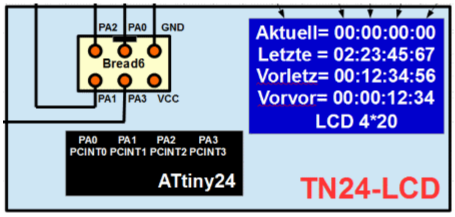
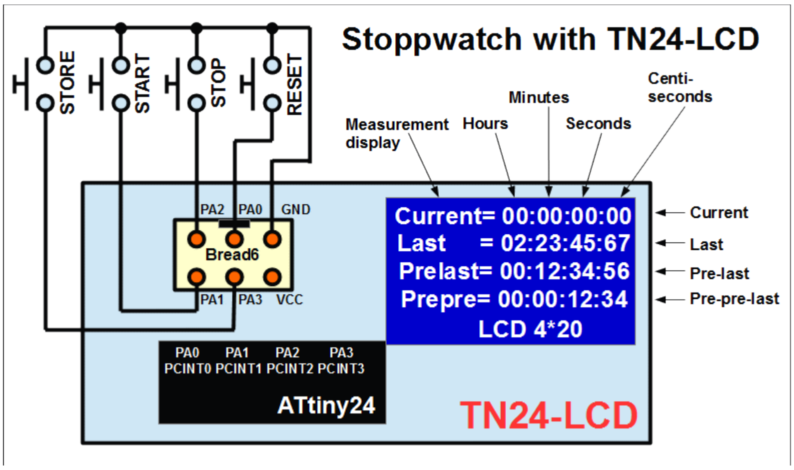
Path: Home => AVR-EN => Applications => Stop watches with AVRs => Stopwatch ATtiny24



AVR Applications

Stopwatch with ATtiny24 in Assembler

Stopwatch with an ATtiny24



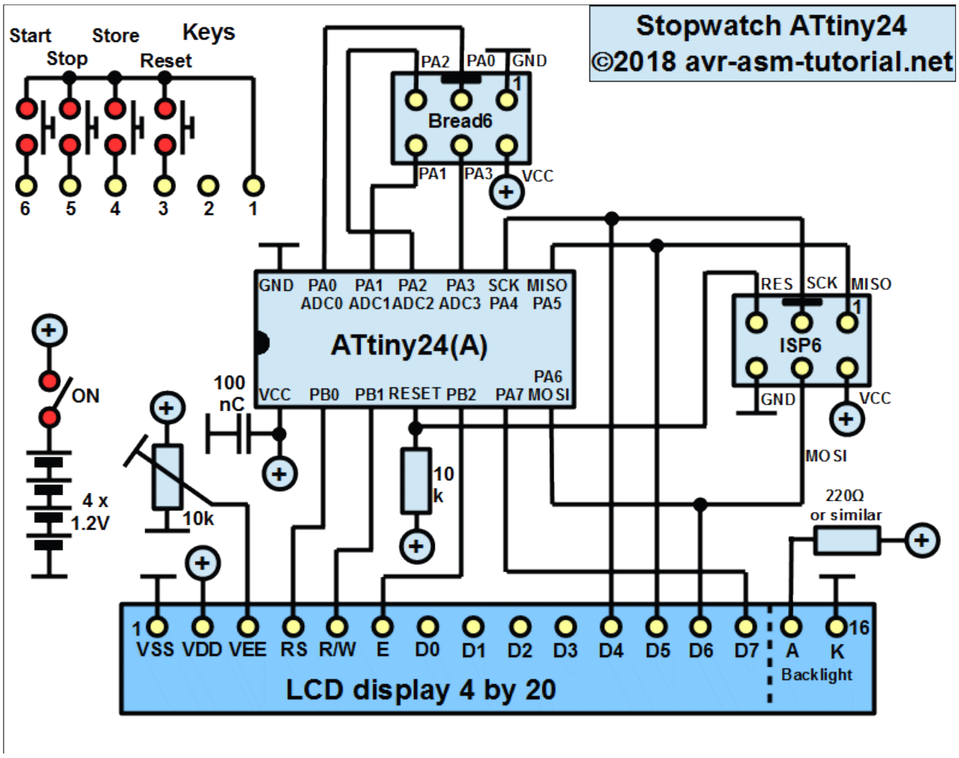
Here a stopwatch with the following properties is described:

- internal RC clock with 10% accuracy, can be adjusted to higher accuracy, no external components necessary,
- four line LCD to display runtime and three stored times,
- four keys (Reset, Start, Stop, Store),
- adjustable key toggling suppression,
- works with the ATtiny24-LCD standard PCB at tn24_lcd and the respective LCD include file (slightly modified),
- Resolution of the timer display 1 ms.

