# Practica 8a

## PROGRAMACION DEL MICROCONTROLADOR EN LENGUAJE C Y COMUNICACIÓN SERIE


Carlos Omar Calderón Meza
MICROPROCESADORES Y MICROCONTROLADORES| INGENIERO EN COMPUTACION

**Objetivo:**

Mediante esta práctica el alumno aprenderá el uso básico de la programación en lenguaje C con las herramientas AVR Studio y WinAVR. Para ello el alumno implementará los procedimientos comunes para inicializar y operar el puerto serie del microcontrolador.

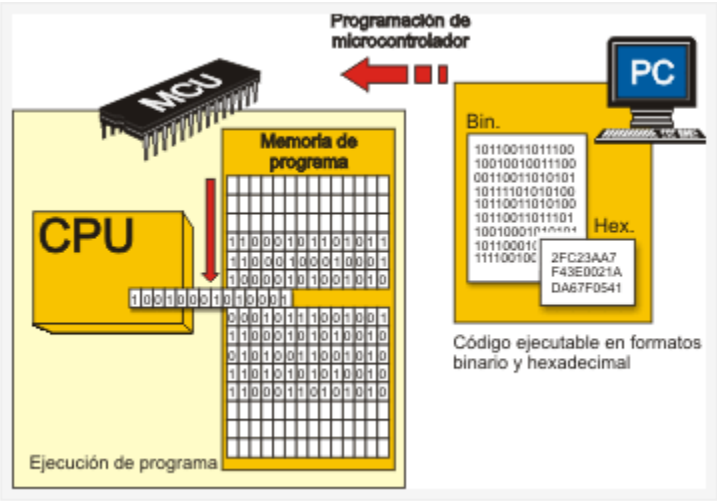
**Equipo:**

**-** Computadora Personal
- Módulo T-Juino

**Teoría:**

- **Programación en lenguaje C en microcontroladores**

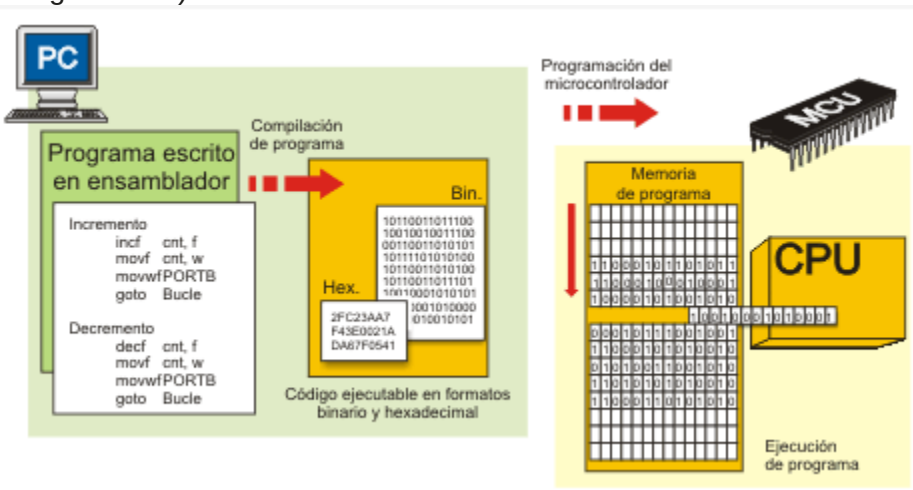
El microcontrolador ejecuta el programa cargado en la memoria Flash. Esto se denomina el código ejecutable y está compuesto por una serie de ceros y unos, aparentemente sin significado. Dependiendo de la arquitectura del microcontrolador, el código binario está compuesto por palabras de 12, 14 o 16 bits de anchura. Cada palabra se interpreta por la CPU como una instrucción a ser ejecutada durante el funcionamiento del microcontrolador. Todas las instrucciones que el microcontrolador puede reconocer y ejecutar se les denominan colectivamente Conjunto de instrucciones. Como es más fácil trabajar con el sistema de numeración hexadecimal, el código ejecutable se representa con frecuencia como una serie de los números hexadecimales denominada código Hex.




