ECE 375 Lab 5

External Interrupts

Lab session: 015
Time: 12:00-13:50

Author: Astrid Delestine
Programming partner: Lucas Plastid

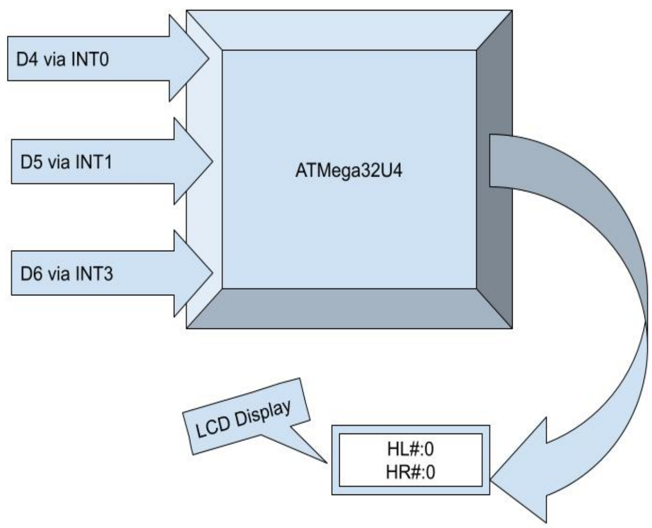
TA Signature

1 Introduction

This is the Fifth lab in the ECE 375 series and it covers using hardware interrupts to preform predescribed "bump bot" operations. Additionally it incorporated use of the LCD Display to show the user how many times the bump bot had been triggered on its left or right side.

2 Design

In this lab Lucas and I setup several different interrupt vectors that were able to trigger certain functions. These functions made the program function similarly to the Lab 1 and 2 bump bot script. Once these interrupts were created and working we moved to creating counters and displays for each of the buttons pressed. In the image seen below, one can see an example of what the LCD display would look like upon boot up.



3 Assembly Overview

As for the Assembly program an overview can be seen below.

3.1 Internal Register Definitions and Constants

Many different constants and registers are assigned in this program, and due to this they will not all be listed. Some more important registers will be highlighted however. The hlcnt and the hrcnt registers were created to count the number of times each button was pressed on either bumper. The strSize is a constant that determines how long the steady state numbers are that need to be patched in every time to the LCD are. All the other values and register assignments are either taken from Lab 1 or Lab 3 and connect to the bump bot script or the LCD scripts.

3.2 Interrupt Vectors

Vectors setup are; hit right on interrupt 0, hit left on interrupt 1, and clear counters on interrupt 3.

3.3 Initialization Routine

Firstly the stack pointer is initialized then ports B and D are initialized for output and input respectively. The LCD is then initialized in its own subroutines as we set it to turn its backlight on and clear any remaining text on the screen. Then we set it such that it displays clear delimiters for each of our button presses. Next we load up the interrupt control for falling edge detection, and configure the interrupt mask for just the 3 interrupts we had setup earlier. Finally we run the sei command to set the interrupt flag in SREG so that the interrupts can work at all.

3.4 Main Routine

The main routine is very simple due to the fact that most operations are handled outside of the main routine by interrupts. All it does is send the move forward command to the LEDs.

3.5 Subroutines

3.6 ClearCounters

This subroutine clears the counters for each button press, then clears the LCD of any overflowing numbers, and resets it back to its initial state. This is done by loading all 16 characters into the data memory that the display looks at for its characters.

3.7 toLCD

The toLCD subroutine is quite simple with regards to what we have already completed. It sets the first four bits of each row to the characters in data memory then uses the built in Bin2ASCII command to take the mpr register and print it to the LCD display. It then enables the LCD to write the characters to the screen.

3.7.1 HitRight

This subroutine is nearly the same as the subroutine built for the original bump bot script. The major changes to it are that it increments a register such that it keeps track of how many times the subroutine is called, and it also calls the toLCD command, allowing the update to be pushed to the LCD. Finally it also has a debounce filter, to disable any interrupts that may have run during the method.

3.7.2 HitLeft

This subroutine is nearly the same as the one built for the original bump bot script. It's major changes are the same as HitRight's, those being the associated counter, the toLCD command and the filter.

3.7.3 Wait

This is the stock wait function. It is unchanged from the original bump bot script.

4 Testing

Tested Each button press and compared to external calculations.