1 Hardware

As the hardware is based on the existing standard PCB tn24_lcd the additional four keys are simply attached to the six-pin bread connector.

Power supply for that all comes either from three 1.5 V batteries or from four rechargeable batteries of 1.2 V. That allows to work with a 5 V standard LCD.



2 Software structure

The software can be programmed in two versions:

1. a 10 ms version: Time measurements are based on a 10 ms cycle, with the time in four registers,
2. a 1 ms version: Measurement in a 1 ms cycle, with the centiseconds in one register (from 0 to 99) and the milliseconds (from 0 to 9) in an additional register.

The version described here is the second version with milliseconds, even though the resolution of 1 ms is a little bit inappropriate, given the large inaccuracy of the internal RC oscillator.

The following properties determine the software design:

- Clocking: setting of the internal clock prescaler CLKPR from its default of 8 to 2 to yield 4 MHz controller clock from the internal 8 MHz RC oscillator by software,
- 1 ms clock: by dividing the controller clock in the time/counter 1 prescaler by 8 and by Clear-Timer-on-compare (CTC) by 500, with Compare A set to 499, and compare match A interrupt,
- Keys: Reading and processing of the keys in each millisecond, with toggle protection following each key event recognition,
- Time measurement: If the stopwatch is running, each compare match interrupt increases the time in ms in the five registers and displays all times on line 1 of the LCD, if necessary (only the registers that were changed during increase are displayed on the LCD),
- Storage: Transfer of the current time in the registers to a storage space in SRAM, shifting of all stored times and display updates for lines 2 to 4 of the LCD.

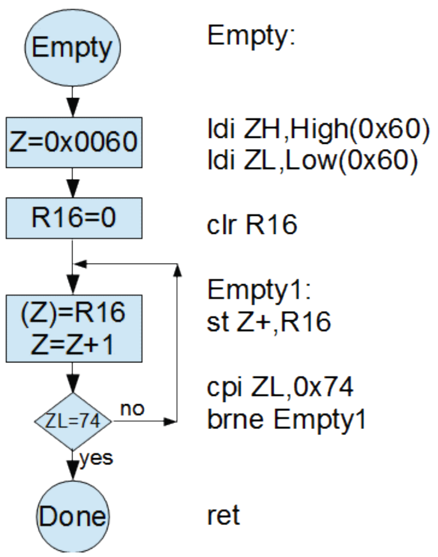
2.1 Clearing the SRAM storage place

At start-up and when resetting the stopwatch the whole storage area in SRAM has to be cleared, which means that zeros have to be written there. As this part is executed two times, it is formulated as a subroutine, so it can be called from different addresses.

The routine uses the register pair ZH:ZL as pointer to SRAM. It is set to the beginning of the storage space.

Clearing is done in a loop that writes, with the AVR instruction *st Z+,R16* the content of the register (zero) to the storage byte addressed in register pair ZH:ZL and automically increases the address in Z by one. If the LSB of the Z pointer reaches the end of the SRAM area to be cleared, the routine stops.

Clearing SRAM storage space



```
Empty:

ldi ZH,High(0x60)
ldi ZL,Low(0x60)

clr R16

Empty1:
st Z+,R16

cpi ZL,0x74
brne Empty1

ret
```

2.2 Shifting values in the storage space

In case the *Store* key has been pressed the

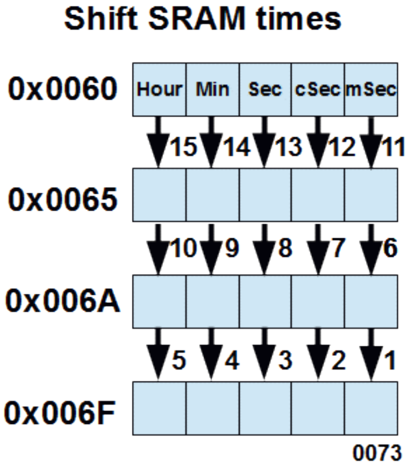
1. current time, located in five registers, has to be copied into the SRAM,
2. all stored times have to be shifted by five bytes to upper addresses, by that overwriting the last five bytes (done with the subroutine *Shift*), and
3. those shifted times have to be displayed in the lines 2 to 4 on the LCD

2.2.1 Shifting of time information

Here is the storage space in SRAM with all addresses in hexadecimal format.