**LENGUAJE ENSAMBLADOR**

Como el proceso de escribir un código ejecutable era considerablemente arduo, en consecuencia fue creado el primer lenguaje de programación denominado ensamblador (ASM). Siguiendo la sintaxis básica del ensamblador, era más fácil escribir y comprender el código. Las instrucciones en ensamblador consisten en las abreviaturas con significado y a cada instrucción corresponde una localidad de memoria. Un programa denominado

ensamblador compila (traduce) las instrucciones del lenguaje ensamblador a código máquina (código binario).




**Ventajas de lenguajes de programación de alto nivel**

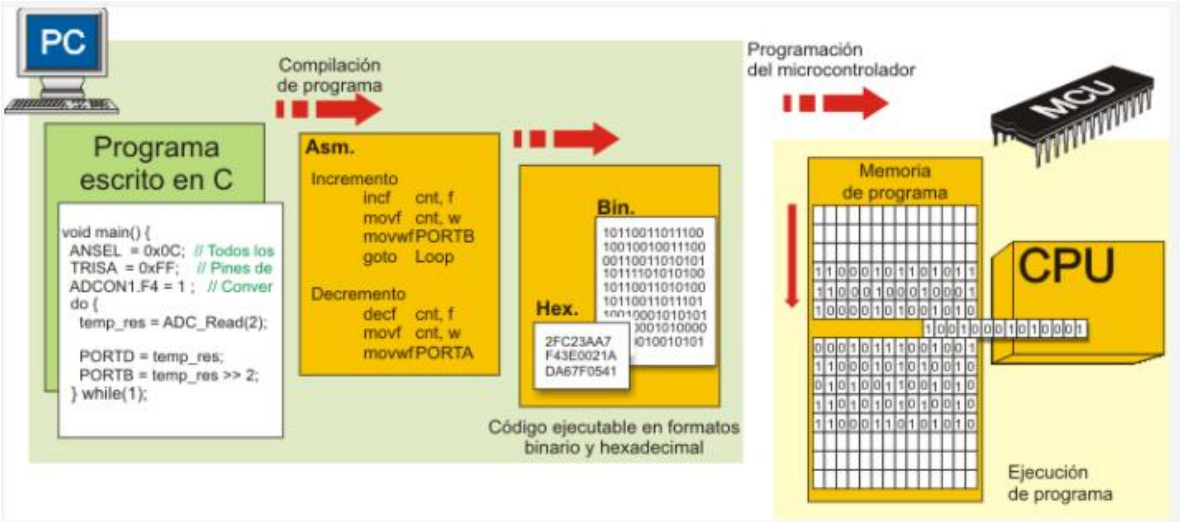
A pesar de todos los lados buenos, el lenguaje ensamblador tiene algunas desventajas:

- Incluso una sola operación en el programa escrito en ensamblador consiste en muchas instrucciones, haciéndolo muy largo y difícil de manejar.
- Cada tipo de microcontrolador tiene su propio conjunto de instrucciones que un programador tiene que conocer para escribir un programa
- Un programador tiene que conocer el hardware del microcontrolador para escribir un programa

Los lenguajes de programación de alto nivel (Basic, Pascal, C etc.) fueron creados con el propósito de superar las desventajas del ensamblador. En lenguajes de programación de alto nivel varias instrucciones en ensamblador se sustituyen por una sentencia. El programador ya no tiene que conocer el conjunto de instrucciones o características del hardware del microcontrolador utilizado. Ya no es posible conocer exactamente cómo se ejecuta cada sentencia, de todas formas ya no importa. Aunque siempre se puede insertar en el programa una secuencia escrita en ensamblador.

