



Universidad autónoma de baja california

Ingeniero en computación

microcontroladores

practica 4

Maestro: Jose Isabel Garcia Rocha

Erik Garcia Chávez 01275863

30 de septiembre del 2024

Teoría:

Programación en lenguaje C en microcontroladores

en lenguaje de programación C es el lenguaje por excelencia cuando se trata de programar estos circuitos integrados, por la alta capacidad que tiene C en el manejo en memoria, manejo de registros, pines de I/O. así como a la facilidad de comunicarse con el ensamblador.

Convención de llamadas a funciones en C en gcc para AVR (avr-gcc)

En el entorno de desarrollo con avr-gcc para microcontroladores AVR, la convención de llamadas a funciones se refiere a cómo se pasan los argumentos a las funciones y cómo se retorna el valor. Aquí hay algunas convenciones comunes:

Convención de llamada por defecto: En C, la convención de llamada por defecto es cdecl, donde los argumentos se pasan de derecha a izquierda y el llamador limpia la pila después de la llamada.

Convención de llamada stdcall: Similar a cdecl, pero el llamador no limpia la pila; la función llamada se encarga de ello. Esta convención es común en las API de Windows.

Convención de llamada fastcall: Los primeros argumentos se pasan en registros, lo que puede hacer la llamada más rápida, pero solo es útil para un número limitado de argumentos.

Convención de llamada naked: No se generan secuencias de entrada/salida por el compilador, lo que permite al programador manejar todo el código de la función en ensamblador

Instrucciones:

Haciendo uso de los pasos descritos en la Practica 1 crear un nuevo proyecto, pero esta vez para hacer uso del lenguaje C con base en el Listado 1 (Prac4.c) y del lenguaje ensamblador con el Listado 2 (Delay.S). El programa tiene como fin el mostrar el llamado a funciones escritas en lenguaje ensamblador desde código escrito lenguaje C. ensamblador.

Desarrollo:

En primera instancia contamos con la declaración de la función, la cual es llamada de C a ensamblador, la función en ASM se encargara de hacer el retardo de 0-255 ms, con la excepción que no se podrá hacer de manera efectiva para 0ms, porque para 0 ms se necesitan hacer 0 ticks del procesador, lo cual no puede realizarse de manera efectiva, por el solo hecho de ir de C a ASM consta de 6 ticks, más aun los Branch así como el ret, al final podría ser alrededor de 20 ticks de procesador, el solo procesar y hacer una excepción sobre que no se puede hacer un delay de 0ms. Para 1ms no es tan exacto, pero para todos los demás se pretendió ser lo más exacto posible.

La función recibe un argumento, el cual es el tiempo que se quiere procesar, el cual será representado en mS., la función es de tipo void ya que no necesito que proese nada, tan solo que queme tiempo de procesador.

```
#include<avr/io.h>
extern void Delay(uint8_t argumento); /* funcion prototype de Delay */
int main(void){

    Delay(52);

    uint8_t encendido, apagado;
    uint8_t variable=100;

    /* configurar de salida el bits 2, 3 y 4 d el PORTB*/
    DDRB = 0x1C; /* DDRC= 0001 1100 */

    /* presentar valor inicial en puerto LEDs off*/
    PORTB = 0x1C;

    while(1){ //DDRC= 00111000

        /* Encender LED PB2 */
        PORTB &= ~(1 << 2);
        for (encendido = 0; encendido < 10; encendido++) {
            Delay(variable);
        }
    }
}
```

C:\practicasUcontralor\practicas\prac4.c

C:\practicasUcontralor\practicas\Delay.S

Delay.S

Establecemos la etiqueta con la cual puede ser llamada por cualquier archivo. Siempre y cuando se enlace el objeto con este archivo claramente, pero eso AVR lo hace por nosotros.

R24 es el registro que trae nuestro parámetro a ASM, por lo que lo guardo, lo realice así porque si no hacia eso me daba errores.

