Plancha 1
C y sistemas de numeración posicional

Arquitectura del Computador
Licenciatura en Ciencias de la Computación

5 de septiembre de 2023

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.

3. Ejercicios con números enteros

1) A continuación se presentan ciertos números enteros expresados en binario utilizando 32 bits 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.

a) `10000000 00000000 00000000 00000000 == ... << ...`

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

c) `11111111 11111111 11111111 00000000 == -1 & ...`

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

e) `00000000 00000000 00000101 00000000 == 5 ... 8`

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

g) `11111111 11111111 11111111 11111111 == 0 ... 1`

h) `00000000 00000000 00000000 00000000 == 0x80000000 + ...........`

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 long n);
```

que tome un entero de 32 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. Evitar utilizar variables auxiliares. Ayuda: Tener en cuenta las propiedades del operador `XOR`.

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` también 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> <cadena a codificar>
```

Por ejemplo, se podría hacer:

```
$ ./prog 12 Mensaje
```

para codificar la cadena "Mensaje" con el código 12. Pruebe el programa codificando con diferentes códigos, por ejemplo, utilizando el código -98.

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

6) *Algoritmo del campesino ruso.* La multiplicación de enteros positivos puede implementarse con sumas, el operador `AND` 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.

4. Ejercicios con números en punto flotante

9) Haga dos funciones o macros de C para extraer la fracción y el exponente de un `float` sin usar variables auxiliares.

Sugerencia: utilice corrimientos de bits y máscaras. Luego use los tipos definidos en la cabecera `ieee754.h` para corroborar.

10) El siguiente programa muestra algunas cualidades de *NaN* (*Not A Number*) y la función `isnan` de C, que indica si un flotante es *NaN*.

```
#include <stdio.h>
#include <math.h>

int main(void)
{
float g = 0.0;
float f = 0.0 / g;
printf("f: %f\n", f);
// ADVERTENCIA: 'NAN' es una extensión de GCC.
if (f == NAN) {
        printf("Es NAN\n");
}
if (isnan(f)) {
        printf("isNaN dice que sí\n");
```

```
}
        return 0;
}
```

d) El programa muestra que comparar con `NAN` retorna siempre falso y para saber si una operación dio *NaN* se puede usar `isnan`. Utilizando las funciones del ejercicio anterior, implemente una función `myisnan` que haga lo mismo que la función `isnan` de C.

e) Implemente otra función, `myisnan2`, que haga lo mismo pero utilizando solo una comparación y sin operaciones de bits.

f) ¿Ocurre lo mismo con $+\infty$?

g) ¿Qué pasa si se suma un valor a $+\infty$?

11) Convierta a `double` y `float` norma *IEEE 754* el número: $N = 6.225$. Realice el cálculo de manera explícita y luego corrobore el resultado mediante un programa que aproveche las herramientas provistas en el ejercicio 9. Analizar en cada caso si se ha cometido error de representación y en caso afirmativo la magnitud del mismo.

12) Dados los números $N_1 = (100000)_{10}$, $N_2 = (0.2)_{10}$ y $N_3 = (0.1)_{10}$:

a) Realizar la operación $N_1 \otimes (N_2 \oplus N_3)$ en simple precisión *IEEE 754*.

b) Realizar la operación $(N_1 \otimes N_2) \oplus (N_1 \otimes N_3)$ en simple precisión *IEEE 754*.

c) Repetir para doble precisión *IEEE 754*.

d) Comparar los resultados.

13) Realice el procedimiento de suma (simple precisión) del número 1.75×2^{-79} con el número `0x19d00000` expresado en IEEE 754 simple precisión.