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.
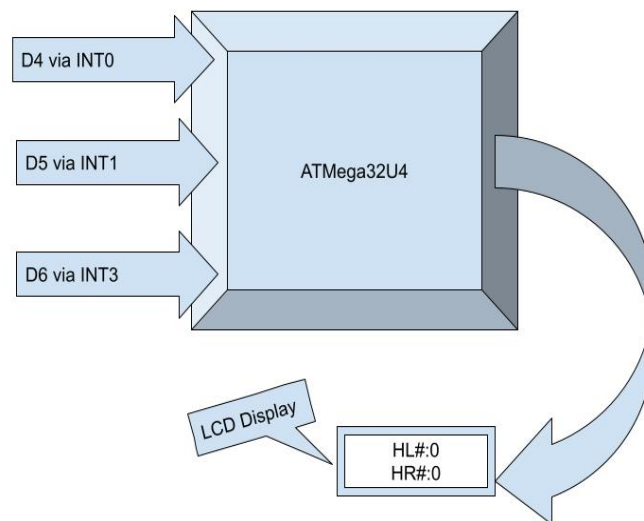
3   Assembly Overview

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

3.1   Internal Register Definitions and Constants

The multipurpose register was setup as r16. At r0 and r1 any multiplication output will be set, such that the outputs of any multiplication operation are automatically assigned to them. A default zero register is set to r2. Two other generic variable registers are defined as r3 and r4. Finally two registers named oloop and iloop are used for counting within the assembly itself.

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

3.4 Main Routine

3.5 Subroutines

3.6 ClearCounters

This subroutine adds two 16 bit numbers together. It does this by taking each X Y and Z registers and setting them to operator 1 operator 2 and the location to save the result respectively. Next, using the A variable register as an intermediary the two inputs were added together and saved into the Z register location. Finally if the carry flag is set at the end of the whole operation then we can assume the most significant bit needs to be set to 1, so it is done.

3.7 toLCD

This subroutine takes two different inputs and subtracts the first from the second. The X Y and Z registers are initialized at the beginning of this subroutine first, the same way that they are in the add subroutine. In the actual operation part of this subroutine A and B registers are loaded with the lower values of X and Y and subtracted from each other. This is then preformed again to account for the second word. Due to the fact that we do not need to worry about signage these are each saved directly to the location pointed to by Z

3.7.1 HitRight

This subroutine takes 4 different data locations and multiplies 24 bits by 24 bits, resulting in a 48 bit number. It must be built differently to the MUL16 operation. Every time addition occurs we need to check for the carry bit and pass it forward if necessary. This will continue until there is no carry bit to pass upward. In reality this can only ever happen up to 4 times. In this subroutine the first operand is loaded into the Z pointer. Then the second operand is loaded into the Y pointer, finally the result is loaded into the X register. For each of these, they load the start of each because they will increment throughout the method. The data in Y an Z are multiplied and the result is stored in r0 and r1. ADDMUL2x is then called. This fixes the carry bit problem of multiplying by 24 bits and as long as we call ADDMUL2x after our multiplication then everything will work out.

3.7.2 HitLeft

This subroutine adds a partial multiplication result to the location x is pointing to. This presumes that x is already pointing to the location where the low result of the current multiplication needs to go. Essentially it takes the multiplication outputs and cycles the carry bit up until it cannot anymore. It utilized a loop moving the carry byte in and out of X when necessary.

3.7.3 Wait

Preforms the operation $((G - H) + I)^2$ Using multiplication, addition and subtraction.

4 Testing

Tested Each input value and compared to external calculations.

Case	Expected	Actual meet expected
$\$FCBA + \$FFFF$	$\$01FCB9$	✓
$\$FCB9 - \$E420$	$\$1899$	✓
$\$00FFFFFF * \$00FFFFFF$	$\$FFFFFE000001$	✓
$((\$FCBA - \$2022) + \$21BB)^2$	$\$FCA8CEE9$	✓

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.

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.

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) out TCNT1H r25 out TCNT1L r24

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

   in r25 TCNT1H in r24 TNCT1L

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 was more challenging than the last, however it did not require us to learn anything outside of lecture. This is a good thing, due to the fact that we re only expected to know exactly what we are taught.

