Lab 13 - Controlling I/O Pins of Microcontroller

13.1 Digital Write

13.1.1 Digital Write Method 1:

The following code will set an I/O pin of microcontroller to high or low:

```
.include "m328pdef.inc"
.cseg
.org 0x0000
      LDI r16, (1<<5)           ; r16 = 00100000
      OUT DDRB, r16             ; PB5 set as an OUTPUT pin

      LDI r17, (1<<5)           ; r17 = 00100000
      OUT PORTB, r17            ; PB5 pin --> HIGH (5v)
      ;delay()
      LDI r17, (0<<5)           ; r17 = 00000000
      OUT PORTB, r17            ; PB5 pin --> LOW  (0V)

loop:
rjmp loop                       ; stay in infinite loop
```

This will turn on the LED attached to D13 pin of Arduino Uno (built-in LED on Arduino Uno development board). Actually, it is *PB5* pin of ATmega328p.

13.1.2 Digital Write Method 2:

In this method we can use predefined names of the I/O pins defined in the "m328pdef.inc" file.

```
.include "m328pdef.inc"
.cseg
.org 0x0000
      LDI r16, (1<<PB5)         ; r16 = 00100000
      OUT DDRB, r16             ; PB5 set as an OUTPUT pin

      LDI r17, (1<<PB5)         ; r17 = 00100000
      OUT PORTB, r17            ; PB5 pin --> HIGH (5v)
      ;delay()
      LDI r17, (0<<PB5)         ; r17 = 00000000
      OUT PORTB, r17            ; PB5 pin --> LOW  (0V)

loop:
rjmp loop                       ; stay in infinite loop
```

13.1.3 Digital Write Method 3:

Without using left-shift operators we can also write the bits explicitly:

```
LDI r16, 0b00100000       ; r16 = 00100000
OUT DDRB, r16             ; PB5 set as an OUTPUT pin

LDI r17, 0b00100000       ; r17 = 00100000
OUT PORTB, r17            ; PB5 pin --> HIGH (5v)
```

So, all the above three methods are exactly same. But the issue with these methods is that whenever you set a pin all other pins of the microcontroller are also affected.

The below method 4 and 5 shows the ways to change the required pin without disturbing other I/O pins of the MCU.

13.1.4 Digital Write Method 4:

This method uses “Bit Masking” method to only change the required bit in the I/O registers.

```
.include "m328pdef.inc"
.cseg
.org 0x0000
; setting PB5 pin as OUTPUT
      IN  r16,DDRB
      ORI r16, (1<<PB5)              ; r16 = ??1?????
      OUT DDRB, r16                  ; PB5 set as an OUTPUT pin

; setting PB5 pin to HIGH
      IN  r16,PORTB
      ORI r16, (1<<PB5)              ; r16 = ??1?????
      OUT PORTB, r16                 ; PB5 pin --> HIGH (5v)

      ;delay()

; setting PB5 pin to LOW
      IN  r16,PORTB
      ANDI r16, ~(1<<PB5)            ; r16 = ??0?????
      OUT PORTB, r16                 ; PB5 pin --> LOW (0v)

loop:
rjmp loop                ; stay in infinite loop
```

13.1.5 Digital Write Method 5:

This is the easiest method that requires only two commands; set bit (SBI) and clear bit (CBI):

```
.include "m328pdef.inc"
.cseg
.org 0x0000
      SBI DDRB,5         ; DDRB  = ??1????? (PB5 set as an OUTPUT pin)
      SBI PORTB,5        ; PORTB = ??1????? (PB5 pin --> HIGH (5v))
      ;delay()
      CBI PORTB,5        ; PORTB = ??0????? (PB5 pin --> LOW  (0V))
loop:
rjmp loop                ; stay in infinite loop
```

SBI – Set Bit in I/O Register and **CBI** – Clear Bit in I/O Register instructions set and clear the specific bit in the I/O register without disturbing other bits of that register.

Other set bit instructions are as follows:

- **SBR** – Set Bit in General-Purpose Register
- **CBR** – Clear Bit in General-Purpose Register
- **SER** – Set whole register (11111111)
- **CLR** – Clear whole register (00000000)

13.1.6 Blink the LED using Procedure

To blink an LED, we have to implement a delay function to stop the execution of program for specific time interval then resume the execution. For that purpose, we have two options:

- **Blocking Delay:** Halt the entire program and wait for the delay (wasting clock cycles to achieve delay).
- **Non-blocking Delay:** CPU performs other activities and when the delay time is passed then perform that task.

In the following example, we implemented blocking delay for LED blinking:

```
.include "m328pdef.inc"
.cseg
.org 0x0000
      SBI DDRB, PB5             ; PB5 set as an OUTPUT pin
loop:
      SBI PORTB, PB5            ; PB5 pin --> HIGH (5v)
      CALL delay                ; Delay of 0.5 second
      CBI PORTB, PB5            ; PB5 pin --> LOW (0v)
      CALL delay                ; Delay of 0.5 second
rjmp loop                       ; stay in infinite loop

;-------------------------------------------------------
; Delay Procedure (0.5 seconds)
;-------------------------------------------------------
delay:
      LDI   r18,71                    ; initialize outer loop count
      L1:
            LDI r24,LOW(28168)        ; intialize inner loop count in inner
            LDI r25,HIGH(28168)       ; loop high and low registers
      L2:
            sbiw r24,1                ; decrement r25:r24
            brne L2                   ; branch to L2 if r25:r24 != 0
            dec r18                   ; decrement r18
            brne L1                   ; branch to L1 if r18 != 0
ret
```

