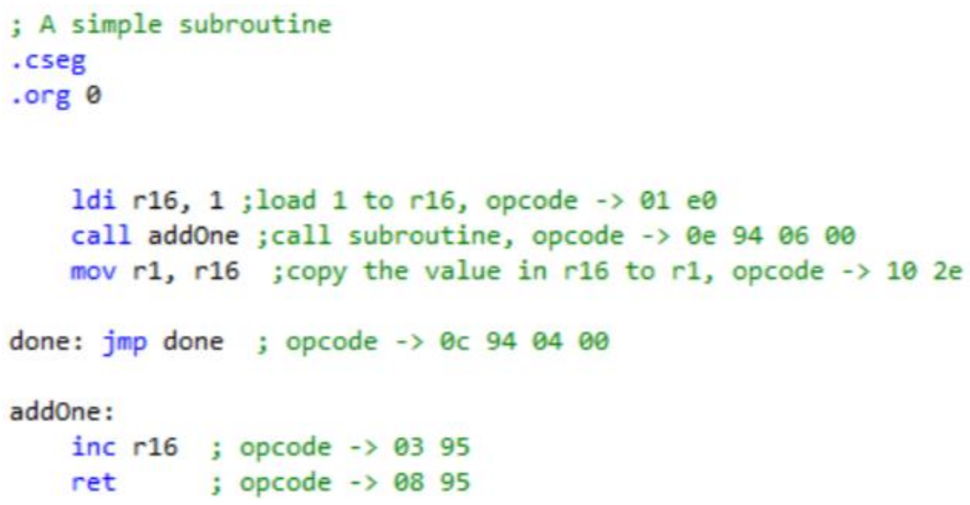
Lab 4 Simple Subroutine Call and Input Instructions

Submit button.asm from the lab4 project at the end of your lab, and not your project file.

I. Simple Function Call

In the first-year programming language courses such as CSC 110, CSC 111, we wrote many functions. In assembly language, the terms procedure, function, and subroutine are interchangeable. A function call implies a transfer of control flow (like branch and jump) to an address representing the entry point of the function. Next, the processor executes the body of the function. When it reaches the end of the function, another transfer occurs to resume execution at the instruction following the initial call. The first transfer is the function call (for invocation), the second transfer is the return (to get back to the calling program). Together, this constitutes the processor's call-return mechanism.