7 Conclusion

This lab cemented the ideas of logical operands and allowed the student to understand how computers operate with large numbers, especially larger numbers than they might be able to handle naively. Additionally the pencil and paper method described in the handout was not how I was taught how to do multiplication, so the solution may be more or less difficult depending on the students type of education.

8 Source Code

```
1   ;*****************************************************************
2   ;*    This  is  the  skeleton  file  for  Lab  5  of  ECE  375
3   ;*
4   ;*      Author: Astrid  Delestine  &  Lucas  Plaisted
5   ;*        Date:  2/23/2023
6   ;*
7   ;*****************************************************************
8
9   .include "m32U4def.inc"          ; Include  definition  file
10
11  ;*****************************************************************
12  ;* Variable  and  Constant  Declarations
13  ;*****************************************************************
14  .def      mpr = r16              ; Multi−Purpose  Register
15  .def      waitcnt = r17              ; Wait  Loop  Counter
16  .def      ilcnt = r18            ; Inner  Loop  Counter
17  .def      olcnt = r19            ; Outer  Loop  Counter
18  .def      hlcnt = r15            ; Hit  Left  Counter
19  .def      hrcnt = r14            ; Hit  Right  Counter
20  ;.def     count = r20            ; needed  for  LCD  binToASCII
21
22  .equ      WTime = 50             ; Time  to  wait  in  wait  loop
23
24  .equ      WskrR = 4              ; Right  Whisker  Input  Bit
25  .equ      WskrL = 5              ; Left  Whisker  Input  Bit
26  .equ      EngEnR = 5             ; Right  Engine  Enable  Bit
27  .equ      EngEnL = 6             ; Left  Engine  Enable  Bit
28  .equ      EngDirR = 4            ; Right  Engine  Direction  Bit
29  .equ      EngDirL = 7            ; Left  Engine  Direction  Bit
30
31  ;//TAKEN FROM LAB3
32
33  .equ      lcdL1 = 0x00     ; Make  LCD  Data  Memory  locations  constants
34  .equ      lcdH1 = 0x01
35  .equ      lcdL2 = 0x10     ; lcdL1  means  the  low  part  of  line  1's  location
36  .equ      lcdH2 = 0x01 ; lcdH2  means  the  high  part  of  line  2's  location
37  .equ      lcdENDH = 0x01   ; as  it  sounds,  the  last  space  in  data  mem
38  .equ      lcdENDL = 0x1F        ; for  storing  lcd  text
39
40  ;//END TAKEN FROM LAB3
41
42  .equ      strSize = 4;
43
```

```
44
45  ;////////////////////////////////////////////////////////////////
46  ;These macros are the values to make the TekBot Move.
47  ;////////////////////////////////////////////////////////////////
48
49  .equ     MovFwd = (1<<EngDirR|1<<EngDirL)      ; Move Forward Command
50  .equ     MovBck = $00                    ; Move Backward Command
51  .equ     TurnR = (1<<EngDirL)               ; Turn Right Command
52  .equ     TurnL = (1<<EngDirR)               ; Turn Left Command
53  .equ     Halt = (1<<EngEnR|1<<EngEnL)        ; Halt Command
54
55  ;*****************************************************************
56  ;*   Start of Code Segment
57  ;*****************************************************************
58  .cseg                                    ; Beginning of code segment
59
60  ;*****************************************************************
61  ;*   Interrupt  Vectors
62  ;*****************************************************************
63  .org     $0000                           ; Beginning of IVs
64          rjmp     INIT                     ; Reset interrupt
65
66          ; Set up interrupt vectors for any interrupts being used
67
68
69          ; This is just an example:
70  ;.org     $002E                           ; Analog Comparator IV
71  ;          rcall    HandleAC               ; Call function to handle interrupt
72  ;          reti                           ; Return from interrupt
73  .org     $0002  ;INT0
74          rcall    HitRight           ;RIGHT WHISKER
75          reti
76  .org     $0004  ;INT1
77          rcall    HitLeft        ;LEFT WHISKER
78          reti
79  ;.org     $0006  ;INT2
80  .org     $0008  ;INT3
81          rcall    ClearCounters          ;CLEAR COUNTERS
82          reti
83  ;.org     $000E  ;INT6
84
85  .org     $0056                           ; End of Interrupt Vectors
