

main.asm

```
1 .INCLUDEPATH "/usr/share/avra" ;set the include path to the correct place
2 ;.DEVICE ATmega16
3 .NOLIST
4 .INCLUDE "m16def.inc"
5 .LIST
6
7 .def MUL_LOW=R0
8 .def MUL_HIGH=R1
9 .def ZERO=R2
10 .def step=R3
11 .def dstep=R4
12 .def slow=R5
13 .def retReg=R6
14 .def tmp1=R16           ; my code is a bit brittle somewhere
15 .def tmp2=R17           ; it seems like tmp1 and tmp2 have to
16 .def tmp3=R18           ; r16 and r17
17 .def arg1=R19
18 .def arg2=R20
19 .def tasknum=R21
20 .def stepCount=R22
21
22 .CSEG                   ; start the code segment
23 .org 0x000              ; locate code at address $000
24     RJMP  START         ; Jump to the START Label
25 .org INT0addr
26     JMP   INT0_ISR
27 .org INT1addr
28     JMP   INT1_ISR
29 .org URXCaddr
30     JMP   urxc_isr
31 .org UTXCaddr
32     JMP   utxc_isr
33 .org OC0addr
34     JMP   t0_OC_ISR
35 .org OVF2addr
36     JMP   t2_OV_ISR
37 .org ADCCaddr
38     JMP   ADC_ISR
39 .org OC1Aaddr
40     JMP   t1_OCA_ISR
41 .org OVF1addr
42     JMP   t1_OV_ISR
43
44 .org $02A               ; locate code past the
    interrupt vectors
```

```

45
46 START:
47     LDI     tmp1, LOW(RAMEND)
48     OUT     SPL, tmp1
49     LDI     tmp1, HIGH(RAMEND)           ; initialise the stack pointer
50     OUT     SPH, tmp1
51
52     EOR     zero, zero                   ; make zero register, well zero
53
54     ; set task num to zero
55     LDI     tasknum, 0x00
56     LDI     XH, HIGH(TASK_NUM_RAM)
57     LDI     XL, LOW(TASK_NUM_RAM)
58     ST      X, tasknum                   ; make the task number buffer zero
59
60     ; initialize the microcontroller
61     CALL    init_LCD
62     CALL    init_EEP
63     CALL    init_UART
64     CALL    init_stepper
65     CALL    init_IO
66     CALL    init_watchdog
67     SEI
68     CALL    do_task
69 MAIN_LOOP:
70     NOP
71     NOP
72     WDR
73     RJMP    MAIN_LOOP
74
75     ; files to include
76     .include "IO.asm"
77     .include "LCD.asm"
78     .include "UART.asm"
79     .include "EEP.asm"
80     .include "stepperMotor.asm"
81     .include "taskHandler.asm"
82     .include "ADC.asm"
83     .include "watchdog.asm"

```

taskHandler.asm

```

1 ;                               taskHandler.asm
2 ; This file handles all the tasks at a high level
3 ; J L Gouws
4 ; found this at
5 ; https://www.avrfreaks.net/forum/how-do-you-make-jump-table-avr-assembly
6 .DSEG
7 TASK_NUM_RAM: .BYTE 2 ;

```

```

8
9 .MACRO jumpto
10     LDI    XH, HIGH(@0)    ; NB Can't use Z register for IJMP
11     LDI    XL, LOW(@0)
12     MOV    tmp1, @1
13     LSL    tmp1
14     ADD    XL, tmp1
15     ADC    XH, ZERO
16     PUSH   XL
17     PUSH   XH
18     RET
19 .ENDMACRO
20
21 .CSEG
22 do_task:
23     LDI    ZH, HIGH(2 * blankline)
24     LDI    ZL, LOW(2 * blankline)
25     CALL   send_chars_terminal
26     jumpto taskTable, tasknum
27 taskTable:                                ; this is a jimmyied jump table
                                           ; that jumps to the correct task
28
29     RCALL  task0
30     RET
31     RCALL  task1
32     RET
33     RCALL  task2
34     RET
35     RCALL  task3
36     RET
37     RCALL  task4
38     RET
39     RCALL  task5
40     RET
41     RCALL  task6
42     RET
43     RCALL  task7
44     RET
45     RCALL  task8
46     RET
47     RCALL  task9
48     RET
49     RCALL  task10
50     RET
51     RCALL  task11
52     RET
53
54 task0:
55     LDI    ZH, HIGH(2 * menu_text)    ; print out menu
56     LDI    ZL, LOW(2 * menu_text)
57     CALL   send_chars_terminal