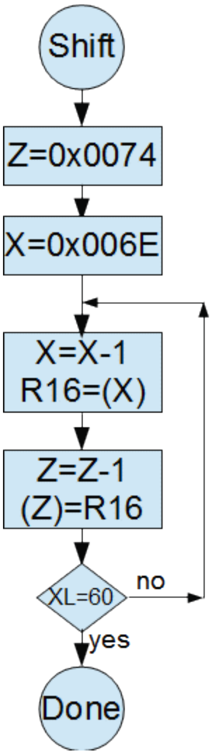
Shifting has to start from the end of the storage space (by overwriting the last time stored) and has to move backwards to lower addresses. The byte transfer is from a source address (at the five byte lower address) to a target address. So two pointers are required that differ by five.

The two pointers are XH:XL or X for the source address and ZH:ZL or Z for the target address.



The flow utilizes a specialty of the AVR instruction set: with *LD R16,-X* the pointer X is first decremented and after that copies the content of the SRAM at this already decremented address to the register. Similarly *ST -Z,R16* first decrements the address in Z and after that writes the register's content to the already decremented location address in SRAM. This makes it easy to read and write backwards, but you'll have to be aware that both pointers, at the beginning, have to point one position higher than were they first transfer from/to.

This operation has been programmed as a subroutine, too, even though it is performed only once (when the Store key is pressed). This eases simulation and debugging.

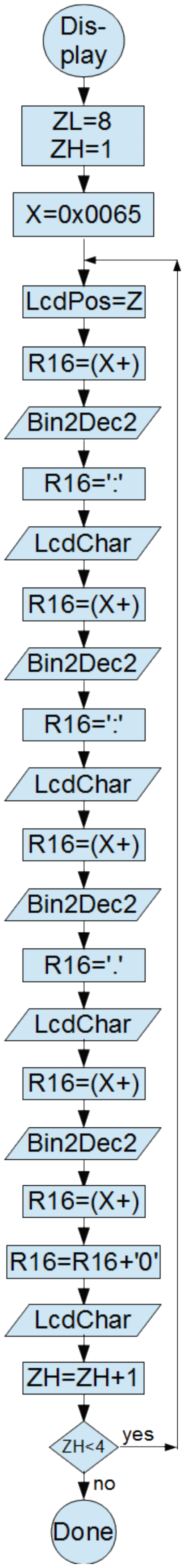


```
Shift:

ldi ZH,High(0x74)
ldi ZL,Low(0x74)

ldi XH,High(0x6E)
ldi XL,Low(0x6E)

Shift1:
ld R16,-X

st -Z,R16

cpi XL,0x60
brne Shift1

ret
```

2.2.2 Displaying the shifted values

When displaying the three lines on LCD the following goes on:

1. The line counter ZH starts with 1 (on line 2) and is increased for each displayed line (if it reaches four, the display is done), as this register is altered by other called subroutines it is pushed to the stack and later restored (not shown here). The column address ZL of the LCD points to the tens of the hour to be displayed. With that positioning is done for each line to be displayed.
2. The SRAM address of the time to be displayed is held in X. Reading one byte of information increases this address (*LD rmp,X+* with subsequent auto-incrementing) and therefore, after five bytes have been read, is already on the address of the next line's time information.
3. Displaying the time advances from the hour to the minutes, then to seconds, centiseconds and milliseconds. Except for the last, a routine named Bin2Dec2 is used to convert the byte to a 2-digit decimal and display those on the LCD. In between hours and minutes and between minutes and seconds a colon is added. Following the seconds a decimal point is added.



