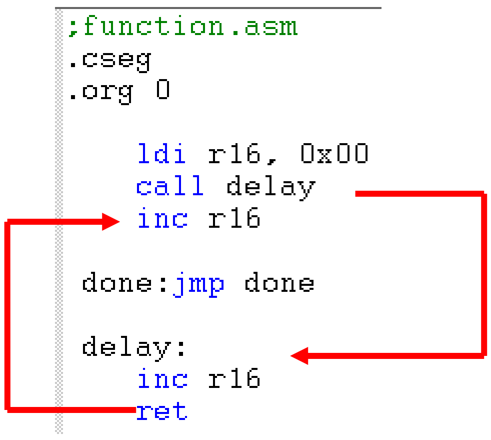
# Lab 4 Simple Function Call and Input Instructions

**Submit lab4Button.asm at the end of your lab, not your project.**

## 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 term procedure, function, or subroutine can be used interchangeably. A function call implies a transfer (branch, jump) to an address representing the entry point of the function, causing execution of the body of the function. When the function is finished, another transfer occurs to resume execution at the statement following the 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 lab4. Type the following code:



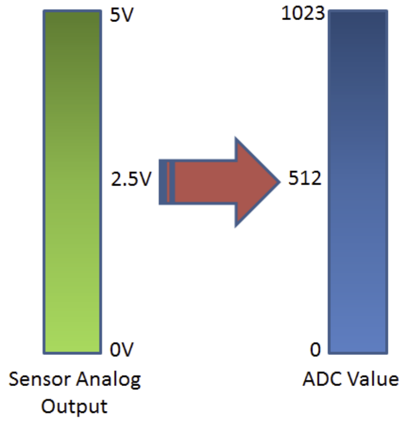
```
;function.asm
.cseg
.org 0

    ldi r16, 0x00
    call delay
    inc r16

 done:jmp done

 delay:
    inc r16
    ret
```

Observe how **`pc`** (program counter) is changed. How program is transferred to the function "delay" and is returned to the statement (**`inc r16`**) after the call. The value in **`r16`** is changed in the function call.

## II. Exercises1: download lab4Blink.asm, change the code to a function call.

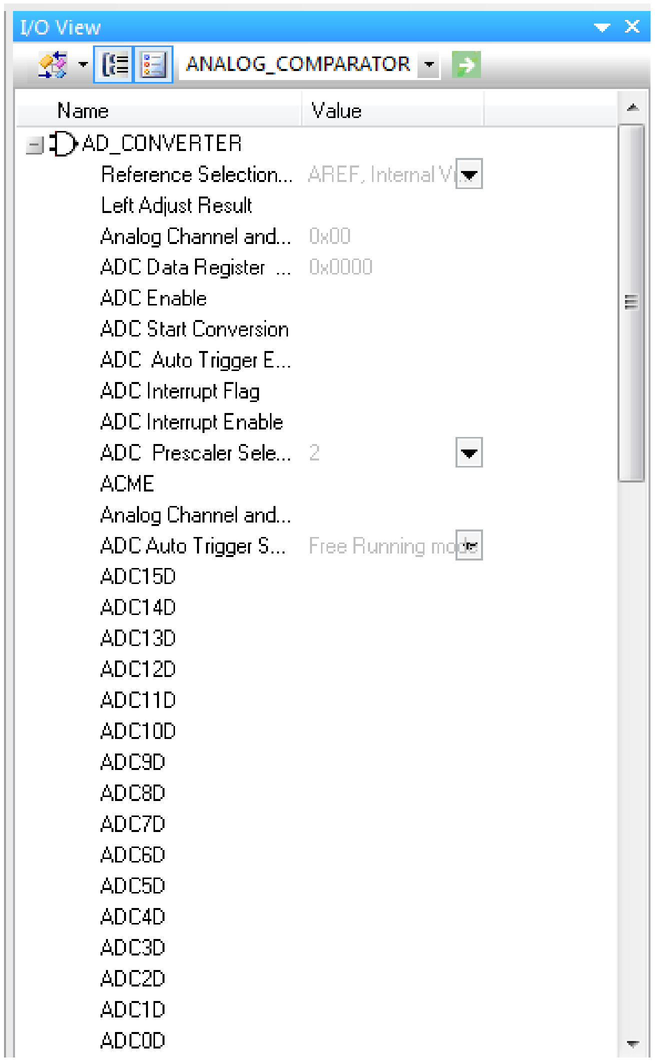
## III. Analog Inputs from Push Buttons

The six push buttons on the board are connected to one analog pin on the board. Analog means continuous time signal and we need to use the built-in **A**nalog to **D**igital **C**onvertor (ADC) to convert these continuous electronic analog signal to digital signal first. When a signal is sampled, the ADC converts the analog value to a 10-bit digital value as shown below:



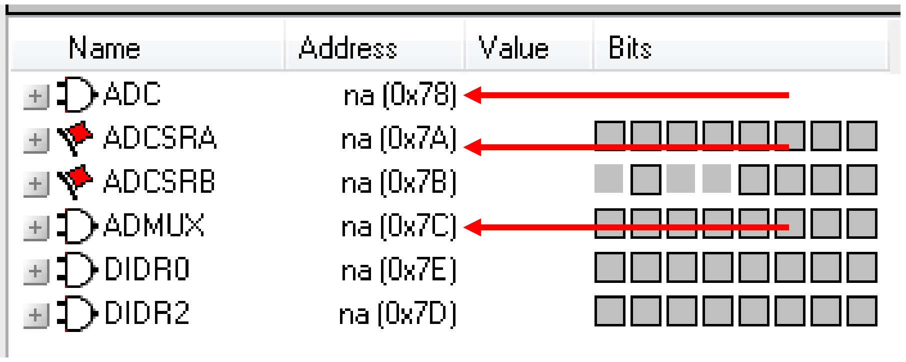
ATMEGA16/32
- 8 channels » 8 pins
- 10 bit resolution
- $2^{10}$ = 1024 steps

The following is the I/O View of the ADC:



| Name | Value |
|---|---|
| AD_CONVERTER | |
| Reference Selection... | AREF, Internal V |
| Left Adjust Result | |
| Analog Channel and... | 0x00 |
| ADC Data Register ... | 0x0000 |
| ADC Enable | |
| ADC Start Conversion | |
| ADC Auto Trigger E... | |
| ADC Interrupt Flag | |
| ADC Interrupt Enable | |
| ADC Prescaler Sele... | 2 |
| ACME | |
| Analog Channel and... | |
| ADC Auto Trigger S... | Free Running mode |
| ADC15D | |
| ADC14D | |
| ADC13D | |
| ADC12D | |
| ADC11D | |
| ADC10D | |
| ADC9D | |
| ADC8D | |
| ADC7D | |
| ADC6D | |
| ADC5D | |
| ADC4D | |
| ADC3D | |
| ADC2D | |
| ADC1D | |
| ADC0D | |

The memory addresses in SRAM are shown at the diagram below:



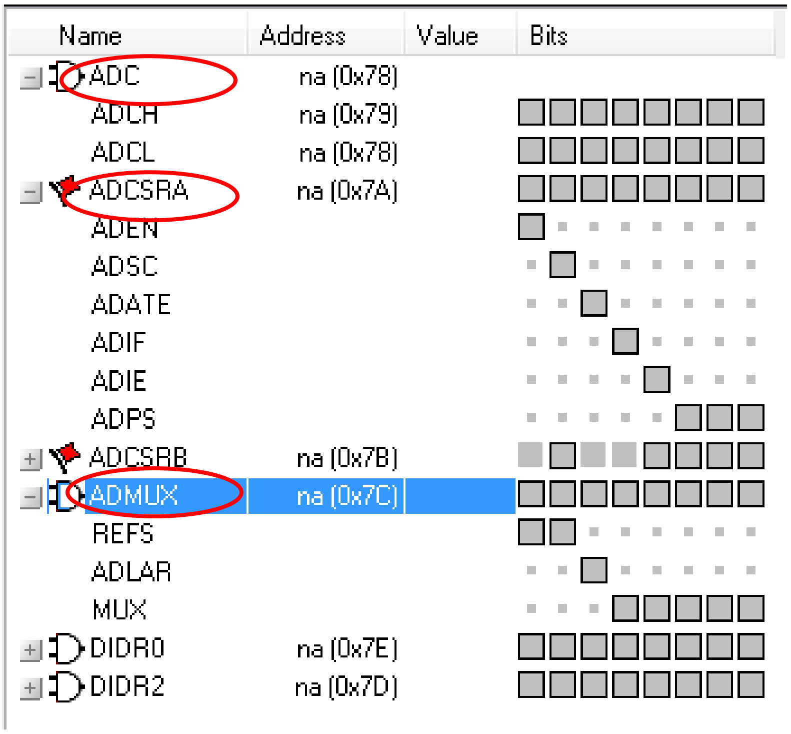
| Name | Address | Value | Bits |
|---|---|---|---|
| ADC | na (0x78) | | |
| ADCSRA | na (0x7A) | | ■■■■■■■■ |
| ADCSRB | na (0x7B) | | ▨■▨▨■■■■ |
| ADMUX | na (0x7C) | | ■■■■■■■■ |
| DIDR0 | na (0x7E) | | ■■■■■■■■ |
| DIDR2 | na (0x7D) | | ■■■■■■■■ |

