Universidad de Buenos Aires
Facultad De Ingeniería
Año 2021 – 1.er Cuatrimestre

(82.07) Laboratorio de microprocesadores

Entrega N.º 3: Puerto serie
Curso 01
22 de Junio de 2021

Docentes a cargo:
Stola, Gerardo Luis
Salaya, Juan Guido
Cofman, Fernando

Integrante	Padrón	Correo electrónico
Lützelschwab, Nahila	100686	nlützelschwab@fi.uba.ar

Índice

1. Objetivos del proyecto

El objetivo del presente trabajo práctico es establecer una comunicación bidireccional serie entre un microcontrolador AVR de 8 bits y una computadora de escritorio. Además, se busca implementar interrupciones por recepción o transmisión y comparar el programa principal con el caso anterior.

2. Descripción del proyecto

El presente trabajo consistió en diseñar un programa que permita transmitir un texto, mostrandose en el terminal serie, el cual le permita al usuario elegir establecer un dígito del 1 al 4 visualizando dicha elección en un display 7 segmentos ánodo común.

2.1. Lista de componentes

A continuación se listan los componentes utilizados para la implementación del proyecto.

- Placa Arduino UNO
- Protoboard de 830 puntos
- Cables macho-macho para protoboard
- Siete resistencias de valor 330Ω
- Un display 7 segmentos ánodo común

3. Esquemático

Como muestra el esquemático 1, se hizo uso de una placa Arduino UNO basada en el microcontrolador ATmega328p, a la cual se conectó un display 7 segmentos ánodo común donde cada segmento va conectado a una resistencia de 330Ω. Dado que el display es ánodo común (nodo común conectado a VCC), se tuvo en cuenta que para encender un segmento se setea un '0' lógico y para apagarlo un '1' lógico. Se le permitió al usuario elegir dígitos del 1 al 4, y en caso que se eliga un valor fuera del rango el display muestra un 0 y aparece un mensaje avisando dicho error en la terminal serie. Como se muestra en la tabla 1, los segmentos se conectaron a los pines de los puertos C y D configurados como salida de la siguiente manera: a - PC0, b - PC1, c - PC2, d - PC3, e - PC4, f - PC5, g - PD6

A su vez, el Arduino se encuentra conectado a una computadora mediante un cable USB con el que se realiza la transmisión y recepción de datos con una conexion de puerto serie en modo full-duplex, es decir que puede recibir y enviar información digital simultáneamente.

