Operaciones con bits

Podemos hacer desde C operaciones entre bits de dos números (AND, OR, XOR, etc). Vamos primero a ver qué significa cada una de esas operaciones.

Operación	Qué hace	En C	Ejemplo
Complemento (not)	Cambiar 0 por 1 y viceversa	~	~1100 = 0011
Producto lógico (and)	1 sólo si los 2 bits son 1	&	1101 & 1011 = 1001
Suma lógica (or)	1 si uno de los bits es 1	|	1101 | 1011 = 1111
Suma exclusiva (xor)	1 sólo si los 2 bits son distintos	^	1101 ^ 1011 = 0110
Desplazamiento a la izquierda	Desplaza y rellena con ceros	<<	1101 << 2 = 110100
Desplazamiento a la derecha	Desplaza y rellena con ceros	>>	1101 >> 2 = 0011

Veamoslo todos en un mismo ejemplo.

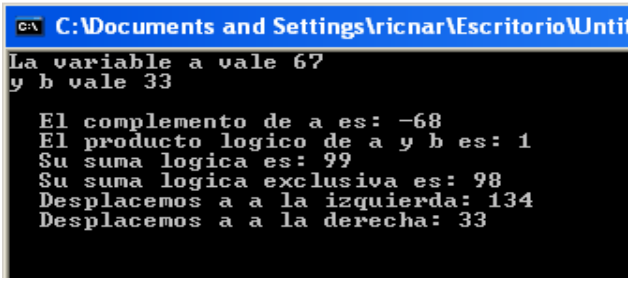
```
#include <stdio.h>

int main() {
   int a   = 67;
   int b   =  33;

   printf("La variable a vale %d\n", a);
   printf("y b vale %d\n\n", b);
   printf("  El complemento de a es: %d\n", ~a);
   printf("  El producto logico de a y b es: %d\n", a&b);
   printf("  Su suma logica es: %d\n", a|b);
   printf("  Su suma logica exclusiva es: %d\n", a^b);
   printf("  Desplacemos a a la izquierda: %d\n", a << 1);
   printf("  Desplacemos a a la derecha: %d\n", a >> 1);
   getchar();
   return 0;

}
```

Al correrlo vemos los resultados de las operaciones.

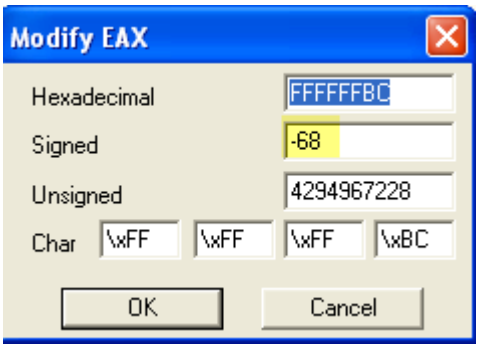


Llevado a binario podemos comprobar fácilmente las operaciones:

A=1000011 = 43h
B=100001 = 21h

El complemento de A o sea NOT (A) es FFFFFFFFh – 43h = **FFFFFFBCh**

Tengo justo un Olly abierto que es rápido para hacer conversiones y veo que FFFFFFBCh es -68h si lo tomamos con signo.



Luego hace el producto lógico o AND entre ambos, los encolumno en forma binaria.
Sabemos que mirando columna por columna solo tendrá la misma resultado uno si ambos bits de la misma son 1 sino cualquier otra combinación da cero.

A=1000011
B=0100001

C=0000001

Allí C sera el resultado y valdrá uno ya que solo la ultima columna de la operación tiene ambos bits encendidos en uno, de esta forma el producto lógico o AND entre ambos da 1h.

Luego viene la suma lógica o OR encolumnamos en binario nuevamente sabiendo en este caso que la suma lógica mientras que haya un uno en algún bit de una columna ya el resultado sera 1.

A=1000011
B=0100001

C=1100011

Que es igual a 63h o sea 99

Luego hacer la suma exclusiva o XOR que sabemos que solo da 1 si ambos bits de la columna son distintos.

