



**CENTRO DE CIENCIAS BÁSICAS**  
**DEPARTAMENTO DE SISTEMAS ELECTRÓNICOS**  
**LENGUAJE ENSAMBLADOR**  
**7° "A"**

**PRÁCTICA 4**

**Profesor: Cristian Jael Mejía Aguirre**

**Alumno: Joel Alejandro Espinoza Sánchez**

**Fecha de Entrega:** Aguascalientes, Ags., 14 de octubre de 2021

# Práctica 4

**Objetivo:** Realizar enmascaramiento AND y OR de un byte en lenguaje ASM.

**Desarrollo:** Anterior a escribir el código se buscaron los dos comandos a usar en la práctica, en este caso, los evidentes AND y OR. Posteriormente se redactó el código con un ejemplo para cada enmascaramiento:

```
%include "io64.inc"

section .text
global CMAIN
CMAIN:
    xor rax, rax
    ; Enmascaramiento AND
    mov rax, 1110b
    mov rbx, 1011b ; Apagamos el segundo bit (de izquierda a derecha)
    PRINT_STRING "rax antes del enmascaramiento"
    NEWLINE
    PRINT_DEC 1, rax
    NEWLINE
    and rax, rbx
    PRINT_STRING "rax despues del enmascaramiento"
    NEWLINE
    PRINT_DEC 1, rax
    NEWLINE
    NEWLINE
    ; Enmascaramiento OR
    mov rax, 1000b
    mov rbx, 0010b ; Encendemos el tercer bit (de izquierda a derecha)
    PRINT_STRING "rax antes del enmascaramiento"
    NEWLINE
    PRINT_DEC 1, rax
    NEWLINE
    or rax, rbx
    PRINT_STRING "rax despues del enmascaramiento"
    NEWLINE
    PRINT_DEC 1, rax
    NEWLINE
    ret
```

El programa anterior daba la siguiente salida:

```
Salida

rax antes del enmascaramiento
14
rax despues del enmascaramiento
10

rax antes del enmascaramiento
8
rax despues del enmascaramiento
10
```

Observemos que para el primer caso, que es el AND, los números 14 y 10 en binario se representan como 1110 y 1010 respectivamente, por lo que sí funcionó el enmascaramiento para apagar el segundo bit de izquierda a derecha.

De igual forma, en el caso de OR, los números 8 y 10 se representan como 1000 y 1010 respectivamente, es decir, se encendió correctamente el tercer bit de izquierda a derecha.

Dentro del programa SASM, el programa se veía de la siguiente forma:

```
1  %include "io64.inc"
2  |
3  section .text
4  global CMAIN
5  CMAIN:
6      xor rax, rax
7
8      ; Enmascaramiento AND
9      mov rax, 1110b
10     mov rbx, 1011b ; Apagaremos el segundo bit (contando de izquierda a derecha)
11     PRINT_STRING "rax antes del enmascaramiento"
12     NEWLINE
13     PRINT_DEC 1, rax
14     NEWLINE
15     and rax, rbx
16     PRINT_STRING "rax despues del enmascaramiento"
17     NEWLINE
18     PRINT_DEC 1, rax
19     NEWLINE
20     NEWLINE
21
```

```

22      ; Enmascaramiento OR
23      mov rax, 1000b
24      mov rbx, 0010b ; Encenderemos el tercer bit (contando de izquierda a derecha)
25      PRINT_STRING "rax antes del enmascaramiento"
26      NEWLINE
27      PRINT_DEC 1, rax
28      NEWLINE
29      or rax, rbx
30      PRINT_STRING "rax despues del enmascaramiento"
31      NEWLINE
32      PRINT_DEC 1, rax
33      NEWLINE
34
35      ret

```

**Conclusión:** Con la práctica pudimos darnos cuenta de la implementación de un enmascaramiento que puede ser realmente sencillo en ensamblador y a su vez muy útil, pues la manipulación de bits nos permite resolver muchos problemas.