



PROGRAMACIÓN EN C

PRÁCTICA 2: Operaciones con bits



Práctica 2: Índice

2.1 Introducción

2.2 Representación de información binaria en C

2.3 Operando con bits en C

2.4 Máscaras

2.5 Desplazamientos

2.6 Ejercicios propuestos



2.1 - Introducción

- El computador digital representa internamente la información, exclusivamente, en forma de números binarios.
- Por tanto, trabajar con los dispositivos hardware del sistema suele consistir en manipular bits.
- Además, cualquier entrada digital proporcionará directamente al sistema una información consistente en un conjunto de bits.
- Análogamente, las salidas digitales tendrán un formato similar.

3



2.2 - Representación de información binaria en C

- C no ofrece un tipo de datos específico para trabajar directamente con números binarios (o sea, con bits).
- Para ello se recurre normalmente a números enteros, considerando su representación interna.
- Por ejemplo, en C, podemos declarar una variable de tipo entero corto y darle un valor mediante distintas representaciones externas del mismo:

```
short int i;
```

```
i=300;
```

```
i=0x12C;
```

```
i=0454;
```

- Pero internamente la representación es la misma, y (suponiendo un tamaño de 2 bytes para el tipo entero corto) se almacenará :

00000001	00101100
----------	----------

4



2.2 - Representación de información binaria en C (II)

- También es posible recurrir a variables tipo carácter, dada su similitud con los enteros (debido a los valores enteros del código ASCII).

- Esto es especialmente útil para representar un único byte. Por ejemplo :

```
unsigned char c;
```

```
c = 'A';
```

```
c = 65; (valor ASCII en decimal del carácter A)
```

```
c = 0x41; (valor ASCII en hexadecimal del carácter A)
```



01000001

- En definitiva, para expresar un valor binario en C recurriremos a las representaciones externas de los tipos entero y carácter (decimal, octal, hexadecimal y carácter).

- Por ejemplo, si necesitamos expresar el valor 1011101b, podemos escribir en C:

– 93 en decimal

– 0x5D en hexadecimal

– 0135 en octal

– ']' como carácter

5



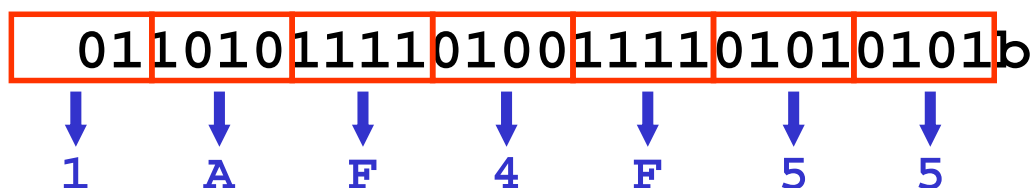
2.2 - Representación de información binaria en C (III)

- La forma más fácil de trabajar con números binarios en C es representarlos en hexadecimal.

- Para representar cualquier número binario en hexadecimal el procedimiento a seguir es:

1º - Hacer grupos de 4 bits empezando desde la derecha.

2º - Sustituir cada grupo por el valor hexadecimal de cada grupo.



1AF4F55h → 0x1AF4F55 en C

6



2.3 - Operando con bits

Operadores C que trabajan a nivel de bit:







-  **&** AND (“y” a nivel de bit)
-  **|** OR (“ó” a nivel de bit)
-  **^** XOR (“ó exclusiva” a nivel de bit)
-  **~** NOT (“no” a nivel de bit)
-  **>>** desplazamiento de bits a la derecha
-  **<<** desplazamiento de bits a la izquierda

Tabla de verdad de las operaciones AND, OR y XOR:

1	&	1	=	1
1	&	0	=	0
0	&	1	=	0
0	&	0	=	0

1		1	=	1
1		0	=	1
0		1	=	1
0		0	=	0

1	^	1	=	0
1	^	0	=	1
0	^	1	=	1
0	^	0	=	0



2.3 - Operando con bits (II)

Ejemplos:

Operación en C (hex.)	Resultado (hex.)	Operación en binario	Resultado (binario)
0xF5 & 0x5A	→ 0x50	11110101b AND 01011010b	→ 01010000b
0xF5 0x5A	→ 0xFF	11110101b OR 01011010b	→ 11111111b
0xF5 ^ 0x5A	→ 0xAF	11110101b XOR 01011010b	→ 10101111b
~ 0xF5	→ 0x0A	NOT 11110101b	→ 00001010b
0x75 >> 2	→ 0x3D	01110101b >> 2	→ 00011101b
0xF5 << 2	→ 0xD4	11110101b << 2	→ 11010100b



2.4 - Máscaras

- Las máscaras son distintas secuencias de bits tales que, combinadas en una operación binaria con cualquier otra secuencia, permiten modificar ésta última, u obtener información sobre ella, a conveniencia del programador.
- El operador **AND**, sabiendo que **X AND 0 = 0** y **X AND 1 = X**, puede emplearse con una máscara para:

Forzar bits a 0.

Por ejemplo, si queremos que los tres bits de menor peso de un byte sean 0:

	b7	b6	b5	b4	b3	b2	b1	b0
AND	1	1	1	1	1	0	0	0
	b7	b6	b5	b4	b3	0	0	0

Para realizarlo en C:

```
unsigned char var;
var = var & 0xF8;
```

9



2.4 - Máscaras (II)

Conocer si un bit está a 1 ó a 0.