Case	Expected	Actual meet expected
d4	an increment on the LCD, and the bump bot right hit function to be called	✓
d5	an increment on the LCD, and the bump bot left hit function to be called	✓
d6	The two numbers listed on the screen reset	✓
d7	nothing	✓

Table 1: Assembly Testing Cases

5 Study Questions

1. As this lab, Lab 1, and Lab 2 have demonstrated, there are always multiple ways to accomplish the same task when programming (this is especially true for assembly programming). As an engineer, you will need to be able to justify your design choices. You have now seen the BumpBot behavior implemented using two different programming languages (AVR assembly and C), and also using two different methods of receiving external input (polling and interrupts). Explain the benefits and costs of each of these approaches. Some important areas of interest include, but are not limited to: efficiency, speed, cost of context switching, programming time, understandability, etc.

   Each of these methods has its drawbacks and most of them have benefits. I will compare and contrast each of these starting from the least complex all the way to the most complex. Firstly C programming using busy waiting. This method is by far the easiest for the programmer to implement and understand what exactly is happening. The main downside for this method is its efficiency, speed, and size. It wins however when compared to all the others with regard to context switching, programming time, and understandability. C programming with interrupts is the next step down the understandability ladder. Due to the average home programmer possibly being inexperienced, or unfamiliar with the ecosystem they are working on using interrupts in a C program will be a little bit more difficult than just using polling in their c program. The programming time, understandability, and ability to switch contexts will decrease when compared to a c program using polling, due to integration constrains and how each system implements their own interrupt systems differently. As for assembly, only those who really know what they are doing can even begin to understand what is happening, so it would make sense to me that the polling version of this bump bot script is worse in every way but understandability. It takes less time to program, less space, and is more efficient, and faster to respond to changes. Understandability falls by the wayside when working in assembly as it is the least important factor. as for portability, using any assembly code will need to change from one system to the next, so there is not that much more of a loss here in regards to that.

2. Instead of using the Wait function that was provided in BasicBumpBot.asm, is it possible to use a timer/counter interrupt to perform the one-second delays that are a part of the BumpBot behavior, while still using external interrupts for the bumpers? Give a reasonable argument either way, and be sure to mention if interrupt priority had any effect on your answer.

   It is possible as the priority of the timers is lower, they would have to be setup in such a way that they were allowed to run while the other interrupt is running. Or this is not possible due to the fact that they have a lower priority status. It truly depends on if the timer continues counting while the main function is interrupted. There is one other case, where the counter continues counting but will not reset without the interrupt, in this case, using an external interrupt timer would be feasible however a prescaler would need to be applied.

3. List the correct sequence of AVR assembly instructions needed to store the contents of registers R25:R24 into Timer/Counter1's 16-bit register, TCNT1. (You may assume that registers R25:R24 have already been initialized to contain some 16-bit value.

   (because its an IO location)

   in r25 TCNT1H in r24 TNCT1L

4. List the correct sequence of AVR assembly instructions needed to load the contents of Timer/-Counter1's 16-bit register, TCNT1, into registers R25:R24

   out TCNT1H r25 out TCNT1L r24

5. Suppose Timer/Counter0 (an 8-bit timer) has been configured to operate in Normal mode, and with no prescaling (i.e., clkT 0 = clkI/O = 8 MHz). The decimal value "128" has just been written into Timer/Counter0's 8-bit register, TCNT0. How long will it take for the TOV0 flag to become set? Give your answer as an amount of time, not as a number of cycles

   it will take 16 microseconds

6 Difficulties

This Lab challenged the thinking power of implementation of ideas we have learned in lecture. It was not too difficult however did require refrenceing both the AVR manual and the atmega32u4 datasheet.

7 Conclusion

In conclusion, this lab introduced and allowed the student to understand many more aspects of how to program a function using interrupts and modify an existing program to work with an LCD in conjunction with those interrupts. This lab proves that the student is learning how to modify certain code structures and is becoming more fluent in the AVR programming scheme

