Ejercitación – Serie F

aritmética a nivel de bits.

Salvo se indique lo contrario, la referencia a una posición dentro de una variable se hará contando desde cero desde el dígito menos significativo.

Realizar las funciones solicitadas en un archivo .c, y el main de la aplicación en otro archivo .c (trabajando con varios archivos fuentes)

Ejercicio 1:

1.a.-

Realizar una función que retorne el valor del *nibble* de una variable de 32 bits, según se lo indique mediante la variable **n** también pasada como parámetro, considerando que el *nibble* cero se corresponde con el *nibble* menos significativo (es decir iniciando a contar desde el *nibble* menos significativo).

Algunos ejemplos:

- Valor = **0x12345678** y **n**=0 se debe retornar: **0x08**
- Valor = **0x12345678** y **n**=5 se debe retornar: **0x03**

1.b.-

Ídem anterior, pero considerando que el *nibble* cero se corresponde al *nibble* más significativo (es decir iniciando a contar desde el *nibble* más significativo)

Algunos ejemplos:

- Valor = **0x12345678** y **n**=0 se debe retornar: **0x01**
- Valor = **0x12345678** y **n**=5 se debe retornar: **0x06**

1.c.-

Fusionar las funciones anteriores en una sola, de manera que al momento de invocar la función se deba indicar si se desea trabajar en modo nibble menos significativo o más significativo.

Ejercicio 2:

2.a.-

Realizar una función que modifique una variable de 32 bits, colocando un determinado *nibble* en cero según se lo indique mediante la variable **n**, iniciando desde el nibble menos significativo.

Algunos ejemplos:

- Valor = **0x12345678** y **n**=0 el valor resultante debe ser: 0x1234567**0**
- Valor = **0x12345678** y **n**=5 el valor resultante debe ser: 0x12**0**45678

2.b.-

Modificar el anterior para que en lugar de colocar un cero en el nibble indicado, coloque un valor pasado por parámetro.

Ejercicio 3:

Realizar una función que retorne el valor del bit de una variable según se lo indique mediante el parámetro **bit**.

Ejercicio 4:

4.a.-

Realizar una función que modifique una variable recibida como parámetro, poniendo en 1 el valor del bit de dicha variable según se lo indique mediante el parámetro **`bit`**, también recibida como parámetro.

4.b.-

Realizar una función que modifique una variable recibida como parámetro, poniendo en 0 el valor del bit de dicha variable según se lo indique mediante el parámetro **`bit`**, también recibida como parámetro.

4.c.-

Realizar una función que modifique una variable recibida como parámetro, conmutando el valor del bit de dicha variable según se lo indique mediante la variable **`bit`**, también recibida como parámetro.

Ejercicio 5:

Realizar una función que modifique un bit de una variable no signada de 32 bits, colocando en 0 o en 1 el valor de dicho bit según se especifique.

Ejercicio 6:

Realizar una función que imprima un valor en formato binario.

Ejercicio 7:

Realizar una función que, utilizando aritmética de bits, indique si el valor recibido es par.

Ejercicio 8: (recomendado)

8.a.-

Analizar que hace la siguiente sentencia.

```
a^=b^=a^=b;
```

8.b.-

Antes de ejecutar el código, intente predecir que se imprimirá en el 2do printf.

```
int main(void) {
unsigned char x=0xAA;
unsigned char y=0x55;

    printf(" x:%X - y:%X\n",x,y);
    x^=y^=x^=y;
    printf(" x:%X - y:%X\n",x,y);
  return 0;
}
```

Nota: *dependiendo del compilador la sentencia propuesta (*`x^=y^=x^=y;`*) puede generar un warning.*

8.c.-

Es común que este tipo de sentencia se escriban en una macro. En tal sentido, si la sentencia bajo análisis la definimos en una macro llamada XYZ, el código anterior quedaría:

```
int main(void) {
unsigned char x=0xAA;
unsigned char y=0x55;
```

```
    printf(" x:%X - y:%X\n",x,y);
    XYZ (x,y);
    printf(" x:%X - y:%X\n",x,y);
  return 0;
}
```

Implementar la macro XYZ.

8.d.-

Realizar una función que realice la misma tarea que la macro.

8.e.-

Analice y documente pros y limitaciones de cada método (macro vs función).

8.f.-

Ejecutar el proceso de XOR sucesivos (a^=b^=a^=b;) con variables de punto flotante de simple precisión (float) - es con un poco de trampa -.

Ejercicio 9: (recomendado)

Realizar una función que imprima en pantalla los valores característicos de una variable tipo punto flotante, pudiendo esta ser de simple, doble o cuádruple precisión.

```
void prt_flotantes (const void *pf, int mode);
```

En donde **pf** es el puntero al dato y **mode** hace referencia al tipo de dato apuntado

Ejercicio 10: (desafío)

Determinar si un procesador es del tipo Big endian o Little endian.