Trabajo práctico N°3

Puerto serie

Laboratorio de microprocesadores
Primer Cuatrimestre del 2021

Becker, Gonzalo Agustín	104291

Docentes:
Gerardo Stola
Fernando Cofman
Guido Salaya

Índice

1. Objetivos

Mediante este proyecto, se logra exponer la funcionalidad de comunicación serie que proveen los microcontroladores. En esta línea, se encuentra en este texto una descripción detallada para obtener una comunicación efectiva entre una computadora de escritorio y el Mcu Atmega328P, utilizado para ello además un conversor USB a TTL. Aún más, en diferentes secciones del presente informe se encontrarán descripciones del código en Assembly, para así permitir al lector desarrollar una comprensión más profunda sobre el tema en estudio.

2. Descripción del proyecto

En el siguiente link se encuentra la explicación del proyecto.

En este caso, se ha utilizado a un Arduino Nano como ISP para programar al microcontrolador. Para más detalles sobre este procedimiento, referirse al "TP N°1: Manejo de puertos", sección 2, de esta misma serie.

El funcionamiento del proyecto es el siguiente: una vez establecida la comunicación serie entre el microcontrolador y la computadora, se transmitirá un mensaje por pantalla indicándole al usuario como proceder para encender los 4 LED presentes en el proyecto. Luego, el usuario utilizará las cuatro teclas numéricas "1", "2", "3", "4"para enviar señales desde la computadora, hacia el conversor USB a TTL, y de allí al terminal Rx del microcontrolador (notar que el terminal Tx del conversor estará conectado al terminal Rx del MCU, y viceversa). De esta forma, cada LED se encenderá o apagará según cuál sea la tecla presionada por el usuario.

Con dicho objetivo, se realizarán dos implementaciones del código en Assembly, una que lee indefinidamente el terminal Rx del microcontrolador, y otra que utiliza la herramienta de las interrupciones.

3. Circuito esquemático

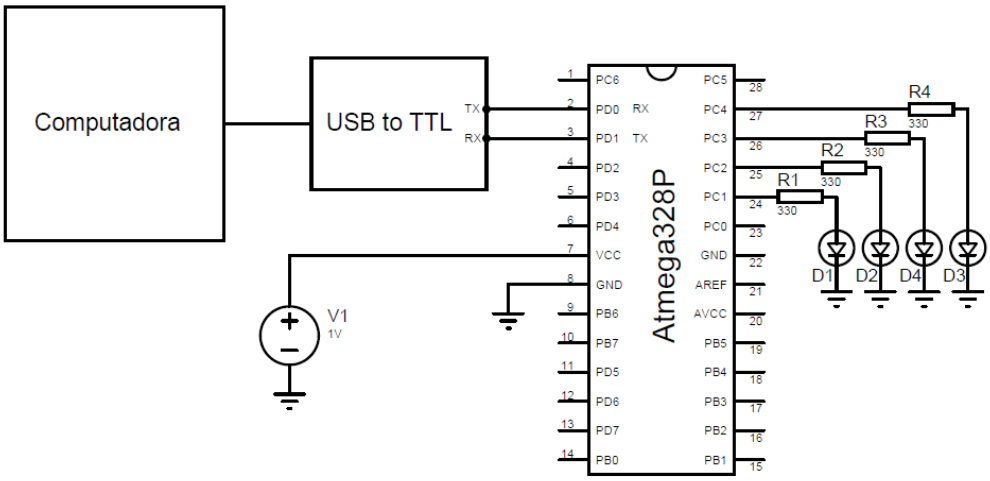

Figura 1: Circuito a implementar

4. Diagrama de flujo

A continuación se encuentran los diagramas de flujo para las diferentes implementaciones del código. En la figura 2, notar que solamente se lee un caracter cuando hay información aún no leída en el buffer de recepción.

