



Universidad autónoma de baja California

Ingeniería en computación

Microcontroladores

Practica 4 C-ASM puertos I/O

Erik garcia Chávez 01275973

Jesus adan Garcia Lopez

7 de marzo del 2025

Ensamblador en línea para GCC:

En formato básico para incluir instrucciones de ASM en C sigue la siguiente sintaxis básica:

```
__asm__("nop")
```

Para dar un ejemplo esto inserta una instrucción nop de ASM en este caso del 2560 que quema 1 ciclo esto en C

Esto mismo se puede hacer como macro, una macro se expande textualmente durante la fase de preprocesamiento antes de la compilación, esto funciona cuando se usa código repetitivo. El procesador reemplaza textualmente la macro por su contenido.

Estas se declaran con **#define**

Como puede ser:

```
#define DELAY_1US() __asm__ volatile ("nop\n\t nop\n\t nop\n\t nop\n\t")
```

En C tan solo se llama al nombre de esta macro como si se llamara desde una función

Llamada en C:

```
DELAY_1US();
```

Cuando se ingresan más de una línea de ASM en C, estas deben de estar en otro formato dado de que compilador las pasa tan y cual como están, entre estos métodos

UInt8\_t resultado

```
__asm__ (  
"add &[res], %[in1], %[in2] \n\t"  
: "=r" (resultado) // salida en el registro ("=r")  
: "r"(a) , "r"(b) // entrada desde variables a y b  
);
```

cada línea va entre comillas dobles y el retorno de carro y el tabulador es porque gcc pasa las líneas tal cual

cómo se asignan variables de C a registros de ASM:

r -> registro general (R0-R31)

d -> registro de bajo orden (R15-R31)

w -> registro para pares (R24,R26,etc.)

l -> constante entera (0-255)

l -> etiqueta de dirección baja

muchas veces de declarar como **volatile**, para evitar que el compilador optimice el acceso al bloque de código ya que este valor puede cambiar de manera impredecible, Es especialmente relevante en sistemas embebidos

### ***convención de llamadas a funciones en avr-gcc***

- 1- Los parámetros se pasan a través de registros o en pila, en orden de izquierda a derecha.

Tipo	Registros Asignados
<b>8-bit</b> (char)	R24 , R22 , R20 , R18 , ...
<b>16-bit</b> (int)	R25:R24 , R23:R22 , R21:R20 , ...
<b>32-bit</b> (long)	R25:R24:R23:R22 , R21:R20:R19:R18 , ...
<b>Punteros</b>	R25:R24 (16-bit), R23:R22 (siguiente), etc.
<b>Structs/Arrays</b>	Por referencia (puntero en R25:R24 , R23:R22 ).

2- El retorno de valores se almacena en registros específicos:

Tipo	Registros
8-bit (char)	R24
16-bit (int)	R25 : R24
32-bit (long)	R25 : R24 : R23 : R22
Punteros	R25 : R24 (dirección 16-bit)
Structs/Arrays	Espacio reservado por el llamante (puntero en R25 : R24 ).

### 3- Los registros preservados (callee-save):

Deben ser guardado por la función llamada si los modifica, R2-R17, R28-R29

**Los registros temporales (caller-save):** pueden ser modificados sin preservarse R18-R27, R30-R31

4- Para la llamada de una subrutina de ASM desde C se hace, en C declarando en el prototipo de la función que esta será una función externa se hace con la palabra reservada **extern**

El archivo con código ASM debe de estar en un archivo con la extensión .s o .S,

Se define las subrutinas con **.global** para que sea visible desde C

Ejemplo:

```
#include <avr/io.h> ; Opcional, pero útil para direcciones de registros
```

```
.section .text ; Define la sección de código
```

```
.global suma_asm ; Hace la función visible para C
```

```
suma_asm:
```

```
; Parámetros: a (R24), b (R22)
```

```
; Retorno: R24 (a + b)
```

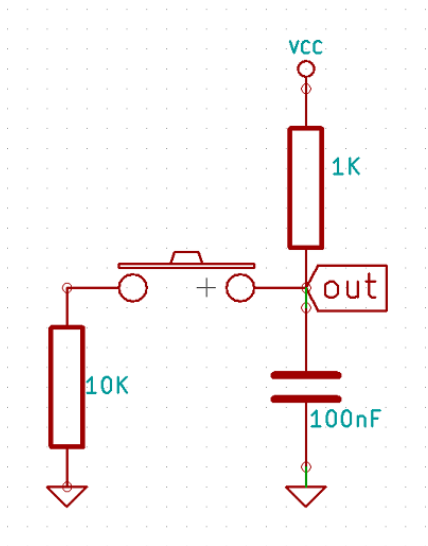
```
ADD R24, R22 ; Suma R24 = R24 + R22
```

