Lab 8: LCD Display and Interrupts

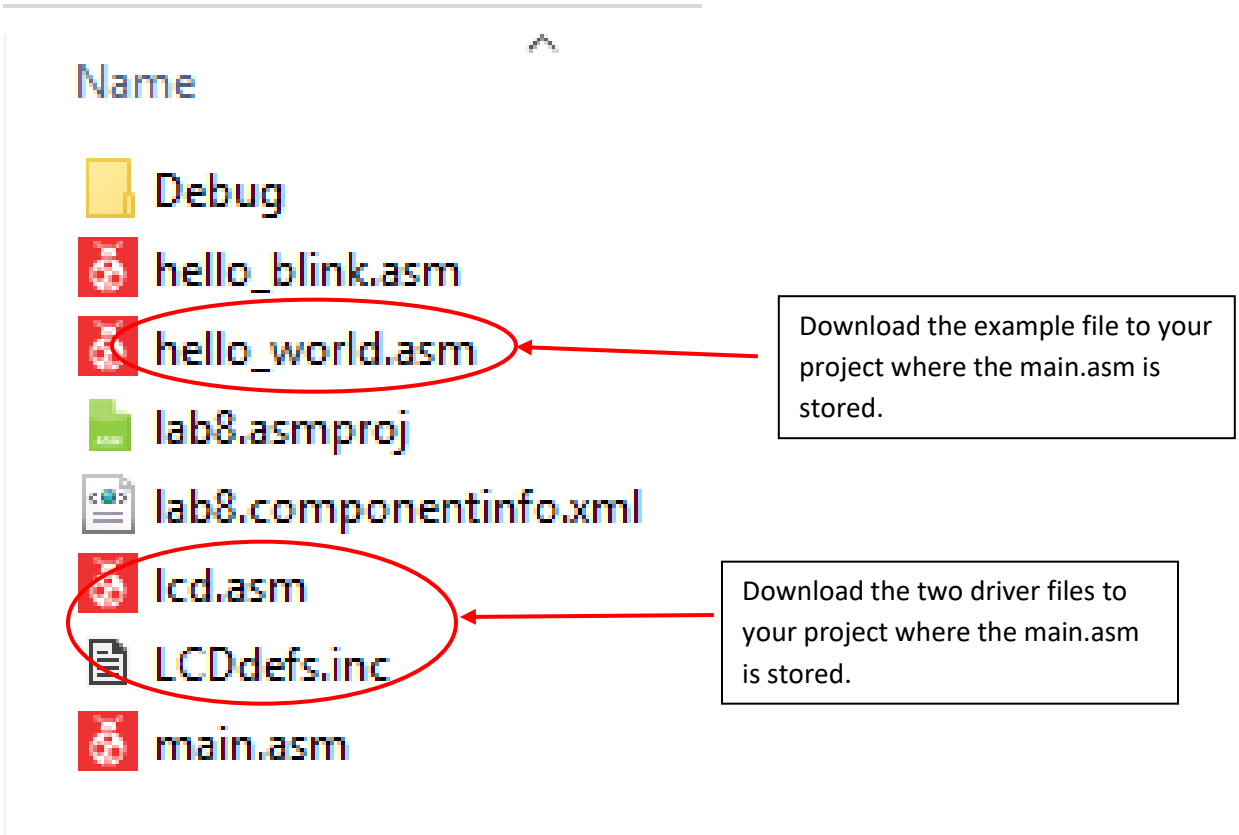
I. LCD Display

In this lab, we are going to display some characters on the LCD display. LCD stands for Liquid Crystal Display. The one that we use has built-in controller and driver chips. What we concern in this lab is the character LCD module. Character modules can display only text and perhaps some special symbols. This LCD display shows 16x2 characters (2 rows and 16 characters per row).