8 Source Code

Listing 1: Assembely Bump Bot Script

```
;***********************************************************
;*   This is the skeleton file for Lab 5 of ECE 375
;*
;*    Author: Astrid Delestine & Lucas Plaisted
;*       Date: 2/23/2023
;*
;***********************************************************

.include "m32U4def.inc"          ; Include definition file

;************************************************************
;* Variable and Constant Declarations
;************************************************************
.def    mpr = r16               ; Multi-Purpose Register
.def    waitcnt = r17               ; Wait Loop Counter
.def    ilcnt = r18             ; Inner Loop Counter
.def    olcnt = r19             ; Outer Loop Counter
.def    hlcnt = r15             ; Hit Left Counter
.def    hrcnt = r14             ; Hit Right Counter
;.def   count = r20             ; needed for LCD binToASCII

.equ    WTime = 50              ; Time to wait in wait loop

.equ    WskrR = 4               ; Right Whisker Input Bit
.equ    WskrL = 5               ; Left Whisker Input Bit
.equ    EngEnR = 5              ; Right Engine Enable Bit
.equ    EngEnL = 6              ; Left Engine Enable Bit
.equ    EngDirR = 4             ; Right Engine Direction Bit
.equ    EngDirL = 7             ; Left Engine Direction Bit

;//TAKEN FROM LAB3

.equ    lcdL1 = 0x00    ; Make LCD Data Memory locations constants
.equ    lcdH1 = 0x01
.equ    lcdL2 = 0x10    ; lcdL1 means the low part of line 1's location
.equ    lcdH2 = 0x01 ; lcdH2 means the high part of line 2's location
.equ    lcdENDH = 0x01  ; as it sounds, the last space in data mem
.equ    lcdENDL = 0x1F      ; for storing lcd text

;//END TAKEN FROM LAB3

.equ    strSize = 4;