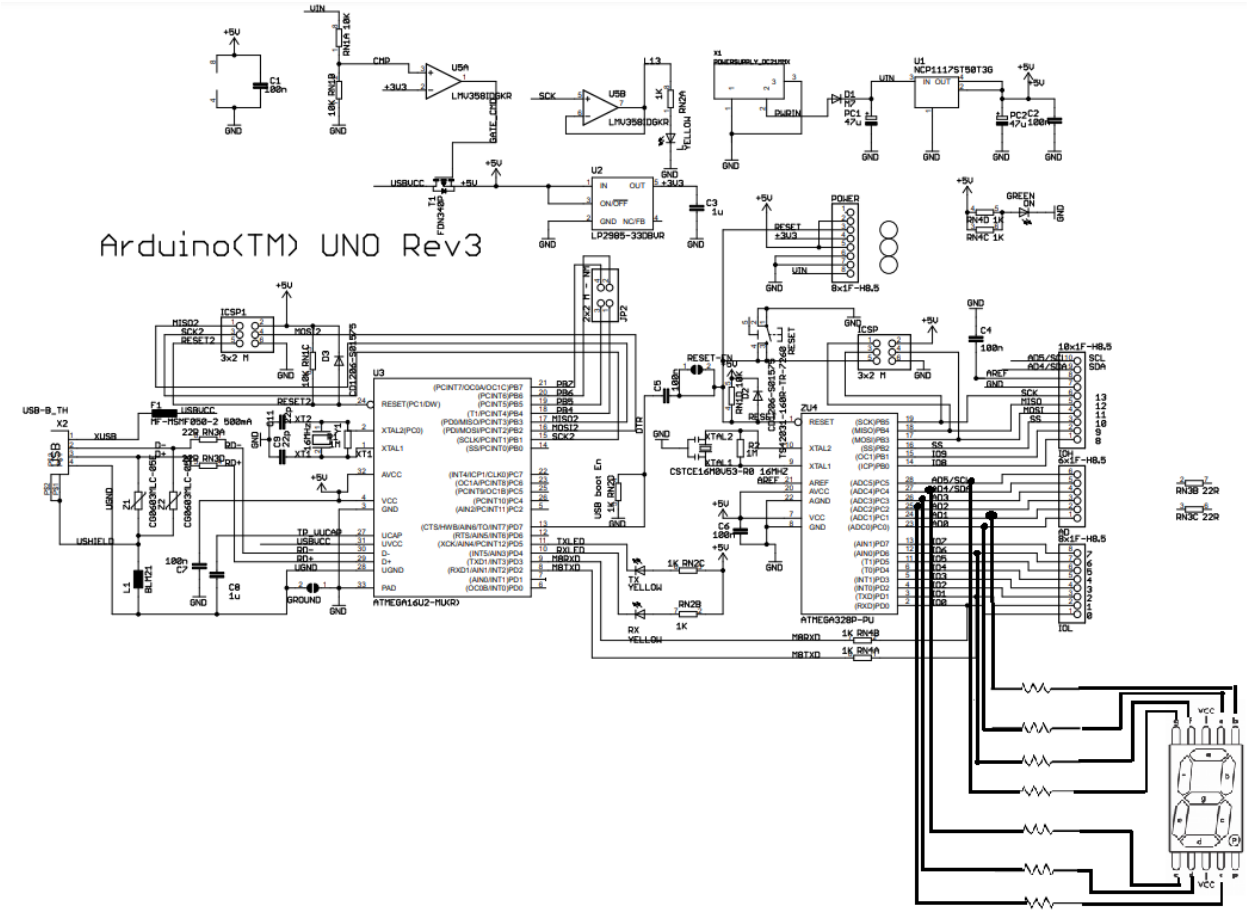

Figura 1: Esquematico completo del circuito implementado

Dígito	dp	PD6	PC5	PC4	PC3	PC2	PC1	PC0
1	1	1	1	1	1	0	0	1
2	1	0	1	0	0	1	0	0
3	1	0	1	1	0	0	0	0
4	1	0	0	1	1	0	0	1

Tabla 1: Valores lógicos de los pines necesarios para encender los segmentos formando los digitos 1 - 4

4. Software

Mediante el software Microchip Studio, se desarrolló el programa a implementar en código Assembly.

Para hacer uso del puerto serie, fue necesario configurar 5 registros asociados al USART (receptor/transmisor universal síncrono/asíncrono): un registro de datos (UDR), tres registros de estado y control (UCSRA, UCSRB y UCSRC), y un registro destinado a la tasa de baudios o baud rate (UBRR).

El valor asignado al registro UBRR determina el baund rate, por lo que para adoptar una velocidad de 9600 bits por segundo con una frecuencia Fosc de 16MHz, fue necesario realizar el siguiente cálculo:

$$UBRR = \frac{Fosc}{16 \cdot baundratedeseado} - 1 = \frac{16MHz}{16 \cdot 9600bps} - 1 = 103,166 \approx 103 \quad (1)$$

A su vez, se decidió configurar una trama de datos con: 8 bits de datos, un bit de inicio y otro de fin de trama, sin paridad. Respecto a la sincronización se adoptó la asíncrona, es por ello que se agrega un bit de inicio cuyo valor siempre es '0' y otro de fin con un valor constante '1' que permiten sincronizar el transmisor con el receptor. Por lo tanto, cada byte de información útil se convierte en 10 bits comunicados, siendo el tiempo de transmisión de cada byte de 10/9600 segundos. Para ello fue necesario mantener en estado bajo('0') el bit 2 del registro UCSRB correspondiente a UCSZ2 y en estado alto('1') los bits 2 y 1 correspondientes a UCSZ1 y UCSZ0 del registro UCSRC.

4.1. Diagrama de flujo

En la figura 2 se puede observar el diagrama de flujo correspondiente al software principal.