A=**1**0000**1**1
B=0**1**00001

C=**11**000**1**0

Que es igual a 62h o sea 98

Luego viene el desplazamiento a la izquierda de A , corremos todo para la izquierda un lugar y le agregamos un cero por la derecha para rellenar el espacio que quedo vacío al correr.

A=1000011

C=10000110

O sea el resultado es 86h o sea 134

Lo mismo el desplazamiento hacia la derecha

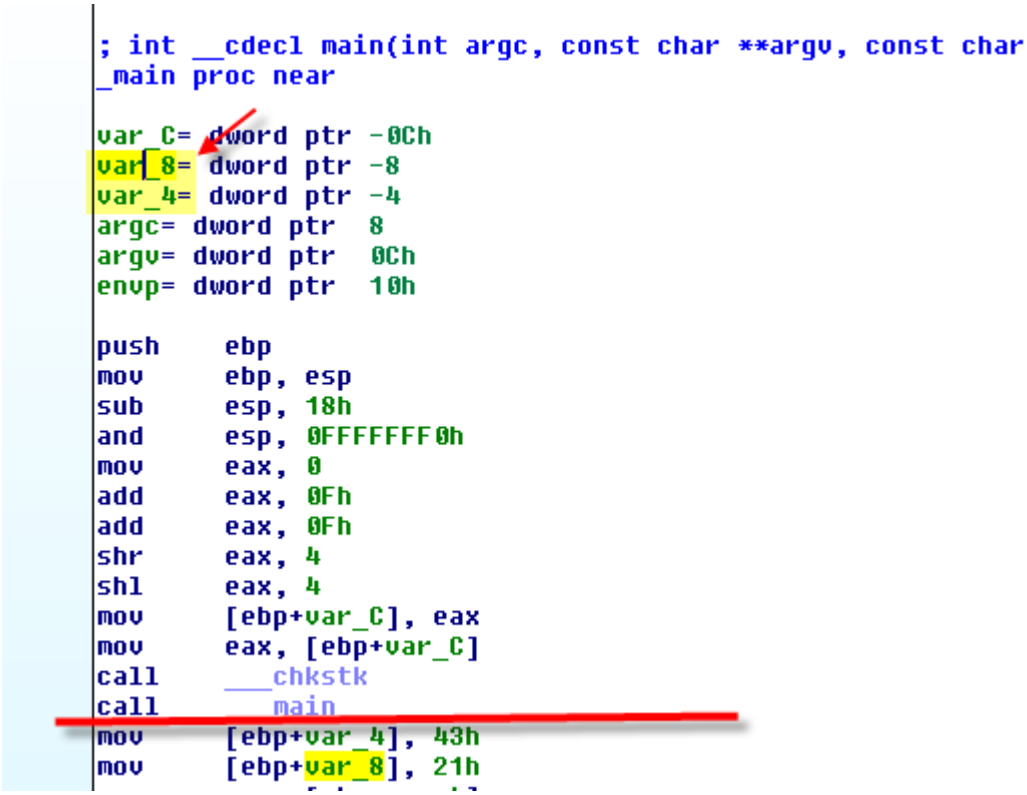
A=100001**1** **-- al desplazar a la derecha ese byte se cae fuera del limite**