**Lenguaje C**

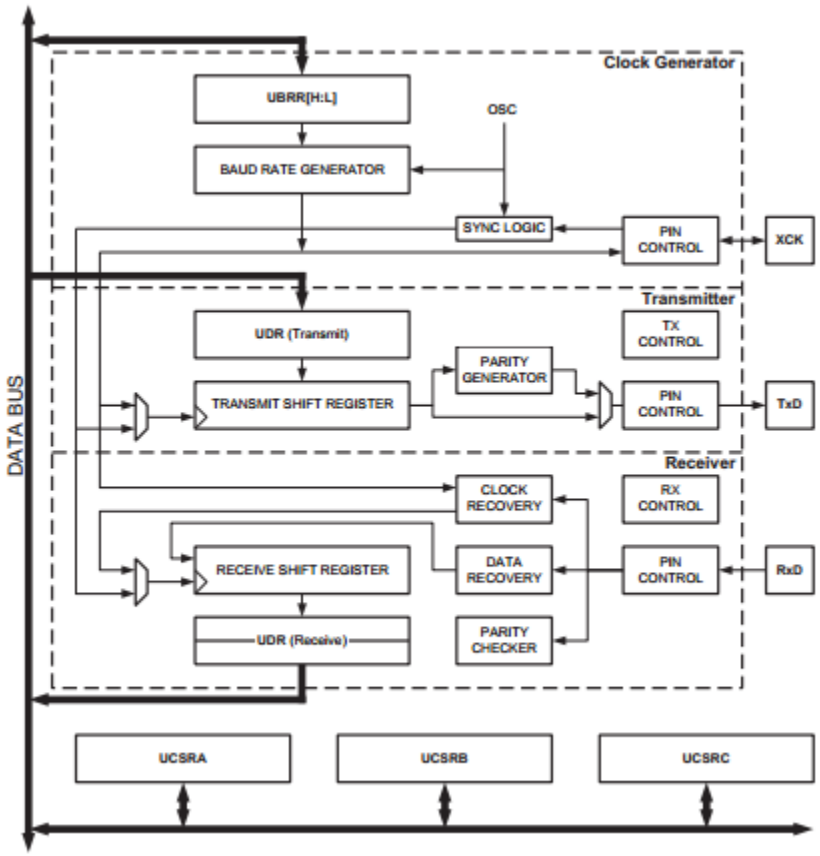
El lenguaje C dispone de todas las ventajas de un lenguaje de programación de alto nivel (anteriormente descritas) y le permite realizar algunas operaciones tanto sobre los bytes como sobre los bits (operaciones lógicas, desplazamiento etc.). Las características de C pueden ser muy útiles al programar los microcontroladores. Además, C está estandarizado (el estándar ANSI), es muy portable, así que el mismo código se puede utilizar muchas veces en diferentes proyectos. Lo que lo hace accesible para cualquiera que conozca este lenguaje sin reparar en el propósito de uso del microcontrolador. C es un lenguaje compilado, lo que significa que los archivos fuentes que contienen el código C se traducen a lenguaje máquina por el compilador. Todas estas características hicieron al C uno de los lenguajes de programación más populares.




La figura anterior es un ejemplo general de lo que sucede durante la compilación de programa de un lenguaje de programación de alto nivel a bajo nivel.

