```
;Authors:     Alejandro Hernandez Sosa          z5178495
;                    Unai Aguilera Lopez                      z5178492
;Date: 25-10-2017

.include "m2560def.inc"

.macro do_lcd_command
       ldi r16, @0
       rcall lcd_command
       rcall lcd_wait
.endmacro
; do_lcd_data and do_lcd_data1 is the same macro, but one uses characters as input
(do_lcd_data1) and the other registers (do_lcd_data)
.macro do_lcd_data
       mov r16, @0
       rcall lcd_data
       rcall lcd_wait
.endmacro
.macro do_lcd_data1
       ldi r16, @0
       rcall lcd_data
       rcall lcd_wait
.endmacro

.macro clear
ldi YL, low(@0) ; load the memory address to Y pointer
ldi YH, high(@0)
clr temp ; set temp to 0
st Y+, temp ; clear the two bytes at @0 in SRAM
st Y, temp
.endmacro

.macro clear_station
ldi YL, low(@0) ; load the memory address to Y pointer
ldi YH, high(@0)
clr temp ; set temp to 0
st Y+, temp ; clear the ten bytes at @0 in SRAM
st Y+, temp
st Y+, temp
st Y+, temp
st Y+, temp
st Y+, temp
st Y+, temp
st Y+, temp
st Y+, temp
st Y, temp
.endmacro

.def maxnumstations=r2 ;Maximum number of stations
.def numletters=r3 ;Number of letters that the station has
.def delay_one = r4 ;Low register of the time to stabilize the keypad
.def counter2=r5 ;Used to count if we have pressed a key more than once
.def flag=r6 ;We use flag for:     1- Is set to 1 when a letter is pressed, to 0
again when a second passes since the last letter was pressed
;                                                      2- In the emulation when flag
is 0 we are between stations, when it is 1 we are in an stop
.def stoptime=r7 ;This is the time the train is has to stop in a station
.def time_next_station=r8 ;Time left for the next station
```

```
.def auxstoptime=r9 ;It is used to restore the stoptime to original value
.def stop_next_station=r10 ;Used to know if a passanger wants to get off or get in
in the next station
.def flag_emergency=r11 ;Used to stop the train at emergency
.def led_state=r12 ;Used to know the state of the leds when blinking
.def countholes=r13 ;Count the holes in the motor
.def revolutions=r14 ;Is used to calculate the revolutions of the motor
.def flag_timer0=r15 ;It is used to actualize the speed of the motor every second
.def temp =r16 ;Temporary register
.def row =r17 ;We also use row as storage of the revolutions in the simulation part
.def col =r18 ;Column of the keypad
.def mask =r19 ;Used to scan the keypad
.def temp2 =r20 ;Temporary register
.def ascii = r21 ;Used to load the ascii value
.def counter = r22 ;Used to know introduced the name for all stations and the time
between all the stations
.def delay_two = r23 ;High register of the time to stabilize the keypad
.def prev=r24 ;It is used:  1-To know if we pressed the same key more than once
;                                                2- In te emulation in order to print
the 10 characters of the station names
.def finalletter=r25 ;Used to store the letter we introduce and set to 0xff if enter
is pressed



.equ PORTLDIR = 0xF0
.equ INITCOLMASK = 0xEF
.equ INITROWMASK = 0x01
.equ ROWMASK = 0x0F

.dseg
TempCounter: .byte 2 ;Two byte number to count the number of cycles until one second
TempCounter2: .byte 2
TempCounter5: .byte 2
Station0: .byte 10
Station1: .byte 10
Station2: .byte 10
Station3: .byte 10
Station4: .byte 10
Station5: .byte 10
Station6: .byte 10
Station7: .byte 10
Station8: .byte 10
Station9: .byte 10
;Store the time between stations
Time0: .byte 1
Time1: .byte 1
Time2: .byte 1
Time3: .byte 1
Time4: .byte 1
Time5: .byte 1
Time6: .byte 1
Time7: .byte 1
Time8: .byte 1
Time9: .byte 1
;Store the dutty cycle after performing feedback
Duttycycle: .byte 1
```

```
.cseg

.org 0x00
jmp RESET


.org INT0addr ; INT0addr is the address of EXT_INT0
jmp EXT_INT0

.org INT1addr ; INT1addr is the address of EXT_INT1
jmp EXT_INT1

.org INT2addr
jmp EXT_INT2

.org OVF2addr
jmp Timer2OVF

.org OVF0addr
jmp Timer0OVF

.org OVF5addr
jmp Timer5OVF

RESET:
ldi temp, low(RAMEND)
out SPL, temp
ldi temp, high(RAMEND)
out SPH, temp
ldi temp, PORTLDIR ; columns are outputs, rows are inputs
STS DDRL, temp     ; cannot use out
ser temp
out DDRF, r16
out DDRA, r16
out DDRC, r16
clr r16
out PORTF, r16
out PORTA, r16

clr temp

out PORTC, temp
out DDRD, temp
ser temp
out PORTD, temp
clr temp
ldi temp, (2 << ISC10) | (2 << ISC00) | (2 << ISC20)
sts EICRA, temp
in temp, EIMSK
ori temp, (1<<INT0) | (1<<INT1) | (1<<INT2)
out EIMSK, temp
do_lcd_command 0b00111000 ; 2x5x7
rcall sleep_5ms
do_lcd_command 0b00111000 ; 2x5x7
rcall sleep_1ms
do_lcd_command 0b00111000 ; 2x5x7
do_lcd_command 0b00111000 ; 2x5x7
do_lcd_command 0b00001000 ; display off?
do_lcd_command 0b00000001 ; clear display
do_lcd_command 0b00000110 ; increment, no display shift
do_lcd_command 0b00001110 ; Cursor on, bar, no blink

;Initialize motor PWM
ldi temp, 0b00010000
out DDRE, temp ; set PL3 (OC5A) as output.
