UNIVERSITAT POLITÈCNICA DE VALÈNCIA

El tratamiento de números enteros bit a bit.

Aplicación a las máscaras con el lenguaje C.

Apellidos, nombre	Perles Ivars, Àngel (aperles@disca.upv.es)
Departamento	Informática de Sistemas y Computadores

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Centro	Escuela Técnica Superior de Ingeniería del Diseño

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

1 Resumen de las ideas clave

Se presentan las técnicas para la manipulación de números enteros para computadores desde el lenguaje de programación C, lo cual es un requisito indispensable en el aprovechamiento de periféricos. A continuación se resumen las ideas clave que se van a trabajar:

- Representar en hexadecimal los números binarios.
- Poner a 0, a 1, complementar y extraer bits de un número entero dado.
- Usar lenguaje C para aplicar estas técnicas.

2 Introducción

Tanto la programación de dispositivos periféricos del computador como la generación/recogida de señales digitales está cargada de números binarios; por tanto, es necesario dominar las técnicas que permiten manipular dichos números a nivel de bit. Esto es fundamental, por ejemplo, en la programación de microcontroladores.

La manipulación a nivel de bit se hace mediante las operaciones del álgebra de Boole AND, OR, X-OR y complementos que, aplicadas sistemáticamente, permitirán obtener 0's, 1's, complementar bits, etc. allí donde se necesite. Al uso de estas operaciones sobre un dato se le suele llamar *máscara*.

A parte de las técnicas, se verá aquí su aplicación práctica desde el lenguaje C.

3 Objetivos

Una vez leído y practicado este documento, el lector será capaz de:

- Manipular los bits de un número entero desde el lenguaje C.

4 Desarrollo

Antes de entrar en materia, es interesante hacer notar que el computador digital es "digital"' como se acaba de decir y, por tanto, representa internamente toda la información como 0's y 1's.

Para dejar claro este concepto, se verá primero el concepto de representación interna-externa empelado en los lenguajes de programación. A continuación se describirá la representación externa en formato hexadecimal como la más adecuada al propósito buscado. A continuación se introducirán los operadores booleanos necesarios para hacer la manipulación de números enteros y su notación en el lenguaje C. Finalmente, se irán describiendo los distintos tipos de máscaras que permiten hacer las manipulaciones.

4.1 Representación externa. Representación interna

La *representación interna* de los números enteros es un caso claro de esta idea. Internamente, un tipo de dato entero es un número binario de una determinada cantidad de bits que se representa habitualmente en *binario natural* para enteros sin signo y en *complemento a dos* para enteros con signo. Hay otras posibles representaciones, pero serán siempre secuencias de 0's y 1's.

Para un humano, usar directamente números binarios no está en su naturaleza, así que los lenguajes de programación permiten escribir los números en otras notaciones para facilitar su introducción; se llama a esto *representación externa*. Insistir en que se trata de un artificio; internamente seguirán siendo números binarios.

Aunque la representación externa decimal es la más comprensible para nosotros, también se proveen otros representaciones más o menos abstractas; por ejemplo, se usa la codificación ASCII para representar símbolos (letras,...), y la representación octal y hexadecimal para representar grupos de bits.

Para ilustrar la idea en el lenguaje C, sea el siguiente programa:

```
#include <stdio.h>
int main(void)
{
    int a;

    a = 'A';       // meter el codigo ASCII de la A
    a = a + 011;   // sumar 11 en octal (9 en decimal)
    a = a + 1;     // sumar 1 en decimal
    a = a + 0xA;   // sumar A en hexadecimal, (10 en decimal)

    printf("En a hay %d en decimal\n", a);
    printf("En a hay %x en hexadecimal\n", a);
    printf("En a hay %c en ASCII\n", a);

    return 0;
}
```

Obsérvese que se "escriben" números empleando distintos artificios, pero internamente todo será binario. Se recomienda probarlo para verificar esta idea, y modificarlo para experimentar.

La elección de una notación u otra es a conveniencia del programador y nunca al contrario.

4.2 Representación hexadecimal

Como el lenguaje C estándar no admite la introducción directa de números binarios, la manera recomendada aquí para escribir un número binario es emplear la representación externa *hexadecimal*. Esta representación se diseñó para representar números binarios en un equivalente que representa un grupo de 4 bits.