Hacemos la primera comparación que es verificar antes que realizar cualquier operación lógica del delay es saber si el número no es 0, si es cero, entonces este tiene que volver a C, porque es un no válido, en el sentido que no se puede realizar un delay de 0 ms con 0 ticks. Por lo que regresamos a C. tan solo, por lo que si no es 0 entonces el programa puede seguir.

```
#define __SFR_ASM_COMPAT 1
#define __SFR_OFFSET 0
#include <avr/io.h>

;r24 contiene el primero argumento en la función en C.
.global Delay
Delay:
    ;ld r20,X
    ;push r27

    ;guardo r24 en 1 pila
    push r24
    mov r23,r24
    ;comparo si el dato que se envió en 1, entonces es un caso especial

    ;antes que nada debo comprobar si el argumento es igual a cero, si es así, debe volver a C
    ;por lo que este es el único que no va a dar exactamente 0ms, tal solo vuelve
    cpi r24,0x00
    breq final

    cpi r24,0x01
```

Se tiene una segunda condicional, en la cual si el parámetro que se recibió es 1, entonces se manda a llamar a una función especial, la cual calcula 16000 ciclos de reloj, pero ese toma en cuenta todos los ticks se se realizan desde que se manda a llamar de C hasta el regreso, el RET, pero si R24 no es 1 entonces es cualquier otro numero por lo que se calcula los ciclos exactos a excepción del último, como se puede ver la etiqueta, lo que hace es un ciclo que manda a llamar a la etiqueta "**delay_general**", el delay_general calcula mS a mS, hace 16000 ciclos exactos regresa a esta etiqueta "ciclo", hace la comparación si R24 no es 1 entonces se vuelve a hacer la llamada al "**delay_general**" cuando ya se está en la última iteración se llama a la etiqueta que tiene el delay que dura igual 16000 ciclos pero toma en cuenta en su cálculo todos los ciclos que se toma para decisiones, así como el CALL y el RET de C a ASM y devuelta.

```
        cpi r24,0x01
        breq un_mSeg

ciclo:

        ;rcall delay_general ;4 ciclos
        rcall delay_general

        dec r24

        cpi r24,0x01
        brne ciclo ; pro esto hacer que el delay de 1ms cuente con los 2 ciclos que este se va
        ;cuando este no se cumpla estara a 15999
        nop ;con este se puede arreglar?

        cpi r24,0x01
        breq un_mSeg

final:
        pop r24 ; 2ciclos
        ret
```

Ciclos un “un_mSeg”:

Se utiliza 2 ciclos, para hacer el retardo de 16000 ciclos, pero esta toma en cuenta, todos los ticks se hicieron hasta antes de llegar a la subrutina, las comparaciones, el tick de ir de C a AMS, etc. por lo que puede ir de 11 a 14 ticks, previos, 11 ticks en el caso de que se mande 1 mS como retardo, por lo que no seguirá a la otra subrutina que calcula los 16000 ticks exactos,

```
un_mSeg:
    ldi r23,27 ;1 ->
    ldi r22,117 ;1

nxt0:
    nop ;
    nop
    dec r22
    brne nxt0
    dec r23
    nop
    nop
    brne nxt1
    rjmp final

nxt1:
    ldi r22, 117
    rjmp nxt0
```

La fórmula calculada a partir del siguiente ciclo para calcular los ticks que necesito, necesito 16000 ticks de reloj para 1mS