ldi temp, 0x00 ; this value and the operation mode determine the PWM duty cycle
sts OCR3BL, temp
clr temp
sts OCR3BH, temp
ldi temp, (1 << CS30) ; CS50=1: no prescaling
sts TCCR3B, temp
ldi temp, (1<< WGM30)|(1<<COM3B1)
; WGM50=1: phase correct PWM, 8 bits
; COM5A1=1: make OC5A override the normal port functionality of the I/O pin PL3
sts TCCR3A, temp
ldi zl,low(Duttycycle)
ldi zh,high(Duttycycle)
ldi temp2,40
st z,temp2
clr temp2



clear TempCounter ; initialize the temporary counter to 0
ldi temp, 0b00000000
out TCCR0A, temp
ldi temp, 0b00000010
out TCCR0B, temp ; set prescalar value to 8
ldi temp, 0<<TOIE0
sts TIMSK0, temp ; disable Timer0 Overflow Interrupt
sei

clear TempCounter2 ; initialize the temporary counter to 0
ldi temp, 0b00000000
sts TCCR2A, temp
ldi temp, 0b00000010
sts TCCR2B, temp ; set prescalar value to 8
ldi temp, 0<<TOIE2
sts TIMSK2, temp ; disable Timer0 Overflow Interrupt
sei

clear TempCounter5 ; initialize the temporary counter to 0
ldi temp, 0b00000000
sts TCCR5A, temp
ldi temp, 0b00000010
sts TCCR5B, temp ; set prescalar value to 8
ldi temp, 0<<TOIE5
sts TIMSK5, temp ; disable Timer0 Overflow Interrupt
sei

clr counter
ldi ascii, '0'
add counter, ascii ;Counter store the number of the station the user is intrdocuing.
if counter is 0, we are at the first station, if it is 1, we are at the sond
station...etc
clr temp2
```

```
ldi ascii,62

clr flag
clr counter2
clr prev
clr finalletter
clr maxnumstations
clr numletters
clr stoptime
clr time_next_station
clr auxstoptime
clr flag_emergency
clr led_state
clr countholes
clr revolutions
clr flag_timer0

do_lcd_data1 'M'
do_lcd_data1 'a'
do_lcd_data1 'x'
do_lcd_data1 'i'
do_lcd_data1 'm'
do_lcd_data1 'u'
do_lcd_data1 'm'
do_lcd_data1 ' '
do_lcd_data1 'n'
do_lcd_data1 'u'
do_lcd_data1 'm'
do_lcd_data1 'b'
do_lcd_data1 'e'
do_lcd_data1 'r'
do_lcd_command 0xC0
do_lcd_data1 'o'
do_lcd_data1 'f'
do_lcd_data1 ' '
do_lcd_data1 's'
do_lcd_data1 't'
do_lcd_data1 'a'
do_lcd_data1 't'
do_lcd_data1 'i'
do_lcd_data1 'o'
do_lcd_data1 'n'
do_lcd_data1 's'
do_lcd_data1 ':'
do_lcd_data1 ' '

jmp main2 ; main keeps scanning the keypad to find which key is pressed.



Timer0OVF: ; interrupt subroutine to Timer0
       push temp
       in temp, SREG
       push temp ; prologue starts
       push YH ; save all conflicting registers in the prologue
       push YL
       push r25
       push r24 ; prologue ends
```

```
        ; Load the value of the temporary counter
        lds r24, TempCounter
        lds r25, TempCounter+1
        adiw r25:r24, 1 ; increase the temporary counter by one
        cpi r24, low(7812) ; check if (r25:r24) = 7812
        ldi temp, high(7812) ; 7812 = 106/128
        cpc r25, temp
        brne NotSecond

        clr flag ; reset the flag to 0
        ldi temp, 0<<TOIE0 ;
        sts TIMSK0, temp ; disable Timer0 Overflow Interrupt
        ldi temp, 0b00000001
        clear TempCounter ; reset the temporary counter
        ; Load the value of the second counter
        rjmp EndIF
        NotSecond: ; store the new value of the temporary counter
        sts TempCounter, r24
        sts TempCounter+1, r25
        EndIF:
        pop r24 ; epilogue starts
        pop r25 ; restore all conflicting registers from the stack
        pop YL
        pop YH
        pop temp
        out SREG, temp
        pop temp
reti ; return from the interrupt

Timer2OVF: ; interrupt subrutine to Timer2
        push temp
        in temp, SREG
        push temp ; prologue starts
        push YH ; save all conflicting registers in the prologue
        push YL
        push r25
        push r24 ; prologue ends
        ; Load the value of the temporary counter
        lds r24, TempCounter2
        lds r25, TempCounter2+1
        adiw r25:r24, 1 ; increase the temporary counter by one
        cpi r24, low(7812) ; check if (r25:r24) = 7812
        ldi temp, high(7812) ; 7812 = 106/128
        cpc r25, temp
        brne NotSecond2

        clr flag_timer0 ; reset the flag to 0
        clr countholes
        clr temp2
        cp flag,temp2 ;We check if the flag is 0 in order to know if we are between
stations or in one station
        brne decrease_stop_time
        dec time_next_station ;Every time a second has passed we decrease time next
station, this is to print in the LCD the time remaining to the next station
        rjmp end_decreasing_time
        decrease_stop_time: ;If we are in an station we decrease the stop time as we
do with the time_next_station
        dec stoptime
```

```
       clr temp2
       inc temp2
       cp stoptime,temp2
       brge end_decreasing_time ;As long as stoptime is greater than 1, we keep
deccrasing it.