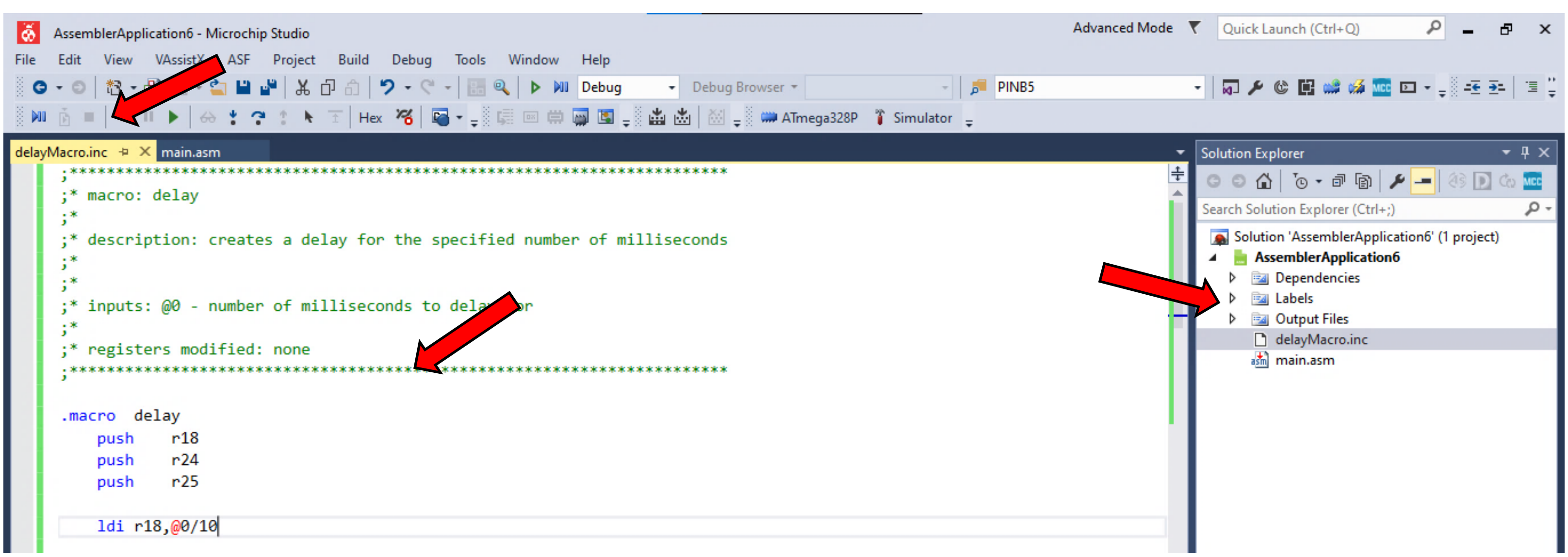
Adding the clock cycles used by these instructions in loops gives a cycle count of 8000000. At 16 MHz, this is exactly 0.5 seconds. So, to change the desired delay we have to change the loop counts accordingly. In the next example we performed these calculations in the code to get variable delay according to the user's need.

13.1.7 Blink the LED using Macro

Unlike subroutines/procedures where the microcontroller will jump to a new section of code, the assembler will simply replace macros with the code it represents during assembly. We can also pass parameters to macros.

Add the following macro in a new include file named "*delay_Macro.inc*":

```
;***********************************************************************
;* macro: delay
;*
;* description: creates a delay for the specified number of milliseconds
;*
;* inputs: @0 - number of milliseconds to delay for
;*
;* registers modified: none
;***********************************************************************

.macro delay
      push   r18
      push   r24
      push   r25

      ldi    r18,@0/10

L1:
      ldi    r24,LOW(39998)      ; intialize inner loop count in inner
      ldi    r25,HIGH(39998)     ; loop high and low registers
L2:
      sbiw r24,1                 ; decrement inner loop registers
      brne L2                    ; branch to L2 if iLoop registers != 0
      dec r18                    ; decrement outer loop register
      brne L1                    ; branch to L1 if outer loop register != 0
      nop                        ; no operation

      pop    r25
      pop    r24
      pop    r18
.endmacro
```



Then include this macro file in our project:

```
.include "delay_Macro.inc"
```

Hence the *main.asm* file becomes:

```
.include "m328pdef.inc"
.include "delay_Macro.inc"

.cseg
.org 0x0000
      SBI DDRB, PINB5           ; PB5 set as an OUTPUT pin
loop:
      SBI PORTB, PINB5          ; PB5 pin --> HIGH (5v)
      delay 1000                ; Delay of 1 second
      CBI PORTB, PINB5          ; PB5 pin --> LOW (0v)
      delay 1000                ; Delay of 1 second
rjmp loop                       ; stay in infinite loop
```

Important: From now, we will use this delay macro our examples throughout the Lab. The maximum delay supported by this macro is 2500 milliseconds (2.5 seconds). So, to get delay larger than 2.5 seconds just reuse the delay macro again and again.

```
delay 4000                ; error

delay 2000
delay 2000                ; success
```

13.1.8 LED Toggling

A shorter method to ON/OFF the LED is to use XOR operator to invert the previous state of the PORTB register. So, we don't have to explicitly set or clear the bits of PORTB register, XOR automatically do this.

```
.include "m328pdef.inc"
.include "delay_Macro.inc"

.cseg
.org 0x0000
   SBI DDRB, PINB5             ; PB5 set as an OUTPUT pin
   CLR r16                     ; r16 = 00000000

loop:
   LDI r17, (1<<PINB5)         ; r17 = 00100000
   EOR r16, r17
   OUT PORTB, r16              ; PB5 pin toggle at every loop count
   delay 1000                  ; Delay of 1 second
rjmp loop
```

Another easiest way to toggle a pin is just set bit in ***PINx*** register again and again while in a loop:

```
loop:
      SBI PINB, PB5             ; Toggle the state of the PB5 pin
      delay 1000
rjmp loop
```

13.2 Digital Read

To read the state of an I/O pin of the microcontroller we have two steps:

- Configure the pin as INPUT using ***DDRx*** register by setting the corresponding bit to 0.
- Read the state of the pin using ***PINx*** register by reading the corresponding bit.
- (optional) Set the pull-up using ***PORTx*** register by setting the corresponding bit to 1.