RET ; Retorna el resultado en R24

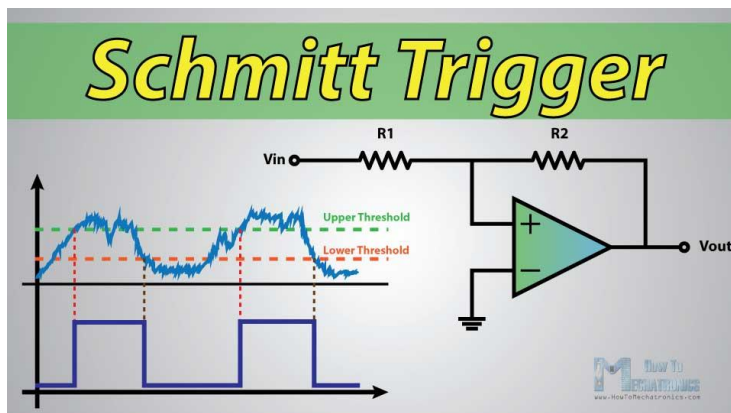
Si se tiene mas de 1 subrutina estas también deben de ir declaradas en con **.global**  
**<nombre\_etiqueta>**

### Técnicas de anti rebote:

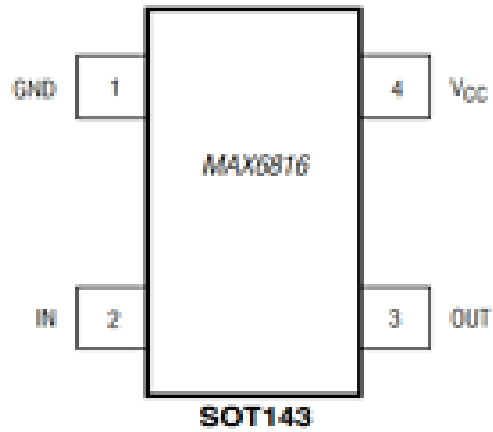
**Filtro RC:** un circuito RC puede ser valores como de  $10K\Omega + 0.1 \mu F$  lo que filtra el ruido al presionar el botón, es simple y barato de implementar solo que su limitación es que presentara retraso en la respuesta



**Schmitt Trigger:** permite reducir los errores producidos por señales ruidosas, nos retorna 2 niveles de umbral, alto y bajo. Lo que nos da una onda cuadrada.



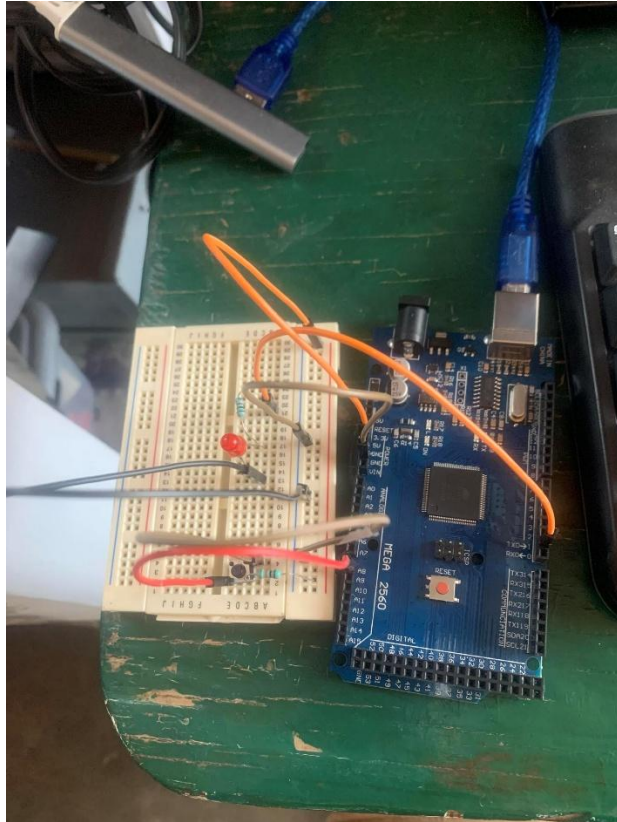
**IC MAX6816:** es un IC que elimina el rebote de los interruptores mecánicos permitiendo que se interconecten los sistemas digitales.