```

```

58  RET
59  task1:
60  LDI    tmp1, 1
61  MOV    dstep, tmp1
62  LDI    arg1, 10           ; set number of steps
63  LDI    arg2, 100         ; number of 5ms per step
64  RJMP   do_motor_task
65  task2:
66  LDI    tmp1, -1
67  MOV    dstep, tmp1
68  LDI    arg1, 10           ; set number of steps
69  LDI    arg2, 100         ; number of 5ms per step
70  RJMP   do_motor_task
71  task3:
72  LDI    tmp1, 2
73  MOV    dstep, tmp1
74  LDI    arg1, 80           ; set number of steps
75  LDI    arg2, 25          ; number of 5ms per step
76  RJMP   do_motor_task
77  task4:
78  LDI    tmp1, -2
79  MOV    dstep, tmp1
80  LDI    arg1, 80           ; set number of steps
81  LDI    arg2, 25          ; number of 5ms per step
82  RJMP   do_motor_task
83  task5:
84  LDI    tasknum, 0x00      ; set to task 0
85  CALL    stepper_disable  ; disable the motor
86  RET
87  task6:
88  LDI    tasknum, 0x00      ; set to task 0
89  CALL    stepper_enable   ; disable the motor
90  RET
91  task7:
92  LDI    tasknum, 0x00
93  CALL    convert_voltage  ; start adc conversion
94  RET
95  task8:
96  LDI    tasknum, 0x00
97  CALL    adc_disable      ; disable the LEDs
98  CALL    clear_lights     ; clear LEDs
99  RET
100 task9:
101 LDI    tasknum, 0x00
102 LDI    XH, HIGH(RAMMESSAGE3) ; point X to ram message
103 LDI    XL, LOW(RAMMESSAGE3)
104 CALL    message_to_LCD      ; print message to LCD
105 RET
106 task10:
107 LDI    tasknum, 0x00

```

```

108  CALL  clear_LCD                ; clear the LCD
109  RET
110 task11:
111  NOP
112  NOP
113  RJMP  task11                  ; infinite loop so WD doesn't get reset
114  RET
115
116 do_motor_task:
117  CALL  stepper_is_enabled      ; check if the motor is actually
118                                     ; enabled
119  SBRS  retReg, 0
120  RET
121  CBI   UCSRB, RXEN
122  LDI   ZH, HIGH(2 * blankterminal)
123  LDI   ZL, LOW(2 * blankterminal)
124  CALL  send_chars_terminal     ; clear the terminal
125  CALL  enable_buttons         ; enable the buttons
126  CALL  step_motor
127  RET
128
129
130 adc_done:
131  MOV   arg1, retReg            ; output the lights to
132  CALL  output_lights
133  RET
134
135 stepper_done:
136  LDI   tasknum, 0x00
137  SBI   UCSRB, RXEN            ; Re-enable receiving
138  CALL  disable_buttons        ; disable buttons
139  RCALL do_task
140  RET
141
142 menu_text:      .db 0x0C,"Project Tasks: ",0x0d,0x0a
143 menu_text1:     .db "-----",0x0d,0x0a
144 menu_text2:     .db "1) Rotate clockwise for 5 seconds ",0x0d,0x0a
145 menu_text3:     .db "2) Rotate anti-clockwise for 5 seconds",0x0d,0
146 menu_text4:     .db "3) Rotate clockwise for 10 seconds",0x0d,0x0a
147 menu_text5:     .db "4) Rotate anti-clockwise for 10 seconds ",0x0d,0
148 menu_text6:     .db "5) Disable stepper",0x0d,0x0a
149 menu_text7:     .db "6) Enable stepper ",0x0d,0x0a
150 menu_text8:     .db "7) Start ADC PWM task ",0x0d,0x0a
151 menu_text9:     .db "8) Stop ADC PWM task ",0x0d,0x0a
152 menu_text10:    .db "9) Print 3rd stored message to LCD",0x0d,0x0a
153 menu_text11:    .db "10) Clear LCD ",0x0d,0x0a
154 menu_text12:    .db "11) Reset Microcontroller ",0x0d,0x0a,0x00,0x00
155 blankterminal: .db 0x0C,0x00