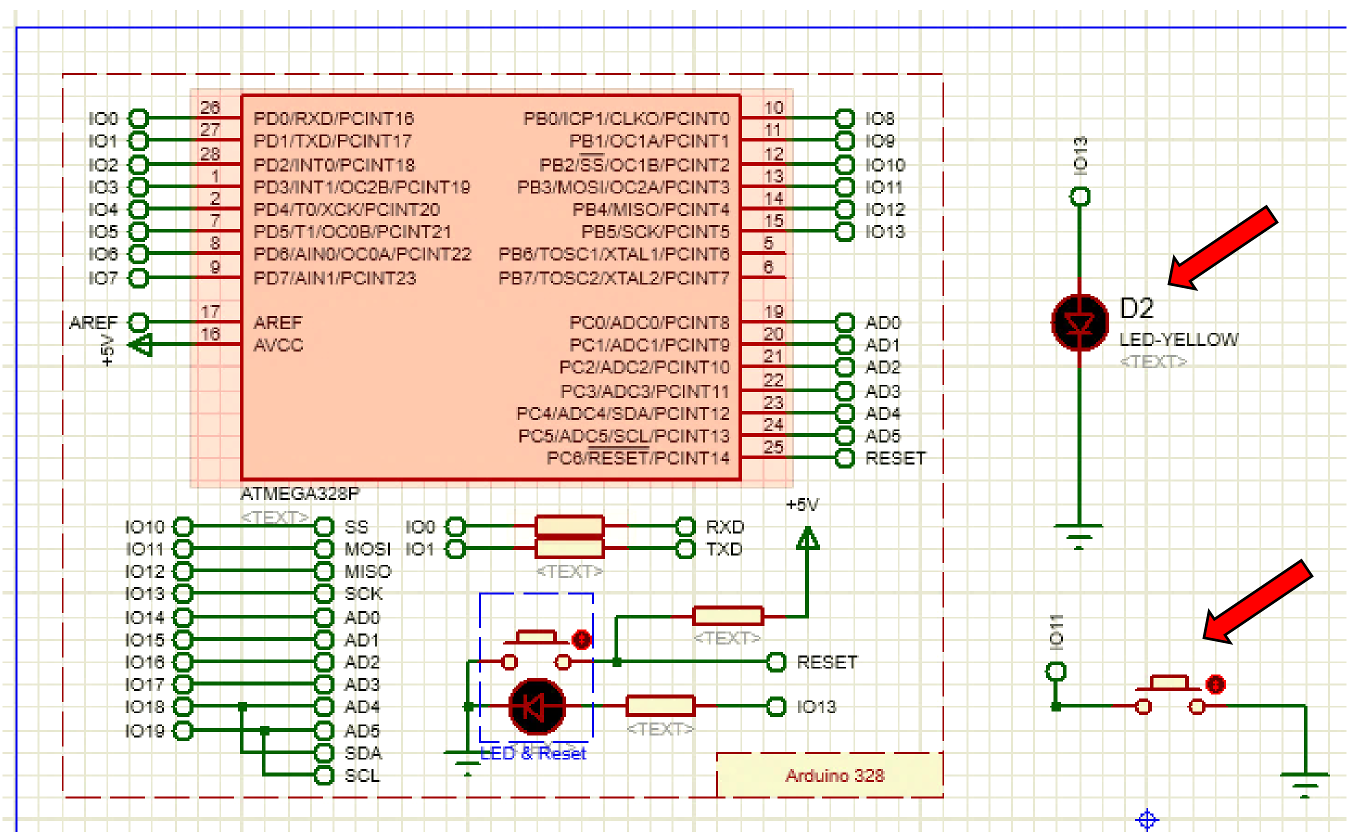
For example, to read the state (HIGH or LOW) of PB5 pin we will read bit number 5 of the PINB register. If the bit number 5 is 1 it means, PB5 pin is high, if the bit number 5 of PINB register is zero then it means PB5 pin in low.

When we declare a pin as input then there are 3 conditions of that pin:

- **High:** means there is high pulse (5v) on that pin from external sensor.
- **Low:** means there is low pulse (0v) or Ground on that pin.
- **High-Impedance (Floating):** means the pin is not connected to either 5v or 0v but it is floating (not connected to anything). This state is not recommended so whenever we configure a pin as an input pin, we should set it default either high or low. This is called pull-up or pull-down respectively. ATmega328p only supports internal pull-up.

13.2.1 Turn ON the LED using Push Button

Let's attach a push button to PB3 pin and then blink LED if the button is pressed. This is the circuit diagram.



This is the assembly code to read the status of a push button attached to the PB3 pin and then turn ON the LED on PB5 pin if the push button is pressed. When the button is released then turn OFF the LED again:

```
.include "m328pdef.inc"
.include "delay_Macro.inc"

.cseg
.org 0x0000
      SBI DDRB, PB5             ; PB5 set as OUTPUT Pin
      CBI PORTB, PB5            ; LED OFF
      CBI DDRB, PB3             ; PB3 set as INPUT pin
      SBI PORTB, PB3            ; Enable internal pull-up resistor
loop:
      ; check if push buttong is pressed
      SBIS PINB,PB3      ; if not pressed, skip next line if the PINB reg. bit# 3 is 1
      rjmp L1
      CBI PORTB, PB5     ; LED OFF
rjmp loop

L1:
      SBI PORTB, PB5     ; LED ON
rjmp loop
```

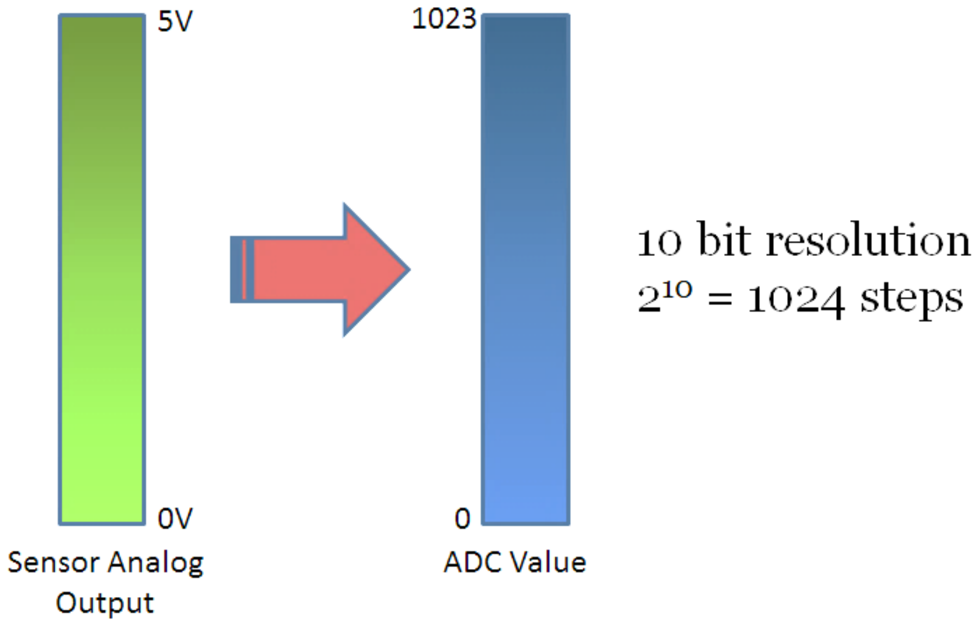
The **SBIS** (**S**kip if **B**it in **I**/O Register is **S**et) instruction checks the bit on I/O register and if that bit is 1 then skip the next coming instruction. (e.g., in our case CALL L1 instruction).

