Electrónica Digital 1
Guía 1: Sistemas numéricos

Nota: todos los ejercicios que consisten en el desarrollo de una función deben estar en un archivo separado con su correspondiente header. Además deben contar con un programa que pruebe el funcionamiento con diferentes valores de entrada.

Ejercicio 1: Tipos de dato y su tamaño en bits.

Realizar un programa en el que se declare una variable de cada tipo **char**, **short int**, **int**, **long int**, **long long int**, **float** y **double**, nombrándolas por su tipo, por ejemplo var_char, var_short_int, etc. Crear además 7 variables del tipo **short int** llamadas tam_char, tam_short_int, tam_int, etc. En estas variables se guardará el tamaño de cada una de las variables definidas en un principio. Mostrar el tamaño de cada variable. (*utilizar **sizeof()***. ¿Cambia el tamaño de las variables si estas son **unsigned**?

Ejercicio 2: Tipos de dato del header <stdint.h>

Repetir el programa anterior para los tipos definidos en la librería <stdint.h> : **int8_t**, **int16_t**, **int32_t**, **int64_t**. ¿Cambia el tamaño de las variables si estas son unsigned (es decir, **uint8_t**, **uint16_t**, …)?

Ejercicio 3: Bases numéricas

La función **printf** permite imprimir caracteres con cierto formato. Dentro de los *caracteres de escape* que permiten hacerlo, existe por ejemplo "%i","%x" y "%o" que permiten representar un número en forma *entera, hexadecimal* y *octal* respectivamente. Realizar un programa que genere una tabla donde se permitan ingresar dos números enteros A y B, y se represente en forma de tabla todos los números entre A y B inclusive, mostrando su representación decimal, hexadecimal y octal (*se pueden encolumnar los valores con el carácter de escape de printf "\t").*

Ejercicio 4: Conversión a binario

Realizar una función llamada **ImprimirBin8** que reciba un entero de 8 bits e imprima en pantalla su representación en binario. Luego agregue una columna al ejercicio anterior, donde se muestre la representación de cada número también en binario. El prototipo es el siguiente:

void ImprimirBin8(**int8_t** num);

a) Realizar la función dividiendo sucesivamente por 2 y obteniendo el resto en cada división. Se sugiere usar el operador **%** para obtener el resto de la división entre 2 números enteros.

b) Realizar además las versiones de 16 y 32 bits de la misma función.

Ejercicio 5: Conversión binario a decimal.

Realizar una función llamada **BinarioDecimal** que reciba un string con los bits de un número binario y devuelva un entero de 32 bits con el valor del mismo. El prototipo de la función es el siguiente:

int32_t BinarioDecimal (**char*** cadena_numero);

Generar un programa que solicite al usuario un número en binario, y lo muestre en pantalla en decimal, binario y octal utilizando la función anterior.

Ejercicio 6: Variables signadas y no signadas.

La función **printf** cuenta con un *sub-especificador* que permite fijar el largo de la variable a imprimir (ver referencia). Se puede utilizar el sub-especificador "h", colocando este carácter entre el signo "%" y el *especificador de tipo* para fijar el largo de la variable a mostrar a 16 bits. Por ejemplo, para fijar un entero signado de 16 bits quedaría **"%hd"**. Cree una función que reciba un entero de 16 bits con signo y que imprima el contenido de esta variable en hexadecimal, como entero no signado de 16 bits y como entero signado de 16 bits, con una leyenda que lo aclare para cada caso. El prototipo de la función es el siguiente:

void ImprimirContenidoEntero (**int16_t** numeroRepresentar);

Luego cree un main donde se declare una variable con signo de 16 bits guardando el número en hexadecimal **"0x7FFF".** Utilizando la función anterior, muestre el contenido de esta variable en los 3 formatos. ¿Hay alguna diferencia en la representación? Luego incremente el valor de esta variable en uno (utilizando ++ por ejemplo). Vuelva a llamar a la función anterior para mostrar el contenido de la variable en los 3 formatos. ¿Por qué se ven diferencias?