Formula=1 +1 + 1XY + 1XY+ 1XY + 2X(Y-1) + 1X+ 1X + 1X+ 2(X-1) +4 + 1X + 4X

Que despejándola nos queda

Ticks = 8X + 5XY + 4

Sustituyendo los valores con X = 27 y Y=117, nos queda

Ticks = 8(27) + 5(27+117) +4 = 16015

Nos acercamos mucho

Se tiene a “**delay_general**” el cual calcula los 16000 ciclos exactos para todos los mS antes de llegar al último, el delay es muy similar a cunado es para el ultimo mS, solo que en este sus valores cambian, y se agregan más nops, dado que se requiere llegar a más ciclos.

```

delay_general:
    ldi r18, 26      ;1 ->X
    ldi r19, 151     ;1 -> Y
Rnxt0:
    nop ;1XY
    dec r19 ;1XY
    brne Rnxt0 ;2X(Y-1)
    dec r18 ;1X
    nop;1X
    nop;1X
    nop;1X
    nop;1X
    nop;1X
    brne recarga ;2(X-1)
    ret             ;5

recarga:
    ldi r19, 151     ;1X
    nop;1X
    rjmp Rnxt0 ;4X

```

La ecuación para calcular la formula quedo de la siguiente manera:

Ticks delay general=

$1 + 1 + 1XY + 1XY + 2X(Y-1) + 1X + 1X + 1X + 1X + 1X + 1X + 2(X-1) + 5 + 1X + 1X + 4X$

Donde si la despejamos nos dejaría con

Ticks delay general= $12X + 4XY + 5$, donde si sustituimos los variables con los valores de X y Y que asignamos a r18 y r19 respectivamente, tenemos:

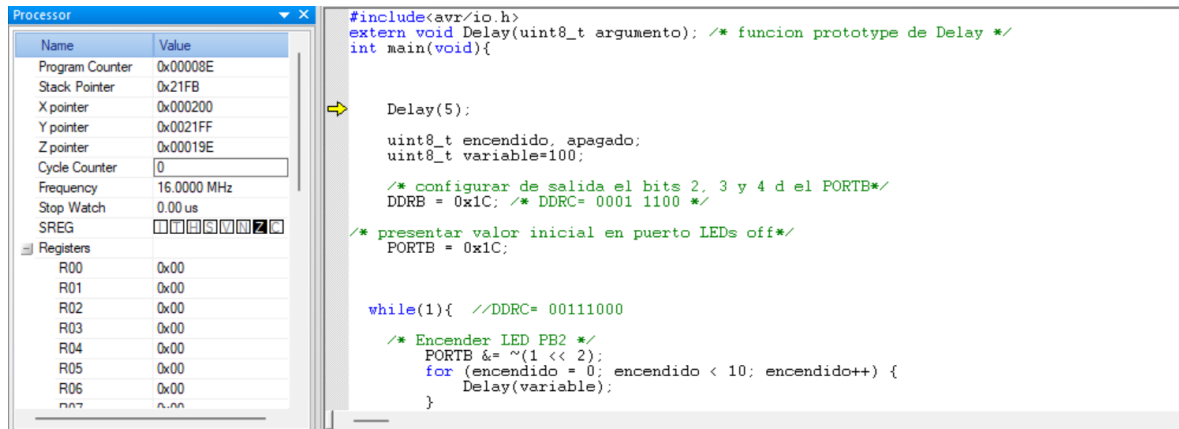
Ticks delay general = $12(26) + 4(26*151) + 5 = 16021$

Igual nos quedamos bastante cercanos al valor que se estima.

Ejecución del programa:

Si queremos un delay para 5mS los ticks totales tendrían que ser de 80000 ticks de ciclo de reloj, desde va de C a ASM hasta que vuelve de ASM a C.

Primero se inicia el programa y limpiamos los ciclos que se ejecutaron para iniciar el programa:



The screenshot shows the AVR Studio IDE with the C code for the delay function and the initial state of the processor registers. The registers window on the left shows the Program Counter at 0x00008E, Stack Pointer at 0x21FB, and various pointers and counters. The code window on the right shows the C code for the delay function, which includes a function prototype, a main function, and a while loop that calls the delay function.