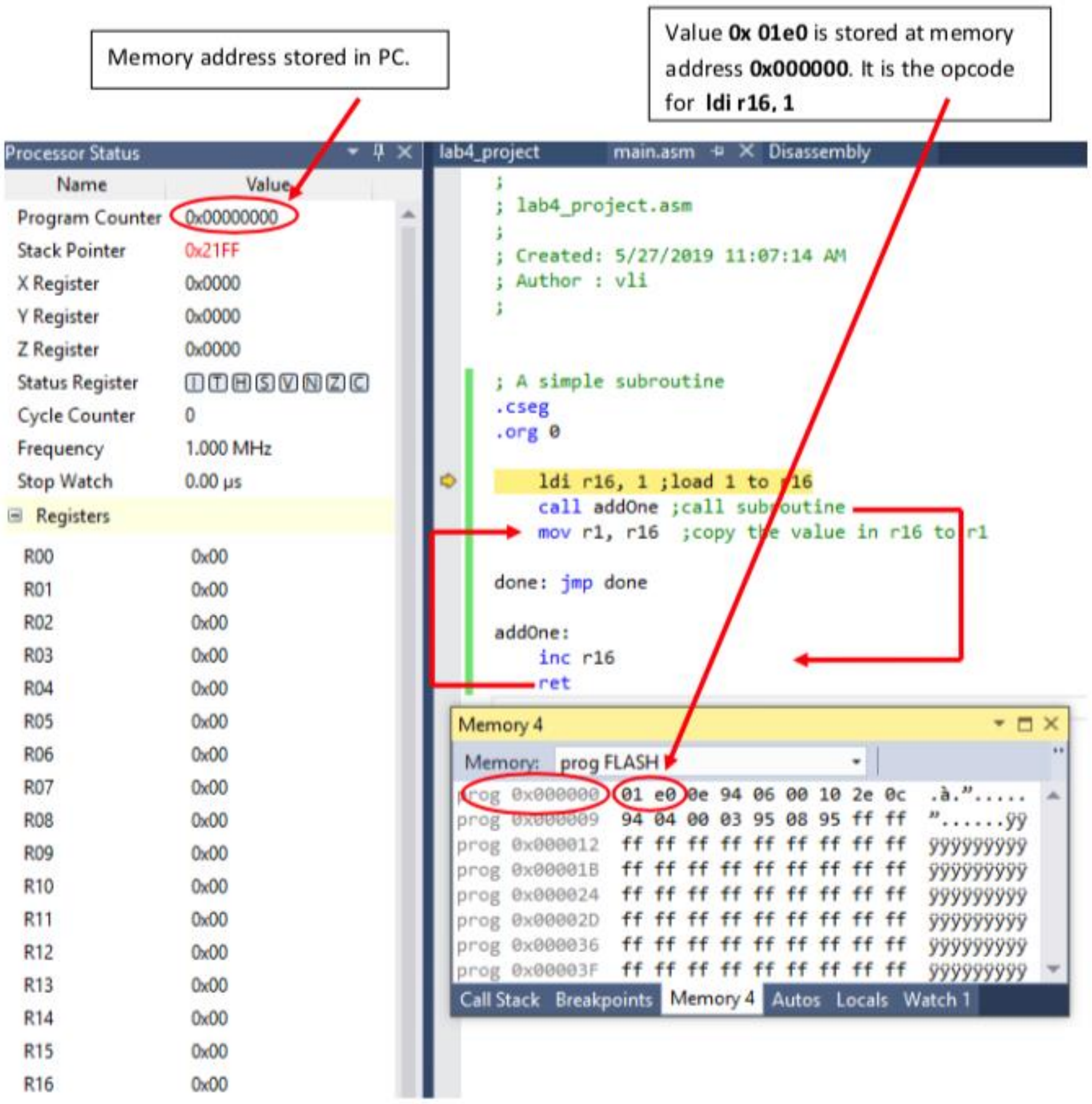
In today's lab, we are going to write a simple function – no parameter passing, no return value. Create a new project named lab4a and write the following code:



```
; A simple subroutine
.cseg
.org 0


    ldi r16, 1 ;load 1 to r16, opcode -> 01 e0
    call addOne ;call subroutine, opcode -> 0e 94 06 00
    mov r1, r16  ;copy the value in r16 to r1, opcode -> 10 2e

done: jmp done  ; opcode -> 0c 94 04 00

addOne:
    inc r16   ; opcode -> 03 95
    ret       ; opcode -> 08 95
```

Build the code and go to the menu, then click on *Debug -> Start Debugging and Break*. The Program Counter (PC) in the Processor Status window contains the memory address of the next instruction that is about to execute. Press the F11 key and observe how the value of PC changes with each keypress. Notice when the CALL instruction executes, the PC is incremented by more than the usual 1.

Pay attention to how the control flow moves to the function "addOne" and then returns to the "MOV r1, r16" instruction after the function increments the value in r16. Review the opcode for the CALL and RET instructions in program memory. What are the values of the operands?



Memory address stored in PC.

Value **0x 01e0** is stored at memory address **0x000000**. It is the opcode for **ldi r16, 1**

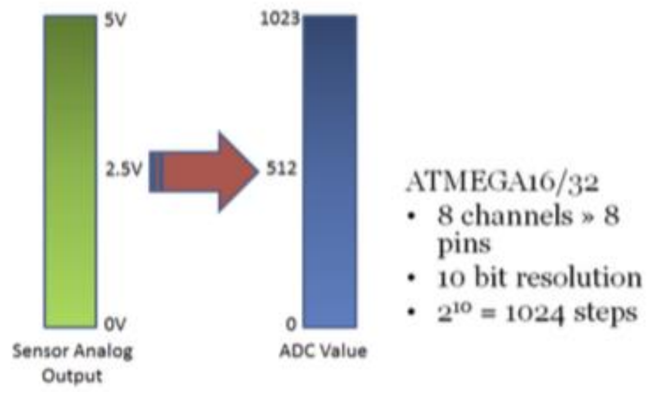
Name	Value
Program Counter	0x00000000
Stack Pointer	0x21FF
X Register	0x0000
Y Register	0x0000
Z Register	0x0000
Status Register	ITHSVNZC
Cycle Counter	0
Frequency	1.000 MHz
Stop Watch	0.00 µs
Registers	
R00	0x00
R01	0x00
R02	0x00
R03	0x00
R04	0x00
R05	0x00
R06	0x00
R07	0x00
R08	0x00
R09	0x00
R10	0x00
R11	0x00
R12	0x00
R13	0x00
R14	0x00
R15	0x00
R16	0x00

```
;
; lab4_project.asm
;
; Created: 5/27/2019 11:07:14 AM
; Author : vli
;


; A simple subroutine
.cseg
.org 0

    ldi r16, 1 ;load 1 to r16
    call addOne ;call subroutine
    mov r1, r16  ;copy the value in r16 to r1

done: jmp done

addOne:
    inc r16
    ret
```