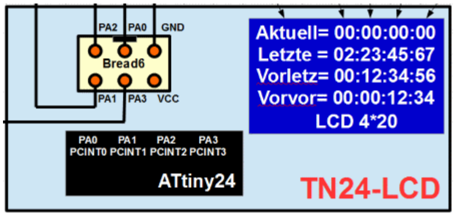
```
Display:
ldi ZL,8
ldi ZH,1
ldi XH,High(0x65)
ldi XL,Low(0x65)
Display1:
rcall LcdPos
ld R16,X+
rcall Bin2Dec2
ldi R16,':'
rcall LcdChar
ld R16,X+
rcall Bin2Dec2
ldi R16,':'
rcall LcdChar
ld R16,X+
rcall Bin2Dec2
ldi R16,'.'
rcall LcdChar
ld R16,X+
rcall Bin2Dec2
ld R16,X+
subi R16,-'0'
rcall LcdChar
inc ZH
cpi ZH,4
brcs Display1
ret
```

3 Software

The software has been written by Jochen Girschik (translation by me) and is completely in assembler. The source code can be viewed in the attachment and downloaded from here in asm format. Assembling requires that the modified LCD include file is stored within the same path.

Praise, error reports, scolding and spam please via the comment page to me.

Path: Home => AVR-EN => Applications => Stop watches with AVRs => Stopwatch ATtiny24 => Source Code