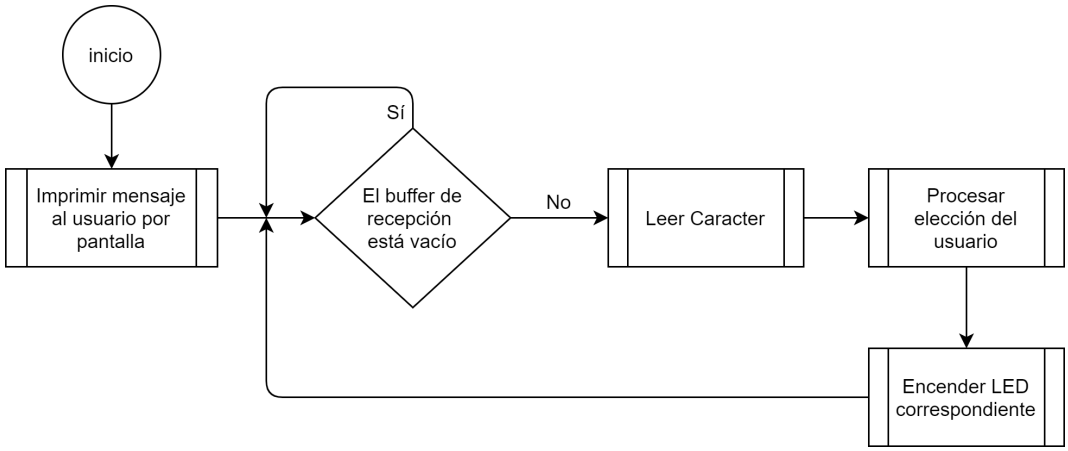

Figura 2: Diagrama de flujo sin interrupciones

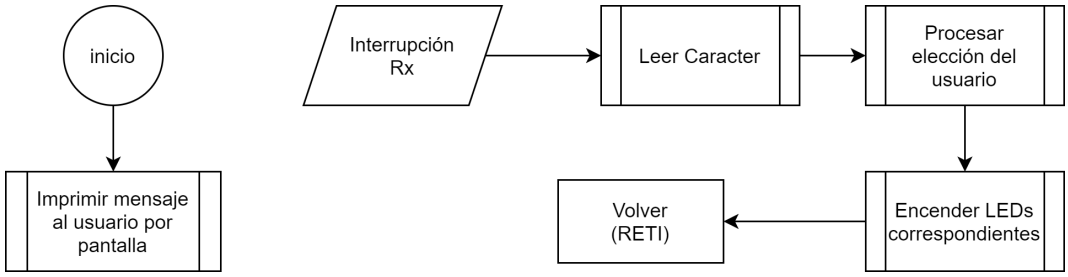

Figura 3: Diagrama de flujo con interrupciones

5. Programación en Assembly

A continuación se describe el código de Assembly del proyecto. Se incluyen explicaciones de las herramientas que provee Assembly, en adición a las descripciones de los principales segmentos del código. Cabe destacar, a fin de simplificar la lectura del código, que se utilizan las siguientes definiciones:

```
1 .def temp = r21
2 .def option = r22
3 .def portc_output = r23
```

5.1. Habilitación de la transmisión UART en formato 8N1

El código que permite configurar la transmisión es el siguiente:

```
;Se incializa el puerto serie
ldi r16, LOW(bps); prescaler de puerto serie
ldi r17, HIGH(bps)
rcall uartInit

uartInit:
    sts UBRR0L, r16; prescaler
    sts UBRR0H, r17
    ; 8N1
    ldi         r16, (0<<UMSEL00) | (0<<UPM00) | (0<<USBS0) | (3<<UCSZ00)
    sts UCSR0C, r16

        ldi r16, (1<<RXEN0) | (1<<TXEN0) | (0<<UCSZ02); enable transmitter and
        ↪  receiver, 8bits
        sts UCSR0B, r16

    ret
```

A continuación, se explica cada registro con más detalle:

- **UBRR0L** Observar que previo a la carga de este registro, el código carga el valor del *prescaler* en los registros r16 y r17.