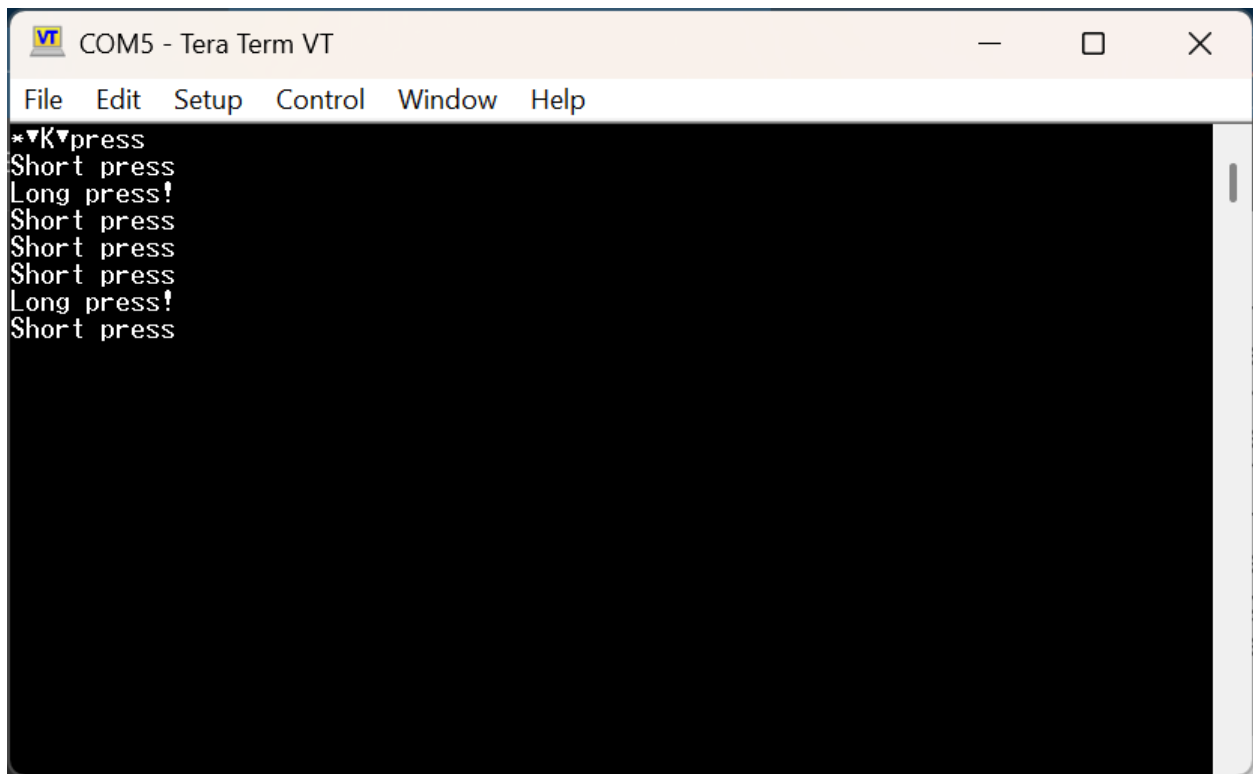


**Técnica por software, Delay:** tras detectar un cambio esperar un tiempo puede ir de 10-50 ms antes de leer el estado, esta función no tiene costo alguno pero bloquea la ejecución en ese lapso de tiempo

Ejecución de código:

Se usa un push button configurado como pull up con una resistencia de  $10K\Omega$ , para dar la señal si se presiona por un largo periodo o corto periodo





```
COM5 - Tera Term VT
File Edit Setup Control Window Help
*▼K▼press
Short press
Long press!
Short press
Short press
Short press
Long press!
Short press
```

### Conclusiones:

Durante la practica pudimos poner en práctica los conocimientos sobre puertos, lo que mas se me dificulto o con que tuve serios problemas es sobre como saber el estado de un puerto cuando es de entrada, pero leyendo con mas detenimiento me puede dar cuenta con el registro PIN ese devuelve el estado físico del puerto, por lo que de ahí en fuera tuve problemas menores sobre la lógica mas que nada, por lo que se pudo realizar la practica y poner a prueba los conocimientos sobre puertos.

### Referencias:

<https://webs.um.es/einiesta/miwiki/lib/exe/fetch.php%3Fmedia%3Dasm-gcc-linea.pdf>