ADC has multiple registers just like other ports we have used and they need to be setup properly. These registers are: ADCSRA (Status Register), ADMUX (selects a channel, as ADC can handle up to 8 channels) and ADC (10-bit Data register). ADC is split into two 8-bit registers ADCL (8-bits) and ADCH (only 2-bits are used)

Expand the "+" sign for details. Give them a symbolic name using the following code:

```
; Definitions for using the Analog to Digital Conversion
.equ ADCSRA=0x7A
.equ ADMUX=0x7C
.equ ADCL=0x78
.equ ADCH=0x79
```

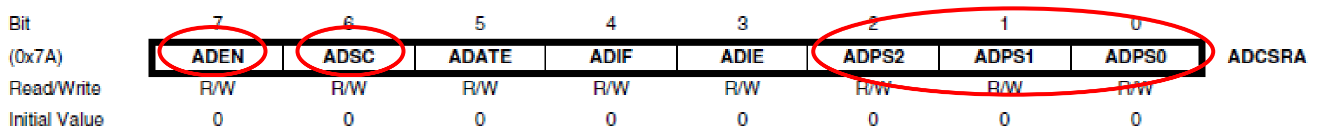
Set the proper values for those I/O registers:



| Name | Address | Value | Bits |
|---|---|---|---|
| ADC | na (0x78) | | |
| ADCH | na (0x79) | | ■■■■■■■■ |
| ADCL | na (0x78) | | ■■■■■■■■ |
| ADCSRA | na (0x7A) | | ■■■■■■■■ |
| ADEN | | | ■······· |
| ADSC | | | ·■······ |
| ADATE | | | ··■····· |
| ADIF | | | ···■···· |
| ADIE | | | ····■··· |
| ADPS | | | ·····■■■ |
| ADCSRB | na (0x7B) | | ▨■▨▨■■■■ |
| ADMUX | na (0x7C) | | ■■■■■■■■ |
| REFS | | | ■■······ |
| ADLAR | | | ··■····· |
| MUX | | | ···■■■■■ |
| DIDR0 | na (0x7E) | | ■■■■■■■■ |
| DIDR2 | na (0x7D) | | ■■■■■■■■ |

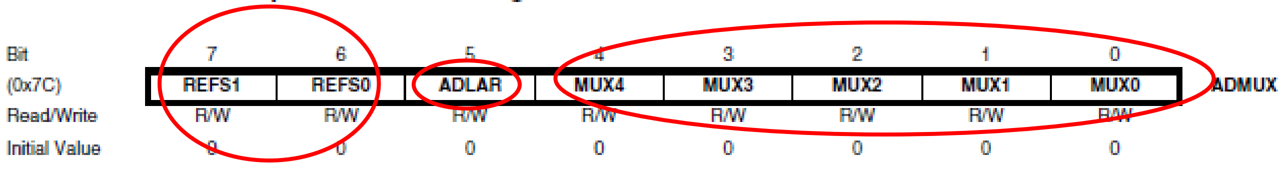
A brief description of the registers:

**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 | |

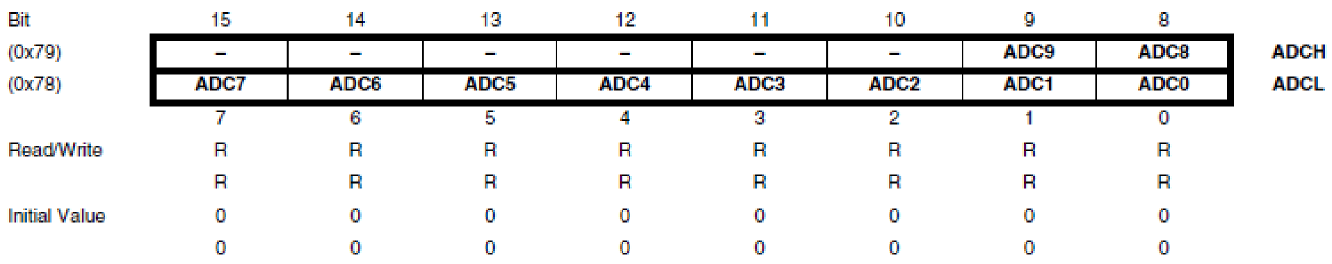
**ADMUX – ADC Multiplexer Selection Register**



| 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 | |
| Read/Write | R | R | R | R | R | R | R | R | |
| | R | R | R | R | R | R | R | R | |
| Initial Value | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |
| | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | |

To initialize the ADC, several I/O registers should be set up properly. These are: **ADEN** bit in **ADCSRA**, reference signal bits **REFS1:S0** in **ADMUX**, the channel to be selected. If **ADLAR** is enabled, then the conversion is just 8-bits and provided in ADCH.

