OPERACIONES CON BITS

Opoeraciones logicas y bit shift con Arduino

OBJETIVOS

- Presentar la forma de rerpresentar caracteres graficos, en una matriz de puntos.
- Como codificar estos caracteres.
- Presentar los operaciones básicas de bits AND, OR, XOR
- Presentar los desplazamientos de bits, a izquierda y derecha.

MATERIAL REQUERIDO.

 (http://www.prometec.net/producto/arduino-uno/)	**Arduino UNO o equivalente (http://www.prometec.net/categoria-producto/arduinos/)**, y estado mental relajado.

DEFINIENDO EL GRAFICO DE UN CARACTER

Si recordáis, en la sesión 31, definimos unos arrays como estos para mostrar los números en el display, algo así:

```
byte Digit[10][8] =
  {  { 1,1,1,1,1,1,0,0 },    // 0
     { 0,1,1,0,0,0,0,0 },    // 1
     { 1,1,0,1,1,0,1,0 },    // 2
     { 1,1,1,1,0,0,1,0 },    // 3
     { 0,0,1,0,0,1,1,0 },    // 4
     { 1,0,1,1,0,1,1,0 },    // 5
     { 1,0,1,1,1,1,1,0 },    // 6
     { 1,1,1,0,0,0,0,0 },    // 7
     { 1,1,1,1,1,1,1,0 },    // 8
     { 1,1,1,0,0,1,1,0 }     // 9
   };
```

Definíamos los segmentos que se tenían que iluminar, especificando un 1 o un 0, con un byte. Lo que significa que ese array ocupa 80 bytes (*Podéis comprobarlo con la función sizeof(Digits)*).

Claro que esto es para unos displays BCDs que tiene 8 segmentos por cada digito. Suponed que quisiéramos utilizar unos displays un poco más sofisticados como una matriz de 8×8 puntos LED, que pudiéramos encender independientemente (Nuestra siguiente sesión). Tendríamos que definir unas matrices de 8×8 puntos, para dibujar una letra concreta.

Si hacemos lo mismo que en el caso de la sesión 31, cada letra ocupara 8 x 8 bytes= 64 , y como si definimos el alfabeto completo, con mayúsculas y minúsculas, números y algunos símbolos como +,-,/,* símbolos de puntuación y demás, nos iremos enseguida a unos 128 caracteres. Que por 64 bytes cada uno, nos da la bonita cifra de 8.192 bytes.

Si recordáis de cuando hablamos de la memoria para las variables, solo tenemos 2k, y resulta que necesitamos 8k para estos arrays, así que por aquí no vamos bien.

Afortunadamente, no somos los primeros en tener este problema (Sorpresa) y más en la época heroica donde ponerle 8k de RAM al ordenador podía costar una cifra con más ceros de los que podíamos pagar. Y la solución es codificar a nivel de bits y operar luego con ellos.

Y esto nos lleva directamente a los objetivos de esta sesión.

DEFINIENDO ARRAYS COMPACTOS DE CARACTERES 8×8

Supongamos que quisiéramos dibujar las letras de PROMETEC en un display de 8×8. Podríamos jugar a dibujar en cuadriculas algo parecido a esto:

	0	1	2	3	4	5	6	7		0	1	2	3	4	5	6	7	Valor binario	Hex
0		■	■	■	■					0	1	1	1	1	0	0	0	0b01111000	0x78
1		■				■				0	1	0	0	0	1	0	0	0b01000100	0x44
2		■				■				0	1	0	0	0	1	0	0	0b01000100	0x44
3		■	■	■	■					0	1	1	1	1	0	0	0	0b01111000	0x78
4		■								0	1	0	0	0	0	0	0	0b1000000	0x40
5		■								0	1	0	0	0	0	0	0	0b1000000	0x40
6		■								0	1	0	0	0	0	0	0	0b1000000	0x40
7		■								0	1	0	0	0	0	0	0	0b1000000	0x40

(http://www.prometec.net/wp-content/uploads/2014/10/bits_P.jpg)
Para la R:

	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	Valor binario	Hex
0									0	1	1	1	1	0	0	0	0b01111000	0x78
1									0	1	0	0	0	1	0	0	0b01000100	0x44
2									0	1	0	0	0	1	0	0	0b01000100	0x44
3									0	1	1	1	1	0	0	0	0b01111000	0x78
4									0	1	1	1	0	0	0	0	0b01110000	0x70
5									0	1	0	1	1	0	0	0	0b01011000	0x58
6									0	1	0	0	1	1	0	0	0b01001100	0x4A
7									0	1	0	0	0	0	1	1	0b01000011	0x43

(http://www.prometec.net/wp-content/uploads/2014/10/bits_R.jpg)
Para la O:

	0	1	2	3	4	5	6	7	0	1	2	3	4	5	6	7	Valor binario	Hex
0									0	1	1	1	1	1	0	0	0b01111100	0x7A
1									0	1	0	0	0	0	1	0	0b01000010	0x42
2									0	1	0	0	0	0	1	0	0b01000011	0x42
3									0	1	0	0	0	0	1	0	0b01000012	0x42
4									0	1	0	0	0	0	1	0	0b01000013	0x42
5									0	1	0	0	0	0	1	0	0b01000014	0x42
6									0	1	0	0	0	0	1	0	0b01000015	0x42
7									0	1	1	1	1	1	0	0	0b01111100	0x7A

(http://www.prometec.net/wp-content/uploads/2014/10/bits_O.jpg)
Fijaros que además de escribir el valor en binario, he añadido una última columna en hexadecimal, porque es más cómodo de manejar.

- Dijimos la última vez que hablamos del tema, que el hexadecimal se usaba mucho porque era muy fácil de pasar al binario y viceversa. Para los que tengáis interés, es tan fácil como coger los dígitos binarios de 4 en 4 y pasarlos directo a hexadecimal con la siguiente tabla:

Bin	0000	0001	0010	0011	0100	0101	0110	0111	1000	1001	1010	1011	1100	1100	1110	1111
Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F

(http://www.prometec.net/wp-content/uploads/2014/10/Img_36_1.jpg)

Una manera compacta de escribir esos valores es:

```
byte P[] = { 0x78, 0x44, 0x44, 0x78, 0x40, 0x40, 0x40, 0x40 };
byte R[] = { 0x78, 0x44, 0x44, 0x78, 0x70, 0x58, 0x4C, 0x46 };
byte O[] = { 0x3C, 0x42, 0x42, 0x42, 0x42, 0x42, 0x42, 0x3C };
```

Fijaros que lo único que hemos hecho, es copiar los números de arriba en hexadecimal (aunque es lo mismo si los ponéis en binario o decimal). De este modo, con 8 bytes codificamos los bits necesarios para dibujar cada letra.

El problema ahora, será sacar los bits cuando los necesitemos, lo que nos lleva de cabeza al siguiente punto

OPERACIONES CON BITS

Vamos a empezar con la P y con el array que la define gráficamente.

```
byte P[] = { 0x78, 0x44, 0x44, 0x78, 0x40, 0x40, 0x40, 0x40 };
```

Lo escribimos en hexadecimal, por comodidad. Pero la idea es que la primera fila de la P está contenida en el primer 0x78 del array. Cada uno de los bits que dibujan su primera fila está allí, contenido en un único bit, no en 8. Y lo mismo con el resto de las filas del dibujo.

Si hacemos esto para definir un juego completo de caracteres, usaríamos

128 caracteres x 8 bytes = 1024 bytes

Eso sí, nos encaja y aún nos queda otro k de memoria para variables corrientes. El problema es que tendremos que desempaquetar los bits para poder iluminar cada punto del display, pero vamos a ver que C++ tiene medios para eso.

