Laboratório 03

Prof Bruno Silvestre

1. Considere uma variável em C, de nome “x”, do tipo “unsigned int”. Escreva expressões que, utilizando operações de manipulação de bits, obtenham os seguintes valores:
   - Um valor de 32 bits com o byte menos significativo igual ao byte correspondente de “x” e todos os outros bits iguais a 0;
   - Um valor de 32 bits com o byte mais significativo com todos os bits iguais a ‘1’ e os outros bytes com o mesmo valor dos bytes correspondentes de “x”.

Como exemplo, se o valor de “x” for 0x87654321, o resultado da avaliação da primeira expressão deverá ser 0x00000021 e o da segunda deverá ser 0xFF654321

Complete o programa abaixo com as suas expressões, compile-o e o execute para verificar se elas estão corretas:

```
#include <stdio.h>

int main(void) {
  unsigned int x = 0x87654321;
  unsigned int y, z;

  /* o byte menos significativo de x e os outros bits em 0 */
  y = ____________________;

  /* o byte mais significativo com todos os bits em '1'
     e os outros bytes com o mesmo valor dos bytes de x */
  z = ____________________;

  printf("%08x %08x\n", y, z);

  return 0;
}
```

2. Escreva uma função que verifique se o número de bits ‘1’ de um inteiro sem sinal é ímpar:

```
int odd_ones(unsigned int x);
```

Obs: sua função NÃO PODE usar o operador de módulo % para testar se um número é par ou ímpar!

Sua função deve retornar 0 (zero) se o número de bits ‘1’ for par, e um valor diferente de 0 (zero) se o número de bits ‘1’ for ímpar.

Teste sua função *odd_ones*() com diferentes valores de entrada, não esquecendo de verificar se os resultados obtidos estão corretos. Por exemplo, teste os valores 0x01010101 (número par de bits 1) e 0x01030101 (número ímpar de bits 1):

```
#include <stdio.h>

int odd_ones(unsigned int x) {
  /* escreva seu código aqui */
}

int main() {
  printf("%x tem numero %s de bits\n", 0x01010101,
         odd_ones(0x01010101) ? "impar" : "par");

  printf("%x tem numero %s de bits\n", 0x01030101,
         odd_ones(0x01030101) ? "impar" : "par");

  return 0;
}
```

3. Algoritmos de criptografia usam tipicamente diversas operações de manipulação de bits. Algumas dessas operações não são oferecidas diretamente por C, mas podem ser implementadas a partir de suas operações básicas (and, or, shift). Faça as seguintes atividades:

a) Implemente em C a operação “switch_byte”, que troca as duas “metades” de um byte. Por exemplo, se sua função receber como parâmetro o valor 0xAB ela deverá retornar 0xBA. O protótipo de sua função deverá ser:

```
unsigned char switch_byte(unsigned char x);
```

Escreva uma função main() e teste sua função “switch_byte”.

b) Implemente agora a operação “rotate_left” (também chamada de “circular left shift”).

```
unsigned char rotate_left(unsigned char x, int n);
```

Assim como um “left shift” (<<), ela desloca seu operando “n” bits para a esquerda. Porém, ao invés de preencher as posições livres à direita com zeros, ela as preenche com os “n” bits removidos pelo deslocamento.

Considere, por exemplo, o valor 0x61 (0110 0001):

- Um “rotate left” de 1 bit resulta no valor 0xc2 (1100 0010)
- Um “rotate left” de 2 bits resulta no valor 0x85 (1000 0101)
- Um “rotate left” de 7 bits resulta no valor 0xb0 (1011 0000)

Incorpore à sua função main() testes para a função “rotate_left”, experimentando alguns valores diferentes para o deslocamento.

Obs: Você pode assumir que o valor do deslocamento será sempre um valor inteiro maior que 0 e menor do que 8.