La siguiente tabla contiene las 16 posibles combinaciones binarias y sus equivalentes en decimal y en hexadecimal.

decimal	binario	hexadecimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0101	6
7	0111	7
8	1000	8
9	1001	9
10	1010	A
11	1011	B
12	1100	C
13	1101	D
14	1110	E
15	1111	F

Lo habitual es que el programador realice sus cálculos en binario y que, después, tenga que traducirlos a hexadecimal. Para pasar un número de binario a hexadecimal, simplemente se dividirá en grupos de 4 bits empezando desde la izquierda y se sustituirá por su equivalente hexadecimal.

Por ejemplo,

Número: *10101010111101010101111110101010*

A grupos de 4: *101 0101 0111 1010 1010 1111 1010 1010*

Equivalente hexadecimal: *5 5 7 A A F A A*

El número resultante es 557AAFAAh. Obsérvese como se ha puesto un "h" al final como notación habitual para hacer notar que el número es hexadecimal. No es lo mismo 1356 que 1356h.

Una vez obtenido el número hexadecimal, se introducirá en el programa utilizando la notación C *0xHH..HHH*, donde *HH...HHH* es el número hexadecimal calculado.

Por ejemplo,

```
int a;
```

```
a = 0x557AAFAA;
```

4.3 Operadores de bit

Para manipular los números enteros a nivel de bit, se recurre a los operadores C contenidos en la siguiente tabla,

v & w	AND a nivel de bit entre v y w
v | w	OR a nivel de bit entre v y w
v ∧ w	X-OR a nivel de bit entre v y w
v << n	desplazar n bits a la derecha a v
v >> n	desplazar n bits a la izquierda v
$\sim v$	complementar los bits de v

Como recordatorio, la siguiente tabla contiene las tablas de verdad de las operaciones AND, OR y X-OR.

AND	OR	X-OR
$0 \,\&\, 0 = 0$	$0 \mid 0 = 0$	$0 \wedge 0 = 0$
$0 \,\&\, 1 = 0$	$0 \mid 1 = 1$	$0 \wedge 1 = 1$
$1 \,\&\, 0 = 0$	$1 \mid 0 = 1$	$1 \wedge 0 = 1$
$1 \,\&\, 1 = 1$	$1 \mid 1 = 1$	$1 \wedge 1 = 0$

4.4 Máscaras

Conociendo los operadores anteriores y aplicando simples reglas repetitivas es fácil manipular adecuadamente los números enteros a nivel de bit. Se dan a continuación las recetas para ello.

4.4.1 Extraer bits

Dada una palabra binaria, una operación muy habitual es extraer unos bits de interés y eliminar el resto (ponerlos a 0). Para lograrlo, se empleará el operador AND aprovechándose de que b AND 0=0 y b AND 1=b.

Ejemplo: Se desean extraer los 3 bits menos significativos de una palabra binaria. Para hacerlo, se empleará el operador AND con el valor binario 0 ... 0111_2.

$$\begin{array}{cccccccccc} & b_N & \dots & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\ \& & 0 & \dots & 0 & 0 & 0 & 1 & 1 & 1 \\ \hline & 0 & \dots & 0 & 0 & 0 & b_2 & b_1 & b_0 \end{array}$$

Expresado en C sería,

```
resultado = dato & 0x7;
```

Como comprobación, se podría mostrar por pantalla el contenido de la variable *resultado* de la siguiente manera.

```
printf("\"resultado\" vale %d en decimal, y %x en hexadecimal\n",
resultado, resultado);
```

Suele ser interesante combinar las extracciones de bits con las operaciones de desplazamiento.

Ejemplo: En los bits 5 al 2 de un número entero está contenido un dato de 4 bits. Para extraerlo, se podrá aplicar la máscara binaria 0 ... 0111100_2 y, a continuación, hacer un desplazamiento de 2 posiciones.

$$\begin{array}{cccccccccc} & b_N & \dots & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\ \& & 0 & \dots & 1 & 1 & 1 & 1 & 0 & 0 \\ \hline & 0 & \dots & b_4 & b_4 & b_3 & b_2 & 0 & 0 \end{array}$$

$$\begin{array}{cccccccccc} >>2 & 0 & \dots & b_5 & b_4 & b_3 & b_2 & 0 & 0 \\ \hline & 0 & \dots & 0 & 0 & b_5 & b_4 & b_3 & b_2 \end{array}$$

Y, expresado en C sería,

```
resultado = (dato & 0x36) >> 2;
```

Si se tiene un poco de picardía, otra manera correcta de hacer la extracción sería,

```
resultado = (dato >> 2 ) & 0xF;
```

4.4.2 Comprobar estado de un bit

La comprobación del valor de un bit es un caso particular de la extracción de bits donde se elige un solo bit y se comprueba si el valor resultante es 0 o distinto de 0.