Vamos en primer lugar con la función AND a nivel de bit. Su operador es & (no confundir con && , que es el operador lógico). Veamos el resultado con un par de números:

```
int a =  92;       // in binary: 0000000001011100
int b = 101;       // in binary: 0000000001100101
int c = a & b;     // result:    0000000001000100,    68 in decimal.
```

AND se efectúa a nivel de bits, si los dos son 1, el resultado es 1. En caso contrario el resultado es 0. Fijaros que realiza la operación bit a bit en binario.

Veamos la función OR. Su símbolo es |

```
int a =  92;        // in binary: 0000000001011100
int b = 101;        // in binary: 0000000001100101
int c = a | b;      // result:    0000000001111101,  125 in decimal.
```

OR se calcula a nivel de bit, del siguiente modo: El resultado es uno si cualquiera de los bits es 1 o ambos lo son, y cero si ambos son 0.

También hay un operador XOR, que es uno si cualquiera de ellos es un 1, pero un 0 si ambos son 1 o 0. Su símbolo es ^

```
int x = 12;     // binary: 1100
int y = 10;     // binary: 1010
int z = x ^ y;  // binary: 0110, ir decimal 6
```

¿Y para qué sirve todo esto? La pregunta del millón.

Pues como siempre, resulta muy complicado explicar la solución a problemas que aun no has tenido, pero tranquilos que enseguida os pondré uno de estos.

- Hasta ahora hemos usado las puertas de Arduino de una en una, leyendo su valor. Pero Arduino contiene registros internos que podemos leer en bloques de 8 y cada bit representa el estado de una puerta digital.
- De hecho, el IDE de Arduino, lee los registros en bloque y después hace operaciones a nivel de bit, para mostrarnos el resultado de un digitalRead, como HIGH o LOW. Y esto queridos amigos es más lento, que hacerlo tú, y hay ocasiones en que mejorar la velocidad de tu programa puede ser crítico.
- No tengo intención de entrar en este tema aún (aunque todo llegará), pero es importante que os vaya sonando.

El caso de representar caracteres, mediante matrices de puntos es uno de estos problemas, en el que estas operaciones serán críticas.

Cuando queramos encender un punto LED en una matriz e 8×8, tendremos que sacar uno, a uno los bits independientes de un byte que representan los puntos a iluminar o no. Para extraer un bit concreto de un byte de datos, tendremos que usar este tipo de operadores y además utilizar los bitwise **operators**.

Asi que, alla vamos.

El primero es el operador >>, que funciona desplazando tantos bits a la derecha un número. Y existe el **bitshif left** equivalente cuyo símbolo es <<. Veamos un ejemplo:

```
int a = 5;          // binary:  00000101
int b = a << 3;     // binary: 00101000,     // 40 en decimal
int c = b >> 3;     // binary: 00000101,     // Vuelta al 5 inicial
```

Si desplazamos 00000101, tres dígitos a la izquierda el resultado es 00101000.

Es muy parecido a lo que pasaba con un shift register, donde los bits eran empujados. Al desplazar a la izquierda, los bits que entran por la derecha son 0s y los que salen se pierden sencillamente.

Cuando haces un shift a la derecha, los bits salientes por la derecha se pierden, y los que entran son 0s.

Bueno, pues armados ya con estas herramientas vamos a ver como sacamos un bit concreto de un byte concreto:

> *Estas operaciones de AND, OR, XOR , right shift y left Shift, funcionan igual con independencia del tipo de la variable, es decir funciona, también con int, long y demás*.

OBTENIENDO BITS INDEPENDIENTES

Supongamos que tenemos un número como este,

```
byte n = 0b00110110 ;       // Hex 0x36         DEC  54
```

Y que queremos obtener el bit 3, empezando por la derecha (posición 2). ¿Cómo lo haremos?

Empecemos definiendo la posición de los bits a así, de izquierda a derecha la posición es

pos = 7 6 5 4 3 2 1 0

Podríamos empezar haciendo un rigt shift de 3 bits:

```
byte n = 0b00110110;
n >>3  = 0b00000110    // Los bits salen por la derecha y entran ceros por la izquierda
```

Pero lo que nos interese es solamente el bit que está a la derecha ahora y no el resto, así pues haremos:

```
byte resultado = n & 0b00000001 // Decimal 1
```

Al hacer el AND a nivel de bits, liquidamos todas las posiciones que llevan 0 y solo si el bit menos significante de n es 1, el resultado será 1, de lo contrario será 0.