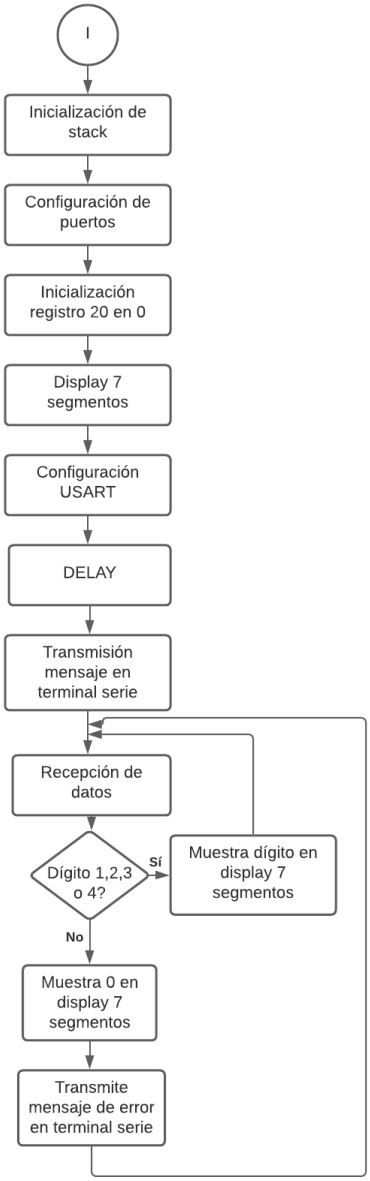

Figura 2: Diagrama de flujo

4.2. Diagrama en bloques

La figura 3 muestra el diagrama en bloques. Como se puede observar se conectó la placa Arduino UNO con una computadora mediante un cable USB como alimentación del circuito y para cargar el software correspondiente.

Figura 3: Diagrama en bloques

5. Resultados

Se lograron los resultados deseados, para ello se hizo uso del porgrama Realterm para realizar la transmisión y recepción de datos a traves de la terminal serie, estos se pueden observar en el video del siguiente link:

https://youtu.be/jXa9MuabgbA

Dado que la terminal serie del programa Realterm muestra 'CR' y 'LF' de los caracteres ASCII, se utilizó el programa Proteus para verificar que dichos caracteres no se muestren por error en el algoritmo. Se muestra en la imagen 4 el resultado de la terminal serie mediante el uso del programa Proteus.

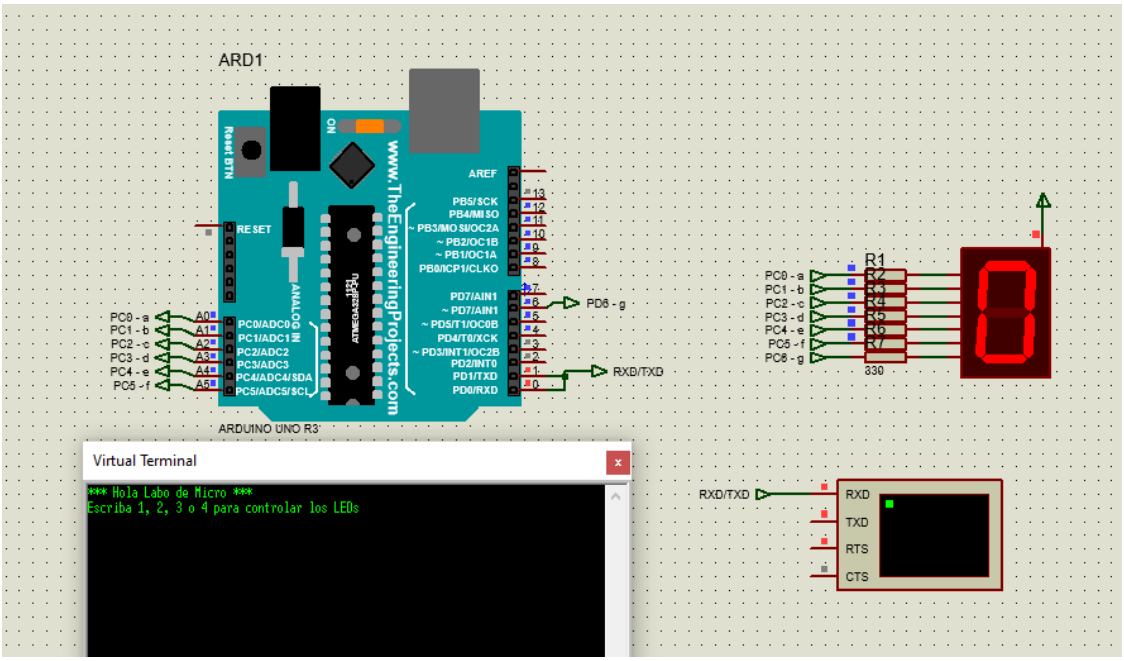

Figura 4: Terminal serie Proteus

6. Conclusiones

Se puede concluir que este trabajo práctico permitió conocer el manejo de transmisión y recepción de datos a partir de la configuración de puerto serie. Se puede decir que los resultados conseguidos fueron los deseados.