       clr flag
       mov stoptime,auxstoptime ;As we decrease the stoptime, we need to reset it
when it is 0, so we use the value in auxstoptime
       end_decreasing_time:

       clear TempCounter2 ; reset the temporary counter
       ; Load the value of the second counter
       rjmp EndIF2
       NotSecond2: ; store the new value of the temporary counter
       sts TempCounter2, r24
       sts TempCounter2+1, r25
       EndIF2:
       pop r24 ; epilogue starts
       pop r25 ; restore all conflicting registers from the stack
       pop YL
       pop YH
       pop temp
       out SREG, temp
       pop temp
reti ; return from the interrupt

Timer5OVF: ; interrupt subroutine to Timer5
       push temp
       in temp, SREG
       push temp ; prologue starts
       push YH ; save all conflicting registers in the prologue
       push YL
       push r25
       push r24 ; prologue ends
       ; Load the value of the temporary counter
       lds r24, TempCounter5
       lds r25, TempCounter5+1
       adiw r25:r24, 1 ; increase the temporary counter by one
       cpi r24, low(10) ; check if (r25:r24) = 10
       ldi temp, high(10)
       cpc r25, temp
       brne NotThirdSecond
       clr temp2
       cp led_state, temp2
       breq leds_off
       clr temp2
       out PORTC, temp2
       clr led_state
       rjmp end_led_state
       leds_off:
       ldi temp2, 3
       out PORTC, temp2
       mov led_state, temp2

       end_led_state:

       clear TempCounter5 ; reset the temporary counter
       ; Load the value of the second counter
       rjmp EndIF5
       NotThirdSecond: ; store the new value of the temporary counter
       sts TempCounter5, r24
       sts TempCounter5+1, r25
       EndIF5:
       pop r24 ; epilogue starts
       pop r25 ; restore all conflicting registers from the stack
       pop YL
       pop YH
       pop temp
       out SREG, temp
       pop temp
reti ; return from the interrupt

EXT_INT2:

push temp
in temp, SREG
push temp
push r30
push r31


inc countholes
clr temp2
cp flag_timer0,temp2 ;When the flag is 0 (that means a second has passed), we
actualize the value of revolutions
brne not_first_hole
mov row,revolutions ;We move revolutions to row to have a fixed value to print in
the lcd

ldi temp2,60
cp revolutions,temp2 ;If the revolutions are less than 60 we increase the dutty
cycle, if not we drecrease
brlo increase_dutty_cicle
ldi zl,low(Duttycycle)
ldi zh,high(Duttycycle)
ld temp,z
ldi temp2,1
sub temp,temp2
sts OCR3BL,temp
st z,temp
clr temp
rjmp end_changing_dutty

increase_dutty_cicle:
ldi zl,low(Duttycycle)
ldi zh,high(Duttycycle)
ld temp,z
ldi temp2,1
add temp,temp2
sts OCR3BL,temp
st z,temp
clr temp

end_changing_dutty:

clr revolutions
```

```
inc flag_timer0


not_first_hole:
clr temp
ldi temp, 5 ;When countholes is 5, one revolution has ocurred
cp countholes,temp
brne not_the_fifth ;If we branch to not_the_fifth, no revolution has ocurred
inc revolutions
clr countholes
inc countholes
not_the_fifth:

pop r31
pop r30
pop temp
out SREG, temp
pop temp
reti

EXT_INT0:
push temp2
in temp2, SREG
push temp2

ldi temp2, 1
mov stop_next_station,temp2  ;Every time the flag is set to 1 we are stopped in one
station
ldi temp2, 0b00001111
out PORTC,temp2

pop temp2
out SREG, temp
pop temp2
reti




EXT_INT1:
push temp2
in temp2, SREG
push temp2

ldi temp2, 1
mov stop_next_station, temp2  ;Every time the flag is set to 1 we are stopped in one
station
ldi temp2, 0b11110000
out PORTC,temp2

pop temp2
out SREG, temp2
pop temp2
reti
```

```
main3: ;We come here once we have to introduce the stop time of the stations
ldi temp2,0xFF ;Once we arrive to the enter stop time part, we set stoptime to 0xFF so that in the convert_number function we know that we are introducing the stoptime
mov stoptime,temp2
clr temp2
do_lcd_command 0b00000001 ; clear display
do_lcd_data1 'E'
do_lcd_data1 'n'
do_lcd_data1 't'
do_lcd_data1 'e'
do_lcd_data1 'r'
do_lcd_data1 ' '
do_lcd_data1 'S'
do_lcd_data1 't'
do_lcd_data1 'o'
do_lcd_data1 'p'
do_lcd_command 0xC0
do_lcd_data1 't'
do_lcd_data1 'i'
do_lcd_data1 'm'
do_lcd_data1 'e'
do_lcd_data1 ':'
do_lcd_data1 ' '
jmp main

halt: ;Halt will be repeated infinit times

ldi temp2, 1<<TOIE2 ;Initialize Timer2
sts TIMSK2,temp2
clr temp2
cp stop_next_station, temp2
breq not_stop_needed
inc flag
clr stop_next_station
not_stop_needed:

do_lcd_command 0b00000001 ; clear display
stop_loop:
       ldi temp2, 1<<TOIE5 ;Initialize Timer5
       sts TIMSK5,temp2
       ldi temp2,0
       sts OCR3BL,temp2

       check_emergency1:
       ldi temp, 0b10111111
       STS PORTL, temp ; set column to mask value
       ; (sets column 0 off)
       ldi temp, 0xFF ; implement a delay so the
       ; hardware can stabilize
       delay_emergency1:
       dec temp
       brne delay_emergency1
       LDS temp, PINL ; read PORTL. Cannot use in
       cpi temp, 0b10110111 ; check if any rows are grounded
       brne no_emergency1
       ldi temp2, 0<<TOIE2 ;Initialize Timer2
       sts TIMSK2,temp2

       delay_bouncing: ;This is a delay to stabilize the bouncing of the keypad
       clr temp2
       ldi temp,0xFF
       mov delay_one,temp
       ldi delay_two,0x05
       delay_emergency_bouncing1:
       cp temp,temp2
       cpc delay_one,temp2
       cpc delay_two,temp2
       breq end_delay_emergency1

       inc temp2
       sub temp,temp2
       clr temp2
       sbc delay_one,temp2
       sbc delay_two,temp2
       brne delay_emergency_bouncing1
       end_delay_emergency1:

       inc flag_emergency ;We increase the flag every time we press the key #
       ldi temp2,2