```

```
156 blankline: .db 0x0D,"", 0x0D,0x00,0x00
```

IO.asm

```

1 ; IO.asm
2 ; J L Gouws
3 ; File for handling the input and output of microcontroller
4
5 .DSEG
6 lightValue: .BYTE 1
7
8 .CSEG
9 init_IO:
10     IN     tmp1, DDRD
11     ANDI   tmp1, 0xF0
12     OUT    DDRD, tmp1           ; set up the d pins data directions
13     LDI    tmp1, 0x7E
14     OUT    DDRA, tmp1         ; set up the d pins data directions
15     IN     tmp1, PORTD
16     ANDI   tmp1, 0xF0
17     ORI    tmp1, 0x0C
18     OUT    PORTD, tmp1        ; enable pull ups for buttons
19     LDI    tmp1, 0x0A
20     OUT    MCUCR, tmp1        ; both falling edge triggered.
21     IN     tmp1, TIMSK        ; timer 1 for PWM
22     ANDI   tmp1, 0xC3
23     ORI    tmp1, 0x14        ; enable overflow and OCA interrupts for
                                ; timer 1
24     OUT    TIMSK, tmp1
25     RET
26
27 enable_buttons:
28     LDI    tmp1, 0xC0        ; 0b11000000
29     OUT    GICR, tmp1        ; int0 and int1 enabled
30     OUT    GIFR, tmp1        ; clear int0 flag and in1 flags
31     RET
32
33 disable_buttons:
34     OUT    GICR, zero        ; int0 and int1 disabled
35     RET
36
37 output_lights:
38     OUT    TCCR1A, zero
39     LDI    tmp1, 0x01        ; prescalar of 1, 8 makes the timer too
                                ; slow
40     OUT    TCCR1B, tmp1
41     LDI    tmp2, 0x3F        ; multiply the lights to get a nice
                                ; range
42
                                ; of brightness

```

```

43  MUL    tmp2, arg1
44  ADD    MUL_LOW, tmp2
45  ADC    MUL_HIGH, zero
46  OUT    OCR1AH, MUL_HIGH           ; set the output compare value
47  OUT    OCR1AL, MUL_LOW
48  LDI    XH, HIGH(lightValue)      ; save this value to RAM
49  LDI    XL, LOW(lightValue)
50  LSR    arg1
51  ANDI   arg1, 0x7E
52  ST     X, arg1
53  RET
54
55  clear_lights:
56  OUT    TCCR1A, zero               ; stop timer 1 for pulse width
      modulation
57  OUT    TCCR1B, zero
58  IN     tmp1, PORTA                ; turn off lights
59  ANDI   tmp1, 0x81
60  OUT    PORTA, tmp1
61  RET
62
63  int0_ISR:
64  CALL   pause_stepper
65  OUT    GICR, ZERO                 ; disable external interrupts
66  IN     tmp1, TIMSK
67  ANDI   tmp1, 0x3F                 ; mask off TIMSK
68  ORI    tmp1, 0x40                 ; enable overflow interrupts
69  OUT    TIMSK, tmp1
70  LDI    tmp1, 0x05
71  OUT    TCCR2, tmp1                ; start timer2 with prescaler set to
      /1024
72  RETI
73
74  int1_ISR:
75  CALL   start_stepper
76  OUT    GICR, ZERO                 ; disable external interrupts
77  IN     tmp1, TIMSK
78  ANDI   tmp1, 0x3F                 ; mask off TIMSK
79  ORI    tmp1, 0x40                 ; enable overflow interrupts
80  OUT    TIMSK, tmp1
81  LDI    tmp1, 0x05
82  OUT    TCCR2, tmp1                ; start timer2 with prescaler set to
      /1024
83  RETI
84
85  t2_OV_ISR:
86  OUT    TCCR2, zero                ; stop counter
87  OUT    TCNT2, zero                ; zero counter
88  RCALL  enable_buttons
89  RETI