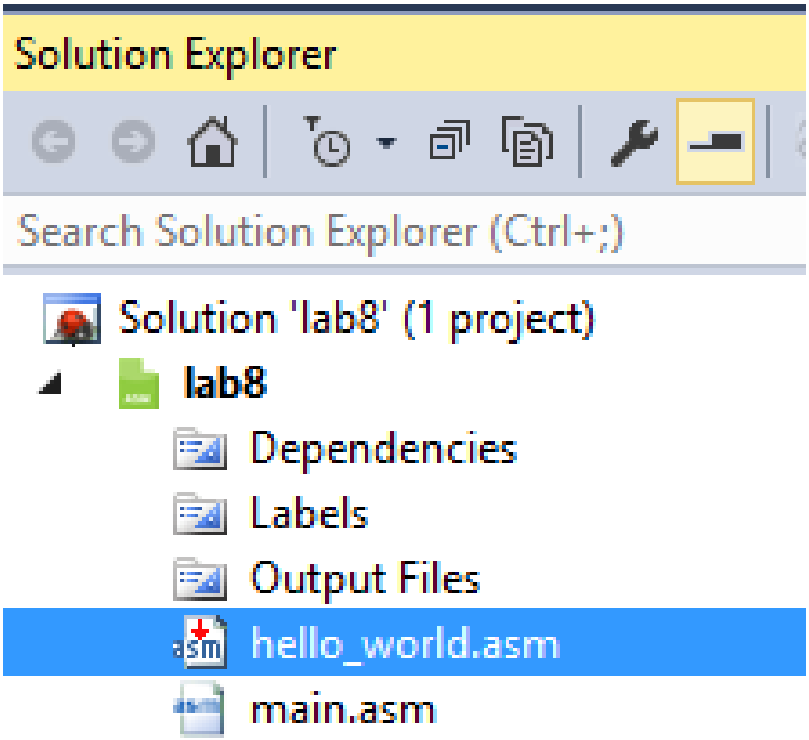
Now create a new project named lab8, then download the two driver files (HD44780 LCD Driver files for ATmega2560) to the same directory containing main.asm file:

LCDdefs.inc (LCD driver)
lcd.asm (LCD driver)

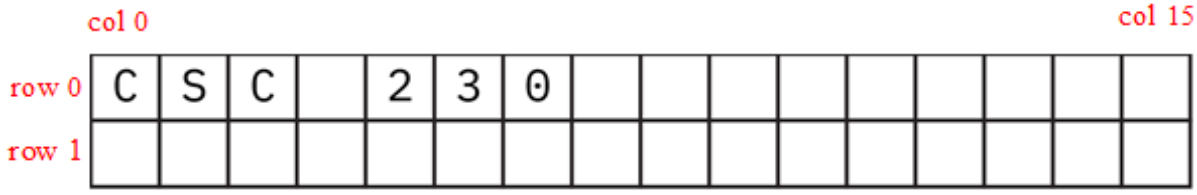
Also download "hello_world.asm" (by Dr. Mike Zastre) to the same directory mentioned above. It demonstrates how to display strings on LCD. Now in the Windows File Explorer, the directory of your project looks like this:



In Microchip Studio 7.0, the Solution Explorer looks like this:



The row and column structure of the LCD display is illustrated below:



To display a character on the LCD, do:

step1 - initialize the LCD according to the specification in the Hitachi HD44780 datasheet;

step2 - set the row, column numbers correctly, for example, in the diagram above, the first 'C' is displayed at row 0, column 0;

step3 - display the character. The following code shows how to display character 'U' at row 1, column 4:

```
.cseg
.org 0
    jmp start

.include "lcd.asm"
.cseg

start:

    rcall lcd_init

    ldi r16, 1  ;row = 1  lcd_gotoxy(row, col)
    ldi r17, 4  ;column = 4
    push r16
    push r17
    rcall lcd_gotoxy
    pop r17
    pop r16

    ldi r16, 'U'  ;lcd_putchar(char), char = 'U'
    push r16
    rcall lcd_putchar
    pop r16

Done: rjmp Done
```

More on interrupt later in this lab.

Must include the driver files. Should be in the same directory as hello_world.asm.

Initialize the LCD according to the specification in the Hitachi HD44780 datasheet.

Set the cursor at (row 1, column 4)

Display ‘U’ at (1, 4) row 1, column 4

II. Exercise1: in hello_world.asm, build the file and observe the output on LCD display, then change the code so that it displays “Hello, world!” starting at row 1, column 3.

III. Timer/Counter1 Interrupt

An interrupt is a signal that can interrupt and alter the flow of current program execution. It is a method that allows programs to respond immediately to external events. Interrupts can be triggered by external or internal signals; they can be caused by software or hardware. When a program is interrupted, the routine that is executed in response to the interrupt is called an **interrupt service routine (ISR)**, or **interrupt handler**. The process of interrupting the current program, executing the interrupt handler, and returning, is called **servicing the interrupt**. Interrupts can be dedicated or shared. If interrupts are shared among multiple devices (I/O), then the ISR further has to check which device or signal caused that interrupt. Usually the interrupts are numbered and when an interrupt occurs, the processor jumps to a pre-defined location assigned to that interrupt. Hence an interrupt vector table has to be setup to jump and branch to appropriate ISR to handle interrupts.