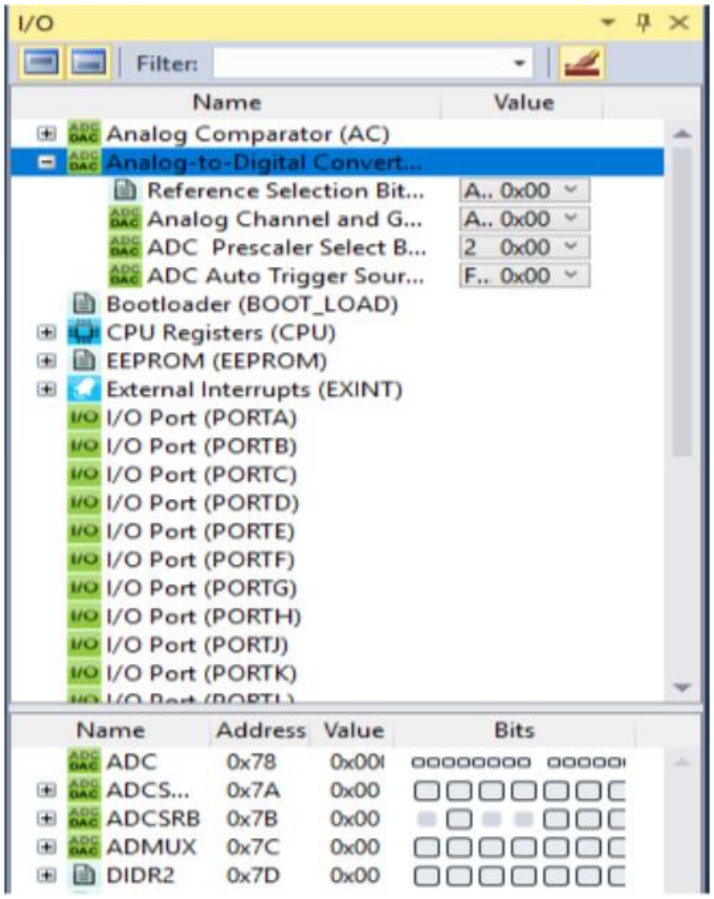
```
prog 0x000000  01 e0 0e 94 06 00 10 2e 0c  .à.”.....
prog 0x000009  94 04 00 03 95 08 95 ff ff  ”......ÿÿ
prog 0x000012  ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿ
prog 0x00001B  ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿ
prog 0x000024  ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿ
prog 0x00002D  ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿ
prog 0x000036  ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿ
prog 0x00003F  ff ff ff ff ff ff ff ff ff  ÿÿÿÿÿÿÿÿÿ
```

II. Analog Inputs from Push Buttons

The five pushbuttons (excluding the RST reset button) on the LCD shield are connected to pins of Port F and Port K on the AVR board, which are designed to read analog input. Analog is a continuously variable signal, but our system is digital (discrete). Thus, we need to use the built-in Analog to Digital Converter (ADC) to transform the continuous analog value into a binary one. ADC samples the signal and converts the analog value to a 10-bit digital value, as shown below:



Several special-purpose registers must be correctly configured for ADC to digitize an analog signal. These are the Control and Status Registers A and B (ADCSRA, ADCSRB) and the Multiplexer Selection Register (ADMUX). The multiplexer selects a channel, as ADC can handle up to 8 channels. The 10-bit binary result of the conversion is in two 8-bit Data registers ADCL and ADCH. You can find these in the I/O view of the ADC:


I/O

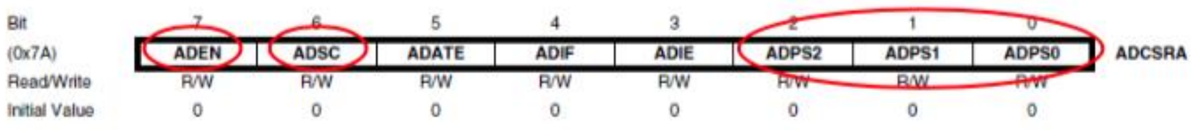
Filter:

Name	Value
⊞ Analog Comparator (AC)	
⊟ Analog-to-Digital Convert...	
Reference Selection Bit...	A.. 0x00
Analog Channel and G...	A.. 0x00
ADC Prescaler Select B...	2 0x00
ADC Auto Trigger Sour...	F.. 0x00
Bootloader (BOOT_LOAD)	
⊞ CPU Registers (CPU)	
⊞ EEPROM (EEPROM)	
⊞ External Interrupts (EXINT)	
I/O Port (PORTA)	
I/O Port (PORTB)	
I/O Port (PORTC)	
I/O Port (PORTD)	
I/O Port (PORTE)	
I/O Port (PORTF)	
I/O Port (PORTG)	
I/O Port (PORTH)	
I/O Port (PORTJ)	
I/O Port (PORTK)	

Name	Address	Value	Bits
ADC	0x78	0x00	00000000 00000
⊞ ADCS...	0x7A	0x00	☐☐☐☐☐☐☐
⊞ ADCSRB	0x7B	0x00	☐☐☐☐☐
⊞ ADMUX	0x7C	0x00	☐☐☐☐☐☐☐
⊞ DIDR2	0x7D	0x00	☐☐☐☐☐☐☐

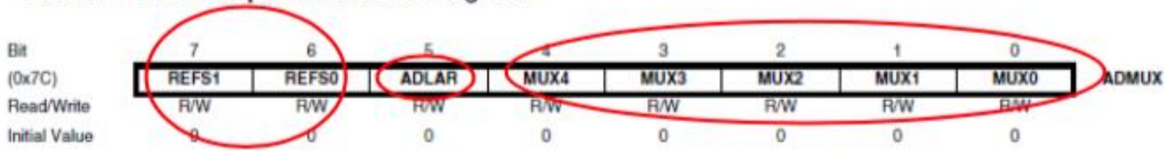
For the purposes of this lab, we will not use the ADCRB register, so only the ADCSRA and ADMUX need to be configured. Following is a brief description of these registers:

ADCSRA – ADC Control and Status Register A



Bit	7	6	5	4	3	2	1	0	
(0x7A)	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0	ADCSRA
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	

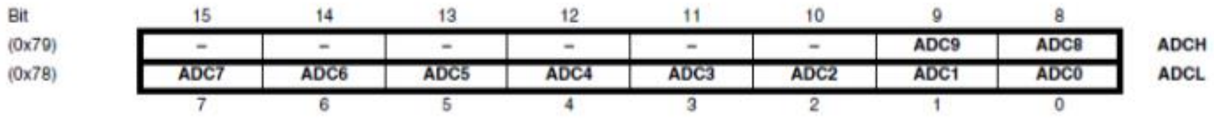
ADMUX – ADC Multiplexer Selection Register



Bit	7	6	5	4	3	2	1	0	
(0x7C)	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	ADMUX
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	

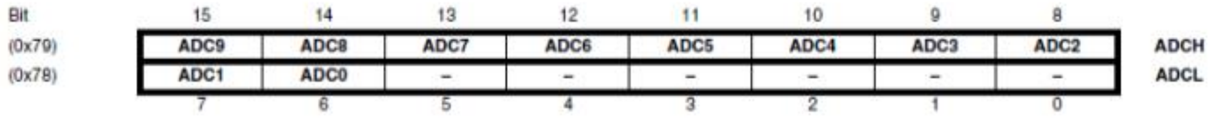
ADCL and ADCH – The ADC Data Register

ADLAR = 0



Bit	15	14	13	12	11	10	9	8	
(0x79)	–	–	–	–	–	–	ADC9	ADC8	ADCH
(0x78)	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	7	6	5	4	3	2	1	0	