Podemos escribir una función de propósito general para obtener un bit concreto de un valor:

```
bool GetBit( byte N, int pos)
   {                            // pos = 7 6 5 4 3 2 1 0
       int b = N >> pos ;       // Shift bits
       b = b & 1 ;              // coger solo el ultimo bit
       return b ;
   }
```

Esta función nos devolverá un TRUE o FALSE según que el bit sea 1 o 0, y vamos a tener ocasión de probarla. A modo de diversión, os incluyo aquí un programita que te va listando los bits consecutivos de un número n cualquiera, que le pases en la primera línea:

```
byte n = 0b01110110 ;

void setup()
   {      Serial.begin(9600);  }

void loop()
   {      for ( int k=7; k>=0 ; k--)
             {   byte m = GetBit( n, k);
                 Serial.print(m);
                 Serial.print(", ");
             }
          Serial.println();
   }

bool GetBit( byte N, int pos)
    {   // pos = 7 6 5 4 3 2 1 0
        int b = N >> pos ;         // Shift bits
        b = b & 1 ;                // coger solo el ultimo bit
        return b ;
     }
```

Por cierto, que tenía pendiente comentaros(y este no parece mal momento), que nuestro Serial.print, acepta modificadores para decirle si queremos que el resultado salga en binario, decimal, hexadecimal,

```
int n = 415 ;

void setup()
   {
      Serial.begin(9600);
   }
void loop()
   {
        Serial.print(n, DEC);
        Serial.print(",    ");
        Serial.print(n, BIN);
        Serial.print(",    ");
        Serial.print(n, HEX);
        Serial.print(",    ");
        Serial.println(n, OCT);
   }
```


Y así espero ahorraros algún dolor de cabeza convirtiendo formatos.

RESUMEN DE LA SESIÓN

- Hemos visto, como rerpresentar caracteres graficos, en una matriz de puntos, y como codificar de modo compacto esta información bit a bit.
- Presentamos los operaciones básicas de bits AND, OR, XOR
- Hemos hecho algún ejemplo de desplazamiento de bits a derecha e izquierda.
- Hemos preparado una función genérica GetBit, que nos permite sacar un bit independiente de un byte previo y que seria muy fácil extender otros tamaños.
- Describimos la forma de usar Serial.print con modificadores, para que formatee la salida en decimal, hexadecimal, binario u octal.

Anterior (http://www.prometec.net/led-bar/)

Siguiente (http://www.prometec.net/matriz-led-8x8/)

(8) COMMENTS

Reply (Http://Www.Prometec.Net/Operaciones-Bits/?Replytocom=18149#Respond)

Ramses (http://www) *01 Feb 2017*

Buenas tardes,

Cuando hablas del Array Digits dices que ocupa 80 bytes y que cada carácter 64 bytes. ¿Es correcto esto o en ambos casos serían bits en vez de bytes?

Saludos y gracias

Reply (Http://Www.Prometec.Net/Operaciones-Bits/?Replytocom=17185#Respond)

Isaac (http://www.prometec.net/members/isaac/) *09 Ene 2017*

Que buena explicación,
Todo lo necesario refrescado y listo para trabajar con sólo una lectura, en serio, ojalá hubiera tenido profesores tan buenos.
Como siempre muchas gracias

Reply (Http://Www.Prometec.Net/Operaciones-Bits/?Replytocom=6515#Respond)

Alejandro *19 Feb 2016*

Hola podrias explicar de nuevo y exclusivamente la parte de la funcion bool GetBit que no la puedo entender bien, gracias.