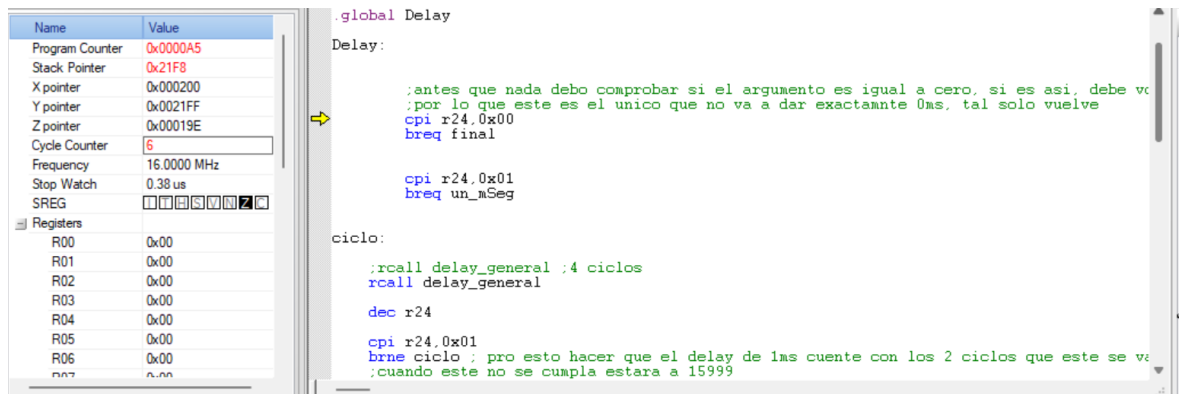
```
#include<avr/io.h>
extern void Delay(uint8_t argumento); /* funcion prototype de Delay */
int main(void){
    Delay(5);
    uint8_t encendido, apagado;
    uint8_t variable=100;

    /* configurar de salida el bits 2, 3 y 4 d el PORTB*/
    DDRE = 0x1C; /* DDRC= 0001 1100 */

    /* presentar valor inicial en puerto LEDs off*/
    PORTB = 0x1C;

    while(1){ //DDRC= 00111000
        /* Encender LED PB2 */
        PORTB &= ~(1 << 2);
        for (encendido = 0; encendido < 10; encendido++) {
            Delay(variable);
        }
    }
}
```

Se llama a la función que esta en ASM desde C lo que consume **6 ticks**.