C=**0**100001 - **y es rellenado con un cero por el otro lado.**

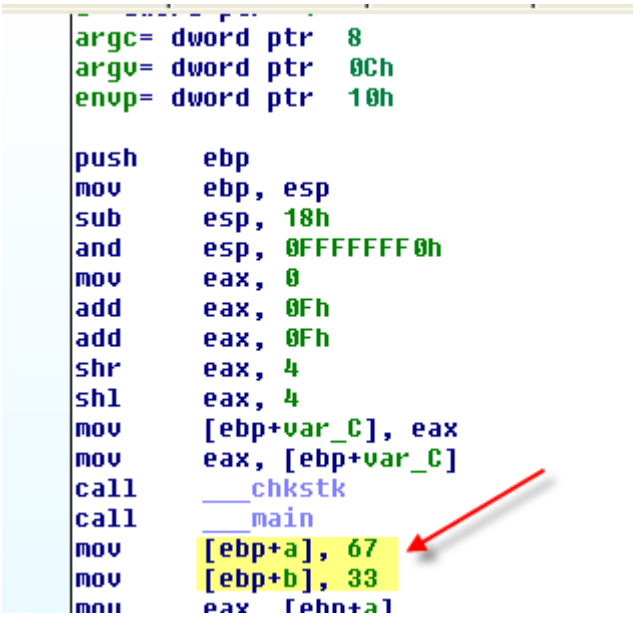
Que es 33 decimal

Es de notar que en el desplazamiento hay que tener en cuenta el limite máximo a cada lado pues al desplazar a la izquierda como solo desplazamos uno no pasamos el limite máximo de bits, si lo hiciéramos, los que se cayeran fuera se perderían siendo reemplazados por los ceros que se rellenan por el otro lado, en el caso del desplazamiento a la derecha es igual solo que allí estamos en el limite, así que al desplazar solo un lugar el ultimo byte se cae fuera siendo rellenado con un cero por delante.

Bueno todo muy lindo veamos como se ve todo esto en IDA.



Allí vemos donde empieza realmente lo nuestro y las dos variables que son int o sea ocupan un dword cada una, comencemos a renombrar.



Lo pusimos en decimal como en el código fuente y le cambiamos los nombres obvio que si no tenemos el fuente y estamos reverseando le pondremos los nombres que se nos antoje.

```
mov     [ebp+a], 67
mov     [ebp+b], 33
mov     eax, [ebp+a]
mov     [esp+4], eax
mov     dword ptr [esp], offset aLaVariableAVal ; "La variable a vale %d\n"
call    printf
mov     eax, [ebp+b]
mov     [esp+4], eax
mov     dword ptr [esp], offset aYBValeD ; "y b vale %d\n\n"
call    printf
```

Lo primero que hace es imprimir los valores de **a** y **b** en dos printf usando **%d** en el format string.

```
mov     eax, [ebp+a]
not     eax
mov     [esp+4], eax
mov     dword ptr [esp], offset aElComplementoD ; "  El complemento de a es: %d\n"
```

Luego imprime el valor del complemento de a, allí vemos como hace NOT que es la instrucción para hallarlo.

```
mov     eax, [ebp+b]
and     eax, [ebp+a]
mov     [esp+4], eax
mov     dword ptr [esp], offset aElProductoLogi ; "  El producto logico de a y b es: %d\n"
```

Luego hace el producto lógico o AND para ellos mueve **b** a **EAX** y luego hace **AND** de **EAX** con **a**, el resultado lo manda a imprimir con printf.

```
mov     eax, [ebp+b]
or      eax, [ebp+a]
mov     [esp+4], eax
mov     dword ptr [esp], offset aSuSumaLogicaEs ; "  Su suma logica es: %d\n"
call    printf
mov     eax, [ebp+b]
xor     eax, [ebp+a]
mov     [esp+4], eax
mov     dword ptr [esp], offset aSuSumaLogicaEx ; "  Su suma logica exclusiva es: %d\n"
call    printf
```

De la misma forma halla la suma lógica **OR** y la suma lógica exclusiva **XOR** y los manda a imprimir.

```
call    printf
mov     eax, [ebp+a]
add     eax, eax
mov     [esp+4], eax
mov     dword ptr [esp], offset aDesplacemosAAL ; "  Desplacemos a a la izquierda: %d\n"
call    printf
```

Luego realiza el desplazamiento a la izquierda que en el caso de desplazar uno es equivalente a multiplicar por 2, recordemos que a valía 67 y el resultado era 134.

Si en el código fuente cambiáramos que se desplace dos posiciones vemos que ahora si necesita usar el SHL o SAL que es la instrucción de desplazamiento a la izquierda.

http://ensam1.blogspot.com/2005/09/64-corrimiento-y-rotacion.html

allí mas de SHL SAL SHR y SAR

y allí usa SAR para el ultimo desplazamiento a la derecha y con eso termina el ejemplo que es bien sencillo.

Una lección sencilla y un ejercicio sencillo para reversear sobre el tema jeje.

Hasta la parte siguiente
ricnar