Operadores de manejo de bits en C

Fernando CLADERA - Eduardo IRIARTE

Version 1 - 24 de abril de 2017

1. Síntesis

Operación	**Código en C**
Poner a `'1'` el bit `n` de la variable `var`	`var |= (1<<n);`
Poner a `'0'` el bit `n` de la variable `var`	`var &= ~(1<<n);`
Toggle del bit `n` de la variable `var`	`var ^= (1<<n);`
Evaluar el bit `n` de la variable `var`	`var & (1<<n)`

2. Introducción

Los operadores de manejo de bits permiten realizar operaciones sobre los bits de una variable. Podemos clasificarlos en:

- Operadores bit a bit
- Operadores de desplazamiento

En ciertos procesadores, las operaciones de manejo de bits están soportadas directamente en el hardware. Por ejemplo, las instrucciones `SBI` y `CLI` en AVR (sólo para E/S) y `BSF` y `BCF` en PIC[1]. En otros procesadores, esta manipulación directa no está soportada, o solamente está permitida para el área de E/S, por lo que se recurre a las técnicas que se explican a continuación[2].

3. Operadores bit a bit

Permiten efectuar operaciones lógicas (AND, OR, NOT, XOR) sobre los bits de una variable. Son particularmente útiles en la programación de microcontroladores, donde deben realizarse operaciones en bits específicos de un registro.

Nombre	**Operador**	**Ejemplo en C**	**Representación en binario**
AND	Ampersand ()	`uint8_t a = 0xF1 & 0x0F;`	`"11110001" & "00001111"`
	`&`	`// a = 0x01`	`= "00000001"`
OR	Pipe	`uint8_t a = 0xF1 | 0x0F;`	`"11110001" | "00001111"`
	`|`	`// a = 0xFF`	`= "11111111"`
XOR	Caret	`uint8_t a = 0xF1 ^ 0x0F;`	`"11110001" ^ "00001111"`
	`^`	`// a = 0xFE`	`= "11111110"`
NOT	Tilde	`uint8_t a = ~0xF1;`	`~"11110001"`
	`~`	`// a = 0x0E`	`= "00001110"`

Nota: `uint8_t` es una variable de tipo *unsigned* de 8 bits. Ver `stdint.h`.

4. Operadores de desplazamiento

Tal como su nombre lo indica, permiten realizar desplazamientos de bits. Por ejemplo, si disponemos del valor `"00010110"` y realizamos un desplazamiento a la izquierda, obtenemos el valor `"00101100"`. Observamos que en este caso, se pierde un bit a la izquierda y se agrega un nuevo bit `'0'` a la derecha.

El desplazamiento de bits a izquierda y derecha, para variables de tipo ***unsigned***, puede entenderse como una multiplicación y una división entera respectivamente. Los

[1] Estas permiten poner a `'1'`, a `'0'` o evaluar el valor de un bit.

[2] La técnica leer/modificar/escribir tiene el inconveniente de que esta secuencia podría ser interrumpida por otro proceso que afecte la misma variable. Esto no pasa en las instrucciones de manipulación directa como las antedichas (que son atómicas). Las de evaluación de bit no tienen este problema.

bits agregados luego del desplazamiento son `'0'`.

- `a << n = a * 2^n`

  Ej: `5 << 2 = "00000101" << 2 = "00010100" = 20`

- `a >> n = a / 2^n`

  Ej: `23 >> 2 = "00010111" = "00000101" = 5`

Nombre	**Operador**	**Ejemplo en C**	**Representación en binario**
Desp. a la izq.	`<<`	`uint8_t a = 0x01 << 4;` `// a = 0x10`	`"00000001" << 4` `= "00010000"`
Desp. a la der.	`>>`	`uint8_t a = 0x10 >> 4;` `// a = 0x01`	`"00010000" >> 4` `= "00000001"`

El desplazamiento de bits en variables de tipo ***signed*** presenta el comportamiento siguiente:

- El **desplazamiento a la derecha** está definido por la implementación del compilador. Los bits agregados luego del desplazamiento pueden ser del valor del bit más significativo (MSB), o pueden ser `'0'`.
- El **desplazamiento a la izquierda** tiene un comportamiento indefinido.