**- Manejo del Periférico de Comunicación Serie 0 (UART0) del microcontrolador ATmega1280/2560 (Resumen)**

El ATmega640/1280/2560 tiene cuatro USART: USART0, USART1, USART2 y USART3



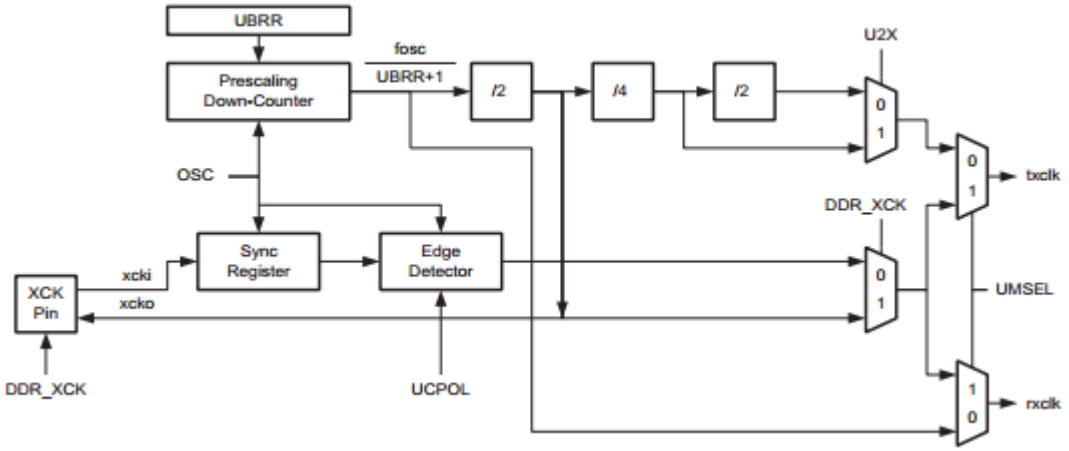
El diagrama lógico del USART se divide en 3 secciones punteadas, el generador de reloj, transmisión y recepción, las tres secciones comparten 3 registros de control y estado UCSRA, UCSRB y UCSRC.

**Generación de reloj**

El USARTn soporta cuatro modos de funcionamiento: el reloj asíncrono normal, asíncronos de doble velocidad, síncronos Maestro y esclavo modo síncrono.

El bit UMSELn en el control USART y registro de estado C (UCSRnC) selecciona entre asíncrono y síncrono. Doble velocidad (modo asíncrono) es controlada por el U2Xn encontrado en el registro UCSRnA. Al usar el modo síncrono (UMSELn = 1), la dirección de registro de datos para el pasador XCKn (DDR_XCKn) controla si la fuente de reloj es Maestro o esclavo externo. El pasador XCKn solo se activa cando se utiliza modo síncrono.

**Figure 22-2.** Clock Generation Logic, Block Diagram




**Txclk** reloj del trasmisor (señal interna)

**Rxclk** reloj base del receptor (señal interna)

**Xcki** entrada de xck (señal interna). Usado para modo esclavo síncrono

**Xcko** reloj de salida para xck (señal interna). Usado para modo maestro sincrono

**Generación de reloj interno – Bauldios**

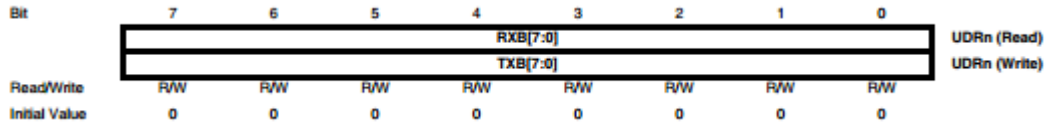
La velocidad en bauldios en el registro USART (UBRRn) se calcula con las siguientes ecuaciones:

**Table 22-1.** Equations for Calculating Baud Rate Register Setting

| Operating Mode | Equation for Calculating Baud Rate[1] | Equation for Calculating UBRR Value |
|---|---|---|
| Asynchronous Normal mode (U2Xn = 0) | $BAUD = \frac{f_{OSC}}{16(UBRRn+1)}$ | $UBRRn = \frac{f_{OSC}}{16BAUD} - 1$ |
| Asynchronous Double Speed mode (U2Xn = 1) | $BAUD = \frac{f_{OSC}}{8(UBRRn+1)}$ | $UBRRn = \frac{f_{OSC}}{8BAUD} - 1$ |
| Synchronous Master mode | $BAUD = \frac{f_{OSC}}{2(UBRRn+1)}$ | $UBRRn = \frac{f_{OSC}}{2BAUD} - 1$ |

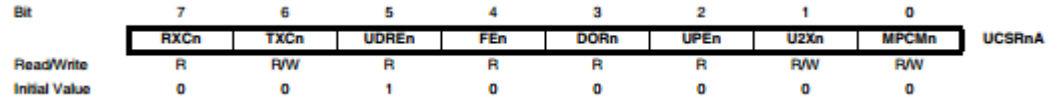
## Descripción de Registros de control y estado de USART

### UDRn – USART I/O Registro de datos




El registro para transferir datos y recibir datos de USART comparte la misma dirección de E/S, se refiere como UDRn.