```

```

;//////////////////////////////////////////////////////////////
;These macros are the values to make the TekBot Move.
;//////////////////////////////////////////////////////////////

.equ    MovFwd = (1<<EngDirR|1<<EngDirL)    ; Move Forward Command
.equ    MovBck = $00                        ; Move Backward Command
.equ    TurnR = (1<<EngDirL)                ; Turn Right Command
.equ    TurnL = (1<<EngDirR)                ; Turn Left Command
.equ    Halt = (1<<EngEnR|1<<EngEnL)        ; Halt Command

;***********************************************************
;*  Start of Code Segment
;***********************************************************
.cseg                           ; Beginning of code segment

;***********************************************************
;*  Interrupt Vectors
;***********************************************************
.org    $0000                   ; Beginning of IVs
        rjmp    INIT            ; Reset interrupt

        ; Set up interrupt vectors for any interrupts being used


        ; This is just an example:
;.org   $002E                   ; Analog Comparator IV
;       rcall   HandleAC        ; Call function to handle interrupt
;       reti                    ; Return from interrupt
.org    $0002 ;INT0
        rcall   HitRight        ;RIGHT WHISKER
        reti
.org    $0004 ;INT1
        rcall   HitLeft     ;LEFT WHISKER
        reti
;.org   $0006 ;INT2
.org    $0008 ;INT3
        rcall   ClearCounters       ;CLEAR COUNTERS
        reti
;.org   $000E ;INT6

.org    $0056                   ; End of Interrupt Vectors

;***********************************************************
;*  Program Initialization
;***********************************************************
```

```
INIT:
	; Initialize the Stack Pointer (VERY IMPORTANT!!!!)
		ldi	mpr, low(RAMEND)
		out	SPL, mpr		; Load SPL with low byte of RAMEND
		ldi	mpr, high(RAMEND)
		out	SPH, mpr		; Load SPH with high byte of RAMEND

	; Initialize Port B for output
		ldi	mpr, $FF		; Set Port B Data Direction Register
		out	DDRB, mpr		; for output
		ldi	mpr, $00		; Initialize Port B Data Register
		out	PORTB, mpr		; so all Port B outputs are low

	; Initialize Port D for input
		ldi	mpr, $00		; Set Port D Data Direction Register
		out	DDRD, mpr		; for input
		ldi	mpr, $FF		; Initialize Port D Data Register
		out	PORTD, mpr		; so all Port D inputs are Tri-State



	;init the LCD
		rcall LCDInit
		rcall LCDBacklightOn
		rcall LCDClr
		rcall toLCD

		rcall ClearCounters


	; Initialize external interrupts
		; Set the Interrupt Sense Control to falling edge
		ldi mpr, 0b10001010
		sts EICRA, mpr;

	; Configure the External Interrupt Mask
		ldi mpr, 0b0000_1011 ; x0xx_0000 ; all disabled
		out EIMSK, mpr;
	; Turn on interrupts
		; NOTE: This must be the last thing to do in the INIT function
	sei ; Turn on interrupts

;***********************************************************
;*	Main Program
;***********************************************************
MAIN:					; The Main program
```

```

        ldi     mpr, MovFwd         ; Load Move Forward Command
        out     PORTB, mpr

        rjmp    MAIN                ; Create an infinite while loop to
                                    ; signify the end of the program.

;***********************************************************
;*  Functions and Subroutines
;***********************************************************

;-----------------------------------------------------------
;   You will probably want several functions, one to handle the
;   left whisker interrupt, one to handle the right whisker
;   interrupt, and maybe a wait function
;------------------------------------------------------------

;-----------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;       beginning of your functions
;-----------------------------------------------------------
ClearCounters:                              ; Begin a function with a label

        ; Save variable by pushing them to the stack

        ; Execute the function here
        clr     hrcnt               ; sets hlcnt and hrcnt to zero by
        clr     hlcnt               ; doing an xor operation with itself

        push ZL                 ; Save vars to stack
        push ZH
        push XL
        push XH
        push mpr
        push ilcnt

        ldi   ZL , low(STRING_BEG<<1)       ; Sets ZL to the low bits
                                    ; of the first string location
        ldi   ZH , high(STRING_BEG<<1)      ; Sets ZH to the first
                                    ; of the first string location
        ldi   XH , lcdH1
        ldi   XL , lcdL1
        ldi   ilcnt , 16

CCl1: ; While ilcnt != zero 1
```

```
        lpm  mpr, Z+
        st   X+ , mpr
        dec  ilcnt
        brne CCl1

        ldi ZL, low(STRING2_BEG<<1)
        ldi ZH, high(STRING2_BEG<<1)
        ;z is already pointing at the second
        ;string due to how memory is stored
        ldi  XH , lcdH2
        ldi  XL , lcdL2
        ldi  ilcnt , 16

CCl2: ; While ilcnt != zero 2
        lpm  mpr, Z+
        st   X+ , mpr
        dec  ilcnt
        brne CCl2

        rcall LCDWrite

        ldi mpr , 0b0000_1011
        out EIFR, mpr

        pop ilcnt
        pop mpr
        pop XH
        pop XL
        pop ZH
        pop ZL                  ; Pop vars off of stack
        ; Restore variable by popping them from the stack
        ; in reverse order

        ret                     ; End a function with RET


;-----------------------------------------------------------
; Func: toLCD
; Desc: Takes various info and pushes it to the LCD
;       *HL#:0
;       *HR#:0
;-----------------------------------------------------------
toLCD:
        push ZL             ; Save vars to stack
        push ZH
        push XL
```

```
        push XH
        push mpr
        push ilcnt

        ; Sets ZL to the low bits of the first string location
        ldi  ZL , low(STRING_BEG<<1)
        ldi  ZH , high(STRING_BEG<<1)
        ;points to the data location where LCD draws from
        ldi  XH , lcdH1
        ldi  XL , lcdL1
        ldi  ilcnt , 4

Line1Loop: ; While ilcnt != zero
