002 )
(a & 3) << 8      = 0000 0000 0000 0000 0000 0010 0000 0000 = ( 0x00000200 )

b                 = 0000 0000 0000 0000 0000 0101 0100 1011 = ( 0x0000054B )
    255           = 0000 0000 0000 0000 0000 0000 1111 1111 = ( 0x000000FF )
b & 255           = 0000 0000 0000 0000 0000 0000 0100 1011 = ( 0x0000004B )

c=(a&3)<<8+(b&255)= 0000 0000 0000 0000 0000 0010 0100 1011 = ( 0x0000024B )
```

- 以下のことについて確認します。
  - 単項演算子の NOT と、二項演算子の AND,OR,XOR
  - AND 演算で、指定した bit のマスキングによる切り出し操作
  - OR 演算で、指定した bit を (無理やり) 1 にしてしまう操作
  - 0xFF$\cdots$FF との XOR 演算で、bit 反転したデータ (これは 1 の補数) を作れること
  - 0x00$\cdots$00 との XOR 演算では、何も変わらないこと
  - 同じデータ同士の XOR 演算では、ゼロで埋められたデータになること
- 応用問題では、a の下 2bit と b の 8bit をつないで、10bit のデータ c を作っています。
  - まず最初に、変数 a の下 2bit を切り出すために、次のマスキング操作を施します
    下 2bit だけが 1 であるような数値 0x00$\cdots$03 との論理積をとります
  - 次に、変数 b の 8bit を納める場所を確保するために、a を左に 8bit シフトしています
  - 最後に、その値と b (これもマスキングによって下 8bit を切り出している) との論理和をとって合計 10bit のデータを作り、それを変数 c に代入しています

この様な操作は、10bit の AD 変換器の出力を受け取る様な場面でしばしば現れます。

- 【演習】
  (1) 12bit の AD 変換器が出力するデータを受け取る場合には、a の下 4bit と b の 8bit をつないで、12bit のデータ c を作る必要が生じます。さて、どんな操作になるかな。
  (2) a の下 6bit と b の下 4bit をつないで、10bit のデータ c を作る場合はどうですか。
  (こんな必要を生じる場面が実際にあるかどうか知らないけど、練習だから)
- 応用問題の演算

```
c = ( (a & 3) << 8 ) | (b & 255)
```

1bit 左にシフトすると、その値は 2 倍になる (右に 1bit シフトすると 1/2)。8bit 左にシフトしているので、変数 a の値は $2^8 = 256$ 倍になりますから、次の様に書いても Ok です。

```
c = ( (a & 3) * 256 ) + (b & 255)
```

- 関数 main() の冒頭

```
sprintf( frm, "%s%c%s", " = ( 0x%0", CHAR_BIT+'0' ,"X )" );
```

この部分では、関数 binhex() の中の printf() で使っている書式文字列 frm を作っています。

```
CHAR_BIT + '0'
```

ここでは、整数の CHAR_BIT を ASCII 文字のコードに変換しています。
数字 ('0','1','2','3','4','5','6','7','8','9') の各文字は、文字コード表の上でこの順番に並んでいますから、数字'0' の文字コードからのオフセットによって、それぞれの文字コードを表す事ができます。(数字 '8' は 数字 '0' のコードから数値 8 だけ離れた所にある)

- 整数変数を 2 進数の形で出力するためには、printb() のような関数を作らなければなりません。最初に左端の bit(MSB) だけを 1 にした mask を用意して、その mask の中の 1 の位置を 1bit ずつ右にシフトしながら後判定反復しています。
(1 の bit が右端から掃き出されたら、mask は 0 で「偽」。反復は終了する)

```
do {
      ・・・・・・・
} while ( mask >>= 1 );   /* mask>>=1 は mask/=2 でも Ok */
```

マスキング操作 (mask と v の論理積) の結果によって、0 と 1 の判定をしています。

```
putchar ( mask & v ? '1' : '0' );
```

この部分を馴染みのある表現で書き直すと、次のようになります。

```
if ( (mask & v) != 0 ) putchar('1');
else                   putchar('0');
```

4bit 毎（カウンタ n を 4 で割った余りが 0 なら）にスペースを 1 つ出力しています。
また、if 文の条件判定の「後」で、 n の値をインクリメント $n++$ しています。

```
if ( n++%4==0 ) putchar ( ' ␣' );
```