These are some "Skip" Branching instructions in AVR assembly:

Mnemonic	Description
SBRC	skip if bit in register cleared
SBRS	skip if bit in register set
SBIC	skip if bit in i/o register cleared
SBIS	skip if bit in i/o register set

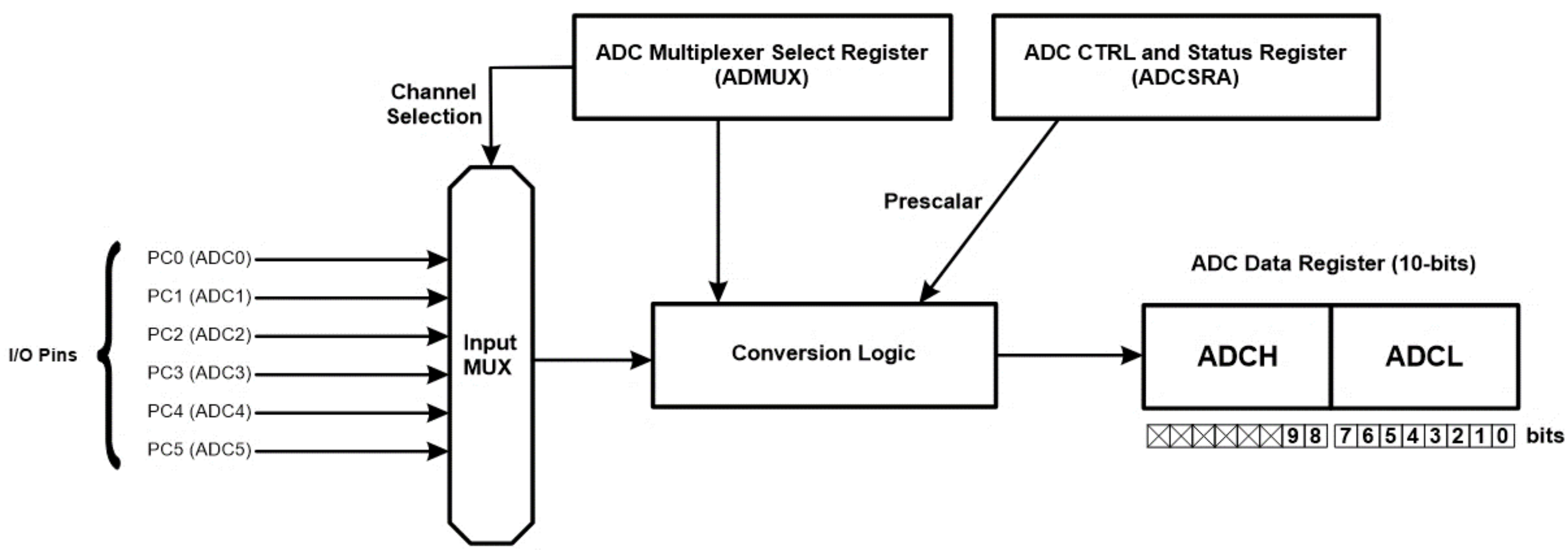
13.3 Analog Read

An Analog to Digital Converter (ADC) is used to turn an analog signal into a digital one. It does this by measuring voltage on its input pin.



ATmega328p has 10-bit resolution ADC which provide output result in the range of 0-1023. It means at a given input voltage to analog pin, the ADC converts that voltage level and map it to a value between 0 and 1023. ATmega328p has 6 analog read pins (PC0 to PC5). Note that analog write is not supported on these pins.

Below is a block diagram of the ADC in the ATmega328p. A *multiplexer* (MUX) selects one of six inputs to be fed into the ADC. A *prescaler* determines the speed of the conversion. The output is 10-bits wide, so the lowest eight bits go into the *ADCL* register; and the *ADCH* register holds the remaining high bits in its low end.



13.3.1 Reading Light Intensity using LDR Sensor

The following example reads the value on ADC0 pin (PC0) of ATmega328p and then turn ON and OFF an LED attached to the PB5 pin by comparing the LDR reading with a threshold value.

```
.include "m328pdef.inc"
.include "delay_Macro.inc"

.def A  = r16
.def AH = r17

.org 0x0000
      ; I/O Pins Configuration
      SBI   DDRB,5              ; Set PB5 pin for Output to LED
      CBI   PORTB,5             ; LED OFF

      ; ADC Configuration
      LDI   A,0b11000111        ; [ADEN ADSC ADATE ADIF ADIE ADIE ADPS2 ADPS1 ADPS0]
      STS   ADCSRA,A
      LDI   A,0b01100000        ; [REFS1 REFS0 ADLAR – MUX3 MUX2 MUX1 MUX0]
      STS   ADMUX,A             ; Select ADC0 (PC0) pin
      SBI   PORTC,PC0           ; Enable Pull-up Resistor

loop:
      LDS   A,ADCSRA            ; Start Analog to Digital Conversion
      ORI   A,(1<<ADSC)
      STS   ADCSRA,A

wait:
      LDS   A,ADCSRA            ; wait for conversion to complete
      sbrc  A,ADSC
      rjmp  wait

      LDS   A,ADCL              ; Must Read ADCL before ADCH
      LDS   AH,ADCH
      delay 100                 ; delay 100ms
      cpi   AH,200              ; compare LDR reading with our desired threshold
      brsh  LED_ON              ; jump if same or higher (AH >= 200)
      CBI   PORTB,5             ; LED OFF
rjmp loop

LED_ON:
      SBI   PORTB,5             ; LED ON
rjmp loop
```

Note: To select the desired input pin of the ATmega328p for analog reading just change the value of the ADMUX register and the PORTC register bit in the above code as follow:

ADMUX	ADC Pin of ATmega328p
0b01100000	ADC0 (PC0)
0b01100001	ADC1 (PC1)
0b01100010	ADC2 (PC2)
0b01100011	ADC3 (PC3)
0b01100100	ADC4 (PC4)
0b01100101	ADC5 (PC5)

13.3.2 Conditional Branching

Use **CP** – Compare two Registers or CPI – Compare register with immediate constant instructions to compare two operands then branch to a label using these instructions:

Mnemonic	Description
brbs	branch if status flag set
brbc	branch if status flag cleared
breq	branch if equal
brne	branch if not equal
brcs	branch if carry set
brcc	branch if carry cleared
brsh	branch if same or higher
brlo	branch if lower
brmi	branch if minus
brpl	branch if plus
brge	branch if greater or equal, signed
brlt	branch if less than, signed
brhs	branch if half carry set
brhc	branch if half carry cleared
brts	branch if t flag set
brts	branch if t flag cleared
brvs	branch if overflow flag set
brvc	branch if overflow flag cleared
brie	branch if interrupt enabled
brid	branch if interrupt disabled

For example:

```
CP      r16, r17                ; compare r16 and r17
        breq L1                 ; branch if r16 == r17
L1:
```