        lpm  mpr, Z+
        st   X+ , mpr
        dec  ilcnt
        brne Line1Loop
        //end loop

        mov mpr, hlcnt; copies the counter to mpr

        rcall Bin2ASCII
        ; Takes a value in MPR and outputs
        ; the ascii equivilant to XH:XL
        ; convineintly X is currently pointing where
        ; I would like this number to go


        ldi ZL, low(STRING2_BEG<<1)
        ldi ZH, high(STRING2_BEG<<1)

        ldi  XH , lcdH2
        ldi  XL , lcdL2
        ldi  ilcnt , 4

Line2Loop: ; While ilcnt != zero 2
        lpm  mpr, Z+
        st   X+ , mpr
        dec  ilcnt
        brne Line2Loop


        mov mpr, hrcnt;
        rcall Bin2ASCII

        rcall LCDWrite
```

```
        pop ilcnt
        pop mpr
        pop XH
        pop XL
        pop ZH
        pop ZL                      ; Pop vars off of stack


        ret
;----------------------------------------------------------------
; Sub:  HitRight
; Desc: Handles functionality of the TekBot when the right whisker
;       is triggered.
;----------------------------------------------------------------
HitRight:
        push    mpr             ; Save mpr register
        push    waitcnt             ; Save wait register
        in      mpr, SREG       ; Save program state
        push    mpr             ;

        ; Move Backwards for a second
        ldi     mpr, MovBck ; Load Move Backward command
        out     PORTB, mpr  ; Send command to port
        ldi     waitcnt, (WTime<<1) ; Shifted bit back by 1,
                                    ; making the wait time two seconds
        rcall   Wait                ; Call wait function

        ; Turn left for a second
        ldi     mpr, TurnL  ; Load Turn Left Command
        out     PORTB, mpr  ; Send command to port
        ldi     waitcnt, WTime  ; Wait for 1 second
        rcall   Wait                ; Call wait function

        ; Move Forward again
        ldi     mpr, MovFwd ; Load Move Forward command
        out     PORTB, mpr  ; Send command to port

        pop     mpr         ; Restore program state
        out     SREG, mpr   ;
        pop     waitcnt     ; Restore wait register
```

```
        pop     mpr     ; Restore mpr

        inc     hrcnt;
        rcall   toLCD;
        ;fix debounce
        ldi mpr , 0b0000_1011
        out EIFR, mpr
        ret             ; Return from subroutine

;----------------------------------------------------------
; Sub:  HitLeft
; Desc: Handles functionality of the TekBot when the left whisker
;       is triggered.
;----------------------------------------------------------
HitLeft:
        push    mpr             ; Save mpr register
        push    waitcnt             ; Save wait register
        in      mpr, SREG       ; Save program state
        push    mpr             ;

        ; Move Backwards for a second
        ldi     mpr, MovBck ; Load Move Backward command
        out     PORTB, mpr  ; Send command to port
        ldi     waitcnt, (WTime<<1) ; Wait for 1 second
        rcall   Wait                ; Call wait function

        ; Turn right for a second
        ldi     mpr, TurnR  ; Load Turn Left Command
        out     PORTB, mpr  ; Send command to port
        ldi     waitcnt, WTime  ; Wait for 1 second
        rcall   Wait                ; Call wait function

        ; Move Forward again
        ldi     mpr, MovFwd ; Load Move Forward command
        out     PORTB, mpr  ; Send command to port

        pop     mpr     ; Restore program state
        out     SREG, mpr   ;
        pop     waitcnt     ; Restore wait register
        pop     mpr     ; Restore mpr

        inc     hlcnt   ;
        rcall   toLCD;
                ;fix debounce
        ldi mpr , 0b0000_1011
        out EIFR, mpr
```

```
        ret                     ; Return from subroutine

;-----------------------------------------------------------
; Sub:  Wait
; Desc: A wait loop that is 16 + 159975*waitcnt cycles or roughly
;       waitcnt*10ms.  Just initialize wait for the specific amount
;       of time in 10ms intervals. Here is the general eqaution
;       for the number of clock cycles in the wait loop:
;           ((((((3*ilcnt)-1+4)*olcnt)-1+4)*waitcnt)-1+16
;-----------------------------------------------------------
Wait:
        push    waitcnt         ; Save wait register
        push    ilcnt           ; Save ilcnt register
        push    olcnt           ; Save olcnt register

Loop:   ldi     olcnt, 224      ; load olcnt register
OLoop:  ldi     ilcnt, 237      ; load ilcnt register
ILoop:  dec     ilcnt           ; decrement ilcnt
        brne    ILoop           ; Continue Inner Loop
        dec     olcnt       ; decrement olcnt
        brne    OLoop           ; Continue Outer Loop
        dec     waitcnt     ; Decrement wait
        brne    Loop            ; Continue Wait loop

        pop     olcnt       ; Restore olcnt register
        pop     ilcnt       ; Restore ilcnt register
        pop     waitcnt     ; Restore wait register
        ret                 ; Return from subroutine



;-----------------------------------------------------------
; Func: Template function header
; Desc: Cut and paste this and fill in the info at the
;       beginning of your functions
;-----------------------------------------------------------
FUNC:                           ; Begin a function with a label

        ; Save variable by pushing them to the stack

        ; Execute the function here

        ; Restore variable by popping them from the stack in reverse order

        ret                     ; End a function with RET

```

```
;***********************************************************
;*  Stored Program Data
;***********************************************************

; Enter any stored data you might need here
;.org
STRING_BEG:
.DB        "HL#:0           "        ; Declaring data in ProgMem
STRING2_BEG:
.DB        "HR#:0           "
STRING_END:


;***********************************************************
;*  Additional Program Includes
;***********************************************************
.include "LCDDriver.asm"          ; Include the LCD Driver
```