ADLAR = 1



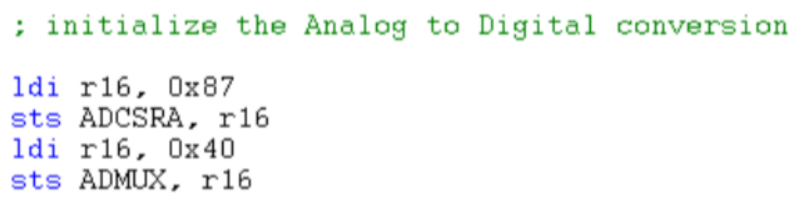
Bit	15	14	13	12	11	10	9	8	
(0x79)	ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
(0x78)	ADC1	ADC0	–	–	–	–	–	–	ADCL
	7	6	5	4	3	2	1	0	

ADCSRA:

- When set, the **ADEN** bit enables the ADC, which means it's ready to convert.
- When set, the **ADSC** bit, tells ADC to start conversion "now!".
- **ADPS0**, **ADPS1**, and **ADPS2** are pre-scalar bits. They reference the ATMEGA2560 clock to set the ADC clock.

ADMUX:

- **REFS0** and **REFS1** select the voltage reference for the ADC.
- **ADLAR** bit affects the presentation of the ADC conversion result in the ADC Data Register. When this bit is 1, the result is left adjusted. Otherwise, it is right adjusted.
- **MUX0**, **MUX1**, **MUX2**, **MUX3**, and **MUX4** select the input channel.

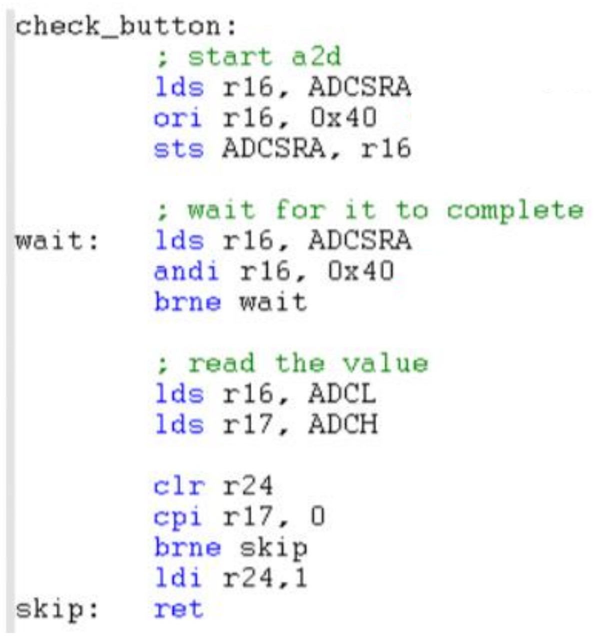


```
; initialize the Analog to Digital conversion

ldi r16, 0x87
sts ADCSRA, r16
ldi r16, 0x40
sts ADMUX, r16
```

ADCSRA: 0x87 = 0b 1000 0111 (means enable the ADC and slow down the ADC clock from 16mHz to ~125kHz, 16mHz/128).

ADMUX: 0x40 = 0b 0100 0000 (means use the AVCC with external capacitor at AREF pin and use the right adjustment since ADLAR is 0, ADC0 channel is used). The reason for the right adjustment is that the analog signal is digitized to a 10 bits value. ADCH:ADCL is 2 bytes (16bits). The digitized value is either left-adjusted or right-adjusted.

To use the ADC, first, initialize it and then enable conversion when needed. For this purpose, use the bit ADSC in ADSRA. When it is set to 1, ADC will immediately start converting the analog signal to digital format. This bit will continue to remain 1 as long as the conversion is in progress. It will be reset to 0 (by the ADC) after the conversion completes. At this point, the 10-bit digital value is available in ADCH:ADCL. That is what the following code is doing.