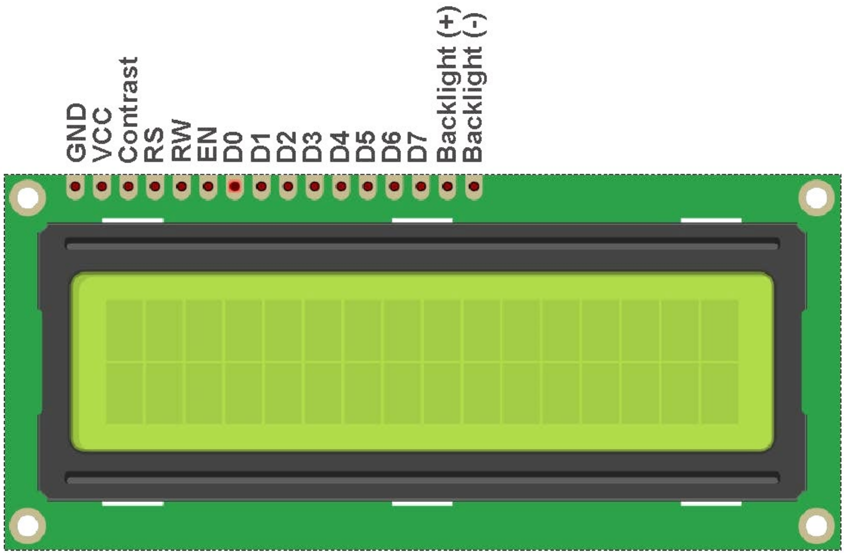
or

```
CPI     r16,50                  ; compare r16 and value 50
        breq L1                 ; branch if r16 == 50
L1:
```

13.4 16x2 LCD Display Module

16×2 LCD is character LCD means it can display ASCII characters. It has 16 Columns and 2 Rows. So, can display 32 ASCII characters in two rows of 16 characters each.

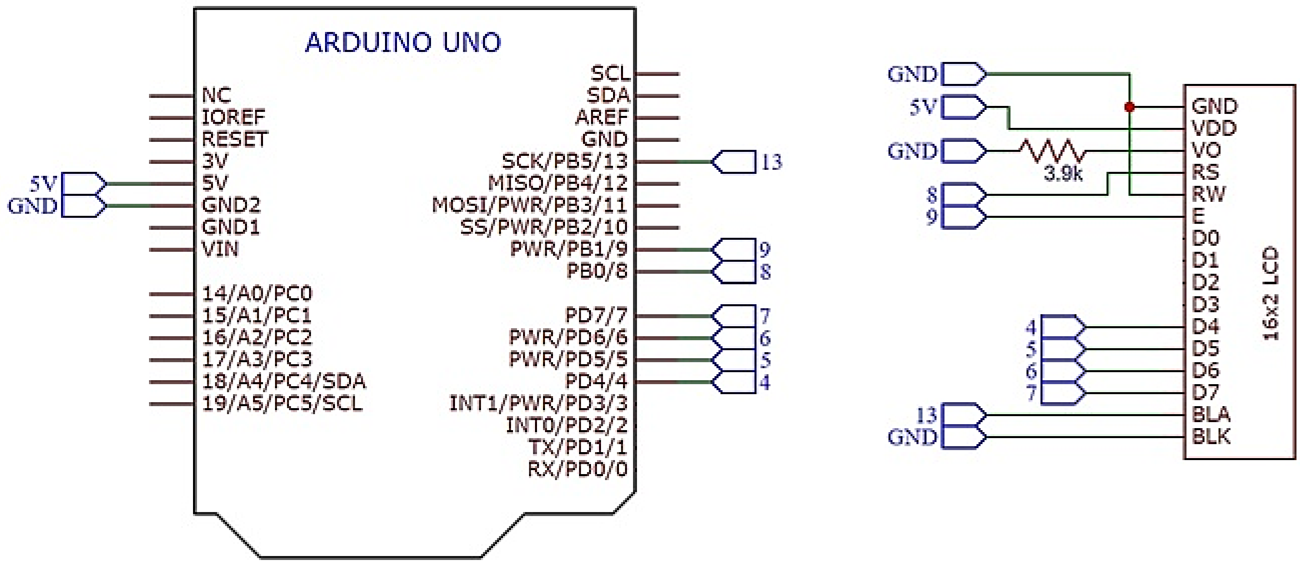
13.4.1 LCD Pinout



Pin #	Pin Name	Description
1	GND	Connected to the ground of the Arduino.
2	VDD or VCC	Power supply for the LCD which we connect to the 5V pin on the Arduino.
3	V0	Controls the contrast and brightness of the LCD. Attach a 3.9Kilo-ohm resistance from this pin to GND of Arduino.
4	RS	Set this pin to LOW when sending commands to the LCD (such as setting the cursor, clearing the display, etc.) and set it to HIGH when sending data to the LCD.
5	RW	Allows you to read data from the LCD or write data to the LCD. Since we are only using this LCD as an output device, we are going to set this pin LOW. This forces it into WRITE mode.
6	EN	Used to enable the display. When this pin is set to LOW, the LCD does do not accepts data. When this pin is set to HIGH, the LCD processes the incoming data.
7	D0	These pins carry the 8-bit data we send to the display. For example, if we want to see an uppercase 'A' character on the display, we set these pins to 0100 0001 (as per the ASCII table)
8	D1	
9	D2	
10	D3	
11	D4	
12	D5	
13	D6	
14	D7	
15	BLA	VCC pin of the backlight of the LCD. Connect with 5V or I/O pin to turn on the LCD backlight.
16	BLK	GND pin of the backlight of the LCD.

