Plancha 1
Representación Computacional de Datos

Arquitectura del Computador
Licenciatura en Ciencias de la Computación

1. Introducción

El objetivo de esta serie de ejercicios es familiarizarse con la representación de datos orientada a la computación, sin depender de la implementación computacional propiamente dicha. Muchos de los ejercicios están diseñados para ser resueltos con papel y lápiz, aunque posteriormente se pueden utilizar herramientas informáticas para verificar los resultados.

2. Ejercicios

1) Utilizando el sistema de numeración posicional denominado **magnitud y signo**:

$$(-1)^s(a_n a_{n-1} \cdots a_1 a_0 . a_{-1} a_{-2} \cdots)_2,$$

determinar la representación binaria de los siguientes números:

a) 29

b) 0.625

c) 5.75

d) -138

e) -15.125

f) 0.1

Analizar en cada caso cuántos dígitos son necesarios para poder representar cada uno de los números. Tener en cuenta el bit de signo s. ¿Qué sucede con el número 0.1?

2) Convertir los siguientes números decimales a binario utilizando la representación en complemento a dos con seis bits:

a) -16

b) 13

c) -1

d) -10

e) 16

f) -31

¿Qué tienen en común todos los números negativos y todos los números positivos al utilizar esta representación?

3) Repita el ejercicio anterior, pero ahora utilizando 8 bits. ¿Qué conclusiones se pueden extraer al comparar los resultados con los del ejercicio anterior?

4) Dados los siguientes números en binario, indicar a qué números corresponden en sistema decimal utilizando la representación en complemento a dos:

a) $(00001101)_2$

b) $(01001101)_2$

c) $(11100001)_2$

d) $(11111001)_2$

e) $(11111111)_2$

f) $(00000000)_2$

5) Mostrar que $(13.25)_{10} = (1101.01)_2 = (15.2)_8 = (D.4)_{16}$

6) Rellenar la siguiente tabla:

Binario	**Octal**	**Hexadecimal**	**Decimal**
1101100.110			
	362.23		
		A1.03	
			74.3

En cada fila encontrarán un valor numérico expresado en la base indicada en la casilla superior de la columna correspondiente. Completen el resto de las casillas con la representación correspondiente según la base indicada en la parte superior, de manera que se mantengan las equivalencias en cada fila. Asuman que los números son sin signo.

7) Determinar el formato hexadecimal que use el mínimo número de dígitos y que permita representar el número $(16.25)_{10}$ de manera exacta. ¿Cuál es el rango y la precisión del formato? Asumir números sin signo.

Ayuda: Tener en cuenta que la precisión se puede interpretar como la diferencia entre un número y el siguiente número representable.

8) Realicen cada una de las siguientes operaciones en complemento a dos, utilizando 8 bits para representar cada número. Para cada operación, analicen si los resultados

obtenidos son correctos o incorrectos, explicando las razones. Ubique los resultados en la recta real, incluyendo los valores mínimos y máximos para esta representación. Además, indiquen el estado de las banderas *Carry* (`CF`) y *Overflow* (`OF`) después de realizar cada operación.

a) $S_1 = 10 - 3$

b) $S_2 = -39 + 92$

c) $S_3 = -19 - 7$

d) $S_4 = 44 + 45$

e) $S_5 = 104 + 45$

f) $S_6 = -75 + 59$

g) $S_7 = -103 - 69$

h) $S_8 = 127 + 1$

i) $S_9 = -1 + 1$

j) $S_{10} = -1 - 1$

k) $S_{11} = -8 - 127$

l) $S_{12} = 127 + 127$

9) Dados $A = 200$, $B = 300$, $C = 500$ y $D = 400$. Calcular $S = A \times B \times C \times D$, interpretando los datos como enteros con signo. Imprimir el resultado por pantalla haciendo un programa en Lenguaje C con datos tipo `int` y analizar el resultado. Repetir para datos tipo `long int`.

Ayuda: Para poder imprimir un `long int` se debe usar el modificador `%ld`.