       cp flag_emergency, temp2
       brne out_of_reach_check_emergency1
       ldi temp2, 1<<TOIE2 ;Initialize Timer2
       sts TIMSK2,temp2
       clr flag_emergency
       rjmp no_emergency1
       out_of_reach_check_emergency1:
       rjmp check_emergency1
       no_emergency1:

       clr temp2
       cp flag,temp2 ;We check the state of the flag
       breq  out_reach_between_stations

       do_lcd_command 0b00000010
       do_lcd_data1 'N'
       do_lcd_data1 'o'
       do_lcd_data1 'w'
       do_lcd_data1 ':'
       do_lcd_data1 ' '
       do_lcd_data1 ' '
;----------------------------------------------------
       ldi temp2,10 ;We substract 10 to the pointer in order to print again the
station name
       sub zl,temp2
       clr temp2
       sbc zh,temp2

       clr prev ;We use prev to create a loop of 10 iterations (The maximun length
of a station name)

       printing_station1:
       cpi prev,10
       breq end_printing_station1
```

```
                ld temp2,z+
                cpi temp2, 0 ;If it is 0, that means that it it empty (The name has
ended), thus, we print a space
                brne print_station_letter1
                ldi temp2,' '
                print_station_letter1:
                do_lcd_data temp2
                inc prev
                rjmp printing_station1

                out_reach_between_stations: rjmp between_stations
        end_printing_station1:

        do_lcd_command 0xC0
        do_lcd_data1 'W'
        do_lcd_data1 'a'
        do_lcd_data1 'i'
        do_lcd_data1 't'
        do_lcd_data1 ' '
        do_lcd_data1 't'
        do_lcd_data1 'i'
        do_lcd_data1 'm'
        do_lcd_data1 'e'
        do_lcd_data1 ':'
        do_lcd_data1 ' '

        ldi ascii,'0'
        add stoptime,ascii
        do_lcd_data stoptime
        sub stoptime,ascii

        rjmp stop_loop

between_stations:
ldi temp2,40 ;We introduce this dutty cicle every time we change station because
otherwise the motor is not able to start spinning
sts OCR3BL,temp2
ldi temp2, 0<<TOIE5 ;Disable timer5
sts TIMSK5,temp2
clr temp2
out PORTC, temp2
clear TempCounter5
cp counter,maxnumstations  ; counter has reacher maxnumstations we reset it, in
order to start again
brlo next_station
ldi counter,'0'
rjmp between_stations
next_station:
clear TempCounter2
ldi temp, 1<<TOIE2
sts TIMSK2, temp ; enable Timer2 Overflow Interrupt
clr temp

do_lcd_command 0b00000001 ; clear display
do_lcd_data1 'N'
do_lcd_data1 'e'
do_lcd_data1 'x'
do_lcd_data1 't'
```

```
do_lcd_data1 ':'
do_lcd_data1 ' '


rcall station_name ;Station name sets the pointer to the correct location in the
memory (Station in Z, and time and Y), according to the station we are in.

clr prev ;We use prev to create a loop of 10 iterations (The maximun length of a
station name)

printing_station:
cpi prev,10
breq end_printing_station
       ld temp2,z+
       cpi temp2, 0 ;If it is 0, that means that it it empty (The name has ended),
thus, we print a space
       brne print_station_letter
       ldi temp2,' '
       print_station_letter:
       do_lcd_data temp2
       inc prev
       rjmp printing_station
end_printing_station:

ld temp2,y
mov time_next_station,temp2

printing_time: ;We come back every time, until time_next_station reaches 0
check_emergency2:
ldi temp, 0b10111111
STS PORTL, temp ; set column to mask value
; (sets column 2 off)
ldi temp, 0xFF ; implement a delay so the
; hardware can stabilize
delay_emergency2:
dec temp
brne delay_emergency2
LDS temp, PINL ; read PORTL. Cannot use in
cpi temp, 0b10110111 ; check if any rows are grounded
brne no_emergency2
do_lcd_data1 ' '
do_lcd_data1 ' '
do_lcd_data1 ' '
ldi temp2, 0<<TOIE2 ;Disable Timer2
       sts TIMSK2,temp2
       ldi temp2, 1<<TOIE5 ;Initialize Timer5
       sts TIMSK5,temp2
       ldi temp2,0 ;Stop the motor
       sts OCR3BL,temp2

       delay_bouncing2: ;Delay to stabilize the keypad
       clr temp2
       ldi temp,0xFF
       mov delay_one,temp
       ldi delay_two,0x05
       delay_emergency_bouncing2:
       cp temp,temp2
       cpc delay_one,temp2
```

```
        cpc delay_two,temp2
        breq end_delay_emergency2
        inc temp2
        sub temp,temp2
        clr temp2
        sbc delay_one,temp2
        sbc delay_two,temp2
        brne delay_emergency_bouncing2
        end_delay_emergency2:
        inc flag_emergency
        ldi temp2, 2
        cp flag_emergency, temp2
        brne out_of_reach_check_emergency2
        ldi temp2, 1<<TOIE2 ;Initialize Timer2
        sts TIMSK2,temp2
        ldi temp2, 0<<TOIE5 ;Disable Timer5
        sts TIMSK5,temp2
        clr temp2
        out PORTC, temp2
        ldi temp2,40
        sts OCR3BL,temp2
        clr flag_emergency
        rjmp no_emergency2
        out_of_reach_check_emergency2:
        rjmp check_emergency2
        no_emergency2:



do_lcd_command 0xC0
do_lcd_data1 'T'
do_lcd_data1 'i'
do_lcd_data1 'm'
do_lcd_data1 'e'
do_lcd_data1 ':'
do_lcd_data1 ' '


ldi temp2,10
cp time_next_station,temp2 ;If time next station is 10, we have to print to characters in the LCD
brne not_ten
do_lcd_data1 '1'
do_lcd_data1 '0'
do_lcd_data1 ' '
do_lcd_data1 ' '
rcall division
do_lcd_data1 ' '
rjmp printing_time
not_ten:
ldi ascii,'0'
add time_next_station,ascii
do_lcd_data time_next_station
do_lcd_data1 ' '
do_lcd_data1 ' '
do_lcd_data1 ' '
rcall division
do_lcd_data1 ' '
```

```
sub time_next_station,ascii

clr temp2 ;Here we check if time_next_station is 0
cp time_next_station,temp2
breq out_reach_halt
clr temp2
rjmp printing_time

out_reach_halt: rjmp halt

rjmp halt ;When we finish the configuration, we come here