13.4.2 Example: Hello World

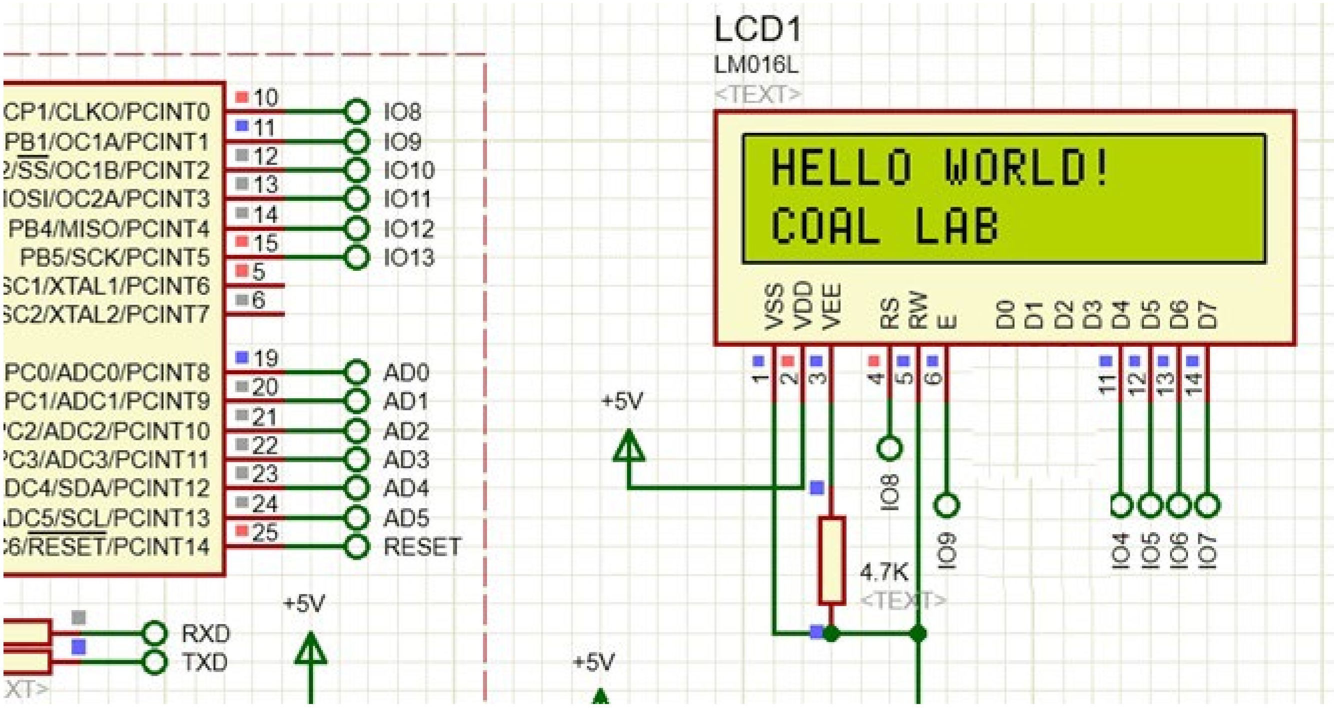
We will display "Hello World" on the LCD using Assembly program. First of all, connect the LCD according to this circuit diagram.



Then in the Microchip Studio project, write the following code in your "main.asm" file:

```
.include "m328pdef.inc"
.include "delay_Macro.inc"
.include "1602_LCD_Macros.inc"
.cseg
.org 0x0000

LCD_init      ; initilize the 16x2 LCD
LCD_backlight_OFF
delay 500
LCD_backlight_ON

loop:
       ; Display a string on LCD
       LDI ZL, LOW (2 * hello_string)
       LDI ZH, HIGH (2 * hello_string)
       LDI R20, string_len
       LCD_send_a_string
       delay 1000
       LCD_send_a_command 0x01 ; clear the LCD

       ; Display an integer on LCD
       LDI r16, 123
       LCD_send_a_register r16
       delay 1000
       LCD_send_a_command 0x01 ; clear the LCD

       ; Display character on LCD
       ; Sending Hello World to LCD character-by-character
       LCD_send_a_character 0x48 ; 'H'
       LCD_send_a_character 0x45 ; 'E'
       LCD_send_a_character 0x4C ; 'L'
       LCD_send_a_character 0x4C ; 'L'
       LCD_send_a_character 0x4F ; 'O'
       LCD_send_a_character 0x20 ; ' ' (space)
       LCD_send_a_character 0x57 ; 'W'
       LCD_send_a_character 0x4F ; 'O'
       LCD_send_a_character 0x52 ; 'R'
       LCD_send_a_character 0x4C ; 'L'
       LCD_send_a_character 0x44 ; 'D'
       LCD_send_a_character 0x21 ; '!'
       LCD_send_a_command 0xC0 ; move curser to next line
       LCD_send_a_character 0x43 ; 'C'
       LCD_send_a_character 0x4F ; 'O'
       LCD_send_a_character 0x41 ; 'A'
       LCD_send_a_character 0x4C ; 'L'
       LCD_send_a_command 0x14 ; move curser one step forward (another way to add space)
       LCD_send_a_character 0x4C ; 'L'
       LCD_send_a_character 0x41 ; 'A'
       LCD_send_a_character 0x42 ; 'B'
       delay 1000
       LCD_send_a_command 0x01 ; clear the LCD
rjmp loop

; it is recommanded to define the strings at the end of the code segment
; The length of the string must be even number of bytes
hello_string: .db    "Tehseen.",0
len: .equ     string_len    = (2 * (len - hello_string)) - 1
```

After uploading the code to Arduino or Porteus, you will see the following result:



The following table contains some important commands that we can send to LCD using the *LCD_send_a_command* macro:

Command HEX Code	Description
0x01	Clear display screen
0x02	Return home
0x04	Decrement cursor (shift cursor to left)
0x06	Increment cursor (shift cursor to right)
0x05	Shift display right (scroll right)
0x07	Shift display left (scroll left)
0x80	Force cursor to beginning of first line
0xC0	Force cursor to beginning of second line
0x38	Set LCD to 8-bit mode, 2 lines and 5×7 matrix
0x28	Set LCD to 4-bit mode, 2 lines and 5×7 matrix
0x3C	Activate second line
0x08	Display OFF, cursor OFF
0x0C	Display ON, cursor OFF
0x0F	Display ON, cursor ON

0x0E	Display ON, cursor blinking

The table of ASCII codes is shown below. Here for the LCD to show a character ‘H’ we need to send a hexadecimal code “0x48” using our custom-defined *LCD_send_a_character* macro. If we send ‘0x45’ to the LCD it will show ‘E’ symbol. In this way we can display any message and data on the LCD.