10) Ejecutar el siguiente programa y analizar lo que imprime por pantalla. Conviene usar GDB para observar el contenido de las variables.

```
#include <stdio.h>
int main()
{
    char a=127;
    printf("%hhd %hhu\n",a,a);
    a=++a;
    printf("%hhd %hhu\n",a,a);
    unsigned char b=128;
    printf("%hhd %hhu\n",b,b);
    b=++b;
    printf("%hhd %hhu\n",b,b);
    return 0;
}
```

Ayuda: Para `signed char` se usa el modificador `%hhd` y para `unsigned char` el modificador `%hhu`.

Repetir usando los modificadores `%d` y `%u` en lugar de `%hhd` y `%hhu`, respectivamente.

11) Dado los siguientes números de 8 bits: `A=0xc4` y `B=0x4e`.

a) Realizar la suma `C=A+B` en hexadecimal.

b) Escriba el valor decimal de `C`, interpretando el resultado como `signed char` y luego como `unsigned char`.

c) Indicar el rango y la cantidad de números representables en ambos casos.

d) Indicar el valor de las banderas `CF`, `OF` y `SF` luego de realizarse la operación suma.

12) Supongamos que en un servidor de almacenamiento los tamaños de los archivos se gestionan usando el sistema octal. La capacidad total del servidor es de `010000` unidades de almacenamiento. Actualmente, el servidor contiene los siguientes archivos:

Archivo	**Unidades**
A	`0345`
B	`0672`
C	`01250`
D	`0507`
E	`0710`

a) Convertir los tamaños de los archivos a su equivalente en decimal.

b) Convertir la capacidad total del servidor a su equivalente en decimal.

c) Calcular el espacio total usado en el servidor en octal y decimal.

d) Calcular el espacio libre en el servidor en octal y decimal.

e) Si se desea almacenar un nuevo archivo de tamaño `0500` unidades, determinar si hay suficiente espacio disponible. Si el archivo se puede almacenar, actualizar la lista de archivos y recalcular el espacio usado y libre en el servidor.

13) Una plataforma de criptomonedas gestiona las transacciones de sus usuarios utilizando identificadores de transacción en sistema hexadecimal. Cada transacción tiene un identificador único y un valor asociado en una criptomoneda llamada CryptoCoin (CC). Actualmente, la plataforma tiene las siguientes transacciones registradas:

Transacción	**Identificador**	**Valor**
A	`0x1A2F`	`0x3B` CC
B	`0x2C4D`	`0x4E` CC
C	`0x3D5A`	`0x25` CC
D	`0x4E7C`	`0x6A` CC
E	`0x5F8E`	`0x72` CC

a) Convierte los identificadores y valores de las transacciones a su equivalente en decimal.

b) Calcula el valor total de todas las transacciones en CryptoCoin en decimal y hexadecimal.

c) La plataforma añade dos nuevas transacciones con los siguientes detalles:

- Transacción F: Identificador `0x6A7B`, valor `0x3C` CC.

- Transacción G: Identificador `0x7B9D`, valor `0x4A` CC.

Actualiza la lista de transacciones y recalcula el valor total de todas las transacciones.

14) 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 + ............`

15) Implemente una función en lenguaje C:

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

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

Ayuda: Pensar en las propiedades de los operadores de bits.

16) Implemente una función `printbin` en lenguaje C:

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

que tome un entero de 32 bits y lo imprima en binario. Utilizar esta función para mostrar en binario los números del Ejercicio 2.

Ayuda: Se puede utilizar la función realizada en el ejercicio anterior.

17) Implemente una función en lenguaje C 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`. Se puede pensar primero en intercambiar dos variables:

```
x = x ^ y;
y = x ^ y;// pero ahora x = x ^ y, entonces y = x ^ y ^ y = x
x = x ^ y;// x = x ^ y ^x = y
```

Luego se puede extender a tres variables.

18) Escriba en lenguaje C 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?

19) *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 en lenguaje C:

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

20) (Opcional) 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`

21) (Opcional) Implemente el algoritmo del campesino ruso para los números anteriores.