86
87  ;*****************************************************************
88  ;*   Program  Initialization
89  ;*****************************************************************
```

```
90   INIT:
91        ; Initialize the Stack Pointer (VERY IMPORTANT!!!!)
92            ldi     mpr, low(RAMEND)
93            out     SPL, mpr          ; Load SPL with low byte of RAMEND
94            ldi     mpr, high(RAMEND)
95            out     SPH, mpr          ; Load SPH with high byte of RAMEND
96
97        ; Initialize Port B for output
98            ldi     mpr, $FF          ; Set Port B Data Direction Register
99            out     DDRB, mpr         ; for output
100           ldi     mpr, $00          ; Initialize Port B Data Register
101           out     PORTB, mpr        ; so all Port B outputs are low
102
103       ; Initialize Port D for input
104           ldi     mpr, $00          ; Set Port D Data Direction Register
105           out     DDRD, mpr         ; for input
106           ldi     mpr, $FF          ; Initialize Port D Data Register
107           out     PORTD, mpr        ; so all Port D inputs are Tri−State
108
109
110
111       ; init the LCD
112           rcall   LCDInit
113           rcall   LCDBacklightOn
114           rcall   LCDClr
115           rcall   toLCD
116
117           rcall   ClearCounters
118
119
120       ; Initialize external interrupts
121           ; Set the Interrupt Sense Control to falling edge
122           ldi mpr, 0b10001010
123           sts EICRA, mpr;
124
125       ; Configure the External Interrupt Mask
126           ldi mpr, 0b0000_1011 ; x0xx_0000 ; all disabled
127           out EIMSK, mpr;
128       ; Turn on interrupts
129           ; NOTE: This must be the last thing to do in the INIT function
130       sei ; Turn on interrupts
131
132   ;******************************************************************
133   ;* Main Program
134   ;******************************************************************
135   MAIN:                             ; The Main program
```

```
136
137             ldi        mpr, MovFwd          ; Load Move Forward Command
138             out        PORTB,  mpr
139
140             rjmp       MAIN                 ; Create  an  infinite  while  loop  to
141                                             ; signify  the  end  of  the  program.
142
143    ;************************************************************
144    ;*   Functions  and  Subroutines
145    ;************************************************************
146
147    ;─────────────────────────────────────────────────────────
148    ;    You  will  probably  want  several  functions ,  one  to  handle  the
149    ;    left  whisker  interrupt ,  one  to  handle  the  right  whisker
150    ;    interrupt ,  and  maybe  a  wait  function
151    ;─────────────────────────────────────────────────────────
152
153    ;─────────────────────────────────────────────────────────
154    ; Func:  Template  function  header
155    ; Desc:  Cut  and  paste  this  and  fill  in  the  info  at  the
156    ;         beginning  of  your  functions
157    ;─────────────────────────────────────────────────────────
158    ClearCounters:                                      ; Begin  a  function  with  a  label
159
160             ; Save  variable  by  pushing  them  to  the  stack
161
162             ; Execute  the  function  here
163             clr        hrcnt                ; sets  hlcnt  and  hrcnt  to  zero  by
164             clr        hlcnt                ; doing  an  xor  operation  with  itself
165
166          push ZL                    ; Save  vars  to  stack
167          push ZH
168          push XL
169          push XH
170          push mpr
171          push ilcnt
172
173             ldi   ZL ,  low(STRING_BEG<<1)      ; Sets  ZL  to  the  low  bits
174                                             ; of  the  first  string  location
175             ldi   ZH ,  high(STRING_BEG<<1)     ; Sets  ZH  to  the  first
176                                             ; of  the  first  string  location
177             ldi   XH ,  lcdH1
178             ldi   XL ,  lcdL1
179             ldi   ilcnt  , 16
180
181    CC11:  ; While  ilcnt  != zero  1
```