7. Anexo

7.1. Código fuente inciso 1

```
;
; Laboratorio de microprocesadores (86.07)
; Nahila Lutzelschwab
; Padron : 100686
; 1er cuatrimestre 2021
; Turno Martes 19hs
;

.INCLUDE "m328pdef.inc"

.CSEG        ;Segmento de codigo


.EQU UBRR = 103 ; Valor asignado a los primeros 12 bits del UBRR siendo
          ; la frecuencia de 16MHz y un baund rate de 9600 bits/segundo

; Redefine tabla ascii con nombres significativos
.EQU CR = $0D    ; Retorno de carro
.EQU LF = $0A    ; Salto de linea
.EQU ONE = 49
```

```
.EQU TWO = 50
.EQU THREE = 51
.EQU FOUR = 52
.EQU INITIALIZE_BCD = 1

; Valores tabla de segmentos
.EQU ZERO_TABLE = 0X00
.EQU ONE_TABLE = 0x01
.EQU TWO_TABLE = 0x02
.EQU THREE_TABLE = 0x03
.EQU FOUR_TABLE = 0x04

; Redefine registros con nombres significativos
.DEF BCD_REG = R21
.DEF RECEIVE_REG = R23
.DEF IN_NUMBER = R24

.ORG 0
  RJMP START
.MACRO USART_CONF
  LDI R16, LOW(@0)
  STS UBRR0L, R16   ; Carga el contenido de R16 en la parte baja del registro baud rate USART
  LDI R17, HIGH(@0)
  STS UBRR0H, R17   ; Carga el contenido de R16 en la parte alta del registro baud rate USART
  LDI R16, (1<<RXEN0)|(1<<TXEN0) ; Habilita transmision y recepcion
  STS UCSR0B,R16
  LDI R16, (1<<UCSZ00)|(1<<UCSZ01) ; Sin bit de paridad, transmision asincronica de 8 bits
  STS UCSR0C, R16
.ENDMACRO



START:

; Inicia el stack en la parte alta de la memoria
  LDI R16, LOW(RAMEND)
  OUT SPL, R16
  LDI R16, HIGH(RAMEND)
  OUT SPH, R16

  RCALL PORTS_CONF  ;Configura puertos

  LDI IN_NUMBER, 0x00
  LDI R20, ZERO_TABLE
  RCALL BCD_SEG
  RCALL DELAY

  USART_CONF UBRR ; Configura USART
  RCALL DELAY   ; Delay de 10ms para abrir el terminal serie

; Inicializa puntero Z al comienzo del mensaje
  LDI ZL, LOW(MESSAGE<<1) ; Carga el puntero Z con la direccion del mensaje
  LDI ZH, HIGH(MESSAGE<<1)
  RCALL USART_TRANSMIT ; Lo transmite a la terminal

  RCALL NUMBER
  RET



/*
; Configuraciones USART
USART_CONF:

; Configura el baud rate USART para  f=16MHz y baud_rate=9600bps -> UBRR_X=103

; | URSEL | - | - | - | UBRR[11:8] | -> UBRRnH (registro baud rate USART)
; | UBRR[7:0] | -> UBRRnL (registro baud rate USART)
  LDI R16, LOW(UBRR)
```

```
90    STS UBRR0L, R16      ; Carga el contenido de R16 en la parte baja del registro baud rate
        USART
91    LDI R17, HIGH(UBRR)
92    STS UBRR0H, R17      ; Carga el contenido de R16 en la parte alta del registro baud rate
        USART
93
94  ; Habilita recepcion y transmision
95  ; | RXCIEn | TXCIEn | UDRIEn | RXENn | TXENn | UCSZn2 | RXB8n | TXB8n | -> UCSRnB (registro B
        de estados y control USART)
96    LDI R16, (1<<RXEN0)|(1<<TXEN0)  ; Habilita transmision y recepcion
97    STS UCSR0B, R16  ; Configura formato
98  ; | URSELn | UMSELn | UPMn1 | UPMn1 | USBSn | UCSZn1 | UCSZn0 | UCPOLn | -> UCSRnC (registro
        C de estados y control USART)
99    LDI R16, (1<<UCSZ00)|(1<<UCSZ01)  ; Setea 8 bits de datos, sin paridad y asincrono
100   STS UCSR0C, R16
101   RET
102 */
103
104 ; Configura puertos
105 PORTS_CONF:
106
107   LDI R17, (1<<DDD7)|(1<<DDD6)|(1<<DDD5)|(1<<DDD4)  ; Habilita el puerto D como entrada
108                                                      ; en la parte baja y en la parte alta como
        salida
109   OUT DDRD, R17
110
111   LDI R17, 0xFF
112   OUT DDRC, R17   ; Habilita el puerto C como salida
113
114   LDI R17, 0x00
115   OUT PORTC, R17
116
117
118   LDI R18, 0B000011111 ; Habilita el puerto B como salida
119             ; en la parte baja y en la parte alta como entrada
120   OUT DDRB, R18
121   RET
122
123
124
125 ; Transmision USART
126 USART_TRANSMIT:
127   LPM R16, Z+       ; Carga un byte de lo apuntado por Z en R16 y post-incrementa
128   TST R16          ; Si el dato es cero -> fin de mensaje
129   BREQ TRANSMIT_RETURN
130 TRANSMIT_BUFFER:
131   LDS  R17, UCSR0A     ; Carga el registro UCSR0A en R17
132   SBRS R17, UDRE0      ; Si UDRE0='1' el bufer de datos de transmision esta vacio