Bit	15	14	13	12	11	10	9	8	
	–	–	–	–	UBRRn[11:8]				UBRRnH
	UBRRn[7:0]								UBRRnL
	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	

Figura 4

Los bits 15 a 12 están reservados para su uso en futuras versiones del microcontrolador, motivo por el cual deben ser incializados a cuatro siempre que se utilize este registro. Los restantes 12 bits definen el *baud rate*, como un número de 12 bits.

- **UCSR0C**: Este registro permite definir la configuración de la codificación que tendrán los bytes que se envían a través del puerto serie.

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	

Figura 5

A partir del esquema en la figura 5, se pueden definir las funcionalidades de cada bit. Los bits 6 y 7 seleccionan el modo de operación del USARTn. Los bits 5 y 4 definen a los bits de paridad, mientras que el bit 3 seleccionan el número de bits de paro (*stop bits*). Los bits 2 y

1 definen el tamaño de los caracteres, y el bit 0 se pone en alto solamente en caso de que se use en modo sincrónico.

Así, se definen los valores de estos bits de forma correcta para obtener la configuración deseada.

- **UCSR0B**

  Nuevamente, se definen aquí los bits necesarios para obtener la configuración buscada.

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	

Figura 6

En este caso, los bits 7 y 6 definen las interrupciones en Rx y Tx. El bit 5 habilita la interrupción en el falg UDREn. Los bits 4 y 3 habilitan al transmisor y receptor n. El bit 2 define la cantidad de bits que usan el transmisor y receptor por segemento. Los bits 1 y 0 son los novenos bits de los caracteres transmitidos, que deben leerse o escribirse antes de realizar cualquier operación de lectura/escritura del caracter.

5.2. Transmitir un mensaje

Aquí se intenta explicar el código asociado a la transmisión de un mensaje hacia el ordenador. Para ello, se define una tabla en la memoria de programa, la cual contiene al mensaje en cuestión. Se toma como convención que una cadena finaliza cuando aparece un 0 en la memoria.

```
message:
        .db "*** Hola Labo de Micro *** Escriba 1, 2, 3 o 4 para controlar los
        ↪  LEDs", 0, 0
```

Luego, se carga el mensaje en el puntero z, y se procede a transmitir el mensaje.

```
ldi zl, LOW(message<<1) ; carga el puntero al mensaje constante en zh:zl
ldi zh, HIGH(message<<1)
rcall show_string ; Transmitir mensaje contenido en flash

show_string:
        lpm r16, Z+
        cpi r16, 0x00
        breq end_show_string

wait:
        lds r17, UCSR0A
        sbrs r17, UDRE0
        rjmp wait ;verificar estado del buffer de transmisión

        sts UDR0, r16
        rjmp show_string
```

```
17
18  end_show_string:
19          ret
```

A continuación, se encuentra una breve explicación de los registros utilizados.

- **UCSR0A**

Bit	7	6	5	4	3	2	1	0	
	RXCn	TXCn	UDREn	FEn	DORn	UPEn	U2Xn	MPCMn	UCSRnA
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	1	0	0	0	0	0	

Figura 7

El bit 7 está en estado alto cuando hay información aún sin leer en el buffer de recepción, y lo mismo para el bit 6 pero con el buffer de transmisión. El bit 5 indica, en caso de estar en estado alto, si es posible recibir nueva data. El bit 4 indica si hubo un error de segmentación. El bit 3 se enciende cuando existe solapamiento de información en el buffer de recepción. Por último, el bit 2 marca errores de paridad, el 1 duplica la velocidad de transmisión, y el 0 habilita el multiprocesamiento en la transmisión.