Vuelva a asignar el valor "**0x0000**" a la variable antes declarada, y muestre el contenido en los 3 formatos utilizando la función anterior. Luego decremente el contenido de la variable (utilizando -- por ejemplo) y vuelva a mostrar el contenido utilizando la función. ¿Qué diferencias se observan?

Ejercicio 7: Overflow en operaciones y "*wrap around*". Carry y borrow.

a) Cree tres variables no signadas de 16 bits, **num1**, **num2** y **resultado**. En las primeras 2 guarde los números en hexadecimal **num1 = 0x8000;** y **num2 = 0x0001;**. Luego asigne a la variable resultado la resta **num1 - num2;.** A continuación muestre la variable num1 en hexadecimal, como entero signado y como entero no signado. Hacer lo mismo para num2 y para la variable resultado. Viendo los resultados obtenidos, se generó overflow tras la operación? cómo podemos darnos cuenta de esto?

b) Repetir lo mismo pero asignando los valores **num1 = 0x7FFF;** y **num2 = 0x0001;** Luego asigne **resultado = num1 + num2;** Muestre las 3 variables en los 3 formatos y saque conclusiones respecto a la validez de los resultados. ¿Hubo overflow?

c) Asignar los valores **num1 = 0xF000;** y **num2 = 0xFF01;** y guardar en resultado la suma de las dos variables. Mostrar las 3 variables en los 3 formatos anteriores. Sacar conclusiones de los resultados. ¿Hubo overflow, carry o nada?

d) Asignar los valores **num1 = 0x0011;** y **num2 = 0xFF01;** y guardar en resultado la suma de las dos variables. Mostrar las 3 variables en los 3 formatos anteriores. Sacar conclusiones de los resultados. ¿Hubo overflow, carry, borrow o nada?

Ejercicio 8: Extensión de signo

a) Definir una variable sin signo de 16 bits llamada *uvar16* y asignarle el valor **0x80FF;** Imprimir este valor en hexadecimal, como entero sin signo y como entero con signo. Crear una variable sin signo de 32 bits llamada *uvar32* y asignarle el valor de **uvar16** realizando el casteo a uint32_t. Imprimir el contenido de **uvar32** en hexadecimal, entero con signo y sin signo.(Tener en cuenta que no se muestran los ceros a la izquierda en el hexadecimal).

b) Repetir el punto anterior pero utilizando variables *con signo*. ¿Qué diferencias se ve al representar el hexadecimal de las dos variables de 32 bits? ¿Para qué es esto necesario? ¿La extensión de signo se hace teniendo en cuenta el tipo de dato de la variable de origen o el tipo de dato de la variable de destino? ¿Interesa si el casteo de 16 a 32 bits es con o sin signo?

Ejercicio 9: Imprimir contenido hexadecimal de un número flotante IEEE-754.

a) La norma IEEE-754 especifica el formato para el punto flotante. Escriba una función llamada **ImprimirContenidoFloat** que permita ver el contenido hexadecimal de una variable float separados por bytes.

Convierta el número (-118,625) en decimal a punto flotante y luego compruebe este valor con el programa generado. ¿Cuál es el número más chico que se puede representar con este valor de exponente?

Repetir el ejercicio para los siguientes valores:

- +118.625
- -1186.25
- +474.5

Ejercicio 10: Codigo ASCII.

- Utilizar la tabla ASCII como referencia. Crear una variable **char** y asignarle el valor 65 en decimal. Imprimir el contenido de la variable utilizando **printf** con los caracteres de escape **“%c”** y “**%d”**. ¿Este valor se corresponde con la tabla ASCII?
- Asignarle ahora a la variable el caracter **‘a’**, e imprimir el contenido de la variable de la misma manera que en el inciso anterior. ¿Qué valor numérico se guarda en la variable char al asignarle un caracter?
- Sumarle uno a la variable con el valor anterior y repetir la muestra del contenido de la variable.
- ¿Se puede realizar cualquier operación matemática o lógica sobre la variable char?