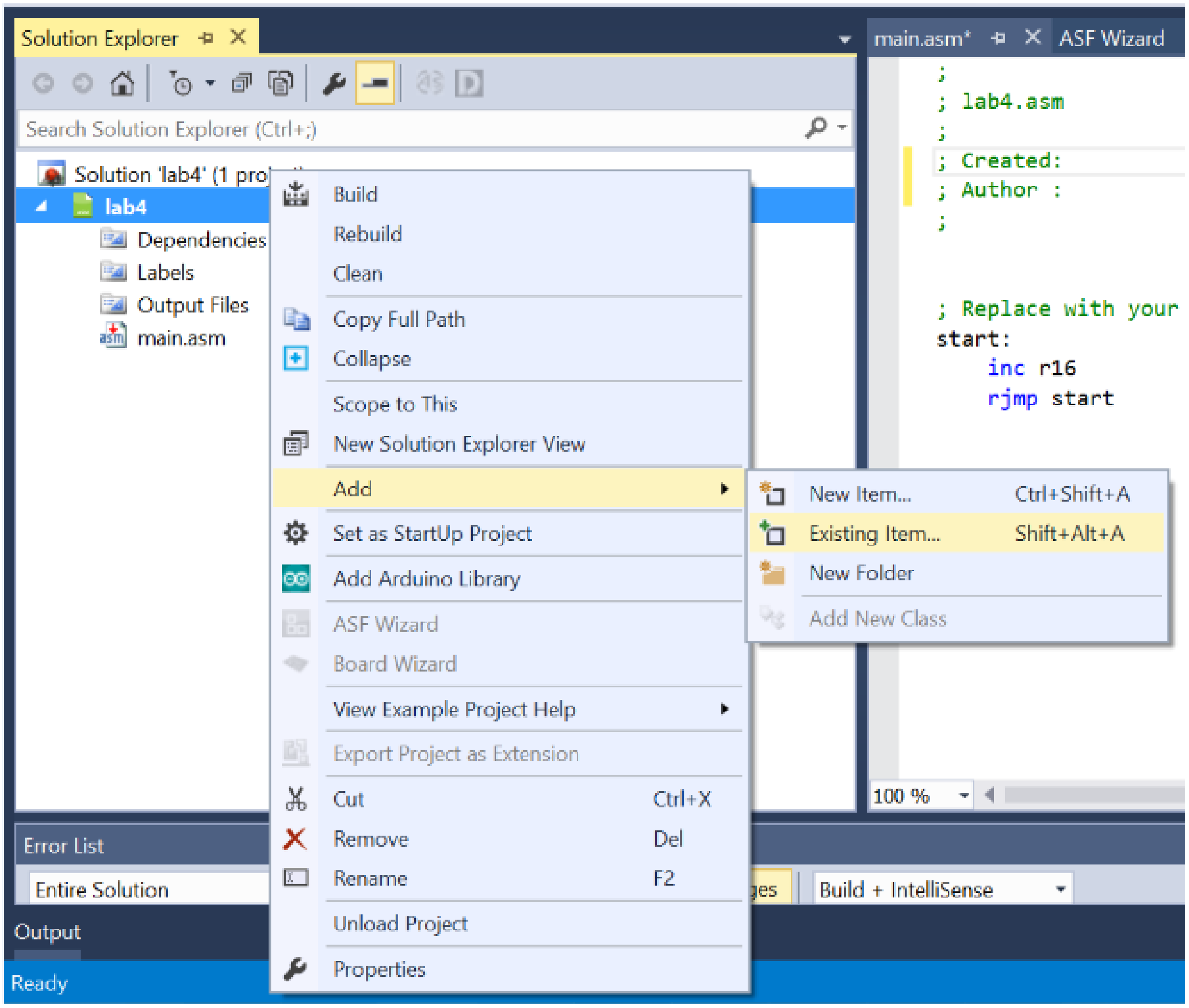
```
check_button:
        ; start a2d
        lds r16, ADCSRA
        ori r16, 0x40
        sts ADCSRA, r16

        ; wait for it to complete
wait:   lds r16, ADCSRA
        andi r16, 0x40
        brne wait

        ; read the value
        lds r16, ADCL
        lds r17, ADCH

        clr r24
        cpi r17, 0
        brne skip
        ldi r24,1
skip:   ret
```

III. Additional Resources on conneX

The file LCDKeypad_Shield_Schematic.pdf contains the wiring diagram of the buttons.
Caution: different AVR boards have different hardware configuration, refer to the LCD_Shield_Buttons.pdf for more information.