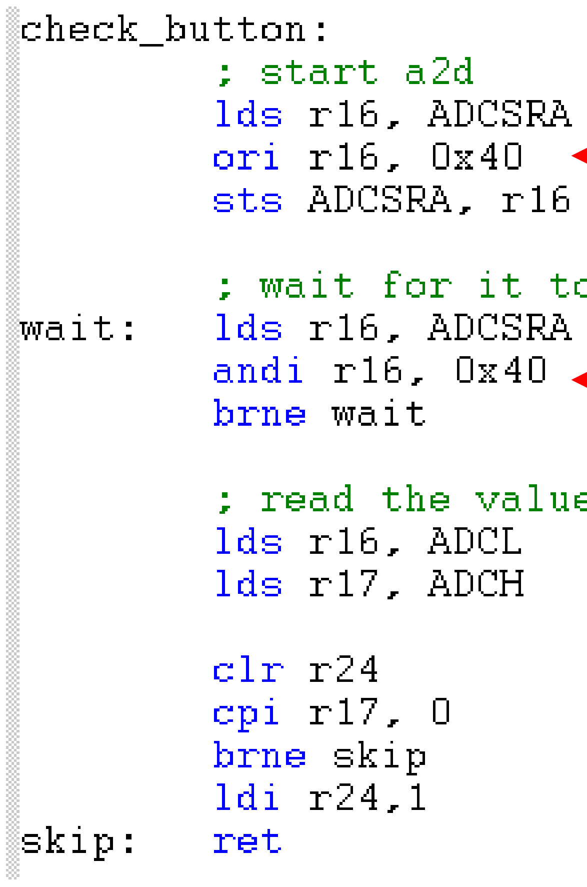
```
; initialize the Analog to Digital conversion

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 ADCLAR 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. You can find more information about these registers in the AVR Datasheet.

In order to use the ADC, first initialize it and then enable conversion when needed. The bit ADSC in ADSRA is used for this purpose. When this is set to 1, ADC will start converting the analog signal to digital format. This bit will continue to remain as 1 as long as the conversion is in progress. It will be reset to 0 after the conversion is done. After the conversion is done, 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
```

*In Single Conversion mode, turn on the ADC start conversion bit (ADSC) to start.*

*In Single Conversion mode, ADSC bit is 1 if the conversion is in progress, 0 if the conversion is completed.*

**IV. Exercises:** download lab4Button.asm and write the delay function you did in the previous exercise. Please see LCD Shield Buttons.pdf document for range of values to be used.

**V. Other Exercises:** Modify the code so that one can detect press of any button. Use the LED lights to display which button is pressed.

**Submit lab4Button.asm at the end of your lab.**

This lab is derived from the Chapter 6 of your textbook (Some Assembly Required by Timothy S. Margush) and section 26 (ADC – Analog to Digital Converter) of the datasheet of ATMEGA 2560 (See AVR Resources on connex)

**LCD Shield Buttons**

The LCD keypad shield we use in the lab is made by DF Robot. It looks like there are two versions of this board (v1.0 and v1.1). As the boards are not marked with version number, there is a high chance that both versions of these boards may be present in the lab. This lab assumed it is v1.0 board. So please use the following table and test your code in case of any problems. The limits are decimal values and corresponding hex values are provided here. Thanks.

```
if (adc_key_in > 1000) return btnNONE;      (1000 = 0x3E8)

// For V1.1 use this threshold
if (adc_key_in < 50)   return btnRIGHT;     (50 = 0x032)
if (adc_key_in < 250)  return btnUP;        (250 = 0x0FA)
if (adc_key_in < 450)  return btnDOWN;      (450 = 0x1C2)
if (adc_key_in < 650)  return btnLEFT;      (650 = 0x28A)
if (adc_key_in < 850)  return btnSELECT;    (850 = 0x352)


// For V1.0 use the one below:
if (adc_key_in < 50)   return btnRIGHT;     (50 = 0x032)
if (adc_key_in < 195)  return btnUP;        (195 = 0x0C3)
if (adc_key_in < 380)  return btnDOWN;      (380 = 0x17C)
if (adc_key_in < 555)  return btnLEFT;      (555 = 0x22B)
if (adc_key_in < 790)  return btnSELECT;    (790 = 0x316)
```

The schematic on the LCD shield is posted below:
http://www.dfrobot.com/image/data/DFR0009/LCDKeypad%20Shield%20V1.0%20SCH.pdf
It's also available on ConneX AVR Resources.