The screenshot shows the AVR Studio IDE with the assembly code for the delay function and the state of the processor registers. The registers window on the left shows the Program Counter at 0x0000A5, Stack Pointer at 0x21FB, and various pointers and counters. The code window on the right shows the assembly code for the delay function, which includes a global declaration, a delay function, and a delay_general function.

```
.global Delay
Delay:
    ;antes que nada debo comprobar si el argumento es igual a cero, si es asi, debe v
    ;por lo que este es el unico que no va a dar exactamnte 0ms, tal solo vuelve
    cpi r24,0x00
    breq final

    cpi r24,0x01
    breq un_mSeg

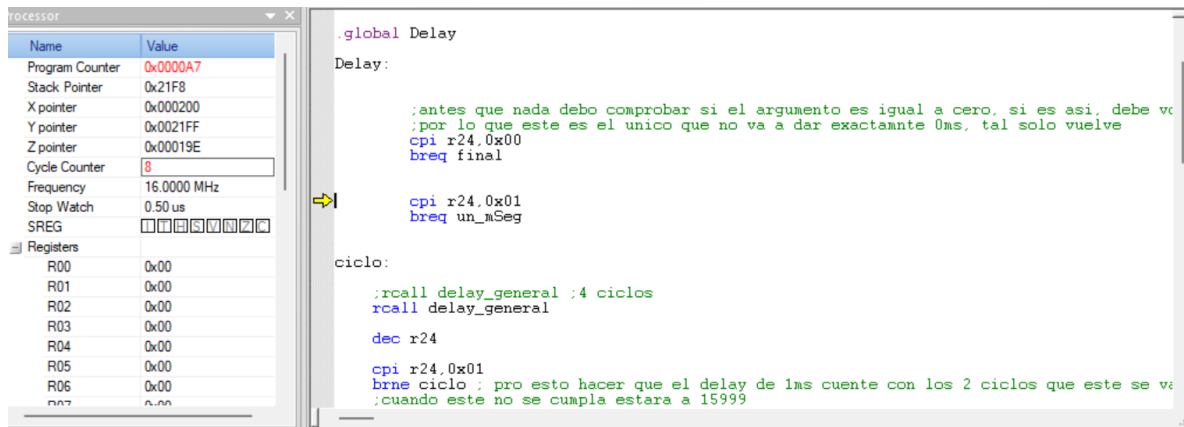
ciclo:
    ;rcall delay_general ;4 ciclos
    rcall delay_general

    dec r24

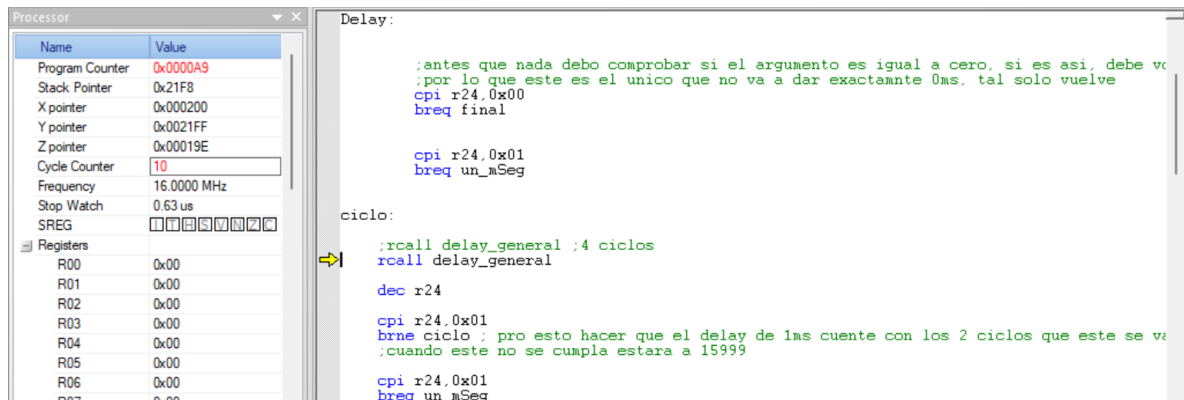
    cpi r24,0x01
    brne ciclo ; pro esto hacer que el delay de 1ms cuente con los 2 ciclos que este se v
    ;cuando este no se cumpla estara a 15999
```

Ahora se tiene que comprobar si el dato que se mandó como parámetro es diferente a 0. Cuando esto suceda, el programa ira inmediatamente a ret a regresar a C, es la única vez cuando no se va a poder cumplir la condición de 0 mS de retardo, es imposible 0 ticks de ciclos de reloj. Tan solo el ir consumió 6 ticks

En este caso no es, por lo que aquí habrán **2 ticks** más, 1 de la comparación y 1 del Branch, dado que no se cumplió por lo que es 1.