```

```

90
91 t1_OCA_ISR:
92     IN     tmp1, PORTA                ; clear lights on output compare
93     ANDI   tmp1, 0x81
94     OUT    PORTA, tmp1
95     RETI
96
97 t1_OV_ISR:
98     LDI    XH, HIGH(lightValue)      ; get adc readout again
99     LDI    XL, LOW(lightValue)
100     LD      tmp1, X
101     IN      tmp2, PORTA                ; get PORTA values
102     ANDI   tmp2, 0x81                ; mask off values
103     OR      tmp1, tmp2
104     OUT    PORTA, tmp1                ; set the lights
105     RETI

```

stepperMotor.asm

```

1  ;           This file is responsible for handling the stepper motor
2  ; J L Gouws 19G4436
3  ;
4
5 .DSEG
6 RAM_STEPS: .BYTE 9
7
8 .CSEG
9 init_stepper:
10     IN      tmp1, DDRD
11     ANDI   tmp1, 0x0F                ; lower bits of the DDRD
12     LDI    tmp2, 0xF0
13     OR      tmp1, tmp2                ; set D4-D7 to output
14     OUT    DDRD, tmp1
15     IN      tmp1, PORTD
16     ANDI   tmp1, 0x0F                ; lower bits of the PORTD
17     ORI    tmp1, 0x10                ; lock motor on first driver
18     OUT    PORTD, tmp1
19     EOR    step, step                ; set step to 0
20     LDI    ZH, HIGH(2 * STEP_TABLE)
21     LDI    ZL, LOW(2 * STEP_TABLE)
22     CALL   read_steps_to_RAM
23     RCALL  init_timer0
24     RET
25
26 init_timer0:
27     IN      tmp1, TIMSK
28     ANDI   tmp1, 0xFC                ; mask off TIMSK
29     ORI    tmp1, 0x02
30     OUT    TIMSK, tmp1

```



```

31  LDI    tmp1, 156                ; output compare  $0.125 \times 10^{-6} \times 256$ 
      x 156 = 4.992ms
32  OUT    OCR0, tmp1
33  LDI    tmp1, 0x00              ; reset timer0s counter
34  OUT    TCNT0, tmp1
35  RET
36
37  step_motor:
38  MOV     slow, ZERO
39  MOV     stepCount, ZERO
40  RCALL   start_stepper
41  RET
42
43  read_steps_to_RAM:
44  LDI     XH, HIGH(RAM_STEPS)
45  LDI     XL, LOW(RAM_STEPS)
46  readPGM:
47  LPM     tmp1, Z+
48  ST      X+, tmp1
49  CPI     tmp1, 0x00
50  BRNE    readPGM
51  RET
52
53  ; makes the motor make one step
54  NEXTSTEP:
55  ADD     step, dstep              ; get to the next step
56  LDI     tmp1, 0x07
57  AND     step, tmp1
58  INC     stepCount                ; get to the next step
59  LDI     XH, HIGH(RAM_STEPS)
60  LDI     XL, LOW(RAM_STEPS)
61  ADD     XL, step
62  ADC     XH, zero
63  LD      tmp2, X
64  IN      tmp1, PORTD
65  ANDI    tmp1, 0x0F              ; tmp1 now contains the masked off
      values                       ; of portD
66
67  OR      tmp1, tmp2
68  OUT     PORTD, tmp1
69  RET
70
71  t0_OC_ISR:
72  OUT     TCCR0, zero              ; stop timer 0
73  INC     slow
74  CP      slow, arg2
75  BRNE    contSteps
76  MOV     slow, zero
77  RCALL   nextstep
78  CP      stepCount, arg1