The ATMega 2560 processor we use in the lab has following interrupt vector table (p101 of the datasheet.[1]):

Table 13-1. Reset and Interrupt Vectors

Vector No.	Program Address(2)	Source	Interrupt Definition
1	$0000(1)	RESET	External Pin, Power-on Reset, Brown-out Reset, Watchdog Reset, and JTAG AVR Reset
2	$0002	INT0	External Interrupt Request 0
3	$0004	INT1	External Interrupt Request 1
4	$0006	INT2	External Interrupt Request 2
5	$0008	INT3	External Interrupt Request 3
6	$000A	INT4	External Interrupt Request 4
7	$000C	INT5	External Interrupt Request 5
8	$000E	INT6	External Interrupt Request 6
9	$0010	INT7	External Interrupt Request 7
10	$0012	PCINT0	Pin Change Interrupt Request 0
11	$0014	PCINT1	Pin Change Interrupt Request 1
12	$0016(3)	PCINT2	Pin Change Interrupt Request 2
13	$0018	WDT	Watchdog Time-out Interrupt
14	$001A	TIMER2 COMPA	Timer/Counter2 Compare Match A
15	$001C	TIMER2 COMPB	Timer/Counter2 Compare Match B
16	$001E	TIMER2 OVF	Timer/Counter2 Overflow
17	$0020	TIMER1 CAPT	Timer/Counter1 Capture Event
18	$0022	TIMER1 COMPA	Timer/Counter1 Compare Match A
19	$0024	TIMER1 COMPB	Timer/Counter1 Compare Match B
20	$0026	TIMER1 COMPC	Timer/Counter1 Compare Match C
21	$0028	TIMER1 OVF	Timer/Counter1 Overflow
22	$002A	TIMER0 COMPA	Timer/Counter0 Compare Match A
23	$002C	TIMER0 COMPB	Timer/Counter0 Compare match B
24	$002E	TIMER0 OVF	Timer/Counter0 Overflow
25	$0030	SPI, STC	SPI Serial Transfer Complete
26	$0032	USART0 RX	USART0 Rx Complete
27	$0034	USART0 UDRE	USART0 Data Register Empty
28	$0036	USART0 TX	USART0 Tx Complete
29	$0038	ANALOG COMP	Analog Comparator

There are 6 built-in timers (two 8-bit and four 16-bit timer/counters) in AVR processor. They can be setup in different modes. Set Timer1 in CTC (Clear Timer on Compare Match) mode. That is, the 16-bit timer is initialized to a value (say 0) and the CPU clock signal is used as source. When the timer reaches the value (called TOP) set in OCR1A (Output Compare Register 1 A), it generates an interrupt (Vector 18, see the table above) causing the processor to jump to a pre-defined location 0x0022. Let's learn the structure of interrupt in AVR assembly language:

```
.cseg
.org 0  ;reset
        jmp start
.org 0x22  ;Timer/Counter1 Compare Match A Interrupt
        jmp swap_chars_isr  ;jump to interrupt service routine


.include "lcd.asm"
.cseg

start:
    ;some command here
    ;enable timer interrupt
    ldi r16, (1 << OCIE1A)
    sts TIMSK1, r16

    ;enamble Interrupt bit in status register
    sei
    ;some command here
    loop:
        ;some command here
    rjmp loop

swap_chars_isr: ;Interrupt service routine
    push r16
    push r17
    lds r16, SREG ;preserve the content of SREG on stack
    push r16

    ;some command here

    ;timer interrupt flag is automatically
    ;cleared when this ISR is executed
    ;(p163 ATmega datasheet)
    pop r16
    sts SREG, r16
    pop r17
    pop r16
    reti
```

Interrupt Vector Table (IVT) for each interrupt

Bit 1 of **TIMSK1** is the Timer1 Output Compare A Match (T1OCAM) Interrupt Enable. Set both this bit and the I flag in SREG to 1 (sei) to enable the T1OCAM interrupt. When timer1 matches the TOP value, a T1OCAM interrupt occurs, that is, OCF1A flag (in TIFR1) is set. Then the corresponding Interrupt Vector is executed. Page 162 of datasheet[1].

The infinite loop is interrupted when timer1 matches the TOP value stored in OCR1A.

At power on (RESET), the program control jumps to location 0x0000. This is where you need to initialize the timer for the desired functionality. In the program above, the control jumps to start which initializes the timer and sets in the desired mode (CTC) and then waits in an infinite loop. After the initialization, timer/counter automatically starts counting with the specified clock rate and

causes an interrupt when the counter reaches the TOP value stored in OCR1A. This interrupt (18) takes the program control to location 0x0022 that further jumps to the ISR (**`swap_chars_isr`**).