133   RJMP TRANSMIT_BUFFER  ; Si UDRE0 era'0'(buffer no vacio)-> no debe escribir el UDR0
134               ; porque anula sus ultimos datos, por lo que chequea el estado
135               ; constantemente hasta recibir datos
136   STS UDR0,R16       ; Escribe el dato leido en el registro de datos
137   RJMP USART_TRANSMIT   ; Repite hasta fin del mensaje
138 TRANSMIT_RETURN:
139   RET
140
141 ; Recepcion USART
142 USART_RECEIVE:
143   LDS R17, UCSR0A      ; Carga el registro UCSR0A en R17
144   SBRS R17, RXC0       ; Si RXC0 = '1' hay datos nuevos en el buffer del receptor aun no leidos
145   RJMP USART_RECEIVE    ; Si RXC0 era '0' -> buffer del receptor vacio, por lo que vuelve a
        leer
146   LDS RECEIVE_REG, UDR0 ; Carga el contenido del registro de datos UDR0 en R23
147   RET
148
149
150 ; Mostrar numeros
151 NUMBER:
152   ;RCALL DELAY
```

```
153   LDI IN_NUMBER, 1     ; Si se encuentra dentro del loop vale 1 sino vale 0
154   RCALL USART_RECEIVE
155
156   LDI R20, ONE_TABLE
157   CPI RECEIVE_REG, ONE  ; Compara con el valor '1' ASCII (49)
158   BREQ BCD_SEG          ; Muestra en el display
159
160   LDI R20, TWO_TABLE
161   CPI RECEIVE_REG, TWO  ; Compara con el valor '2' ASCII (50)
162   BREQ BCD_SEG          ; Muestra en el display
163
164   LDI R20, THREE_TABLE
165   CPI RECEIVE_REG, THREE  ; Compara con el valor '3' ASCII (51)
166   BREQ BCD_SEG          ; Muestra en el display
167
168   LDI R20, FOUR_TABLE
169   CPI RECEIVE_REG, FOUR ; Compara con el valor '4' ASCII (52)
170   BREQ BCD_SEG          ; Muestra en el display
171
172   LDI R20, ZERO_TABLE   ; Si el valor ingresado no es ni '1','2','3' o '4'
173   RCALL ERROR_MSG       ; muestra en el display un '0' y un mensaje de error
174
175   RET
176
177
178
179 ; BCD a 7 segmentos
180 BCD_SEG:               ; Inicializa puntero Z al comienzo de la tabla
181   LDI ZL, LOW(SEGMENTS << 1)  ; R30 apunta a la parte baja de la direccion
182   LDI ZH, HIGH(SEGMENTS << 1) ; R31 apunta a la parte alta de la direccion
183   LDI R19, 0
184   ADD ZL, R20           ; Le suma la posicion a  ZL
185   ADC ZH, R19           ; Si la suma tiene carry se lo asigna a ZH
186   LPM R19, Z            ; Carga lo apuntado por Z en R16
187   RCALL SHOW_SEG          ; Muestra los segmentos en el display a traves de PORTD
188   CPI IN_NUMBER, 1        ; Si era la inicializacion del display no salta
189                 ; sino se puede volver a ingresar un numero
190   BREQ NUMBER
191   RET
192
193
194
195 ; Muestra display
196 SHOW_SEG:
197   OUT PORTC, R19      ; Carga lo apuntado por Z en PORTC
198   SBRS R19, 6
199   CBI PORTD, 6
200   SBRC R19, 6
201   SBI PORTD, 6
202   RET
203
204
205 ERROR_MSG:
206   LDI ZL, LOW(ERROR<<1) ; Carga el puntero Z con la direccion del mensaje
207   LDI ZH, HIGH(ERROR<<1)
208   RCALL USART_TRANSMIT  ; Lo transmite a la terminal
209   BREQ BCD_SEG
210   RET
211
212
213
214 ; Delay de 10ms para abrir el terminal serie
215 DELAY:
216   LDI  R21, 208
217     LDI  R22, 202
218 LOOP1:
219   DEC  R22
220     BRNE LOOP1
221     DEC  R21
```

```
222     BRNE LOOP1
223     NOP
224   RET
225
226
227
228 .ORG 0X500     ; Vector del mensaje
229 MESSAGE:
230 .db "*** Hola Labo de Micro ***", CR, LF, "Escriba 1, 2, 3 o 4 para controlar los LEDs", CR,
      LF, 0
231
232 .ORG 0X600     ; Vector del mensaje de error
233 ERROR:
234 .db "Error, valor ingresado invalido. Debe ingresar 1, 2, 3 o 4", CR, LF, 0
235
236 ; PUERTO C, PC0->a, PC1->b, ..., PC5->f, PD6->g
237 ; Display 7 segmentos anodo comun -> enciende con '0'
238 .ORG 0X700       ; Vector de la tabla
239
240 SEGMENTS:
241 .db 0b11000000, 0b11111001,  0b10100100, 0b10110000
242 ;     0              1             2            3
243 .db 0b10011001 , 0b10010010, 0b10000010, 0b11111000
244 ;     4              5             6            7
245 .db 0b10000000, 0b10010000
246 ;     8              9
```