- **UDR0**

Bit	7	6	5	4	3	2	1	0	
	RXB[7:0]								UDRn (Read)
	TXB[7:0]								UDRn (Write)
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	0	0	0	

Figura 8

Los valores escritos sobre estos registros serán enviados al buffer de transmisión, mientras que leer el valor de estos registros devolverá un valor del buffer de recepción. Se utiliza para lograr la comunicación efectiva en los pines Rx y Tx del microcontrolador.

5.3. Recibir un mensaje

A partir de los registros ya explicados, se puede leer un caracter:

```
1  rcall getc
2
3  getc:
4          lds temp, UCSR0A; cargar UCSR0A en temp
5          sbrs temp, RXC0; Esperar a que se vacie el buffer de recepción
6          rjmp getc
7          lds option, UDR0; recibir carácter deseado
8          ret
```

5.4. Lógica principal

Aquí se observa la lógica principal del código. Una vez impreso el mensaje al usuario, se comienza a leer continuamente el valor en el buffer de recepción (Rx). En caso de poder leerse un valor, se lo compara contra la codificación ASCII de los símbolos "1", "2", "3", "4". En caso de coincidir, se decide si apagar o encender el led asociado, según corresponda.

```
reset:

        ;Se inicializa al Stack
        ldi temp, LOW(RAMEND)
        out SPL, temp
        ldi temp, HIGH(RAMEND)
        out SPH, temp

        ;Se configuran a C0, C1, C2 y C3 como pines de salida

        ldi temp, 0b00011110
        out DDRC, temp
        ldi temp, 0
        out PORTC, temp

        ;Se incializa el puerto serie
        ldi r16, LOW(bps); prescaler de puerto serie
    ldi r17, HIGH(bps)
        rcall uartInit

        rcall delay_s
        ldi zl, LOW(message<<1); carga el puntero al mensaje constante en zh:zl
    ldi zh, HIGH(message<<1)
    rcall show_string; Transmitir mensaje contenido en flash
        rjmp main


main:
        rcall getc
        rcall process_option;En ASCII, el 1 es 0b00110001, por lo que se hace
        ↪  el análsisis pertinente

        in temp, PINC
        eor portc_output, temp

        out PORTC, portc_output
    rjmp main; Volver a leer continuamente

process_option:

        cpi option, 0x31
        breq led1
        cpi option, 0x32
        breq led2
        cpi option, 0x33
        breq led3
```

```
        cpi option, 0x34
        breq led4
        clr portc_output
        ret

led1:
        ldi portc_output, 0b00000010
        ret

led2:
        ldi portc_output, 0b00000100
        ret

led3:
        ldi portc_output, 0b00001000
        ret

led4:
        ldi portc_output, 0b00010000
        ret
```

5.5. Interrupciones

Las interrupciones son una herramienta poderosa que proveen los microcontroladores para permitir que el programa se desvíe de su funcionamiento en caso de que un pin de entrada cambie de estado lógico, y así evitar tener que leer constantemente su valor.

Para implementar una interrupción en el pin Rx, es necesario utilizar el vector de interrupción USART, RX, localizado en la posición de memoria 0x0024. De esta forma, el código a utilizar es el siguiente:

```
.org 0x0000
    rjmp reset

.org URXCaddr        ;RX
        rjmp handle_instruction
```

Luego, la lógica implementada para lograr leer la selección del usuario y su procesamiento se encontrará en este caso dentro de la interrupción, como se observa a continuación.

```
handle_instruction:
        in r25, SREG
        push r25

        getc:
        lds temp, UCSR0A    ; cargar UCSR0A en temp
        sbrs temp, RXC0     ; Esperar a que se vacie el buffer de recepción
        rjmp getc
        lds        option, UDR0        ; recibir caracter deseado
```

```
10
11          cpi option, 0x31
12          breq led1
13
14          cpi option, 0x32
15          breq led2
16
17          cpi option, 0x33
18          breq led3
19
20          cpi option, 0x34
21          breq led4
22
23          reti
24
25          led1:
26                  ldi portc_output, 0b00000010
27                  rjmp handle_instruction_end
28
29          led2:
30                  ldi portc_output, 0b00000100
31                  rjmp handle_instruction_end
32
33          led3:
34                  ldi portc_output, 0b00001000
35                  rjmp handle_instruction_end
36
37          led4:
38                  ldi portc_output, 0b00010000
39                  rjmp handle_instruction_end
40
41          handle_instruction_end:
42
43                  in temp, PINC
44                  eor portc_output, temp
45                  out PORTC, portc_output
46
47                  pop         r25
48                  out         SREG, r25
49                  reti
```