Como conclusión, podemos decir que **no es una buena idea** realizar desplazamientos de bits en variables *signed*, salvo que se conozca con precisión las características de la plataforma y el compilador.

5. Operaciones sobre registros de un microcontrolador

Si se desea configurar el Timer 0 del microcontrolador ATmega328 (AVR 8 - Arduino Uno/Nano), es necesario modificar el registro `TCCR0A`. De la hoja de datos del microcontrolador observamos los diferentes campos del registro:

15.9.1 TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Se observa que mediante este registro pueden modificarse 3 parámetros del Timer 0 (*la función de estos parámetros no es importante en este momento*):

- `COM0Ax` (bits 7:6) - Compare Match Output A Mode
- `COM0Bx` (bits 5:4) - Compare Match Output B Mode
- `WGM0x` (bits 1:0) - Waveform Generation Mode

5.1. Ejemplo 1 - Escritura de registros con AND y OR

Se plantea el problema de modificar los bits de `COM0Bx`, sin modificar el resto de los bits del registro. Específicamente, se requiere escribir un `'1'` en el bit 5 (`COM0B1`) y un `'0'` en el bit 4 (`COM0B0`). Se presenta el siguiente código de ejemplo, utilizando operadores bit a bit:

```
TCCR0A &= 0xEF  // Mediante AND con el valor "11101111" (máscara)
                // se coloca a '0' el bit 4
TCCR0A |= 0x20  // Mediante OR con "00100000" se coloca a '1' el bit número 5
                // los otros bits permanecen en el estado anterior
```

El código propuesto, si bien cumple con el objetivo de la consigna, es complejo debido a la necesidad de especificar los valores hexadecimales de las máscaras.

5.2. Ejemplo 2 - Escritura de registros con desplazamiento de bits

Podemos utilizar los desplazamientos de bits para analizar el ejemplo anterior:

```
TCCR0A &= ~(1<<4);  // Se coloca a '0' el bit 4
TCCR0A |= 1<<5;     // Se coloca a '1' el bit número 5
```

Observamos que

- `~(1<<4) = 0xEF`
- `1<<5 = 0x20`

No obstante, este ejemplo no es fácilmente legible. Es necesario contar con la hoja de datos del microcontrolador para saber qué campos están siendo modificados en `TCCR0A`.

5.3. Ejemplo 3 - Escritura de registros mediante headers

Generalmente, las cabeceras (*headers*, .h) que definen los registros de un microcontrolador especifican los diferentes campos. Para el ATmega 328, la cabecera que define el microcontrolador establece las siguientes constantes:

```
#define WGM00 0
#define WGM01 1
#define COM0B0 4
#define COM0B1 5
#define COM0A0 6
#define COM0A1 7
```

Por consiguiente, podemos reescribir el ejemplo anterior como

```
TCCR0A &= ~(1<<COM0B0); // Colocamos a '0' el bit correspondiente a COM0B0
TCCR0A |= 1<<COM0B1;    // Colocamos a '1' el bit correspondiente a COM0B1
```

En este código es fácil ver la operación sobre los registros, y analizar el resultado final de `COM0Bx` en `TCCR0A`. Por consiguiente, **se trata de la respuesta más adecuada para el problema propuesto**.

5.4. Ejemplo 4 - Toggle de bits de un registro

Utilizando el operador XOR podemos cambiar el estado de un bit (*toggle*). Por ejemplo, si deseamos cambiar el estado del bit `COM0B1` de `TCCR0A`, podemos escribir:

```
TCCR0A ^= (1<<COM0B1);    // Toggle de bit COM0B1
```

5.5. Ejemplo 5 - Evaluación del estado de un bit de un registro

Finalmente, si deseamos evaluar el estado de un bit, podemos hacer un simple AND bit a bit. Por ejemplo, para evaluar el estado de `COM0B0` de `TCCR0A`, podemos escribir

```
if (TCCR0A & (1<<COM0B0)) {
  // Detectamos que COM0B0 está en '1'
  printf("TCCR0A - COM0B0: 1\n");
} else {
  // Detectamos que COM0B0 está en '0'
  printf("TCCR0A - COM0B0: 0\n");
}
```

6. Links útiles

http://www.catonmat.net/blog/low-level-bit-hacks-you-absolutely-must-know/
http://graphics.stanford.edu/~seander/bithacks.html