### UCSRnA – Registro de control y estado n A




### Bit 7 – RXCn: USART Recepción completa

1 indica que hay datos leídos en el buffer

0 El buffer está vacío

### Bit 6 – TXCn: USART Transmisión completa

1 indica que se transmitió completamente los datos del buffer.

### Bit 5 – UDREn: USART Registro de datos vacío

1 indica que el buffer de transmisión UDRn está listo para recibir nuevos datos.

### Bit 4 – FEn: error de Frame

1 indica que el siguiente carácter de recepción tenía un error cuando se recibió

### Bit 3 – DORn: USART OverRun

1 indica que ocurre un desbordamiento de datos, es decir cuando el buffer está lleno y llega otro dato.

**Bit 2 – UPERn: USART Error de paridad**

1 indica que ocurrió un error de paridad

**Bit 1 – U2Xn: USART Doble velocidad de transmisión**

Este bit solo funciona en el modo asíncrono.

1 activa doble velocidad de transferencia para la comunicación asíncrona

**Bit 0 – MPCMn: Modo de comunicación multi-procesador**

1 activa el modo de comunicación multi-procesador.

**UCSRnB – Registro de control y estado n B**

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | RXCIEn | TXCIEn | UDRIEn | RXENn | TXENn | UCSZn2 | RXB8n | TXB8n | UCSRnB |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

**Bit 7 – RXCIEn: habilitación de interrupción completa RX**

1 permite interrumpir en la bandera RXCn

**Bit 6 – TXCIEn: habilitación de interrupción completa TX**

1 permite interrumpir la bandera TXCn

**Bit 5 – UDRIEn: habitación de interrupción de registro de datos n**

1 permite interrumpir en la bandera UDREn

**Bit 4 – RXENn: Habilita receptor n**

1 permite al receptor USART.

**Bit 3 – TXENn: Habilita Transmisor n**

1 habilita el transmisor USART

**Bit 2 – UCSZn2: Tamaño de carácter n**

0 UCSRnC establece el número de bits de datos.

**Bit 1 – RXB8n: Recibe bits de datos 8 n**

Es el noveno bit de datos del frame recibido cuando se opera con los marcos de seria con 9 bits de datos.

**Bit 0 – TXB8n: Transmision de bits de datos 8 n**

Es el noveno bit de datos del frame que se transmite cuando se opera con marcos de serie con 9 bits de datos.

### UCSRnC – USART registro de control y estado n C

| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|---|---|---|---|---|---|---|---|---|---|
| | UMSELn1 | UMSELn0 | UPMn1 | UPMn0 | USBSn | UCSZn1 | UCSZn0 | UCPOLn | UCSRnC |
| Read/Write | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | |

### Bits 7:6 – UMSELn1: 0 USART selección de modo

Estos bits seleccionan el modo de funcionamiento de la USARTn

**Table 22-4.** UMSELn Bits Settings

| UMSELn1 | UMSELn0 | Mode |
|---|---|---|
| 0 | 0 | Asynchronous USART |
| 0 | 1 | Synchronous USART |
| 1 | 0 | (Reserved) |
| 1 | 1 | Master SPI (MSPIM)[1] |

### Bits 5:4 – UPMn1: 0: Modo paridad

**Table 22-5.** UPMn Bits Settings

| UPMn1 | UPMn0 | Parity Mode |
|---|---|---|
| 0 | 0 | Disabled |
| 0 | 1 | Reserved |
| 1 | 0 | Enabled, Even Parity |
| 1 | 1 | Enabled, Odd Parity |