AVR Applications

Stopwatch with ATtiny24
Assembler source code

Assembler source code for the ATtiny24 stopwatch

The source code in asm format can be downloaded from here and requires the modified include file Lcd4Busy_mod.inc that can be downloaded from here.

```
;
; *******************************************
;  Stopwatch with LCD and Textlines Vers.04
;  Date: 25.07.2018 / 20:22
;  (C)2018 by Jochen Girschik
; *******************************************
;
     .nolist
     .include "tn24adef.inc"
     .list
;
;    1000 Hz clock (1ms) generation:
;    Internal RC Oscillator clock 8 MHz
;      divided with clock prescaler = 2 to
;      effective controller Clock 4 MHz
;    Timer/Counter1 as millisecond generator:
;      Timer/Counter prescaler = 8 yields 500.000 Hz
;      Clear timer on Compare at 500-1 (CTC)
;      Interrupt at Compare Match A
;
;******************
;*   Constants    *
;******************
;
        .equ Clock      =  4000000         ; clock frequency
        .equ cstopclock =  1               ; clock stopwatch in ms
        .equ cToggle    =  255/cstopclock  ; Number of CTC cycles,
                                           ; Toggle suppression
;
;****************
;*   REGISTER   *
;****************
;       Register R0 used by lcd include
;               Registers R1 - R7 free
;
        .def rmSec      =  R8   ; Milliseconds
        .def rToggle    =  R9   ; Toggle counter
        .def rKey       =  R10  ; Key value
        .def rcSec      =  R11  ; Centiseconds
        .def rSec       =  R12  ; Seconds
        .def rMin       =  R13  ; Minutes
        .def rHour      =  R14  ; Hours
        .def rSreg      =  R15  ; Save/Restore SREG
        .def rmp        =  R16  ; Multi purpose register
        .def rFlag      =  R17  ; Flag register
            .equ bKey   =  0    ; Key flag
            .equ bToggle=  1    ; Toggle time flag
            .equ b1ms   =  2    ; 1 ms over
            .equ bRun   =  3    ; Stopwatch runs
        .def rimp       =  R18  ; Multi purpose register in interrupts
        .def rmo        =  R19  ; Multi purpose register LCD
        .def rLine      =  R20  ; Line counter LCD
        .def rRead      =  R21  ; Read register LCD
;
;       Registers R22 - R25 free
;       Registers R27:R26 used as XH:XL (in LCD and in Shift)
;       Registers R29:R28 free
;       Registers R31:R30 used as ZH:ZL for multiple purposes
;
;
;*******************************
;*     Define data segment     *
;*******************************
;
     .DSEG
        .ORG SRAM_START
;
     Times:
        .byte 20
;
;
;
;*************************************
;*   Reset- and Interrupt vectors    *
;*************************************
     .CSEG
     .ORG 0
        rjmp START              ; Reset vector
        reti ; EXT_INT0
        reti ; PCI0
        reti ; PCI1
        reti ; WATCHDOG
        reti ; ICP1
        RJMP INT_COMPA          ; OC1A
        reti ; OC1B
        reti ; OVF1
        reti ; OC0A
        reti ; OC0B
        reti ; OVF0
        reti ; ACI
        reti ; ADCC
        reti ; ERDY
        reti ; USI_STR
        reti ; USI_OVF
;
;*********************************
;*   Interrupt Service Routine   *
;*********************************
;
     INT_COMPA:
        IN rSreg, SREG          ; Save Status
        SBRS rFlag, bToggle     ; bToggle set? if yes jump over next instruction
        RJMP KeyPress           ; No, jump to key press
        IN rimp, PINA           ; Read key status
        ANDI rimp, 0x0F         ; Clear upper bits
        CPI rimp, 0x0F          ; Any key pressed?
        BREQ NoKey              ; No key pressed, jump
        LDI rimp, cToggle       ; Load toggle constant
        MOV rToggle, rimp       ; Write to rToggle
        RJMP MsFlag             ; Done, move to MsFlag
;
     NoKey:
        DEC rToggle              ; Count toggle counter down
        BRNE MsFlag              ; Not yet zero
        CBR rFlag,1<<bToggle     ; Clear bToggle flag (toggle time over)
        RJMP MsFlag              ; Done, move to MsFlag
;
     KeyPress:
        IN rimp, PINA           ; Read key state
        ANDI rimp, 0x0F         ; Clear upper bits
        CPI rimp, 0x0F          ; No key pressed?
        BREQ MsFlag             ; Yes, move to MsFlag
        MOV rKey, rimp          ; Copy current key status
        SBR rFlag,(1<<bKey)|(1<<bToggle)    ; Set flags
        LDI rimp, cToggle        ; Load toggle constant
        MOV rToggle, rimp        ; Copy to rToggle
;
     MsFlag:
        SBR rFlag, 1<< b1ms     ; Set b1ms flag
        OUT SREG, rSreg         ; Restore status
        RETI                    ; Return from interrupt
;
;**************************
;*   Main program init    *
;**************************
;
     START:
;