main2:; Every time a new station is entered, main2 must be executed

clr temp
cp maxnumstations, temp ;We create maxnumstations, which contains the maximum number of statiosn. If it is 0, it means that the user has to introduce a number and not letters
brne introduce_letters




jmp main

introduce_letters:
cp counter,maxnumstations ;We keep comparing counter with maximum stations. Once counter is same or higher than maxnumstations, we have introduced allt he statioons needed and we can stop introducing more stations
brlo enter_stations
sub counter,maxnumstations
ldi ascii,'0'
add counter,ascii
cp counter,maxnumstations ; Once we finish entering the names we still use counter to introduce the time between stations. So we keep incresing counter, and substract maxnumstations to get the value of counter as if it was reset but still is higher that maxnumstations in the previous comparation
brlo enter_time_estations
jmp main3
enter_time_estations:
sub counter,ascii
add counter,maxnumstations
rcall time ;Time is used to store the time between the stations in a correct manner
jmp main
enter_stations:

do_lcd_command 0b00000001 ; clear display
do_lcd_data1 'E'
do_lcd_data1 'n'
do_lcd_data1 't'
do_lcd_data1 'e'
do_lcd_data1 'r'
do_lcd_data1 ' '
do_lcd_data1 'S'
do_lcd_data1 't'
do_lcd_data1 'a'
do_lcd_data1 't'
do_lcd_data1 'i'
```

```
do_lcd_data1 'o'
do_lcd_data1 'n'
do_lcd_data1 ' '


do_lcd_data counter
do_lcd_data1 ':'
do_lcd_command 0xC0 ;This is to allow 2 line display in the LCD
clr temp
; In this part of the code, we select the location in the data memory to store the
name of our station
cpi counter, '0'  ;We set the pointers to save the data
brne keep_comparing_0
clear_station Station0
ldi zl, low(Station0)
ldi zh, high(Station0)
keep_comparing_0:
cpi counter, '1'  ;We set the pointers to save the data
brne keep_comparing_1
clear_station Station1
ldi zl, low(Station1)
ldi zh, high(Station1)
keep_comparing_1:
cpi counter, '2'  ;We set the pointers to save the data
brne keep_comparing_2
clear_station Station2
ldi zl, low(Station2)
ldi zh, high(Station2)
keep_comparing_2:
cpi counter, '3'  ;We set the pointers to save the data
brne keep_comparing_3
clear_station Station3
ldi zl, low(Station3)
ldi zh, high(Station3)
keep_comparing_3:
cpi counter, '4'  ;We set the pointers to save the data
brne keep_comparing_4
clear_station Station4
ldi zl, low(Station4)
ldi zh, high(Station4)
keep_comparing_4:
cpi counter, '5'  ;We set the pointers to save the data
brne keep_comparing_5
clear_station Station5
ldi zl, low(Station5)
ldi zh, high(Station5)
keep_comparing_5:
cpi counter, '6'  ;We set the pointers to save the data
brne keep_comparing_6
clear_station Station6
ldi zl, low(Station6)
ldi zh, high(Station6)
keep_comparing_6:
cpi counter, '7'  ;We set the pointers to save the data
brne keep_comparing_7
clear_station Station7
ldi zl, low(Station7)
ldi zh, high(Station7)
```

```
keep_comparing_7:
cpi counter, '8'  ;We set the pointers to save the data
brne keep_comparing_8
clear_station Station8
ldi zl, low(Station8)
ldi zh, high(Station8)
keep_comparing_8:
cpi counter, '9'  ;We set the pointers to save the data
brne keep_comparing_9
clear_station Station9
ldi zl, low(Station9)
ldi zh, high(Station9)
keep_comparing_9:
clr temp



clr temp2
ldi ascii,62
clr flag
clr counter2
clr prev
clr finalletter
clr numletters



main:
ldi mask, INITCOLMASK ; initial column mask
clr col ; initial column
colloop:
STS PORTL, mask ; set column to mask value
; (sets column 0 off)
ldi temp, 0xFF ; implement a delay so the
; hardware can stabilize
delay:
dec temp
brne delay
LDS temp, PINL ; read PORTL. Cannot use in
andi temp, ROWMASK ; read only the row bits
cpi temp, 0xF ; check if any rows are grounded
breq nextcol ; if not go to the next column
ldi mask, INITROWMASK ; initialise row check
clr row ; initial row
rowloop:
mov temp2, temp
and temp2, mask ; check masked bit
brne skipconv ; if the result is non-zero,
; we need to look again
clr temp2
cp maxnumstations,temp2
breq call_numbers
cp counter,maxnumstations
brsh call_numbers
rcall convert_letters ;We have two functions, one for introducing letters with the
keypad and the other for introducing numbers
rjmp check_enter
call_numbers:
```

```
rcall convert_numbers ; if bit is clear, convert the bitcode
clr temp2
cp stoptime,temp2 ;We check stop time in order to know if we are introducing it or
not. If we are introducing it it will be 0xFF
breq check_enter
ldi temp2, 0xFF
cp stoptime,temp2
breq error_stoptime
ldi counter,'0' ;We restart the counter before going to the simulation
ldi zl,low(Station0+10) ;We set the pointer for the first time we enter
ldi zh,high(Station0+10)
ldi temp2, 0<<TOIE0 ;
sts TIMSK0, temp2
clr temp2
jmp halt
error_stoptime:
jmp main3
check_enter:
cpi finalletter, 0xFF ;If finalletters is set, we know that enter has been pressed
and we go back to main2
brne continue_to_main
jmp main2
continue_to_main:
jmp main ; and start again



skipconv:
inc row ; else move to the next row
lsl mask ; shift the mask to the next bit
jmp rowloop
nextcol:
cpi col, 3 ; check if we are on the last column
breq main ; if so, no buttons were pushed,
; so start again.

sec ; else shift the column mask:
; We must set the carry bit
rol mask ; and then rotate left by a bit,
; shifting the carry into
; bit zero. We need this to make
; sure all the rows have
; pull-up resistors
inc col ; increment column value
jmp colloop ; and check the next column
; convert function converts the row and column given to a
; binary number and also outputs the value to PORTC.