### Bit 3 – USBSn: bit de selección de bit de stop

Selecciona el número de bits de parada que se inserta por el transmisor

**Table 22-6.** USBS Bit Settings

| USBSn | Stop Bit(s) |
|---|---|
| 0 | 1-bit |
| 1 | 2-bit |

### Bit 2:1 – UCSZn1:0: tamaño de caracteres

**Table 22-7.** UCSZn Bits Settings

| UCSZn2 | UCSZn1 | UCSZn0 | Character Size |
|---|---|---|---|
| 0 | 0 | 0 | 5-bit |
| 0 | 0 | 1 | 6-bit |
| 0 | 1 | 0 | 7-bit |
| 0 | 1 | 1 | 8-bit |
| 1 | 0 | 0 | Reserved |
| 1 | 0 | 1 | Reserved |
| 1 | 1 | 0 | Reserved |
| 1 | 1 | 1 | 9-bit |

### Bit 0 – UCPOLn: paridad de reloj

Este bit se utiliza solamente para el modo síncrono

**Table 22-8.** UCPOLn Bit Settings

| UCPOLn | Transmitted Data Changed (Output of TxDn Pin) | Received Data Sampled (Input on RxDn Pin) |
|---|---|---|
| 0 | Rising XCKn Edge | Falling XCKn Edge |
| 1 | Falling XCKn Edge | Rising XCKn Edge |

### UBRRnL y UBRRnH – USART registros Baud Rate

| Bit | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | |
|---|---|---|---|---|---|---|---|---|---|
| | - | - | - | - | UBRR[11:8] | | | | UBRRHn |
| | UBRR[7:0] | | | | | | | | UBRRLn |
| | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| Read/Write | R | R | R | R | R/W | R/W | R/W | R/W | |
| | R/W | R/W | R/W | R/W | R/W | R/W | R/W | R/W | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

### Bit 15:12 – Bits reservados

Estos bits están reservados para usos futuros, se deben escribir a cero cuando UBRRH este escrito

Bit 11:0 – UBRR11:0: Registro de Velocidad de transmisión USART

Contiene la velocidad de transmisión USART. El UBRRH contiene los 4 bits más significativos y UBRRL contiene los 8 bits menos significativos.

El USART tiene que ser inicializado antes de cualquier comunicación.

**Actividades a realizar:**

1. **Calcular el periodo total de la función Delay(void) del Listado 1 con optimización *Os*.**

**Análisis:**

**[ x ]: Veces que se ejecuta la instrucción**

**( y ): Numero de ciclos de la instruccion**

```
00000099  SBI 0x05,7        Set bit in I/O register
        delay();
0000009A  RCALL PC-0x001D       Relative call subroutine
        PORTB &= ~( 1 << PB7 ); /* apagar LED */
0000009B  CBI 0x05,7        Clear bit in I/O register
        delay();
0000009C  RCALL PC-0x001F       Relative call subroutine
    }
0000009D  RJMP PC-0x0004        Relative jump
```

Desde la llamada de la función Delay():

RCALL **[1] * (4)**

```
0000007D  PUSH R28      Push register on stack
0000007E  PUSH R29      Push register on stack
0000007F  PUSH R1       Push register on stack
00000080  PUSH R1       Push register on stack
00000081  IN R28,0x3D       In from I/O location
00000082  IN R29,0x3E       In from I/O location
--- C:\Users\LILIANA\Documents\Atmel Studio\7.0\p8\p8\Debug/../sr
    for(i=0;i<0xffff;i++);
00000083  STD Y+2,R1        Store indirect with displacement
00000084  STD Y+1,R1        Store indirect with displacement
00000085  LDD R24,Y+1       Load indirect with displacement
00000086  LDD R25,Y+2       Load indirect with displacement
00000087  ADIW R24,0x01     Add immediate to word
00000088  BREQ PC+0x0A      Branch if equal
```

Cuando es llamada la función Delay realiza las instrucciones que se muestran en la imagen.