7.2. Código fuente inciso 2

```
1  ;
2  ; Laboratorio de microprocesadores (86.07)
3  ; Nahila Lutzelschwab
4  ; Padron : 100686
5  ; 1er cuatrimestre 2021
6  ; Turno Martes 19hs
7  ;
8
9
10 ; CON INTERRUPCIONES
11
12
13 .INCLUDE "m328pdef.inc"
14
15 .CSEG        ;Segmento de codigo
16
17
18 .EQU UBRR = 103 ; Valor asignado a los primeros 12 bits del UBRR siendo
19         ; la frecuencia de 16MHz y un baund rate de 9600 bits/segundo
20
21 ; Redefine tabla ascii con nombres significativos
22 .EQU CR = $0D    ; Retorno de carro
23 .EQU LF = $0A    ; Salto de linea
24 .EQU ONE = 49
25 .EQU TWO = 50
26 .EQU THREE = 51
27 .EQU FOUR = 52
28 .EQU INITIALIZE_BCD = 1
29
30 ; Valores tabla de segmentos
31 .EQU ZERO_TABLE = 0X00
32 .EQU ONE_TABLE = 0x01
33 .EQU TWO_TABLE = 0x02
34 .EQU THREE_TABLE = 0x03
35 .EQU FOUR_TABLE = 0x04
36
37 ; Redefine registros con nombres significativos
38 .DEF BCD_REG = R21
```

```
.DEF RECEIVE_REG = R23
.DEF IN_NUMBER = R24
.DEF STATUS_REG = R25

.ORG 0
  RJMP START

.ORG UDREaddr    ; Vector de interrupcion
  RJMP ISR_UDRE


.MACRO USART_CONF
  LDI R16, LOW(@0)
  STS UBRR0L, R16    ; Carga el contenido de R16 en la parte baja del registro baud rate USART
  LDI R17, HIGH(@0)
  STS UBRR0H, R17    ; Carga el contenido de R16 en la parte alta del registro baud rate USART
  LDI R16, (1<<RXEN0)|(1<<TXEN0) ; Habilita transmision y recepcion
  STS UCSR0B,R16
  LDI R16, (1<<UCSZ00)|(1<<UCSZ01) ; Sin bit de paridad, transmision asincronica de 8 bits
  STS UCSR0C, R16
.ENDMACRO



START:

; Inicia el stack en la parte alta de la memoria
  LDI R16, LOW(RAMEND)
  OUT SPL, R16
  LDI R16, HIGH(RAMEND)
  OUT SPH, R16

  RCALL PORTS_CONF  ;Configura puertos

  RCALL ENABLE_ISR ; Habilita las interrupciones por recepcion RXC en UCSRA
  SEI         ; Habilita las interrupciones globales

  LDI IN_NUMBER, 0x00
  LDI R20, ZERO_TABLE
  RCALL BCD_SEG
  RCALL DELAY

  USART_CONF UBRR ; Configura USART
  RCALL DELAY    ; Delay de 10ms para abrir el terminal serie


; Inicializa puntero Z al comienzo del mensaje
  LDI ZL, LOW(MESSAGE<<1) ; Carga el puntero Z con la direccion del mensaje
  LDI ZH, HIGH(MESSAGE<<1)
  RCALL USART_TRANSMIT ; Lo transmite a la terminal



LOOP:
  RJMP LOOP



/*
; Configuraciones USART
USART_CONF:

; Configura el baud rate USART para  f=16MHz y baud_rate=9600bps -> UBRR_X=103

; | URSEL | - | - | - | UBRR[11:8] | -> UBRRnH (registro baud rate USART)
; | UBRR[7:0] | -> UBRRnL (registro baud rate USART)
  LDI R16, LOW(UBRR)
  STS UBRR0L, R16     ; Carga el contenido de R16 en la parte baja del registro baud rate
    USART
```

```
  LDI R17, HIGH(UBRR)
  STS UBRR0H, R17      ; Carga el contenido de R16 en la parte alta del registro baud rate USART

; Habilita recepcion y transmision
; | RXCIEn | TXCIEn | UDRIEn | RXENn | TXENn | UCSZn2 | RXB8n | TXB8n | -> UCSRnB (registro B de estados y control USART)