; Inputs come from registers row and col and output is in
; temp.


.equ LCD_RS = 7
.equ LCD_E = 6
.equ LCD_RW = 5
.equ LCD_BE = 4

.macro lcd_set
       sbi PORTA, @0
```

```
.endmacro
.macro lcd_clr
       cbi PORTA, @0
.endmacro

;
; Send a command to the LCD (r16)
;

lcd_command:
       out PORTF, r16
       nop
       lcd_set LCD_E
       nop
       nop
       nop
       lcd_clr LCD_E
       nop
       nop
       nop
       ret

lcd_data:
       out PORTF, r16
       lcd_set LCD_RS
       nop
       nop
       nop
       lcd_set LCD_E
       nop
       nop
       nop
       lcd_clr LCD_E
       nop
       nop
       nop
       lcd_clr LCD_RS
       ret

lcd_wait:
       push r16
       clr r16
       out DDRF, r16
       out PORTF, r16
       lcd_set LCD_RW
lcd_wait_loop:
       nop
       lcd_set LCD_E
       nop
       nop
        nop
       in r16, PINF
       lcd_clr LCD_E
       sbrc r16, 7
       rjmp lcd_wait_loop
       lcd_clr LCD_RW
       ser r16
       out DDRF, r16
       pop r16
       ret

.equ F_CPU = 16000000
.equ DELAY_1MS = F_CPU / 4 / 1000 - 4
; 4 cycles per iteration - setup/call-return overhead

sleep_1ms:
       push r24
       push r25
       ldi r25, high(DELAY_1MS)
       ldi r24, low(DELAY_1MS)
delayloop_1ms:
       sbiw r25:r24, 1
       brne delayloop_1ms
       pop r25
       pop r24
       ret

sleep_5ms:
       rcall sleep_1ms
       rcall sleep_1ms
       rcall sleep_1ms
       rcall sleep_1ms
       rcall sleep_1ms
       ret




//----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----

                                                ;CONVERT LETTERS FUNCTION
//----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----

convert_letters:

cpi col, 3 ; if column is 3 we have a letter
breq out_of_reach_letters
cpi row, 3 ; if row is 3 we have a symbol or 0
breq out_of_reach_symbols
mov temp, row ; otherwise we have a number (1-9)
lsl temp ; temp = row * 2
add temp, row ; temp = row * 3
add temp, col ; add the column address
; to get the offset from 1
inc temp ; add 1. Value of switch is
mov temp2, temp
lsl temp2
add temp, temp2
clr temp2
ldi ascii,62
add temp,ascii
; row*3 + col + 1+ '0'
```

```
rjmp keypad
out_of_reach_letters:
rjmp letters
out_of_reach_symbols: ;We do this because symbols is out of reach, so we need to
jump here and then to symbols
rjmp symbols

/////////////////////////Here we program the keypad to display multiple
characters//////////////

keypad:
ldi temp2, 1<<TOIE0 ;
sts TIMSK0, temp2 ;Activate the timer0 for counting up to one second every time a
letter is pressed
mov temp2,temp
clear TempCounter ; initialize the temporary counter to 0
mov temp,temp2

clr temp2
cp flag, temp2
breq newletter ; We go to newletter every time a new key is pressed
cp prev,temp
brne newletter ;If the key pressed is the same, we keep going with the program
ldi temp2,2 ;Once we press the key 3 times, the characters displayed start again
cpi temp, 89 ;If the letter pressed was a Y, we only have Y and Z in that key, so we
restart the characters after pressing twice
brne threeletters
ldi temp2,1
threeletters:
do_lcd_command 0x10 ; Here, if the user pressed a key more than once, we move the
cursor to the left by one
cp counter2, temp2
brne increase_final_letter
sub finalletter,counter2
clr counter2
clr temp2

rjmp end_letter_comparison

increase_final_letter: ;This is to follow the alphabet if we press one key more than
once
inc finalletter
inc counter2
rjmp end_letter_comparison

newletter:
clr counter2
clr temp2
cp prev, temp2
breq notsave ;Here we save the presious letter pressed. The first time we press any
key, we dont have anything to save, so we skip this step
inc numletters
ldi temp2,10
cp numletters, temp2 ;We store in numletters the number of letters that a station
name has. When the number of letters reaches 10, we go directly to enter.
ldi row, 1
breq enter
st z, finalletter
```

```
ld temp2,z+

notsave:
clr finalletter
mov finalletter,temp
mov prev,temp ;Here we move temp to prev to store the prevous letter

end_letter_comparison:
inc flag ;The flag is so that we know if the user wants to represent a B for
example, or 2 A's. We set the flag to 1 every time we press a letter, and to 0 when
a certain amount of time has passed.
jmp convert_end


letters:


cpi row, 0
brne enter
inc numletters
st z, finalletter
ld temp2, z+
clr finalletter
ldi temp2, ' ' ;Letter A represents a space
add finalletter, temp2

ldi temp2,10
cp numletters, temp2
ldi row,1
breq enter


jmp convert_end


enter:;When the user presses the key B, we know that he has finished entering the
name of the station
cpi row, 1
brne zero ; If the row pressed is 2 or 3, the user pressed C or D--> error
st z, finalletter
ld temp2,z
;Here we end entering the name
inc counter
ldi finalletter,0xFF
jmp prueba


symbols:

zero:
do_lcd_command 0b00000001 ; clear display

do_lcd_data1 'I'
do_lcd_data1 'n'
do_lcd_data1 'c'
do_lcd_data1 'o'
do_lcd_data1 'r'
do_lcd_data1 'r'
```

```
do_lcd_data1 'e'
do_lcd_data1 'c'
do_lcd_data1 't'
do_lcd_data1 '!'