Para completar el proceso de configuración de la interrupción, es necesario usar la instrucción sei (set global interrupt flag).

6. Lista de materiales

Material	costo ($)
4x LED	88.8
Atmega328P	504.7
Arduino nano	539.0
7 x Resistor 330 Ω	13.85
Conversor USB a TTL	305.60
Total	1451.95

NOTA: Se incluyen los costos del programador. Los precios son a Abril de 2021.

7. Resultados

En la figura 9 se encuentra la implementación que se ha realizado del circuito explicado en este informe. No se han encontrado grandes inconvenientes, salvo que el puerto de la computadora tuvo dificultades para detectar al módulo conversor USB a TTL.

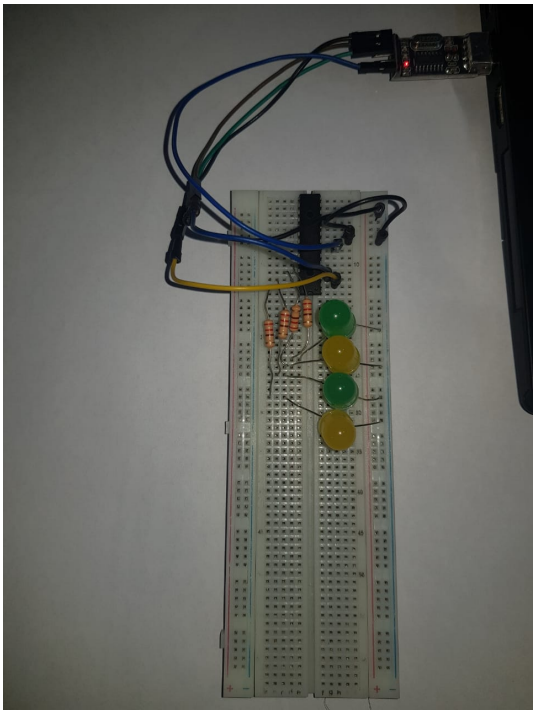

Figura 9: Circuito del proyecto

8. Conclusión

En el presente informe, se ha realizado una descripción detallada para lograr la efectiva comunicación serie entre el microcontrolador Atmega328P y una computadora, a partir del lenguaje de programación Assembly. A modo de ejemplo, se ha desarrollado un simple proyecto para permitir

controlar LEDs desde la computadora. Queda así en evidencia la potente herramienta que es el puerto serie para lograr intercomunicar a diferentes microcontroladores en una red compleja, y la ventaja que suponen todos los diferentes modos de operación con los cuales se lo puede configurar en la programación.