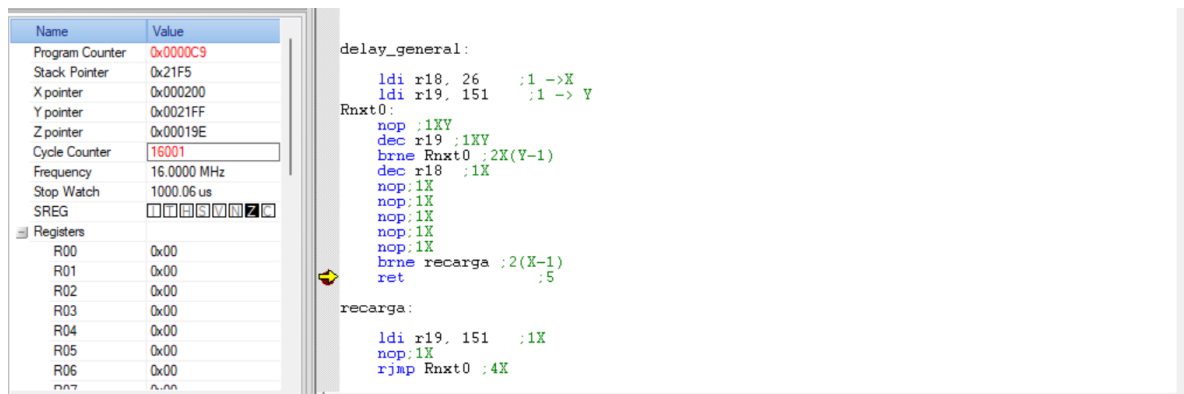


Ahora se compara si lo que se mando es 1mS, en caso de serlo llama a la subrutina. En nuestro caso no por lo que serán otros **2 Ticks** de reloj.



En este punto tenemos 10 ticks de reloj, por lo que siempre se se vuelva a llamar a **delay_general** serán 14 ticks más, con rcall con 4 ticks más.

Por lo que se calculan 16000 ciclos, junto con ret, y las otras operaciones siempre y cuando r24 no sea igual 1.



Decrementamos R24

Name	Value
Program Counter	0x0000AA
Stack Pointer	0x21F8
X pointer	0x000200
Y pointer	0x0021FF
Z pointer	0x00019E
Cycle Counter	16006
Frequency	16.0000 MHz
Stop Watch	1000.38 us
SREG	00000000

```

ciclo:
    ;rcall delay_general ;4 ciclos
    rcall delay_general
    dec r24
    cpi r24,0x01
    brne ciclo ; pro esto hacer que el delay de 1ms cuente con los 2 ciclos que este se va
    ;cuando este no se cumpla estara a 15999
    cpi r24,0x01
    breq un_mSeg

final:
    ret

un_mSeg:
    ldi r23,27 ;1 -> X
    ldi r22,117 ;1 -> Y

nxt0:
    nop ; 1XY
  
```

En caso de no ser 1 vuelve a *ciclo*

Name	Value
Program Counter	0x0000AC
Stack Pointer	0x21F8
X pointer	0x000200
Y pointer	0x0021FF
Z pointer	0x00019E
Cycle Counter	16008
Frequency	16.0000 MHz
Stop Watch	1000.50 us
SREG	00000000

```

ciclo:
    ;rcall delay_general ;4 ciclos
    rcall delay_general
    dec r24
    cpi r24,0x01
    brne ciclo ; pro esto hacer que el delay de 1ms cuente con los 2 ciclos que este se va
    ;cuando este no se cumpla estara a 15999
    cpi r24,0x01
    breq un_mSeg

final:
    ret

un_mSeg:
    ldi r23,27 ;1 -> X
    ldi r22,117 ;1 -> Y

nxt0:
    nop ; 1XY
  
```

La operación es la misma hasta llegar cuando $R24 = 1$;

Name	Value
Program Counter	0x0000A9
Stack Pointer	0x21F8
X pointer	0x000200
Y pointer	0x0021FF
Z pointer	0x00019E
Cycle Counter	16010
Frequency	16.0000 MHz
Stop Watch	1000.63 us
SREG	00000000

```

ciclo:
    ;rcall delay_general ;4 ciclos
    rcall delay_general
    dec r24
    cpi r24,0x01
    brne ciclo ; pro esto hacer que el delay de 1ms cuente con los 2 ciclos que este se va
    ;cuando este no se cumpla estara a 15999
    cpi r24,0x01
    breq un_mSeg

final:
    ret

un_mSeg:
    ldi r23,27 ;1 -> X
    ldi r22,117 ;1 -> Y

nxt0:
    nop ; 1XY
  