PUSH R28 **[1] * (2)**

PUSH R29 **[1] * (2)**

PUSH R1 **[1] * (2)**

PUSH R1 **[1] * (2)**

IN R28,0x3D **[1] * (1)**

IN R29,0x3E **[1] * (1)**

STD Y+2, R1 **[1] * (2)**

STD Y+1, R1 **[1] * (2)**

LDD R24,Y+1 **[1] * (2)**

LDD R25,Y+2 **[1] * (2)**

ADIW R24,0x01 **[1] * (2)**

BREQ PC+0x0A **[1] * (1)**

```
00000089  LDD R24,Y+1       Load indirect with displacement
0000008A  LDD R25,Y+2       Load indirect with displacement
0000008B  ADIW R24,0x01     Add immediate to word
0000008C  STD Y+2,R25       Store indirect with displacement
0000008D  STD Y+1,R24       Store indirect with displacement
0000008E  LDD R24,Y+1       Load indirect with displacement
0000008F  LDD R25,Y+2       Load indirect with displacement
00000090  ADIW R24,0x01     Add immediate to word
00000091  BRNE PC-0x08      Branch if not equal
00000092  POP R0          Pop register from stack
00000093  POP R0          Pop register from stack
00000094  POP R29         Pop register from stack
00000095  POP R28         Pop register from stack
00000096  RET         Subroutine return
```

Cuando entra a las iteraciones del for realiza las siguientes instrucciones

LDD R24, Y+1 **[65535] * (2)**

LDD R24,Y+2 **[65535] * (2)**

ADIW R24,0x01 **[65535] * (2)**

STD Y+2,R25 **[65535] * (2)**

STD Y+1,R24 **[65535] * (2)**

LDD R24, Y+1 **[65535] * (2)**

LDD R25,Y+2 **[65535] * (2)**

ADIW R24,0x01 **[65535] * (2)**

BRNE PC-0x08 **[65535] * (2) + [1] * (1)**

Despues recupera de la pila lo que guardo y regresa al main:

POP R0 **[1] * (2)**

POP R0 **[1] * (2)**

POP R29 **[1] * (2)**

POP R28 **[1] * (2)**

RET **[1] * (4)**

**Total de ciclos de reloj y tiempo:**

**1,179,664 de ciclos @ 16Mhz**

**Delay = (1/16Mhz) * 1,179,664 = 73.729 ms**

**2. Implementar las siguientes funciones para la comunicación serie mediante encuesta (polling):**

a) void **UART0_Init**(uint16_t mode**)**: Función para inicializar el puerto serie del ATmega1280/2560 según el valor del parámetro modo. Si mode es 0 entonces la inicialización es **9600,8,N,1** de lo contrario **19200,8,N,1.**

b) char **UART0_getchar**(void): Función que retorna el byte recibido por el puerto serie UART0.

c) void **UART0_putchar**(char data): Función que transmite un byte por el puerto serie UART0.

d) void **UART0_gets**(char *str): Función que retorna una cadena mediante **UART0_getchar**, la cadena se retorna en el apuntador str**.**

e) void **UART0_puts**(char *str): Función que transmite una cadena mediante **UART0_putchar.**

f) void **itoa**(char* str, uint16_t number, uint8_t base) : Función que convierte un número de 16 bits a su representación alfanumérica en la base dada, y la retorna en el apuntador str.

g) unsigned int **atoi**(char *str) : Función que convierte una cadena numérica (de base 10) y retorna su valor numérico en 16 bit

**Conclusiones y comentarios:**

El puerto serie USART de los microcontroladores de Atmel es sencillo de utilizar una vez que ya se entendió bien cómo funciona, al principio no lograba entender bien cómo funcionaba pero con apoyo de la hoja de datos logre entender cómo funciona, está muy bien documentado los microcontroladores de Atmel, de hecho no se necesita buscar información por otra fuente, todo lo necesario para utilizar los puertos USART viene bien documentados.

Las aplicaciones para los puestos serie USART de Atmel veo que son inmensas, con lo puertos se puede interactuar con componentes o cosas alrededor del micro, controlar maquinas etc. Es por ello que es fundamental conocer cómo funciona y como aplicarlo en los sistemas embebidos.

Bibliografía:

[1] https://learn.mikroe.com/ebooks/microcontroladorespicc/chapter/lenguajes-de-programacion/

[2] Hoja de datos del Atmel ATmega640/V-1280/V-1281/V-2560/V-2561/V