9. Código Fuente

9.1. Código sin interrupciones

```
.include "m328Pdef.inc"
.dseg


.def temp = r21
.def option = r22
.def portc_output = r23

.equ    F_CPU = 8000000
.equ    baud = 9600                     ; baudrate
.equ    bps    = (F_CPU/16/baud) - 1    ; prescaler


.cseg
;Punto incial del código
.org 0x0000
        rjmp reset

message:
        .db "*** Hola Labo de Micro *** Escriba 1, 2, 3 o 4 para controlar los
        ↪  LEDs", 0, 0

reset:

        ;Se inicializa al Stack
        ldi temp, LOW(RAMEND)
        out SPL, temp
        ldi temp, HIGH(RAMEND)
        out SPH, temp

        ;Se configuran a C0, C1, C2 y C3 como pines de salida

        ldi temp, 0b00011110
        out DDRC, temp
        ldi temp, 0
        out PORTC, temp

        ;Se incializa el puerto serie
        ldi r16, LOW(bps); prescaler de puerto serie
    ldi r17, HIGH(bps)
```

```
41          rcall uartInit
42
43          rcall delay_s
44          ldi zl, LOW(message<<1); carga el puntero al mensaje constante en zh:zl
45      ldi zh, HIGH(message<<1)
46      rcall show_string; Transmitir mensaje contenido en flash
47          rjmp main
48
49
50  main:
51          rcall getc
52          rcall process_option;En ASCII, el 1 es 0b00110001, por lo que se hace
            ↪  el análsisis pertinente
53
54          in temp, PINC
55          eor portc_output, temp
56
57          out PORTC, portc_output
58      rjmp main; Volver a leer continuamente
59
60  process_option:
61
62          cpi option, 0x31
63          breq led1
64          cpi option, 0x32
65          breq led2
66          cpi option, 0x33
67          breq led3
68          cpi option, 0x34
69          breq led4
70          clr portc_output
71          ret
72
73  led1:
74          ldi portc_output, 0b00000010
75          ret
76
77  led2:
78          ldi portc_output, 0b00000100
79          ret
80
81  led3:
82          ldi portc_output, 0b00001000
83          ret
84
85  led4:
86          ldi portc_output, 0b00010000
87          ret
88
89
90
91  getc:
92          lds         temp, UCSR0A; cargar UCSR0A en temp
93          sbrs temp, RXC0; Esperar a que se vacie el buffer de recepción
```

```
94          rjmp getc
95          lds         option, UDR0; recibir carácter deseado
96          ret
97
98  show_string:
99
100         lpm         r16, Z+
101         cpi         r16, 0x00
102         breq end_show_string
103
104 wait:
105         lds                 r17, UCSR0A
106         sbrs         r17, UDRE0
107         rjmp         wait
108
109         sts                 UDR0, r16
110         rjmp         show_string
111
112 end_show_string:
113         ret
114
115 ;************************************************************
116 ;* uartInit
117 ;*
118 ;* entradas: r17:r16 - prescaler de baud rate
119 ;*
120 ;* Habilita la transmission de UART para 8N1
121 ;*
122 ;* Registros modificados: r16
123 ;************************************************************
124 uartInit:
125     sts         UBRR0L, r16; prescaler
126     sts         UBRR0H, r17
127     ; 8N1
128     ldi                 r16, (0<<UMSEL00) | (0<<UPM00) | (0<<USBS0) |
        ↪  (3<<UCSZ00)
129     sts         UCSR0C, r16
130
131         ldi                 r16, (1<<RXEN0) | (1<<TXEN0) | (0<<UCSZ02)        ;
            ↪  enable transmitter and receiver, 8bits
132         sts                 UCSR0B, r16
133
134     ret
135
136 ; Assembly code auto-generated
137 ; by utility from Bret Mulvey
138 ; Delay 8 000 000 cycles
139 ; 1s at 8.0 MHz
140
141 delay_s:
142
143     ldi  r18, 41
144     ldi  r19, 150
145     ldi  r20, 128
```

```
L1: dec  r20
    brne L1
    dec  r19
    brne L1
    dec  r18
    brne L1
ret
```