```

```

79  BRNE  contSteps
80  LDI   stepCount, 0x00;
81  MOV   slow, zero
82  CALL  stepper_done
83  RETI                                     ; this is done
84  contSteps:
85  RCALL start_stepper
86  RETI
87
88  stepper_disable:
89  IN     tmp1, TIMSK
90  ANDI   tmp1, 0xFC                       ; mask off TIMSK
91  OUT    TIMSK, tmp1
92  IN     tmp1, PORTD
93  ANDI   tmp1, 0x0F                       ; tmp1 now contains the masked off
      values
94                                              ; of portD
95  OUT    PORTD, tmp1                       ; the lower bits of PORTD are now off
96  RET
97
98  stepper_is_enabled:
99  IN     tmp1, TIMSK
100 ANDI   tmp1, 0x03                       ; mask off TIMSK
101 MOV    retReg, tmp1
102 ANDI   tmp1, 0x01                       ; mask off TIMSK
103 LSR    retReg
104 OR     retReg, tmp1
105 RET
106
107 stepper_enable:
108 RCALL  init_timer0
109 LDI    XH, HIGH(RAM_STEPS)
110 LDI    XL, LOW(RAM_STEPS)
111 ADD    XL, step
112 ADC    XH, zero
113 LD     tmp2, X
114 IN     tmp1, PORTD
115 ANDI   tmp1, 0x0F
116 OR     tmp1, tmp2
117 OUT    PORTD, tmp1
118 RET
119
120 pause_stepper:
121 OUT    TCCR0, zero                       ; stop timer 0
122 RET
123
124 start_stepper:
125 OUT    TCNT0, zero
126 LDI    tmp1, 0x04                       ; 0b00000100 | clock prescalar 256
127 OUT    TCCR0, tmp1                       ; stop timer 0

```

```

128  RET
129
130 STEP_TABLE:      .db 0x10, 0x30, 0x20, 0x60, 0x40, 0xC0, 0x80, 0x90, 0
    x00, 0x00

```

UART.asm

```

1  ;                               UART.asm
2  ; J L Gouws
3  ;                               File for handling UART and related things
4
5  init_UART:
6      ;set baud rate (9600,8,n,2)
7      LDI    tmp1, 51
8      LDI    tmp2, 0x00
9      OUT    UBRRH, tmp2                ; Baudrate
10     OUT     UBRRL, tmp1
11     ; set rx and tx enable
12     SBI     UCSRB, RXEN
13     SBI     UCSRB, TXEN
14     ; enable uart interrupts , both transmit and receive
15     SBI     UCSRB, RXCIE
16     SBI     UCSRB, TXCIE
17     RET
18
19 send_chars_terminal:
20     LPM     tmp1, Z+                    ; outputs the characters in program
21                                           ; memmory to terminal
22     OUT     UDR, tmp1
23     RET
24
25 clear_line:                            ; clears a line in the terminal
26     LDI     ZH, HIGH(2 * blankline)    ; this is one method because it gets
27     LDI     ZL, LOW(2 * blankline)     ; called often
28     LPM     tmp1, Z+
29     OUT     UDR, tmp1
30     RET
31
32 wait_transmit:
33     SBIS    UCSRA, TXC                  ; wait for bit data to be sent
34     RJMP    wait_transmit
35     SBI     UCSRA, TXC
36     RET
37
38 URXC_ISR: ; on receive
39     LDI     XH, HIGH(TASK_NUM_RAM)
40     LDI     XL, LOW(TASK_NUM_RAM)
41     IN      tmp1, UDR
42     CPI     tmp1, '.'                  ; check for stop character

```

```

43  BREQ  set_task
44  OUT   UDR, tmp1                ; echo to terminal
45  RCALL wait_transmit
46  LD    tmp3, X                  ; get current stored number
47  LDI   tmp2, 10
48  MUL   tmp3, tmp2              ; multiply tmp3 by tmp2
49  CP    MUL_HIGH, zero
50  BRNE  reset_task             ; bad input this has bad interaction
51                                     ; but whatever
52  MOV   tmp3, MUL_LOW
53  SUBI  tmp1, '0'
54  BRMI  reset_task
55  ADD   tmp3, tmp1
56  ST    X, tmp3
57  RETI
58  set_task:
59  LD    tasknum, X
60  CPI   tasknum, 0x00
61  BRLT  reset_task
62  CPI   tasknum, 0x0C
63  BRGE  reset_task
64  RJMP  do_set_task
65  reset_task:
66  LDI   tasknum, 0x0           ; set task num to zero
67  do_set_task:
68  ST    X, zero               ; clear the input register.
69  CALL  do_task
70  RETI
71
72
73  UTXC_ISR:                      ; continue transmitting
74  LPM   tmp1, Z+
75  CPI   tmp1, 0x00            ; check for byte
76  BREQ  donetx
77  OUT   udr, tmp1
78  donetx:
79  RETI