```

2mS= 32000 ticks

Name	Value
Program Counter	0x0000C9
Stack Pointer	0x21F5
X pointer	0x000200
Y pointer	0x0021FF
Z pointer	0x00019E
Cycle Counter	32001
Frequency	16.0000 MHz
Stop Watch	2000.06 us
SREG	00000000

```

delay_general:
    ldi r18, 26    ;1 -> X
    ldi r19, 151   ;1 -> Y
Rnxt0:
    nop ;1XY
    dec r19 ;1XY
    brne Rnxt0 ;2X(Y-1)
    dec r18 ;1X
    nop:1X
    nop:1X
    nop:1X
    nop:1X
    brne recarga ;2(X-1)
    ret ;5

recarga:
    ldi r19, 151 ;1X
    nop:1X
    rjmp Rnxt0 ;4X
  
```

R24-=1;

Name	Value
R15	0x00
R16	0x00
R17	0x02
R18	0x00
R19	0x00
R20	0x00
R21	0x00
R22	0x00
R23	0x00
R24	0x03
R25	0x00
R26	0x00
R27	0x02
R28	0xFF
R29	0x21
R30	0x9E
R31	0x01

```

ciclo:
    rcall delay_general ;4 ciclos
    rcall delay_general
    dec r24
    cpi r24,0x01
    brne ciclo ; pro esto hacer que el delay de 1ms cuente con los 2 ciclos que este se va
    ;cuando este no se cumpla estara a 15999
    cpi r24,0x01
    breq un_mSeg

final:
    ret

un_mSeg:
    ldi r23,27 ;1 -> X
    ldi r22,117 ;1 -> Y
nxt0:
    nop ; 1XY
  
```

R24!=1?

Name	Value
Program Counter	0x0000C9
Stack Pointer	0x21F5
X pointer	0x000200
Y pointer	0x0021FF
Z pointer	0x00019E
Cycle Counter	48001
Frequency	16.0000 MHz
Stop Watch	3000.06 us
SREG	00000000

```

delay_general:
    ldi r18, 26    ;1 -> X
    ldi r19, 151   ;1 -> Y
Rnxt0:
    nop ;1XY
    dec r19 ;1XY
    brne Rnxt0 ;2X(Y-1)
    dec r18 ;1X
    nop:1X
    nop:1X
    nop:1X
    nop:1X
    brne recarga ;2(X-1)
    ret ;5

recarga:
    ldi r19, 151 ;1X
    nop:1X
    rjmp Rnxt0 ;4X
  
```

Así ira hasta que r24=1;

En este punto R24 en efecto es igual a 1. Por lo que ira el branch es aceptado e ira a *un_mSeg*

Name	Value
Program Counter	0x0000B0
Stack Pointer	0x21F8
X pointer	0x000200
Y pointer	0x0021FF
Z pointer	0x00019E
Cycle Counter	64012
Frequency	16.0000 MHz
Stop Watch	4000.75 us
SREG	0x00

```

ret

un_mSeg:
    ldi r23, 27 ;1 -> X
    ldi r22, 117 ;1 -> Y

nxt0:
    nop ;1XY
    nop ;1XY
    dec r22 ;1XY
    brne nxt0 ;2X(Y-1)
    dec r23 ;1X
    nop ;1x
    nop ;1x
    brne nxt1 ;1:2(x-1)
    rjmp final ;4

nxt1:
    ldi r22, 117 ;1X
    rjmp nxt0 ;4X
  
```

Son los ultimo 16000 ciclos, pero esta toma en cuenta todos los ticks se hicieron antes de estar llamando a las distintas etiquetas.