  LDI R16, (1<<RXEN0)|(1<<TXEN0)  ; Habilita transmision y recepcion
  STS UCSR0B, R16  ; Configura formato
; | URSELn | UMSELn | UPMn1 | UPMn1 | USBSn | UCSZn1 | UCSZn0 | UCPOLn | -> UCSRnC (registro C de estados y control USART)
  LDI R16, (1<<UCSZ00)|(1<<UCSZ01)  ; Setea 8 bits de datos, sin paridad y asincrono
  STS UCSR0C, R16
  RET
*/

; Habilita las interrupciones por recepcion (RXC en UCSR0A)
ENABLE_ISR:
  LDS R26, UCSR0B
  SBR R26, (1<<RXCIE0)
  STS UCSR0B, R26
  RET

; Configura puertos
PORTS_CONF:

  LDI R17, (1<<DDD7)|(1<<DDD6)|(1<<DDD5)|(1<<DDD4)  ; Habilita el puerto D como entrada
                                                    ; en la parte baja y en la parte alta como salida
  OUT DDRD, R17

  LDI R17, 0xFF
  OUT DDRC, R17   ; Habilita el puerto C como salida

  LDI R17, 0x00
  OUT PORTC, R17


  LDI R18, 0B000011111 ; Habilita el puerto B como salida
            ; en la parte baja y en la parte alta como entrada
  OUT DDRB, R18
  RET



; Transmision USART
USART_TRANSMIT:
  LPM R16, Z+       ; Carga un byte de lo apuntado por Z en R16 y post-incrementa
  TST R16          ; Si el dato es cero -> fin de mensaje
  BREQ TRANSMIT_RETURN
TRANSMIT_BUFFER:
  LDS  R17, UCSR0A    ; Carga el registro UCSR0A en R17
  SBRS R17, UDRE0     ; Si UDRE0='1' el bufer de datos de transmision esta vacio
  RJMP TRANSMIT_BUFFER  ; Si UDRE0 era'0'(buffer no vacio)-> no debe escribir el UDR0
              ; porque anula sus ultimos datos, por lo que chequea el estado
              ; constantemente hasta recibir datos
  STS UDR0,R16      ; Escribe el dato leido en el registro de datos
  RJMP USART_TRANSMIT   ; Repite hasta fin del mensaje
TRANSMIT_RETURN:
  RET

; Recepcion USART
USART_RECEIVE:
  LDS R17, UCSR0A     ; Carga el registro UCSR0A en R17
  SBRS R17, RXC0      ; Si RXC0 = '1' hay datos nuevos en el buffer del receptor aun no leidos
  RJMP USART_RECEIVE    ; Si RXC0 era '0' -> buffer del receptor vacio, por lo que vuelve a leer
  LDS RECEIVE_REG, UDR0 ; Carga el contenido del registro de datos UDR0 en R23
  RET
```

```
171
172
173 ; Mostrar numeros
174 ISR_UDRE:
175   IN STATUS_REG, SREG
176   PUSH STATUS_REG
177   ;RCALL DELAY
178   LDI IN_NUMBER, 1     ; Si se encuentra dentro del loop vale 1 sino vale 0
179   RCALL USART_RECEIVE
180
181   LDI R20, ONE_TABLE
182   CPI RECEIVE_REG, ONE  ; Compara con el valor '1' ASCII (49)
183   BREQ BCD_SEG       ; Muestra en el display
184
185   LDI R20, TWO_TABLE
186   CPI RECEIVE_REG, TWO  ; Compara con el valor '2' ASCII (50)
187   BREQ BCD_SEG       ; Muestra en el display
188
189   LDI R20, THREE_TABLE
190   CPI RECEIVE_REG, THREE  ; Compara con el valor '3' ASCII (51)
191   BREQ BCD_SEG       ; Muestra en el display
192
193   LDI R20, FOUR_TABLE
194   CPI RECEIVE_REG, FOUR ; Compara con el valor '4' ASCII (52)