8

```
182          lpm   mpr, Z+
183          st    X+ , mpr
184          dec   ilcnt
185          brne  CCl1
186
187          ldi ZL,  low(STRING2_BEG<<1)
188          ldi ZH,  high(STRING2_BEG<<1)
189          ;z  is  already  pointing  at  the  second
190          ;string  due  to  how  memory  is  stored
191          ldi   XH , lcdH2
192          ldi   XL , lcdL2
193          ldi   ilcnt  , 16
194
195   CCl2:  ; While  ilcnt != zero  2
196          lpm   mpr, Z+
197          st    X+ , mpr
198          dec   ilcnt
199          brne  CCl2
200
201          rcall  LCDWrite
202
203          pop ilcnt
204          pop mpr
205          pop XH
206          pop XL
207          pop ZH
208          pop ZL                      ; Pop  vars  off  of  stack
209          ; Restore  variable  by  popping  them  from  the  stack
210          ; in  reverse  order
211
212          ret                         ; End  a  function  with  RET
213
214
215   ;─────────────────────────────────────────────
216   ; Func: toLCD
217   ; Desc: Takes  various  info  and  pushes  it  to  the  LCD
218   ;        *HL#:0
219   ;        *HR#:0
220   ;─────────────────────────────────────────────
221   toLCD:
222          push ZL                  ; Save  vars  to  stack
223          push ZH
224          push XL
225          push XH
226          push mpr
227          push ilcnt
```

```
228
229             ; Sets ZL to the low bits of the first string location
230             ldi   ZL , low(STRING_BEG<<1)
231             ldi   ZH , high(STRING_BEG<<1)
232             ;points to the data location where LCD draws from
233             ldi   XH , lcdH1
234             ldi   XL , lcdL1
235             ldi   ilcnt , 4
236
237   Line1Loop: ; While ilcnt != zero
238             lpm   mpr, Z+
239             st    X+ , mpr
240             dec   ilcnt
241             brne  Line1Loop
242             //end loop
243
244             mov mpr, hlcnt; copies the counter to mpr
245
246             rcall Bin2ASCII
247             ; Takes a value in MPR and outputs
248             ; the ascii equivilant to XH:XL
249             ; convineintly X is currently pointing where
250             ; I would like this number to go
251
252
253             ldi ZL, low(STRING2_BEG<<1)
254             ldi ZH, high(STRING2_BEG<<1)
255
256             ldi   XH , lcdH2
257             ldi   XL , lcdL2
258             ldi   ilcnt , 4
259
260   Line2Loop: ; While ilcnt != zero 2
261             lpm   mpr, Z+
262             st    X+ , mpr
263             dec   ilcnt
264             brne  Line2Loop
265
266
267             mov mpr, hrcnt;
268             rcall Bin2ASCII
269
270             rcall LCDWrite
271
272
273
```

```
274
275
276
277          pop ilcnt
278          pop mpr
279          pop XH
280          pop XL
281          pop ZH
282          pop ZL                              ; Pop vars off of stack
283
284
285          ret
286   ;─────────────────────────────────────────────────────────────
287   ; Sub:  HitRight
288   ; Desc: Handles functionality of the TekBot when the right whisker
289   ;       is triggered.
290   ;─────────────────────────────────────────────────────────────
291   HitRight:
292          push     mpr           ; Save mpr register
293          push     waitcnt         ; Save wait register
294          in       mpr, SREG    ; Save program state
295          push     mpr            ;
296
297          ; Move Backwards for a second
298          ldi      mpr, MovBck ; Load Move Backward command
299          out      PORTB, mpr   ; Send command to port
300          ldi      waitcnt, (WTime<<1) ; Shifted bit back by 1,
301                                         ; making the wait time two seconds
302          rcall    Wait             ; Call wait function
303
304          ; Turn left for a second
305          ldi      mpr, TurnL   ; Load Turn Left Command
306          out      PORTB, mpr   ; Send command to port
307          ldi      waitcnt, WTime   ; Wait for 1 second
308          rcall    Wait             ; Call wait function
309
310          ; Move Forward again
311          ldi      mpr, MovFwd ; Load Move Forward command
312          out      PORTB, mpr   ; Send command to port
313
314          pop      mpr       ; Restore program state
315          out      SREG, mpr    ;
316          pop      waitcnt        ; Restore wait register
317          pop      mpr       ; Restore mpr
318
319          inc      hrcnt;
```

```
320            rcall    toLCD;
321            ;fix debounce
322            ldi mpr , 0b0000_0011
323            out EIFR, mpr
324            ret                  ; Return from subroutine
325
326    ;————————————————————————————————————————————————————
327    ; Sub:   HitLeft
328    ; Desc: Handles functionality of the TekBot when the left whisker
329    ;       is triggered.
330    ;————————————————————————————————————————————————————
331    HitLeft:
332            push     mpr              ; Save mpr register
333            push     waitcnt          ; Save wait register
334            in       mpr, SREG        ; Save program state
335            push     mpr              ;
336
337            ; Move Backwards for a second
338            ldi      mpr, MovBck ; Load Move Backward command
339            out      PORTB, mpr  ; Send command to port
340            ldi      waitcnt, (WTime<<1) ; Wait for 1 second
341            rcall    Wait             ; Call wait function
342
343            ; Turn right for a second
344            ldi      mpr, TurnR   ; Load Turn Left Command
345            out      PORTB, mpr   ; Send command to port
346            ldi      waitcnt, WTime  ; Wait for 1 second
347            rcall    Wait             ; Call wait function
348
349            ; Move Forward again
350            ldi      mpr, MovFwd ; Load Move Forward command
351            out      PORTB, mpr   ; Send command to port
352
353            pop      mpr       ; Restore program state
354            out      SREG, mpr    ;
355            pop      waitcnt      ; Restore wait register
356            pop      mpr       ; Restore mpr
357
358            inc      hlcnt    ;
359            rcall    toLCD;
360                     ;fix debounce
361            ldi mpr , 0b0000_0011
362            out EIFR, mpr
363            ret                  ; Return from subroutine
364
365    ;————————————————————————————————————————————————————
```