Name	Value
Program Counter	0x0000BA
Stack Pointer	0x21F8
X pointer	0x000200
Y pointer	0x0021FF
Z pointer	0x00019E
Cycle Counter	79994
Frequency	16.0000 MHz
Stop Watch	4999.63 us
SREG	0x00

```

ret

un_mSeg:
    ldi r23, 27 ;1 -> X
    ldi r22, 117 ;1 -> Y

nxt0:
    nop ;1XY
    nop ;1XY
    dec r22 ;1XY
    brne nxt0 ;2X(Y-1)
    dec r23 ;1X
    nop ;1x
    nop ;1x
    brne nxt1 ;1:2(x-1)
    rjmp final ;4

nxt1:
    ldi r22, 117 ;1X
    rjmp nxt0 ;4X
  
```

Name	Value
Program Counter	0x000091
Stack Pointer	0x21FB
X pointer	0x000200
Y pointer	0x0021FF
Z pointer	0x00019E
Cycle Counter	80001
Frequency	16.0000 MHz
Stop Watch	5000.06 us
SREG	0x00

```

#include<avr/io.h>
extern void Delay(uint8_t argumento); /* funcion prototype de Delay */
int main(void){

    Delay(5);

    uint8_t encendido, apagado;
    uint8_t variable=100;

    /* configurar de salida el bits 2, 3 y 4 d el PORTE*/
    DDRB = 0x1C; /* DDRC= 0001 1100 */

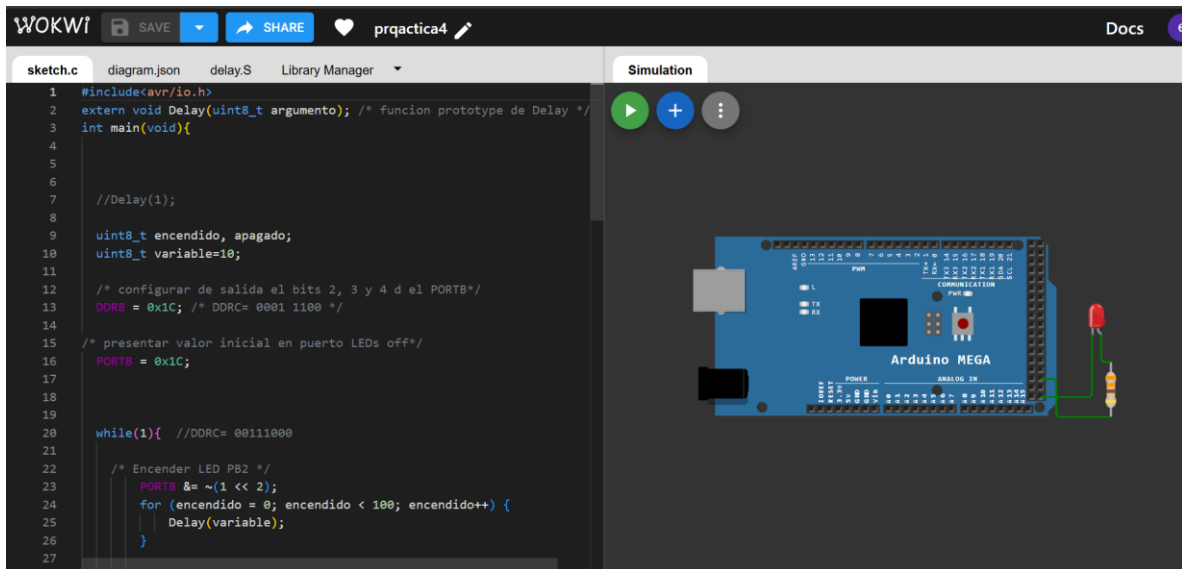
    /* presentar valor inicial en puerto LEDs off*/
    PORTB = 0x1C;

    while(1){ //DDRC= 00111000

        /* Encender LED PB2 */
        PORTB &= ~(1 << 2);
        for (encendido = 0; encendido < 10; encendido++) {
            Delay(variable);
        }
    }
}
  
```

Programa cargado a Arduino Atmega 2560:

En este caso esta en un simulador:



Debería de prender y apagar entre 1 segundo, así infinitamente ya que se encuentran dentro de una función while.

Video: <https://www.youtube.com/watch?v=9czb5wVnDQw>