195   BREQ BCD_SEG       ; Muestra en el display
196
197   LDI R20, ZERO_TABLE   ; Si el valor ingresado no es ni '1','2','3' o '4'
198   RCALL ERROR_MSG      ; muestra en el display un '0' y un mensaje de error
199 RETURN:
200   POP STATUS_REG
201   OUT SREG, STATUS_REG
202   RETI
203
204
205
206 ; BCD a 7 segmentos
207 BCD_SEG:             ; Inicializa puntero Z al comienzo de la tabla
208   LDI ZL, LOW(SEGMENTS << 1)  ; R30 apunta a la parte baja de la direccion
209   LDI ZH, HIGH(SEGMENTS << 1) ; R31 apunta a la parte alta de la direccion
210   LDI R19, 0
211   ADD ZL, R20          ; Le suma la posicion a  ZL
212   ADC ZH, R19          ; Si la suma tiene carry se lo asigna a ZH
213   LPM R19, Z           ; Carga lo apuntado por Z en R16
214   RCALL SHOW_SEG         ; Muestra los segmentos en el display a traves de PORTD
215   CPI IN_NUMBER, 1       ; Si era la inicializacion del display no salta
216                 ; sino se puede volver a ingresar un numero
217   BREQ ISR_UDRE
218   RJMP RETURN          ; Devuelve el registro de estados
219   RET
220
221
222
223 ; Muestra display
224 SHOW_SEG:
225   OUT PORTC, R19     ; Carga lo apuntado por Z en PORTC
226   SBRS R19, 6
227   CBI PORTD, 6
228   SBRC R19, 6
229   SBI PORTD, 6
230   RET
231
232
233 ERROR_MSG:
234   LDI ZL, LOW(ERROR<<1) ; Carga el puntero Z con la direccion del mensaje
235   LDI ZH, HIGH(ERROR<<1)
236   RCALL USART_TRANSMIT  ; Lo transmite a la terminal
237   BREQ BCD_SEG
238   RET
239
```

```
240
241
242 ; Delay de 10ms para abrir el terminal serie
243 DELAY:
244   LDI   R21, 208
245     LDI   R22, 202
246 LOOP1:
247   DEC   R22
248     BRNE LOOP1
249     DEC   R21
250     BRNE LOOP1
251     NOP
252   RET
253
254
255
256 .ORG 0X500     ; Vector del mensaje
257 MESSAGE:
258 .db "*** Hola Labo de Micro ***", CR, LF, "Escriba 1, 2, 3 o 4 para controlar los LEDs", CR,
      LF, 0
259
260 .ORG 0X600     ; Vector del mensaje de error
261 ERROR:
262 .db "Error, valor ingresado invalido. Debe ingresar 1, 2, 3 o 4", CR, LF, 0
263
264 ; PUERTO C, PC0->a, PC1->b, ..., PC5->f, PD6->g
265 ; Display 7 segmentos anodo comun -> enciende con '0'
266 .ORG 0X700       ; Vector de la tabla
267
268 SEGMENTS:
269 .db 0b11000000, 0b11111001,  0b10100100, 0b10110000
270 ;     0              1             2            3
271 .db 0b10011001 , 0b10010010, 0b10000010, 0b11111000
272 ;     4              5             6            7
273 .db 0b10000000, 0b10010000
274 ;     8              9
```

8. Bibliografía

- Mazidi, M. A., Naimi, S., Naimi, S. (2010). AVR Microcontroller and Embedded Systems: Using Assembly and C (Pearson Custom Electronics Technology). New Jersey, United States of America: Pearson
- ATMEGA328P Datasheet (PDF) - ATMEL.http://ww1.microchip.com/downloads/ Corporation. en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P$_{D}atasheet.pdf$