12

```
366  ; Sub:   Wait
367  ; Desc:  A wait loop that is 16 + 159975*waitcnt cycles or roughly
368  ;        waitcnt*10ms. Just initialize wait for the specific amount
369  ;        of time in 10ms intervals. Here is the general eqaution
370  ;        for the number of clock cycles in the wait loop:
371  ;               (((((3*ilcnt)-1+4)*olcnt)-1+4)*waitcnt)-1+16
372  ;─────────────────────────────────────────────────────────────────
373  Wait:
374          push    waitcnt              ; Save wait register
375          push    ilcnt                ; Save ilcnt register
376          push    olcnt                ; Save olcnt register
377
378  Loop:   ldi     olcnt, 224           ; load olcnt register
379  OLoop:  ldi     ilcnt, 237           ; load ilcnt register
380  ILoop:  dec     ilcnt                ; decrement ilcnt
381          brne    ILoop                ; Continue Inner Loop
382          dec     olcnt             ; decrement olcnt
383          brne    OLoop                ; Continue Outer Loop
384          dec     waitcnt           ; Decrement wait
385          brne    Loop                 ; Continue Wait loop
386
387          pop     olcnt             ; Restore olcnt register
388          pop     ilcnt             ; Restore ilcnt register
389          pop     waitcnt           ; Restore wait register
390          ret                ; Return from subroutine
391
392
393
394  ;─────────────────────────────────────────────────────────────────
395  ; Func:  Template function header
396  ; Desc:  Cut and paste this and fill in the info at the
397  ;        beginning of your functions
398  ;─────────────────────────────────────────────────────────────────
399  FUNC:                                  ; Begin a function with a label
400
401          ; Save variable by pushing them to the stack
402
403          ; Execute the function here
404
405          ; Restore variable by popping them from the stack in reverse order
406
407          ret                            ; End a function with RET
408
409  ;****************************************************************
410  ;*   Stored Program Data
411  ;****************************************************************
```

13

```
412
413   ;  Enter  any  stored  data  you  might  need  here
414   ;.org
415   STRING_BEG:
416   .DB        "HL#:0␣␣␣␣␣␣␣␣␣␣␣"              ;  Declaring  data  in  ProgMem
417   STRING2_BEG:
418   .DB        "HR#:0␣␣␣␣␣␣␣␣␣␣␣"
419   STRING_END:
420
421
422   ;****************************************************************
423   ;*   Additional  Program  Includes
424   ;****************************************************************
425   .include  "LCDDriver.asm"              ;  Include  the  LCD  Driver
```