Por ejemplo, para comprobar el valor del bit 12 de una variable *var* de tipo *short int*:

&	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
	0	0	0	b12	0	0	0	0	0	0	0	0	0	0	0	0



```
if (var & 0x1000)
    printf("Es 1");
else
    printf("Es 0");
```

Extraer un grupo de bits.

Por ejemplo, para extraer el número binario de 3 bits contenido en los tres bits de menor peso de una variable *var*:

&	b15	b14	b13	b12	b11	b10	b9	b8	b7	b6	b5	b4	b3	b2	b1	b0
	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1
	0	0	0	0	0	0	0	0	0	0	0	0	0	b2	b1	b0



```
short int numero, var;
numero = var & 0x7;
```

10



2.4 - Máscaras (III)

- El operador **OR**, sabiendo que $X \text{ OR } 0 = X$ y $X \text{ OR } 1 = 1$, puede emplearse para:

Forzar bits a 1.

Por ejemplo, para conseguir que los tres bits de menor peso de un byte sean 1:

	b7	b6	b5	b4	b3	b2	b1	b0
OR	0	0	0	0	0	1	1	1
	b7	b6	b5	b4	b3	1	1	1



```
unsigned char var;  
var = var | 0x7;
```



2.4 - Máscaras (IV)

- El operador **XOR**, sabiendo que $X \text{ XOR } 0 = X$ y $X \text{ XOR } 1 = \bar{X}$, puede emplearse para:

Complementar bits.

Por ejemplo, para complementar los tres bits de menor peso de un byte:

	b7	b6	b5	b4	b3	b2	b1	b0
XOR	0	0	0	0	0	1	1	1
	b7	b6	b5	b4	b3	\bar{b}_2	\bar{b}_1	\bar{b}_0



```
unsigned char var;  
var = var ^ 0x7;
```



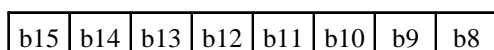
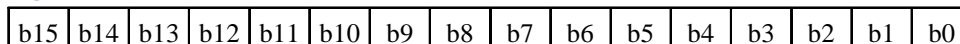
2.5 - Desplazamientos

Los operadores de desplazamiento de bits \gg y \ll equivalen a dividir o multiplicar por potencias de 2

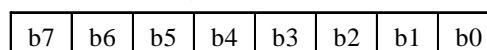
Un uso común de ambos es para componer/descomponer números.

Por ejemplo, para separar un número de 16 bits en dos bytes:

```
unsigned short int dato;
```



```
unsigned char alto;
```



```
unsigned char bajo;
```

```
alto = dato >> 8;  
bajo = dato & 0xFF;
```

O bien

```
alto = dato / 256;  
bajo = dato % 256;
```

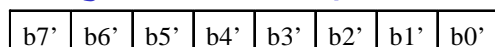
13



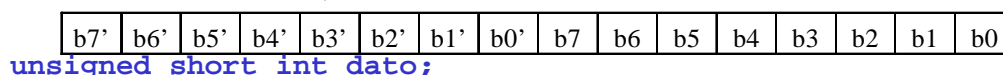
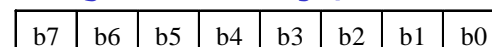
2.5 - Desplazamientos (II)

Por ejemplo, para componer un número de 16 bits a partir de 2 bytes:

```
unsigned char alto;
```



```
unsigned char bajo;
```



```
unsigned short int dato;
```

```
dato = alto;  
dato = dato << 8;  
dato = dato | bajo;
```

O bien


```
dato = alto * 256;  
dato = dato + bajo;
```

14




2.6 - Ejercicios propuestos

1. Realizar (en papel) las siguientes operaciones:

 Pasar 10101001011010101010101010101010101011b a hexadecimal.

 $0xAB \ \& \ 0x03$

 $0xAB \ | \ 0x03$

 $0xAB \ \wedge \ 0x03$

 $\sim 0xAB$

 $0xAB \ \gg \ 2$

 $0xAB \ \ll \ 2$

2. Hacer un programa que realice las operaciones del **Ejercicio 1** (salvo la primera), mostrando el resultado en pantalla, y comprobar los resultados.
3. Dada una variable *valor* de tipo **unsigned char**, escribir la expresión C que ponga a 1 los 2 bits de mayor peso y a 0 los 2 bits de menor peso.

15



2.6 - Ejercicios propuestos (II)

4. En los bits 10,9,8,7 de una variable de tipo **short int** hay codificado un número. Almacena su valor en una variable de tipo **unsigned char** y muéstralo por pantalla en formato decimal usando **printf()**.
5. Componer en una variable de tipo **unsigned char** un dato de 8 bits a partir de los 4 bits de menor peso contenidos en las variables de tipo **unsigned short int** *n_low* y *n_high*.
6. Una tarjeta de adquisición de datos proporciona a nuestro ordenador las lecturas de tres sensores (temperatura, acidez y nivel de un depósito de líquidos) agrupadas en una sola variable de tipo entero. Los datos recibidos del sensor de temperatura ocupan los 12 bits menos significativos de dicha variable (bits 0 al 11); los datos del sensor de acidez los 8 bits siguientes (bits 12 al 19), y los del sensor de nivel los 10 bits siguientes (bits 20 al 29). El resto de bits de la variable entera se descartan. Realizar un programa capaz de extraer las lecturas de los distintos sensores, mostrando cada una de ellas por pantalla (simular los datos recibidos en la tarjeta introduciendo por teclado valores de la variable entera).

16