Plancha 1:
C y sistemas de numeración posicional

2019 – Arquitectura del Computador

Licenciatura en Ciencias de la Computación

Entrega: viernes 13 de setiembre

1. Introducción

Esta plancha trata los sistemas de representación de números enteros en el lenguaje de programación C y los operadores de bits para manipular estos números a bajo nivel.

2. Procedimiento

Resuelva cada ejercicio en computadora. Cree un subdirectorio dedicado para cada ejercicio, que contenga todos los archivos del mismo. Para todo ejercicio que pida escribir código, genere un programa completo, con su función `main` correspondiente; evite dejar fragmentos sueltos de programas.

Asegúrese de que todos los programas que escriba compilen correctamente con `gcc`. Se recomienda además pasar a este las opciones `-Wall` y `-Wextra` para habilitar advertencias sobre construcciones cuestionables en el código.

Una vez terminada la plancha, suba una entrega al repositorio Subversion de la materia, creando una etiqueta en el subdirectorio correspondiente a su grupo:
`https://svn.dcc.fceia.unr.edu.ar/svn-no-anon/lcc/R-222/Alumnos/2019/`

3. Ejercicios

1) A continuación se presentan ciertos números enteros expresados en binario y a su derecha, expresiones en lenguaje C incompletas. Complete estas expresiones de forma que la igualdad sea cierta. Utilice operadores de bits, operadores enteros y constantes de enteros literales según considere necesario.

```
10000000 00000000 00000000 00000000  ==  ... << ...

10000000 00000000 10000000 00000000  ==  (1 << ...) | (1 << ...)

11111111 11111111 11111111 00000000  ==  -1 & ...

10101010 00000000 00000000 10101010  ==  0xAA ... (0xAA << ...)

00000000 00000000 00000101 00000000  ==  5 ... 8

11111111 11111111 11111110 11111111  ==  -1 & (... (1 << 8))
```

2) Implemente una función:

```
int is_one(long n, int b);
```

que indique si el bit `b` del entero `n` es 1 o 0.

3) Implemente una función `printbin`:

```
void printbin(unsigned n);
```

que tome un entero de 64 bits y lo imprima en binario.

4) Implemente una función que tome tres parámetros a, b y c y que rote los valores de las variables de manera que al finalizar la función el valor de a se encuentre en b, el valor de b en c y el de c en a. Evite utilizar variables auxiliares.

5) Escriba un programa que tome la entrada estándar, la codifique e imprima el resultado en salida estándar. La codificación deberá ser hecha carácter a carácter utilizando el operador *XOR* y un código que se pase al programa como argumento de línea de comando.

El código adicional para el operador *XOR* se debe pasar como argumento de línea de comandos al programa. Es decir, suponiendo que el ejecutable se llame `prog`, la línea de comando para ejecutar el programa tendría el formato:

```
$ ./prog <código>
```

Por ejemplo, se podría hacer:

```
$ ./prog 12
$ ./prog 4321
```

¿Qué modificaciones se tendrían que hacer al programa para que decodifique? ¿Se gana algo codificando más de una vez?

Pruebe el programa codificando el código fuente del programa y utilizando, por ejemplo, el código -98. Si el archivo fuente se llama `prog.c`, esto se podría hacer con la siguiente línea de comando:

```
$ ./prog -98 <prog.c
```

6) *Algoritmo del campesino ruso.* La multiplicación de enteros positivos puede implementarse con sumas, el operador textitAND y desplazamientos de bits usando las siguientes identidades:

$$a.b = \begin{cases} 0 & \text{si } b = 0 \\ a & \text{si } b = 1 \\ 2a.k & \text{si } b = 2k \\ 2a.k + a & \text{si } b = 2k + 1 \end{cases}$$

Úselas para implementar una función:

```
unsigned mult(unsigned a, unsigned b);
```

7) Muchas arquitecturas de CPU restringen los enteros a un máximo de 64 bits. ¿Qué sucede si ese rango no nos alcanza? Una solución es extender el rango utilizando más de un entero (en este caso enteros de 16 bits) para representar un valor. Así podemos pensar que:

```
typedef struct {
    unsigned short n[16];
} nro;
```

representa el valor:

$$\begin{aligned} N \;=\; & nro.n[0] + \\ & nro.n[1] * 2^{sizeof(short)*8} + \\ & nro.n[2] * 2^{2*sizeof(short)*8} + \\ & ... + \\ & nro.n[15] * 2^{15*sizeof(short)*8} \end{aligned}$$

Podemos pensar en la estructura `nro` como un entero de 256 bits. Lamentablemente la arquitectura no soporta operaciones entre valores de este tipo, por lo cual debemos realizarlas en software.

a) Implemente funciones que comparen con 0 y con 1 y determinen paridad para valores de este tipo.

b) Realice funciones que corran a izquierda y derecha los valores del tipo `nro`.

c) Implemente la suma de valores del tipo `nro`.

Nota: en el repositorio Subversion de la materia hay una función para imprimir valores de este tipo. Esta función utiliza la biblioteca GMP (*GNU Multiple Precision Arithmetic Library*), por lo cual deberá compilar el código agregando la opción `-lgmp`. Puede encontrar la función en el archivo `código/enteros_grandes/gmp1.c`:
`https://svn.dcc.fceia.unr.edu.ar/svn/lcc/R-222/Public/cdigo/enteros_grandes/gmp1.c`

8) Implemente el algoritmo del campesino ruso para los números anteriores.