



# Práctica 13a

## Objetivo

Demostrar el conocimiento necesario para implementar y ejecutar un código en lenguaje c y ensamblador, así mismo acceder a los puertos de la tarjeta T-Juino por medio de las bibliotecas.

## Desarrollo

Realice lo siguiente.

1. Tomando de ejemplo el archivo main.c

```
#include <stdio.h> extern unsigned int _test

(unsigned int, unsigned int);

int main(void)
{ printf("%d\n", _test (85,5)); /* you are calling div here, not _test */
  return 0;
}
```

2. Archivo fun.asm

```
global _test

_test:
push    ebp mov    ebp, esp
mov     eax, [ebp+8] xor    edx,
edx div    dword [ebp+12]
mov     esp, ebp pop    ebp ret
```

3. Correr en terminar las siguientes líneas

```
nasm -f elf -o fun.o fun.asm
gcc -m32 -c -o main.o main.c
gcc -m32 -o run fun.o main.o
./run
```

4. Agregar captura de pantalla del resultado de la ejecución impreso en terminal.

5. Apoyándose en el ejemplo anterior crear un archivo P13.asm donde estarán las subrutinas y el código necesario para hacer el llamado a estas desde lenguaje c:

- a) **setBit**: activa un bit del 0 al 7 a un dato de 1 Byte recibidos ambos por parámetro, está retorna únicamente el dato modificado.
- b) **clearBit**: desactiva un bit del 0 al 7 a un dato de 1 Byte recibidos ambos por parámetro, está retorna únicamente el dato modificado.
- c) **notBit**: invierte un bit del 0 al 7 a un dato de 1 Byte recibidos ambos por parámetro, está retorna únicamente el dato modificado.
- d) **testBit**: revisa el estado de un bit del 0 al 7 a un dato de 1 Byte recibidos por parámetro, está retorna únicamente el valor del bit.

6. Crear un archivo P13C.c donde se llamarán a las funciones de ensamblador como funciones externas.

- a) Se les enviará el dato que será modificado de un Byte y el número del bit en el que se trabajara.
- b) Para probar el funcionamiento se deberá hacer lo siguiente: c)

- 1. Comenzar con un dato que será 0B3h al cual se le debe activar el bit 3.
- 2. El dato resultante del paso 1 desactivarle los bits 1, 4 y 7.
- 3. El dato resultante del paso 2 invertirle los bits 2, 3, 4, 5 y 7.
- 4. El dato resultante del paso 3 revisar el estado de los bits 1, 4 y 7.

- d) Mostrar en terminar como hexadecimal cada cambio que se va realizando por cada modificación de bit.

## Screenshot de consola

```
lnasm -f elf -o p13.o p13.asm
lgcc -m32 -c -o p13C.o p13C.c
lgcc -m32 -o run p13.o p13C.o
!./run
```

```
ERROR: ld.so: object '/usr/lib/x86_64-linux-gnu/libtcmalloc.so.4' from LD_PRELOAD cannot be preloaded (wrong ELF class: ELF
x = b3
_setBit(x,3) = bb
_clearBit(x,1) = b9
_clearBit(x,4) = a9
_clearBit(x,7) = 29
_notBit(x,2) = 2d
_notBit(x,3) = 25
_notBit(x,4) = 35
_notBit(x,5) = 15
_notBit(x,7) = 95
_testBit(x,1) = 0
_testBit(x,4) = 1
_testBit(x,7) = 1
```

0s completed at 10:43 AM

### Codigo p13C.c

```
#include <stdio.h>
extern unsigned char _setBit (unsigned char, unsigned char);
extern unsigned char _clearBit (unsigned char, unsigned char);
extern unsigned char _notBit (unsigned char, unsigned char);
extern unsigned char _testBit (unsigned char, unsigned char);
int main(void)
{
    unsigned char x = 0x0b3;
    printf("x = %x\n", x);
    printf("_setBit(x,3) = %x\n", x = _setBit(x,3));
    printf("_clearBit(x,1) = %x\n", x = _clearBit(x,1));
    printf("_clearBit(x,4) = %x\n", x = _clearBit(x,4));
    printf("_clearBit(x,7) = %x\n", x = _clearBit(x,7));
    printf("_notBit(x,2) = %x\n", x = _notBit(x,2));
    printf("_notBit(x,3) = %x\n", x = _notBit(x,3));
    printf("_notBit(x,4) = %x\n", x = _notBit(x,4));
    printf("_notBit(x,5) = %x\n", x = _notBit(x,5));
    printf("_notBit(x,7) = %x\n", x = _notBit(x,7));
    printf("_testBit(x,1) = %x\n", _testBit(x,1));
    printf("_testBit(x,4) = %x\n", _testBit(x,4));
    printf("_testBit(x,7) = %x\n", _testBit(x,7));

    return 0;
}
```

### Codigo p13.asm

```
global _setBit
global _clearBit
global _notBit
global _testBit
_setBit:
    push ebp
    mov ebp, esp
    mov cl,[ebp+12]
    mov eax,1
    rol al,cl
    or al,[ebp+8]
    pop ebp
ret
_clearBit:
    push ebp
    mov ebp, esp
    mov cl,[ebp+12]
    mov eax,0FFFFFFFh
    rol al,cl
    and al,[ebp+8]
    pop ebp
ret
_notBit:
    push ebp
    mov ebp, esp
    mov cl,[ebp+12]
    mov eax,1
```

```
    rol al,cl
    xor al,[ebp+8]
    pop ebp
ret
_testBit:
    push ebp
    mov ebp, esp
    mov cl,[ebp+12]
    mov eax,1
    rol al,cl
    test al,[ebp+8]
    jz .cero
    mov eax,1
    jmp .end
.cero:
    mov eax,0
.end:
    pop ebp
ret
```

### Conclusiones y comentarios

Es bastante interesante que exista la posibilidad de mezclar código ensamblador con otros lenguajes, en este caso lenguaje C ya que ofrece la posibilidad de crear un código lo más eficiente posible y correrlo en tu programa en lenguaje C