```

EEP.asm

```

1  ;                               EEP.asm
2  ; This file handles EEPROM
3  ; J L Gouws
4  .ESEG
5  EEMSG: .DB "Why am I here",0x00,"Not this message",0x00,"The message is
        long",0x00
6
7  .DSEG
8  RAMMESSAGE1: .BYTE 20

```

```

9  RAMMESSAGE2: .BYTE 20
10 RAMMESSAGE3: .BYTE 20
11
12 .CSEG
13 ; Reads the EEPROM messages into RAM
14 init_EEP:
15     LDI    XH, HIGH(RAMMESSAGE1)      ; point X to the ram message
16     LDI    XL, LOW(RAMMESSAGE1)
17     LDI    YH, HIGH(EEMSG)            ; set Y to point to the messages in
18                                         ; EEPROM
19     LDI    YL, LOW(EEMSG)
20     CALL   read_EEP                    ; read from EEPROM
21     LDI    XH, HIGH(RAMMESSAGE2)      ; rinse an repeat
22     LDI    XL, LOW(RAMMESSAGE2)
23     CALL   read_EEP
24     LDI    XH, HIGH(RAMMESSAGE3)
25     LDI    XL, LOW(RAMMESSAGE3)
26     CALL   read_EEP
27     RET
28
29 ; Reads an individual message into RAM, stops on null byte
30 ; Y -- Location of first byte in EEPROM to read
31 ; X -- Location to store byte in RAM
32 read_EEP:
33     OUT    EEARH, YH
34     OUT    EEARL, YL
35     SBI    EECR, EERE                    ; read from EEPROM
36     IN     tmp1, EEDR
37     ST     X+, tmp1
38     ADIW   YL, 1
39     CPI    tmp1, 0x00                    ; see if we are at end of message
40     BRNE   read_EEP                      ; read next byte
41     RET

```

ADC.asm

```

1  ;                               ADC.asc
2  ; Handles the ADC.
3  ; J L Gouws
4  convert_voltage:
5     LDI    tmp1, 0b01100000            ; AVCC selected as reference
6     OUT    ADMUX, tmp1                  ; ADLAR set so most significant 8 bits
        are in ADCH
7     LDI    tmp1, 0b11001111            ; ADC enabled conversion started no
        auto trigger
8                                         ; 128 prescaler interrupt enabled
9     OUT    ADCSRA, tmp1
10    RET
11

```

```

12 adc_disable:
13     OUT    ADMUX, zero           ; pretty self explanatory , sets the adc
        to
14     OUT    ADCSRA, zero         ; stop working
15     RET
16
17 ADC_ISR:
18     IN     retReg , ADCH
19     CALL   adc_done
20     RETI

```

LCD.asm

```

1  ;                               LCD file
2  ; Handles the LCD screen
3  ; J L Gouws and Mr. Sullivan
4  .MACRO LCD_WRITE
5      CBI PORTB, 1
6  .ENDMACRO
7  .MACRO LCD_READ
8      SBI PORTB, 1
9  .ENDMACRO
10 .MACRO LCD_E_HI
11     SBI PORTB, 0
12 .ENDMACRO
13 .MACRO LCD_E_LO
14     CBI PORTB, 0
15 .ENDMACRO
16 .MACRO LCD_RS_HI
17     SBI PORTB, 2
18 .ENDMACRO
19 .MACRO LCD_RS_LO
20     CBI PORTB, 2
21 .ENDMACRO
22
23 ;This is a one millisecond delay
24 Delay:
25     PUSH   r16
26     LDI    r16, 11
27 Delayloop1:
28     PUSH   r16
29     LDI    r16, 239           ; for an 8MHz xtal
30 Delayloop2:
31     DEC    r16
32     BRNE   Delayloop2
33     POP    r16
34     DEC    r16
35     BRNE   Delayloop1
36     POP    r16