;    Increase clock frequency
;
        LDI rmp,1<<CLKPCE       ; Controller clock setting with CLKPR
                                ; Enable with CLKPCE set (other bits cleared)
        OUT CLKPR, rmp          ; Write to CLKPR schreiben
;
        LDI rmp, 1<<CLKPS0      ; Set the CLKPS0 bit (Prescaler = 2)
                                ; (CLKPCE cleared)
        OUT CLKPR, rmp          ; Write to port CLKPR
;
     .ifdef SPH                 ; For ATtiny44/84
        LDI rmp, HIGH(RAMEND)   ; Init MSB stack pointer
        OUT SPH, rmp
     .endif
        LDI rmp, LOW(RAMEND)    ; Init LSB stack pointer
        OUT SPL, rmp
;
;
;    Init ports for LCD
;
        LDI rmp,(1<<bLcdCRE)|(1<<bLcdCRRS)|(1<<bLcdCRRW)
;
        OUT pLcdCR,rmp          ; Direction of control ports
        CLR rmp                 ; Clear outputs
        OUT pLcdCO,rmp          ; to output port
        LDI rmp,mLcdDRW         ; Data port output mask
        OUT pLcdDR,rmp          ; Write to direction port
;
        RCALL LcdInit           ; Call subroutine LCDInit
        RCALL LcdCurs0          ; LcdCurs0 aufrufen, disable cursor
;
;
;    Write Text lines to LCD
```

```
;
;
        LDI ZH, HIGH(2*TextTable)   ; Lade text table
        LDI ZL, LOW(2*TextTable)    ; Load text table
        RCALL LcdText                 ; Call subroutine LcdText
;
;    Init time and output
;
        RCALL Empty             ; Call subroutine Unterprogramm SRAM empty
        RCALL LcdOut_Hour       ; Call subroutine LcdOut_Hour to display
;
;    Init the keys
;
        IN rmp, DDRA            ; Read direction port
        ANDI rmp, 0xF0          ; Clear lower nibble
        OUT DDRA, rmp           ; Write to direction register Port A
        IN rmp, PORTA           ; Read output port A to rmp
        ORI rmp, 0x0F           ; Set lower nibble (Pull-up resistors on)
        OUT PORTA, rmp          ; To output port A
;
;
;    Millisecond clock generation
;
        LDI rmp, High(500-1)    ; MSB CTC value
        OUT OCR1AH, rmp         ; MSB to output compare port 1A
        LDI rmp, Low(500-1)     ; LSB CTC value
        OUT OCR1AL, rmp         ; LSB to output compare port 1A
;
;    CTC mode and prescaler to 8
;
        LDI rmp, (1<<WGM12)|(1<<CS11)  ; WGM12 = CTC mode, CS11=Prescaler = 8
        OUT TCCR1B, rmp         ; To Timer/Counter1 Control Register B
;
        LDI rmp, 1<<OCIE1A      ; Interrupt on timer compare match
        OUT TIMSK1, rmp         ; To Timer/Counter 1 Interrupt Mask
;
        LDI rmp, 1<<SE          ; Set Sleep Enable
        OUT MCUCR, rmp          ; In MCU Control Register
;
        SEI                     ; Global Interrupt Enable in SREG
;
;**************************************
;*   Loop == Start the program loop   *
;**************************************
;
     Loop:
        SLEEP
        NOP
        SBRC rFlag, b1ms        ; b1ms flag set? If not skip next instruction
        RCALL FlagTime          ; Yes, handle flag
        SBRC rFlag, bKey        ; Flag bKey set? If not skip next instruction
        RCALL FlagKey           ; Yes, handle flag
        SBRC rFlag, b1ms        ; Flag b1ms now set? If not skip next
instruction
        RCALL FlagTime          ; Yes, handle flag again
        RJMP Loop               ;
;
;************************
;*   FlagTime set       *
;************************
;
     FlagTime:
        CBR rFlag, 1<<b1ms      ; Clear fFlag
        SBRS rFlag, bRun        ; Is the sopwatch running? If yes, skip next
        RET                     ; Jump back
;
;    MilliSek
        INC rmSec               ; Increase milliseconds
        LDI rmp, 10             ; Load max value
        CP rmSec, rmp           ; Compare with 10
        BRNE LcdOut_Milli       ; If not equal jump to LCD display milliseconds
;
;    Centi_Sek
        CLR rmSec               ; Restart milliseconds
        INC rcSec               ; Increase centiseconds
        LDI rmp, 100            ; Load compare value (100)
        CP rcSec, rmp           ; Compare with 100
        BRNE LcdOut_Centi       ; If not equal jump to display centiseconds
;
;    Sekunden
        CLR rcSec               ; Clear centiseconds
        INC rSec                ; Increase seconds
        LDI rmp, 60             ; Load compare value (60)
        CP rSec, rmp            ; Compare with 60
        BRNE LcdOut_Sek         ; If not equal jump to display seconds
;
;    Minuten
        CLR rSec                ; Clear seconds
        INC rMin                ; Increase minutes
        CP rMin, rmp            ; Compare with 60
        BRNE LcdOut_Min         ; If not equal jump to display minutes
;
;    Stunden
        CLR rMin                ; Clear minutes
        INC rHour               ; Increase hours
        LDI rmp, 24             ; Load compare value (24)