do_lcd_command 0xC0
do_lcd_data1 'T'
do_lcd_data1 'r'
do_lcd_data1 'y'
do_lcd_data1 ' '
do_lcd_data1 'a'
do_lcd_data1 'g'
do_lcd_data1 'a'
do_lcd_data1 'i'
do_lcd_data1 'n'
do_lcd_data1 ':'
do_lcd_data1 ' '

ser finalletter ;We set finalletter to 0xFF because the error erases all the
characters and we have to introduce a new word again
ser delay_two
mov delay_one, delay_two
ldi delay_two, 0x1F
rjmp delay1

convert_end:
do_lcd_data finalletter

prueba:
clr temp2
ldi temp,0xFF
mov delay_one,temp
ldi delay_two,0x05
delay1:
cp temp,temp2
cpc delay_one,temp2
cpc delay_two,temp2
breq end_delay1
inc temp2
sub temp,temp2
clr temp2
sbc delay_one,temp2
sbc delay_two,temp2
brne delay1
end_delay1:


ret ; return to caller


//----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----

                                                ;CONVERT LETTERS END
//----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----
```

```
//----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----

                                                ;CONVERT NUMBERS
//----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----


convert_numbers:
cpi col, 3 ; if column is 3 we have a letter
breq letters_n
cpi row, 3 ; if row is 3 we have a symbol or 0
breq symbols_n
mov temp, row ; otherwise we have a number (1-9)
lsl temp ; temp = row * 2
add temp, row ; temp = row * 3
add temp, col ; add the column address
; to get the offset from 1
inc temp ; add 1. Value of switch is

clr temp2
cp stoptime,temp2 ;Once we go to main 3 stop time is 0xFF, but before is always 0
breq not_introducing_stoptime

cpi temp,2 ;We compare the pressed key with 2, 3, 4, 5 in order to see if it is
correct
breq correct_stoptime
cpi temp,3
breq correct_stoptime
cpi temp,4
breq correct_stoptime
cpi temp,5
breq correct_stoptime
jmp letters_n

correct_stoptime:
mov stoptime,temp
mov auxstoptime,stoptime
ldi ascii, '0'
add temp,ascii
jmp convert_end_numbers

symbols_n:
rjmp symbols_nn

not_introducing_stoptime:
clr temp2
cp maxnumstations,temp2 ;If maxnumstations is 0, we know that is time to introduce
the station numbers, if not, we know that we are introducing the time between
stations
brne insert_time
ldi ascii, '0'
add temp,ascii
mov maxnumstations,temp
```

```
jmp convert_end_numbers
insert_time:
st z+,temp ;We store the time
inc counter
ldi ascii, '0'
add temp,ascii
jmp convert_end_numbers
letters_n:
do_lcd_command 0b00000001 ; clear display
do_lcd_data1 'I'
do_lcd_data1 'n'
do_lcd_data1 'c'
do_lcd_data1 'o'
do_lcd_data1 'r'
do_lcd_data1 'r'
do_lcd_data1 'e'
do_lcd_data1 'c'
do_lcd_data1 't'
do_lcd_data1 '!'
do_lcd_command 0xC0
do_lcd_data1 'T'
do_lcd_data1 'r'
do_lcd_data1 'y'
do_lcd_data1 ' '
do_lcd_data1 'a'
do_lcd_data1 'g'
do_lcd_data1 'a'
do_lcd_data1 'i'
do_lcd_data1 'n'
do_lcd_data1 ':'
do_lcd_data1 ' '

ser delay_two ;We insert a delay so that the user has time to read the error
mov delay_one, delay_two
ldi delay_two, 0x1F
rjmp delay1_n

jmp convert_end_numbers
symbols_nn:
cpi col, 1 ; or if we have zero
brne zero_n
clr temp ; set to zero
ldi temp, 10

clr temp2
cp maxnumstations,temp2 ; If maxnumstations is equal to zero, that means that we are in the part of the code where we have to introduce the number of stations, so we dont want to save it in the memory
breq insert_time_1
st z+,temp
inc counter
rjmp insert_time_0
insert_time_1:
mov maxnumstations, temp
ldi ascii,'0'
add maxnumstations, ascii
insert_time_0:
clr temp
```

```
do_lcd_data1 '1'
clr temp
add temp,ascii
jmp convert_end_numbers

zero_n:

do_lcd_command 0b00000001 ; clear display
do_lcd_data1 'I'
do_lcd_data1 'n'
do_lcd_data1 'c'
do_lcd_data1 'o'
do_lcd_data1 'r'
do_lcd_data1 'r'
do_lcd_data1 'e'
do_lcd_data1 'c'
do_lcd_data1 't'
do_lcd_data1 '!'
do_lcd_command 0xC0
do_lcd_data1 'T'
do_lcd_data1 'r'
do_lcd_data1 'y'
do_lcd_data1 ' '
do_lcd_data1 'a'
do_lcd_data1 'g'
do_lcd_data1 'a'
do_lcd_data1 'i'
do_lcd_data1 'n'
do_lcd_data1 ':'
do_lcd_data1 ' '

ser delay_two
mov delay_one, delay_two
ldi delay_two, 0x1F
rjmp delay1_n

convert_end_numbers:

do_lcd_data temp

prueba_n:
clr temp2
ldi temp,0xFF
mov delay_one,temp
ldi delay_two,0x05
delay1_n:
cp temp,temp2
cpc delay_one,temp2
cpc delay_two,temp2
breq end_delay1_n
inc temp2
sub temp,temp2
clr temp2
sbc delay_one,temp2
sbc delay_two,temp2
brne delay1_n
end_delay1_n:
```

```
ser finalletter
ret ; return to caller





//----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----

                                                ;CONVERT NUMBERS END
//----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----

//----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----

                                                ;TIME
//----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----
time:

do_lcd_command 0b00000001 ; clear display
;do_lcd_command 0xC0 ;This is to allow 2 line display in the LCD
;do_lcd_command 0b00000001 ; clear display

do_lcd_data1 'E'
do_lcd_data1 'n'
do_lcd_data1 't'
do_lcd_data1 'e'
do_lcd_data1 'r'
do_lcd_data1 ' '
do_lcd_data1 't'
do_lcd_data1 'i'
do_lcd_data1 'm'
do_lcd_data1 'e'
do_lcd_data1 ' '
sub counter,maxnumstations
ldi ascii,'0'
add counter,ascii
do_lcd_data counter
do_lcd_data1 ':'
do_lcd_command 0xC0 ;This is to allow 2 line display in the LCD
clr temp
; In this part of the code, we select the location in the data memory to store the
name of our station
cpi counter, '0'  ;We set the pointers to save the data