```

```

37  RET
38  ; waits 800 clock cycles (0.1ms on 8MHz clock)
39  Waittenth:
40  PUSH  r16
41  LDI   r16, 255
42  decloop:
43  DEC   r16
44  NOP
45  NOP
46  BRNE  decloop
47  POP   r16
48  RET
49
50  ; return when the lcd is not busy
51  Check_busy:
52  PUSH  r16
53  LDI   r16, 0b00000000
54  OUT   DDRC, r16 ; portc lines input
55  LCD_RS_LO ; RS lo
56  LCD_READ ; read
57  Loop_Busy:
58  RCALL Delay ; wait 1ms
59  LCD_E_HI ; E hi
60  RCALL Delay
61  IN     r16, PINC ; read portc
62  LCD_E_LO ; make e low
63  SBRC   r16, 7 ; check the busy flag in bit 7
64  RJMP   Loop_busy
65  LCD_WRITE ;
66  LCD_RS_LO ; rs lo
67  POP    r16
68  RET
69
70  ; write char in r16 to LCD
71  Write_char: ;rcall Check_busy
72  PUSH   r17
73  RCALL  Check_busy
74  LCD_WRITE
75  LCD_RS_HI
76  SER    r17
77  OUT    DDRC, r17 ; c output
78  OUT    PORTC, R16
79  LCD_E_HI
80  LCD_E_LO
81  CLR    r17
82  OUT    DDRC, r17
83  POP    r17
84  RET
85  ;write instruction in r16 to LCD
86  Write_instruc:

```

```

87  PUSH  r17
88  RCALL Check_busy
89  LCD_WRITE
90  LCD_RS_LO
91  SER    r17
92  OUT    DDRC, r17                ; c output
93  OUT    PORTC, R16
94  LCD_E_HI
95  LCD_E_LO
96  CLR    r17
97  OUT    DDRC, r17
98  POP    r17
99  RET
100
101
102 Init_LCD:
103  PUSH  r16
104  CLR    r16
105  OUT    DDRC, r16
106  OUT    PORTC, r16
107  SBI    DDRB, 2                ; reg sel output
108  SBI    DDRB, 0                ; enable output
109  SBI    PORTB, 2
110  SBI    PORTB, 0
111  SBI    DDRB, 1                ; rw output
112  LDI    r16, 0x38
113  RCALL Write_instruc
114  LDI    r16, 0x0E                ; turn lcd on with cursor
115  RCALL Write_instruc
116  LDI    r16, 0x06
117  RCALL Write_instruc
118  LDI    r16, 0x01
119  RCALL Write_instruc
120  POP    r16
121  RET
122
123 message_to_LCD:
124  LDI    tmp2, 0x00                ; counter to keep track of chars on
125                                          ; LCD, this works, but is brittle in
126                                          ; general
127 one_char:
128  CPI    tmp2, 16                ; check if we need to line break
129  BRNE   write_one_char
130  LDI    tmp1, 0b11000000        ; instruction for next line
131  CALL   Write_instruc            ; see data sheet
132 write_one_char:
133  LD     tmp1, X+                ; write a char from the address in X
134  CPI    tmp1, 0x00
135  BREQ   m2ldone                ; end on null
136  CALL   write_char

```



```
137     INC    tmp2
138     RJMP   one_char           ; do next chars
139 m2ldone:
140     RET
141
142 clear_LCD:
143     CBI    PORTB, 0
144     CBI    PORTB, 1
145     LDI    R16, 0x01           ; write instruction to clear display
146     RCALL  write_instruc
147     RET
```

watchdog.asm

```
1 init_watchdog:
2     LDI    tmp1, 0x0A           ; Enable watchdog 65ms time out
3     OUT    WDTCR, tmp1
4     RET
```