        CP rHour, rmp           ; Compare with 24
        BRNE LcdOut_Hour        ; If not equal jump to display hours
;
        CLR rHour               ; Clear hours
;
;    The positions of times
;
;    Line 1 is    "Current=00:00:00.000"
;    Position LCD "01234567890123456789"
;
     LcdOut_Hour:               ; Display hours
        LDI ZH, 0               ; on line 1
        LDI ZL, 8               ; Position starts at column 8
        RCALL LcdPos            ; Call LcdPos(ition)
        MOV rmp, rHour          ; Copy rHour to rmp
        RCALL Bin2Dec2          ; Subroutine binary to
;                               ; 2 decimal digits conversion
        LDI rmp, ':'            ; Colon to rmp
        RCALL LcdChar           ; and display
;
     LcdOut_Min:                ; Display minutes
        LDI ZH, 0               ; in line 1
        LDI ZL, 11              ; Position starts at column 12
        RCALL LcdPos            ; Call subroutine LcdPos
        MOV rmp, rMin           ; Copy minutes to rmp
        RCALL Bin2Dec2          ; Subroutine binary to
;                               ; 2 decimal digits conversion
        LDI rmp, ':'            ; Colon to rmp
        RCALL LcdChar           ; and display
;
     LcdOut_Sek:                ; Display seconds
        LDI ZH, 0               ; in line 1
        LDI ZL, 14              ; Position starts at column 15
        RCALL LcdPos            ; Call subroutine LcdPos
        MOV rmp, rSec           ; Copy seconds to rmp
        RCALL Bin2Dec2          ; Subroutine binary to
;                               ; 2 decimal digits conversion
        LDI rmp, ','            ; Colon to rmp
        RCALL LcdChar           ; and display
;
     LcdOut_Centi:              ; Display centiseconds
        LDI ZH, 0               ; in line 1
        LDI ZL, 17              ; Position starts at column 18
        RCALL LcdPos            ; Call subroutine LcdPos
        MOV rmp, rcSec          ; Copy centiseconds to rmp
        RCALL Bin2Dec2          ; Call binary-to-2-digit-decimal
;
     LcdOut_Milli:              ; Display milliseconds
        LDI ZH, 0               ; in line 1
        LDI ZL, 19              ; Position starts at column 20
        RCALL LcdPos            ; Call subroutine LcdPos
        MOV rmp, rmSec          ; Copy milliseconds to rmp
        SUBI rmp, -'0'          ; Add ASCII-zero
        RCALL LcdChar           ; Display character
;
        RET                     ; Return from subroutine
;
;*************************
;*   FlagKey set         *
;*************************
;
     FlagKey:
        CBR rFlag, 1<<bKey      ; Clear flag
        LSR rKey                ; Bit 0 of key input to carry
        BRCS Flag1              ; Carry not clear, jump to Flag1
        ; The reset button has been pushed
        CBR rFlag, 1<<bRun      ; Clear bit bRun in rFlag
                                ; Time increase off
        CLR rHour               ; Clear rHour
        CLR rMin                ; Clear rMin
        CLR rSec                ; Clear rSec
        CLR rcSec               ; Clear rcSec
        CLR rmSec               ; Clear rmSec
        RJMP LcdOut_Hour        ; Display hours and all other time info
;
      Flag1:
        LSR rKey                ; Bit 1 of key input to Carry
        BRCS Flag2              ; Carry not zero, jump to Flag2
        ; Start button has been pushed
        SBR rFlag, 1<<bRun      ; Set bRun flag active
        RET                     ; Done, back to Loop
;
     Flag2:
        LSR rKey                ; Bit 2 of key input to Carry
        BRCS Flag3              ; Carry not zero, jump to Flag3
        ; Stop button has been pressed
        CBR rFlag, 1<<bRun      ; Clear bRun flag
        RET                     ; Done, back to Loop
;
     Flag3:
        LSR rKey                ; Bit 3 of key input to Carry
        BRCS Flag3Ret           ; Carry not zero, jump to RET
;
;    Store time at reserved space in SRAM
;
        STS Times, rHour        ; Store rHour to SRAM 0x0060
```

```
         STS Times+1, rMin       ; Store rMin to SRAM 0x0061
         STS Times+2, rSec       ; Store rSec to SRAM 0x0062
         STS Times+3, rcSec      ; Store rcSec to SRAM 0x0063
         STS Times+4, rmSec      ; Store rmSec to SRAM 0x0064
         RCALL Shift             ; Call subroutine Shift
         RJMP Display            ; Jump to display all times
;
      Flag3Ret:                  ; Return from subroutine
         RET                     ; Back to RCALL +1
;
;
;**********************
;*   Subroutines      *
;**********************
;
;    Convert binary in register rmp
;    to 2 decimal digits and display on LCD
;
      Bin2Dec2:
         MOV ZH, rmp             ; Copy rmp to ZH