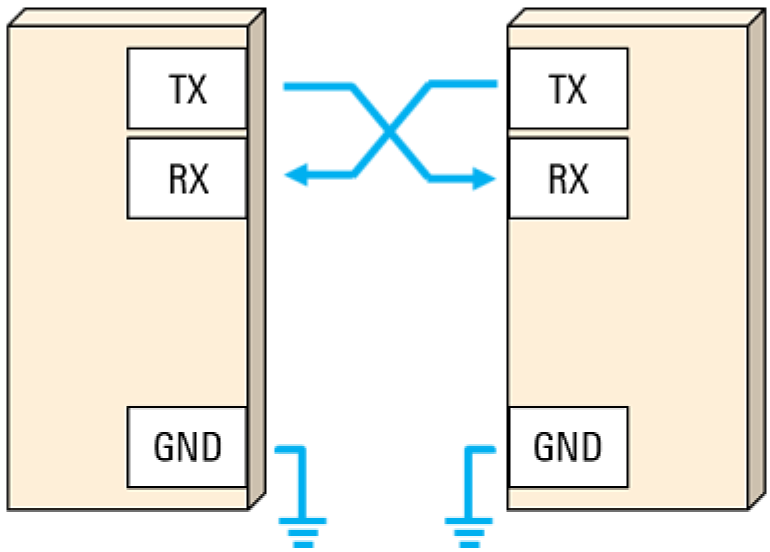
ASCII Table

Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char	Dec	Hex	Oct	Char
0	0	0		32	20	40	[space]	64	40	100	@	96	60	140	`
1	1	1		33	21	41	!	65	41	101	A	97	61	141	a
2	2	2		34	22	42	"	66	42	102	B	98	62	142	b
3	3	3		35	23	43	#	67	43	103	C	99	63	143	c
4	4	4		36	24	44	$	68	44	104	D	100	64	144	d
5	5	5		37	25	45	%	69	45	105	E	101	65	145	e
6	6	6		38	26	46	&	70	46	106	F	102	66	146	f
7	7	7		39	27	47	'	71	47	107	G	103	67	147	g
8	8	10		40	28	50	(	72	48	110	H	104	68	150	h
9	9	11		41	29	51	)	73	49	111	I	105	69	151	i
10	A	12		42	2A	52	*	74	4A	112	J	106	6A	152	j
11	B	13		43	2B	53	+	75	4B	113	K	107	6B	153	k
12	C	14		44	2C	54	,	76	4C	114	L	108	6C	154	l
13	D	15		45	2D	55	-	77	4D	115	M	109	6D	155	m
14	E	16		46	2E	56	.	78	4E	116	N	110	6E	156	n
15	F	17		47	2F	57	/	79	4F	117	O	111	6F	157	o
16	10	20		48	30	60	0	80	50	120	P	112	70	160	p
17	11	21		49	31	61	1	81	51	121	Q	113	71	161	q
18	12	22		50	32	62	2	82	52	122	R	114	72	162	r
19	13	23		51	33	63	3	83	53	123	S	115	73	163	s
20	14	24		52	34	64	4	84	54	124	T	116	74	164	t
21	15	25		53	35	65	5	85	55	125	U	117	75	165	u
22	16	26		54	36	66	6	86	56	126	V	118	76	166	v
23	17	27		55	37	67	7	87	57	127	W	119	77	167	w
24	18	30		56	38	70	8	88	58	130	X	120	78	170	x
25	19	31		57	39	71	9	89	59	131	Y	121	79	171	y
26	1A	32		58	3A	72	:	90	5A	132	Z	122	7A	172	z
27	1B	33		59	3B	73	;	91	5B	133	[	123	7B	173	{
28	1C	34		60	3C	74	<	92	5C	134	\	124	7C	174	|
29	1D	35		61	3D	75	=	93	5D	135	]	125	7D	175	}
30	1E	36		62	3E	76	>	94	5E	136	^	126	7E	176	~
31	1F	37		63	3F	77	?	95	5F	137	_	127	7F	177	

13.5 UART Serial Communication

UART (Universal Asynchronous Receiver / Transmitter) defines a protocol, or set of rules, for exchanging serial data between two devices. UART is very simple and only uses two wires between transmitter and receiver to transmit and receive in both directions.

Both ends also have a ground connection. Communication in UART can be simplex (data is sent in one direction only), half-duplex (each side speaks but only one at a time), or full-duplex (both sides can transmit simultaneously). Data in UART is transmitted in the form of frames.



13.5.1 Timing and synchronization of UART protocols

One of the big advantages of UART is that it is asynchronous – the transmitter and receiver do not share a common clock signal. Although this greatly simplifies the protocol, it does place certain requirements on the transmitter and receiver. Since they do not share a clock, both ends must transmit at the same, pre-arranged speed in order to have the same bit timing.

The most common UART baud rates in use today are 4800, 9600, 19200, 57600, and 115200. In addition to having the same baud rate, both sides of a UART connection also must use the same frame structure and parameters.

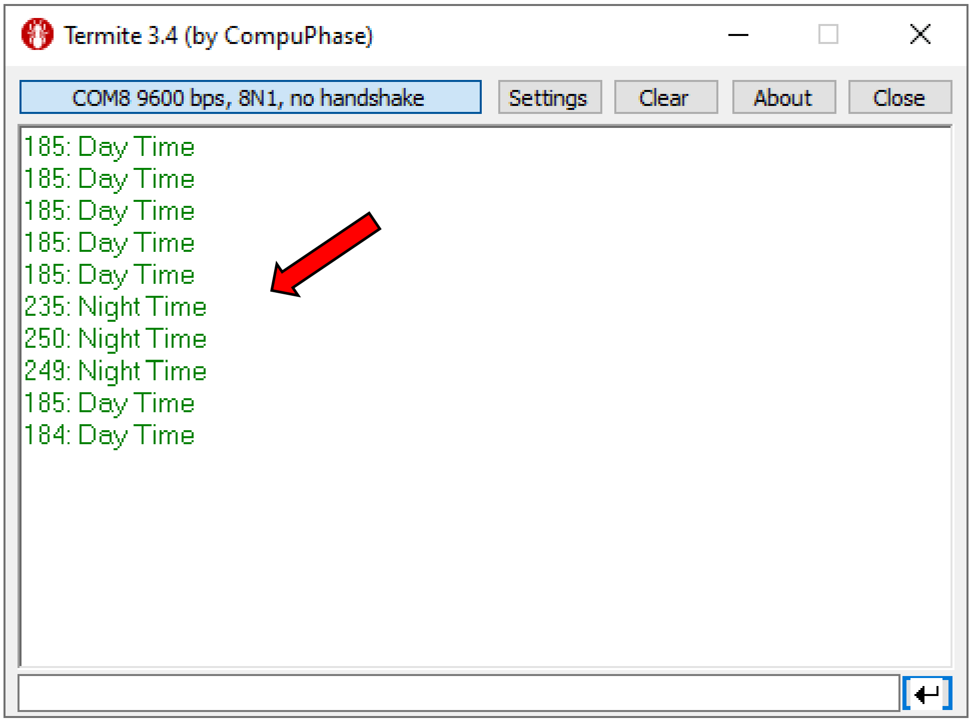
Example 1:

The following example reads the LDR value and sends that value to the UART protocol through *Tx* and *Rx* pins of the Arduino UNO. It also sends custom strings to UART when it is day or night times.

```
.include "m328pdef.inc"
.include "delay_Macro.inc"
.include "UART_Macros.inc"
.include "div_Macro.inc"

.def A = r16
.def AH = r17
.cseg
.org 0x0000
```

```
      ; ADC Configuration
      LDI A,0b11000111          ; [ADEN ADSC ADATE ADIF ADIE ADIE ADPS2 ADPS1 ADPS0]
      STS ADCSRA,A
      LDI A,0b01100000          ; [REFS1 REFS0 ADLAR – MUX3 MUX2 MUX1 MUX0]
      STS ADMUX,A               ; Select ADC0 (PC0) pin
      SBI PORTC,PC0             ; Enable Pull-up Resistor

      Serial_begin              ; initilize UART serial communication

; Reading Analog value from LDR Sensor
loop:
      LDS A,ADCSRA              ; Start Analog to Digital Conversion
      ORI A,(1<<ADSC)
      STS ADCSRA,A
wait:
      LDS A,ADCSRA              ; wait for conversion to complete
      sbrc A,ADSC
rjmp wait
      LDS A,ADCL                ; Must Read ADCL before ADCH
      LDS AH,ADCH
      delay 100                 ; delay 100ms

      Serial_writeReg_ASCII AH  ; sending the received value to UART
      Serial_writeChar ':'      ; just for formating (e.g. 180: Day Time or 220: Night
Time)
      Serial_writeChar ' '

      cpi AH,200                ; compare LDR reading with our desired threshold
      brsh LED_ON               ; jump if same or higher (AH >= 200)
      CBI PORTB,5               ; LED OFF
      ; writes the string "Day Time" to the UART
      LDI ZL, LOW (2 * day_string)
      LDI ZH, HIGH (2 * day_string)
      Serial_writeStr
      delay 500
rjmp loop

      LED_ON:
      SBI PORTB,5               ; LED ON
      ; writes the string "Night Time" to the UART
      LDI ZL, LOW (2 * night_string)
      LDI ZH, HIGH (2 * night_string)
      Serial_writeStr
      delay 500
rjmp loop


; It is recommanded to define the strings at the end of the code segment.
; Optionally you can use CRLF (carriage return/line feed) characters 0x0D and 0x0A at the
end of the string.
; The string should be terminated with 0.
; The overall length of the string (including CRLF and ending zero) must be even number of
bytes.

day_string:         .db    "Day Time ",0x0D,0x0A,0
night_string:       .db    "Night Time ",0x0D,0x0A,0
```

After successfully uploading the code to Atmega328p, open the Serial Terminal software on your computer and then set the serial port to the COM port of Arduino. And set the Baudrate to 9600 then connect the serial terminal to the COM port and you will start receiving the data from the Arduino UNO.



Example 2:

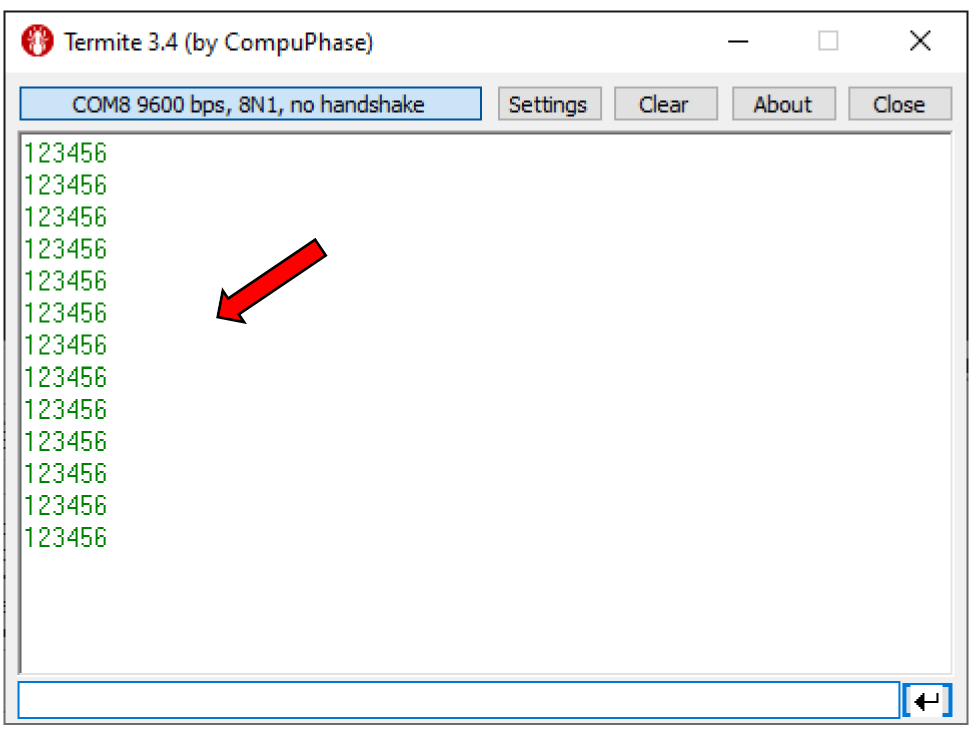
This following code sends an array to the UART:

```
.include "m328pdef.inc"
.include "delay_Macro.inc"
.include "UART_Macros.inc"
.include "div_Macro.inc"
.cseg
.org 0x0000

Serial_begin              ; initilize UART serial communication

loop:
      ; writes the integer array to the UART
      LDI ZL, LOW (2 * hello_buffer)
      LDI ZH, HIGH (2 * hello_buffer)
      LDI r20, hello_buffer_len
      Serial_writeBuffer

      delay 500
rjmp loop


; it is recommanded to define the array,strings,etc. at the end of the code segment
hello_buffer:       .db    1,2,3,4,5,6
len: .equ    hello_buffer_len    = 2 * (len - hello_buffer)
```

Example 3:

The following code sends an integer array to the UART but after converting the array to ASCII-encoded version for better display on serial terminal.

```
.include "m328pdef.inc"
.include "delay_Macro.inc"
.include "UART_Macros.inc"
.include "div_Macro.inc"
.cseg
.org 0x0000

Serial_begin              ; initilize UART serial communication

loop:
      ; writes the ASCII-encoded integer array to the UART
      LDI ZL, LOW (2 * hello_buffer)
      LDI ZH, HIGH (2 * hello_buffer)
      LDI r20, hello_buffer_len
      Serial_writeBuffer_ASCII
      Serial_writeNewLine

      delay 500
rjmp loop


; it is recommanded to define the array,strings,etc. at the end of the code segment
hello_buffer:      .db    1,2,3,4,5,6
len: .equ    hello_buffer_len    = 2 * (len - hello_buffer)
```



13.6 Macros and Example Codes Download Link

You can download all the macros and example codes used in this Lab from this link: Link