Ejemplo: Se desea comprobar el valor del bit 2 de una palabra binaria. Para hacerlo, se empleará el operador AND con el valor binario $0 ... 0100_2$.

$$\begin{array}{cccccccccc} & b_N & \dots & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\ \& & 0 & \dots & 0 & 0 & 0 & 1 & 1 & 1 \\ \hline & 0 & \dots & 0 & 0 & 0 & b_2 & 0 & 0 \end{array}$$

A continuación, se puede comprobar la igualdad o no con 0 del valor resultante. Expresado en C sería,

```
if ((dato & 0x4) == 0) {
    printf("Vale 0\n");
} else {
    printf("Vale 1\n");
}
```

O, también,

```
if ((dato & 0x4) != 0) {
    printf("Vale 1\n");
} else {
    printf("Vale 0\n");
}
```

Es habitual combinar esta operación con la de desplazamiento para facilitar la escritura de código. Por ejemplo,

```
if ((dato & (1 << 2) == 0) {
    printf("Vale 0\n");
} else {
    printf("Vale 1\n");
}
```

4.4.3 Poner bits a 0

Otra necesidad típica es la de poner determinados bits a 0. Para lograrlo, se empleará el operador AND aprovechándose de que b AND 0=0 y b AND 1=b.

Ejemplo: Se desea poner a 0 los bits 4, 3 y 2 de una palabra binaria de 16 bits. Para hacerlo se empleará el operador AND con el valor binario 1 ... 100011_2.

$$\begin{array}{cccccccccc} & b_{15} & \dots & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\ \& & 1 & \dots & 1 & 0 & 0 & 0 & 1 & 1 \\ \hline & b_{15} & \dots & b_5 & 0 & 0 & 0 & b_1 & b_0 \end{array}$$

Expresado en C sería,

```
resultado = dato & 0xFFE3;
```

4.4.4 Poner bits a 1

También suele ser necesario poner determinados bits a 1. Para lograrlo, se empleará el operador OR aprovechándose de que b OR 0=b y b OR 1=1.

Ejemplo: Se desea poner a 1 los bits 5, 2 y 1 de una palabra binaria. Para hacerlo se empleará el operador OR y con el valor binario 0 ... 0100110_2.

$$\begin{array}{cccccccccc} & b_N & \dots & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\ | & 0 & \dots & 1 & 0 & 0 & 1 & 1 & 0 \\ \hline & b_N & \dots & 1 & b_4 & b_3 & 1 & 1 & b_0 \end{array}$$

Expresado en C sería,

```
resultado = dato | 0x26;
```

4.4.5 Complementar bits

Otra operación de máscara muy habitual es la que permite complementar bits individuales. Para lograrlo, se emplea el operador X-OR aprovechándose de que b X-OR 0=b y b X-OR 1=/b.

Ejemplo: Se desean complementar los bits 3, 1 y 0 de una palabra binaria. Para hacerlo, se empleará el operador X-OR con el valor binario 0 ... 001011_2.

$$\begin{array}{cccccccccc} & b_N & \dots & b_5 & b_4 & b_3 & b_2 & b_1 & b_0 \\ \wedge & 0 & \dots & 0 & 0 & 1 & 0 & 1 & 1 \\ \hline & b_N & \dots & b_5 & b_4 & \bar{b_3} & b_2 & \bar{b_1} & \bar{b_0} \end{array}$$

Expresado en C sería,

```
resultado = dato ^ 0xB;
```

5 Cierre

A lo largo de este objeto de aprendizaje se ha visto cómo manipular adecuadamente los bits de un número entero para

Como se ha visto, la manipulación a nivel de bit se hace mediante las operaciones del álgebra de Boole AND, OR, X-OR y complementos que, aplicadas sistemáticamente, permitirán obtener 0's, 1's, complementar bits, etc. allí donde se

necesite. Estas técnicas se llaman máscaras en el ámbito informático y, como se ha visto, son de inmediata aplicación práctica desde el lenguaje C.

6 Bibliografía

La siguiente referencias permiten ampliar/practicar lo visto aquí.

Mask (computing). From Wikipedia, the free enciclopedia. http://en.wikipedia.org/wiki/Mask_%28computing%29

Bit manipulation. From Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Bit_manipulation

Bitwise operation. From Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Bitwise_operation