IV. Exercises:

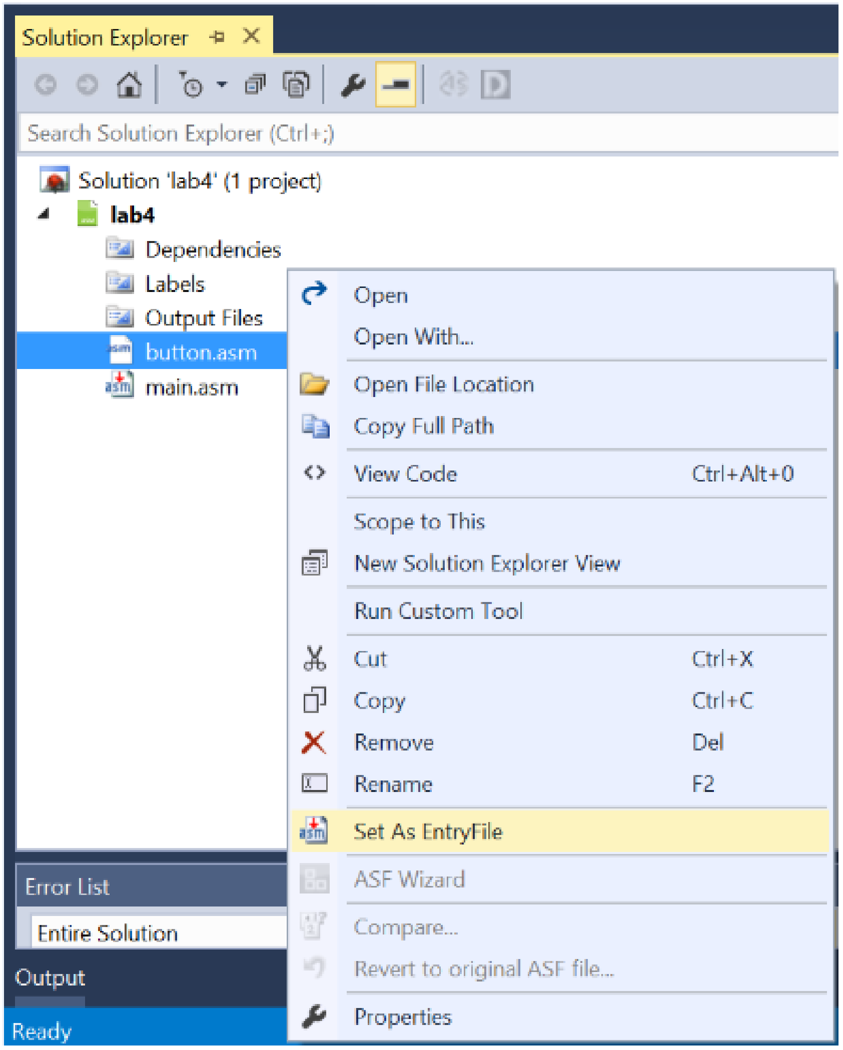
1. Download the button.asm file and put it in the same location as the main.asm file in the lab4 project folder. Then, add it to the project in Atmel Studio following the instructions below:
   a. In the Solution Explorer panel, right-click on the project name "lab4", then choose *Add* -> *Existing item* in the pulldown menus.



```
;
; lab4.asm
;
; Created:
; Author :
;


; Replace with your
start:
    inc r16
    rjmp start
```

   b. Then select the button.asm file, which you just downloaded.

   c. In the Solution Explorer panel, right-click on the button.asm file and choose the *Set As EntryFile* option. Notice there is a red arrow in the icon corresponding to the button.asm file. When you build and run your code, Atmel Studio will use the button.asm file instead of the main.asm file.



Modify the code in button.asm by adding contents to the delay function from the code you wrote for exercise 2 during the last lab. If your code isn't working as expected, you may use the code provided in the blink.asm file (on conneX).

2. Modify the code so that it detects ANY button press (except the RST reset button).

3. Modify the check_button function so that different buttons light up different LEDs. Notice that the function returns a value corresponding to which button was pressed via register r24. Your task is to write the necessary if-else control flow described in the LCD_Shield_Buttons.pdf file so that r24 contains a value 0 when no button was pressed, 1 for btnRIGHT, 2 for btnUP, 3 for btnDOWN, 4 for btnLEFT, 5 for btnSELECT. In the main program loop (not in the check_button function), write the complimentary code to turn on the corresponding LED. It's up to you to determine which LED will light up for which button.

This last exercise (#3) is optional for the lab, but it will be required for your next assignment. So, this is a chance to get the assignment started earlier and to get help with this component (if needed) as part of your lab exercises.

Submit button.asm from the lab4 project at the end of your lab, and not your project file.