         LDI rmp, '0'-1          ; Load ASCII-0 minus 1 (=0x29) to reg rmp
;
      Bin2Dec2A:
         INC rmp                 ; Increase ASCII-number
         SUBI ZH, 10             ; Subtract 10 from ZH
         BRCC Bin2Dec2A          ; Repeat as long as Carry is clear
;
         RCALL LcdChar           ; Display ASCII character in rmp (first digit)
         LDI rmp, '0'+10         ; Load Ascii-0 plus 10 to register rmp
         ADD rmp, ZH             ; Add ZH to register content in rmp
         RJMP LcdChar            ; Display ASCII character in rmp (second digit)
;
;
;    Text lines on LCD as table
;
      TextTable:
         .db "Current=", 0x0D,0xFF       ; Line 1
         .db "Last   =", 0x0D,0xFF       ; Line 2
         .db "Prelast=", 0x0D,0xFF       ; Line 3
         .db "PrePre ="                  ; Line 4
         .db 0xFE, 0xFE                  ; End of text
;
;
;    Clear SRAM data in 0x0060 to 0x0074
;
      Empty:                          ; Subroutine Empty
         LDI ZH, HIGH(Times)          ; Load ZH with MSB address SRAM(0x0060)
         LDI ZL, LOW(Times)           ; Load ZL with LSB address SRAM(0x0060)
;
      Empty1:                         ;
         CLR R16                      ; Clear register rmp
         ST Z+, R16                   ; Store rmp (=0) in SRAM
                                      ; Address Z with Auto-Increment
         CPI ZL, LOW(Times+20)        ; Compare ZL with Times+20
                                      ; (0x0060 + 0x0014 = 0x0074)
         BRNE Empty1                  ; If not equal: repeat
         RET                          ; Back to calling address +1
;
;
;    Shift SRAM storage
;      Shifts all times in storage by 5 positions
;      Works backwards from higher to lower addresses
;
      Shift:                          ; Subroutine Shift
```

```
        LDI ZH, HIGH(Times+20)       ; Load MSB target address ZH with 0x0074
        LDI ZL, LOW(Times+20)        ; Load LSB target address ZL with 0x0074
        LDI XH, HIGH(Times+15)       ; Load MSB source address XH with 0x006F
        LDI XL, LOW(Times+15)        ; Load LSB source address XL with 0x006F
;
     Shift1:                         ;
        LD R16, -X                   ; Auto-dec and load rmp with source byte
        ST -Z, R16                   ; Auto-dec and store rmp to target address
        CPI XL, LOW(Times)           ; Compare source XL with Times (0x0060)
        BRNE Shift1                  ; If not equal, Then repeat
        RET                          ; Back to caller address +1
;
;
;    Display displays all three times on LCD
;
     Display:                        ; Subroutine Display
        LDI ZH, 1                    ; Load ZH with 1, start at line 2 LCD
        LDI ZL, 8                    ; Load ZL with 8, column position 9 LCD
        LDI XH, HIGH(Times+5)        ; Load XH with MSB SRAM address 0x0065
        LDI XL, LOW(Times+5)         ; Load XL with LSB
;
     Display1:                       ;
        RCALL LcdPos                 ; Call subroutine LcdPos for positioning
;
        PUSH ZH                      ; Push line to stack
        LD R16, X+                   ; Load SRAM byte from address and inc addr
        RCALL Bin2Dec2               ; Convert and display two digit decimal
;
        LDI R16, ':'                 ; Load colon to rmp
        RCALL LcdChar                ; rmp to display
        LD R16, X+                   ; Load next byte from SRAM and inc address
        RCALL Bin2Dec2               ; Convert and display two digit decimal
;
        LDI R16, ':'                 ; Load colon to rmp
        RCALL LcdChar                ; rmp to display
        LD R16, X+                   ; Load next byte from SRAM and inc address
        RCALL Bin2Dec2               ; Convert and display two digit decimal
;
        LDI R16, '.'                 ; Load decimal dot to rmp
        RCALL LcdChar                ; rmp to display
        LD R16, X+                   ; Load next byte from SRAM and inc address
        RCALL Bin2Dec2               ; Convert and display two digit decimal
;
        LD rmp, X+                   ; Load next byte from SRAM and inc address
        SUBI rmp, -'0'               ; Convert to ASCII (add ASCII-0)
        RCALL LcdChar                ; rmp to display
;
        POP ZH                       ; Restore line from stack
        INC ZH                       ; Next line
;
        CPI ZH, 4                    ; Already at line 5?
        BRCS Display1                ; Repeat if not
;
        RET                          ; Back to caller address +1
;
;
;***************************************
;*   Include the 4 bit LCD routines    *
;***************************************
;
     .include "Lcd4Busy_mod.inc"
;
```