Saludos

Reply (Http://Www.Prometec.Net/Operaciones-Bits/?Replytocom=6519#Respond)

admin (http://www.prometec.net/members/admin/) *20 Feb 2016*

Hola Alejandro, GetBit es solo una funcion simple para coger un bit definido de un byte que le pasamos. La idea es que si queremos coger el terecer bit del numero, entonces hacemos un shifting a la derecha de tantos bits como posicion buscamos, por eso si hacemos N >> 3 es el bit mas a la derecha el que contine lo que buscamos pero a la izquierda

hay mas digitos que tenemos que limpiar y para eso hacemos el and logico con 1 o si prefieres con 0000 0001 de modo que nos a seguramos de poner a 0 todo el resultado menos el bit que nos interesa, y ahora el resultante ya solo es 0 o 1 que corresponde al bit que boscabamos en la posicion inical pos

Reply (Http://Www.Prometec.Net/Operaciones-Bits/?Replytocom=4039#Respond)

Secundino Fernández (http://medrocracia.blogspot.com.es/) *14 Dic 2015*

Hay una placa de diseño español, vinciDuino, compatible con Alduino Leonardo, que permite programar el chip con el software Atmel.

Reply (Http://Www.Prometec.Net/Operaciones-Bits/?Replytocom=4037#Respond)

Secundino Fernández (http://medrocracia.blogspot.com.es/) *14 Dic 2015*

No hay problema; como dije, ahora me tomo la vida con calma.
He podido encontrar la documentación. Hay un PDF de unas 500 páginas, para el Mega2560, en la web de Atmel.
Gracias por contestar. Seguiré por aquí y os pediré algún material, cuando me considere un poco preparado.

Reply (Http://Www.Prometec.Net/Operaciones-Bits/?Replytocom=3913#Respond)

Secundino Fernández *10 Dic 2015*

Hace bastantes años que supe de la existencia de Arduino y siempre he sentido interés. Ahora, casi jubilado, dispongo de tiempo.
Primero, he de decir que, este, me parece un excelente manual, que combina la pedagogía con el sentido del humor; dos visiones que no deberían ser antónimos.
Como procedo del mundo de la automatización y los PLCs, una de las instrucciones más útiles es el manejo de registros. Las órdenes para hacerlo con Arduino están (bitRead (x, n) bitWrite(x, n, b) bitSet(x, n)) pero, ¿como escribir algo parecido a esto?: ” Si la entrada 1 está a 1, la 2 la dos está a 0 y la salida 3 se pone a 1, pon a set(1) el registro X ” para poder usar ese registro en otras partes del programa.
¿Que direcciones de memoria se pueden utilizar para ello?.
Estas son las direcciones de memoria del chip Atmel para el temporizador 3. (0x9D) (0x9C) (0x9B) (0x9A) (0x99) (0x98) (0x97) (0x96) (0x95) (0x94)
¿Como acceder, por ejemplo, al 0x95 (Counter Register High Byte)?.

Reply (Http://Www.Prometec.Net/Operaciones-Bits/?Replytocom=3983#Respond)

admin (http://www.prometec.net/members/admin/) *12 Dic 2015*

Hola secundino,

Perdona el retraso de la respuesta pero he tenido unos dias complicados y queria leer despacio tu pregunta. No tengo experiencia accediendo a los registros internos del ATMEL, porque lo he evitado por pura vagancia y la ventaja de poder usar el entorno arduino es precisamente poder olvidarte de bajar a el barro de los registros internos, pero no hay ningun problema en ello y simplemente necesitarias una buena guia al respecto

Normalmente a programar a bajo nivel el procesador atmel 328 de un arduino UNO se requiere usar la librería AVR que es la base de todo esto y si buscas por internet en seguida encontraras libros sobre ello.

Si no los encuentras, pásame un correo privado y te enviare alguno que tengo por algún rincón de mi disco duro

Un saludo y hasta pronto

GIVE A REPLY

Conectado como Javier (http://www.prometec.net/members/jmartina/profile/edit/). ¿Quieres salir? (http://www.prometec.net/wp-login.php?action=logout&redirect_to=http%3A%2F%2Fwww.prometec.net%2Foperaciones-bits%2F&_wpnonce=a8a274ddda)

Message

Post comment