brne keep_comparing_00
ldi zl, low(Time0)
ldi zh, high(Time0)
keep_comparing_00:
cpi counter, '1'  ;We set the pointers to save the data
brne keep_comparing_11
ldi zl, low(Time1)
ldi zh, high(Time1)
keep_comparing_11:
cpi counter, '2'  ;We set the pointers to save the data
brne keep_comparing_22
ldi zl, low(Time2)
ldi zh, high(Time2)
keep_comparing_22:
cpi counter, '3'  ;We set the pointers to save the data
brne keep_comparing_33
ldi zl, low(Time3)
ldi zh, high(Time3)
keep_comparing_33:
cpi counter, '4'  ;We set the pointers to save the data
brne keep_comparing_44
ldi zl, low(Time4)
ldi zh, high(Time4)
keep_comparing_44:
cpi counter, '5'  ;We set the pointers to save the data
brne keep_comparing_55
ldi zl, low(Time5)
ldi zh, high(Time5)
keep_comparing_55:
cpi counter, '6'  ;We set the pointers to save the data
brne keep_comparing_66
ldi zl, low(Time6)
ldi zh, high(Time6)
keep_comparing_66:
cpi counter, '7'  ;We set the pointers to save the data
brne keep_comparing_77
ldi zl, low(Time7)
ldi zh, high(Time7)
keep_comparing_77:
cpi counter, '8'  ;We set the pointers to save the data
brne keep_comparing_88
ldi zl, low(Time8)
ldi zh, high(Time8)
keep_comparing_88:
cpi counter, '9'  ;We set the pointers to save the data
brne keep_comparing_99
ldi zl, low(Time9)
ldi zh, high(Time9)
keep_comparing_99:
sub counter,ascii
add counter,maxnumstations

ret

//----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----

                                                ;TIME END
//----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----
```

```
//---------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----

                                                ;STATION NAME
//---------------------------------------------------------------------------------
-----------------------------------------------------------------------------------
-----
station_name:

cpi counter, '0'  ;We set the pointers to save the data
brne keep_comparing_000
ldi zl, low(Station1)
ldi zh, high(Station1)
ldi yl, low (Time0)
ldi yh, high(Time0)
keep_comparing_000:
cpi counter, '1'  ;We set the pointers to save the data
brne keep_comparing_111
ldi zl, low(Station2)
ldi zh, high(Station2)
ldi yl, low (Time1)
ldi yh, high(Time1)
keep_comparing_111:
cpi counter, '2'  ;We set the pointers to save the data
brne keep_comparing_222
ldi zl, low(Station3)
ldi zh, high(Station3)
ldi yl, low (Time2)
ldi yh, high(Time2)
keep_comparing_222:
cpi counter, '3'  ;We set the pointers to save the data
brne keep_comparing_333
ldi zl, low(Station4)
ldi zh, high(Station4)
ldi yl, low (Time3)
ldi yh, high(Time3)
keep_comparing_333:
cpi counter, '4'  ;We set the pointers to save the data
brne keep_comparing_444
ldi zl, low(Station5)
ldi zh, high(Station5)
ldi yl, low (Time4)
ldi yh, high(Time4)
keep_comparing_444:
cpi counter, '5'  ;We set the pointers to save the data
brne keep_comparing_555
ldi zl, low(Station6)
ldi zh, high(Station6)
ldi yl, low (Time5)
ldi yh, high(Time5)
keep_comparing_555:
cpi counter, '6'  ;We set the pointers to save the data
brne keep_comparing_666
ldi zl, low(Station7)
ldi zh, high(Station7)
ldi yl, low (Time6)
ldi yh, high(Time6)
```

```
keep_comparing_666:
cpi counter, '7'  ;We set the pointers to save the data
brne keep_comparing_777
ldi zl, low(Station8)
ldi zh, high(Station8)
ldi yl, low (Time7)
ldi yh, high(Time7)
keep_comparing_777:
cpi counter, '8'  ;We set the pointers to save the data
brne keep_comparing_888
ldi zl, low(Station9)
ldi zh, high(Station9)
ldi yl, low (Time8)
ldi yh, high(Time8)
keep_comparing_888:
cpi counter, '9'  ;We set the pointers to save the data
brne keep_comparing_999
ldi zl, low(Station0)
ldi zh, high(Station0)
ldi yl, low (Time9)
ldi yh, high(Time9)
keep_comparing_999:
clr temp
inc counter
cp counter,maxnumstations
brlo not_the_last_station
ldi zl, low(Station0)
ldi zh, high(Station0)
not_the_last_station:

ret
//----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----

                                                ;STATION NAME END
//----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----

//----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----

                                                ;DIVISION
//----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----

division:
       ;our row will be our dividend
       ;r19 will be our quotient
       ;r13 will be our divisor
       ;r18 will be our bit number

       ;Prologue
       push r16
       push row
push r19
push r13
push r18
push r21

;body
clr r16
clr r13
clr r18
clr r19
clr r21

inc r18
ldi r21,10
mov r13,r21
clr r21

cp row,r13
brlo not_more_division

for2:
	cp r13,row
	brsh end_loop2

	ldi r21,0x80
	cp r13,r21
	brsh end_loop2

	lsl r13

	lsl r18
	rjmp for2

end_loop2:

for3:
	clr r21
	cp r21,r18
	brsh end_loop3

	cp row,r13
	brlo end_if1
	sub row,r13
	add r19,r18
	end_if1:

	lsr r13

	lsr r18
	rjmp for3

end_loop3:

mov r21,row
mov row,r19


rcall division
```

```
       ldi r16,48
       add r16,r21
       do_lcd_data r16
       rjmp epilogue

not_more_division:
       ldi r16,48
       add r16,row
       do_lcd_data r16

       ;epilogue
       epilogue:
       pop r21
       pop r18
       pop r13
       pop r19
       pop row
       pop r16
ret

//----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----

                                                ;DIVISION END
//----------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-----
```