9.2. Código con interrupciones

```
.include "m328Pdef.inc"
.dseg

.def temp = r21
.def option = r22
.def portc_output = r23

.equ    F_CPU = 8000000
.equ    baud = 9600; baudrate
.equ    bps    = (F_CPU/16/baud) - 1; prescaler

.cseg
;Punto incial del código
.org 0x0000
        rjmp reset

.org URXCaddr         ;RX
        rjmp handle_instruction

message:
        .db "*** Hola Labo de Micro *** Escriba 1, 2, 3 o 4 para controlar los
        ↪  LEDs", 0, 0

reset:

        ;Se inicializa al Stack
        ldi temp, LOW(RAMEND)
        out SPL, temp
        ldi temp, HIGH(RAMEND)
        out SPH, temp

        ;Se configuran a C0, C1, C2 y C3 como pines de salida

        ldi temp, 0b00011110
        out DDRC, temp
        ldi temp, 0
        out PORTC, temp

        ; Se incializa el puerto serie
        ldi r16, LOW(bps); prescaler de puerto serie
    ldi r17, HIGH(bps)
```

```
41          rcall uartInit
42
43          rcall delay_s
44          ldi zl, LOW(message<<1); carga el puntero al mensaje constante en zh:zl
45      ldi zh, HIGH(message<<1)
46      rcall show_string; Transmitir mensaje contenido en flash
47          sei
48          rjmp main
49
50
51  main:
52      rjmp main; Volver a leer continuamente
53
54
55  show_string:
56
57          lpm         r16, Z+
58          cpi         r16, 0x00
59          breq end_show_string
60
61  wait:
62          lds                 r17, UCSR0A
63          sbrs        r17, UDRE0
64          rjmp        wait
65
66          sts                 UDR0, r16
67          rjmp        show_string
68
69  end_show_string:
70          ret
71
72  uartUDREIEnable:
73      lds         temp, UCSR0B
74      sbr         temp, (1<<UDRIE0)
75      sts         UCSR0B, temp
76      ret
77  ;************************************************************
78  ;* uartInit
79  ;*
80  ;* entradas: r17:r16 - prescaler de baud rate
81  ;*
82  ;* Habilita la transmission de UART para 8N1
83  ;*
84  ;* Registros modificados: r16
85  ;************************************************************
86  uartInit:
87      sts         UBRR0L, r16; prescaler
88      sts         UBRR0H, r17
89      ; 8N1
90      ldi                 r16, (0<<UMSEL00) | (0<<UPM00) | (0<<USBS0) |
        ↪  (3<<UCSZ00)
91      sts         UCSR0C, r16
92
```

```
93          ldi                 r16, (1<<RXEN0) | (1<<TXEN0) | (0<<UCSZ02) |
            ↪  (1<<TXCIE0) | (1<<RXCIE0); enable transmitter and receiver, 8bits
94          sts                 UCSR0B, r16
95
96      ret
97
98  ; Assembly code auto-generated
99  ; by utility from Bret Mulvey
100 ; Delay 8 000 000 cycles
101 ; 1s at 8.0 MHz
102 delay_s:
103     ldi  r18, 41
104     ldi  r19, 150
105     ldi  r20, 128
106 L1: dec  r20
107     brne L1
108     dec  r19
109     brne L1
110     dec  r18
111     brne L1
112 ret
113
114 handle_instruction:
115         in r25, SREG
116         push r25
117
118         getc:
119         lds         temp, UCSR0A; cargar UCSR0A en temp
120         sbrs temp, RXC0; Esperar a que se vacie el buffer de recepción
121         rjmp getc
122         lds         option, UDR0; recibir caracter deseado
123
124         cpi option, 0x31
125         breq led1
126
127         cpi option, 0x32
128         breq led2
129
130         cpi option, 0x33
131         breq led3
132
133         cpi option, 0x34
134         breq led4
135
136         reti
137
138         led1:
139                 ldi portc_output, 0b00000010
140                 rjmp handle_instruction_end
141
142         led2:
143                 ldi portc_output, 0b00000100
144                 rjmp handle_instruction_end
145
```

```
146         led3:
147                 ldi portc_output, 0b00001000
148                 rjmp handle_instruction_end
149
150         led4:
151                 ldi portc_output, 0b00010000
152                 rjmp handle_instruction_end
153
154         handle_instruction_end:
155
156                 in temp, PINC
157                 eor portc_output, temp
158                 out PORTC, portc_output
159
160                 pop         r25
161                 out         SREG, r25
162                 reti
```