The timer of the AVR can be specified to monitor several events. Status flags in the TIFR (Timer Interrupt Flag Register) show if an event has occurred. Some of the modes the timers can operate are: a) Timer Overflow mode; b) **Compare Match** and c) Input capture. **For the purpose of this lab, we are going to setup the timer to operate only in b) - Clear Timer on Compare Match (CTC) mode.** The OCF1A flag is set when timer1 has counted up to the TOP value (stored in OCR1A) and is reset to zero in the next timer clock cycle. Below is the summary of the registers associated with Timer/Counter 1:

- TIMSK1 is the interrupt enable register. In order to enable the interrupt by the timer, a flag has to be set in the register as well as enable the global interrupt in SREG (using SEI instruction).
- The 16-bit timer counter value is loaded at: TCNT1H:TCNT1L. The counter will start counting up from the initial value loaded here and count up to TOP in OCR1A. In the next clock pulse, the 16-bit counter will be reset to 0x0000 and causes an interrupt.
- The timer mode (CTC mode) is set by configuring control registers TCCR1A and TCCR1B.
- To count, the timer must use a clock signal (either internal or external). In this lab, the timer is setup to use an internal clock. Note that the CPU clock is operating at 16Mhz. We will use this clock but reduce the counting speed by setting up a pre-scaler of 1024. That is the clock signal is divided by 1024 to reduce the counting speed. In another words, when the CPU clock clicks 1024 times, the timer clicks once. This is done by configuring the TCCR1B control register.

IV. Timer/Counter 1 in AVR ATmega 2560

In this lab, the 16-bit Timer/Counter 1 is used. The full names of the registers and the corresponding memory addresses in the data memory are:

.equ TCCR1A = 0x80 Timer/Counter 1 Control Register A
.equ TCCR1B = 0x81 Timer/Counter 1 Control Register B
.equ TCCR1C = 0x82 Timer/Counter 1 Control Register C (no need to set it)
.equ TCNT1H = 0x85 high byte of the Timer/Counter 1 (no need to set it)
.equ TCNT1L = 0x84 low byte of the Timer/Counter 1 (no need to set it)
.equ TIFR1 = 0x36 Timer/Counter 1 Interrupt Flag Register (no need to set it)
.equ TIMSK1 = 0x6F Timer/Counter 1 Interrupt Mask Register
.equ SREG   = 0x5F Status Register

For the detailed description, refer to the lab 7 note.

Timer1 Output Compare A Match Interrupt Enable, set it to 1.

TIMSK1 – Timer/Counter 1 Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	OCIE1C	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

"**Bit 1 – OCIE1A: Timer/Counter1, Output Compare A Match Interrupt Enable**

When this bit is written to one, and the I-flag in the Status Register is set (interrupts globally enabled), the Timer/Counter Output Compare A Match interrupt is enabled. The corresponding Interrupt Vector is executed when the OCF1A Flag (in TIFR1) is set (p162 of datasheet[1]). The program control jumps to address 0x0022 and further jumps to swap_chars_isr.

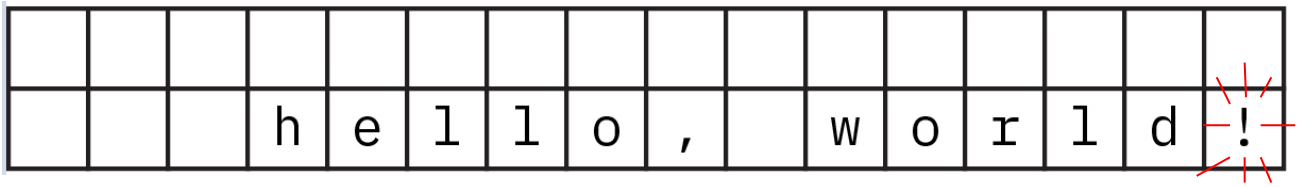
Combine the four bits in TCCR1A and TCCR1B configure the timer/counter1 mode (CTC):

TCCR1B TCCR1A

Table 17-2. Waveform Generation Mode Bit Description(1)

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM,Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

V. Exercise2: Download hello_blink.asm. Implement the interrupt service routine called swap_chars_isr. Display the following message and make '!' blink.



1. Section 14 (Interrupts, p101) and section 17(16-bit Timer/Counter, p133) of the ATmega2560-datasheet.pdf at http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-2549-8-bit-AVR-Microcontroller-ATmega640-1280-1281-2560-2561_datasheet.pdf.