Øving 6 løsningsforslag

1. Løsningsalternativ A:Tabell ligger i SRAM

```
seg7:
        ldi xl,low(tabell)
        ldi xh,high(tabell)
        clr r22
        add xl,r16
        adc xh,r22
        ld r17,x
        ret
```

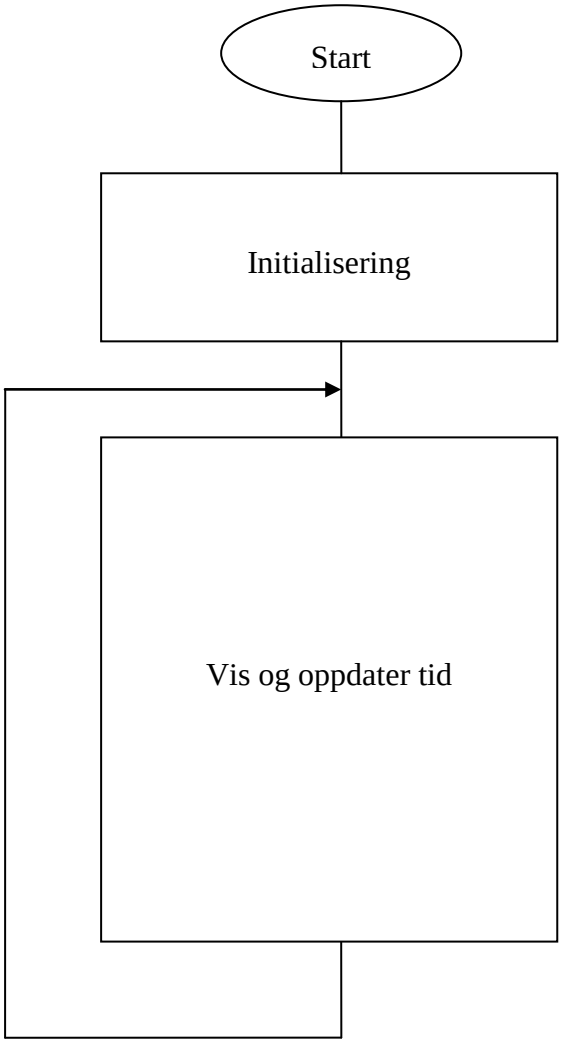
Løsningsalternativ B: Tabellen ligger i flash-minnet

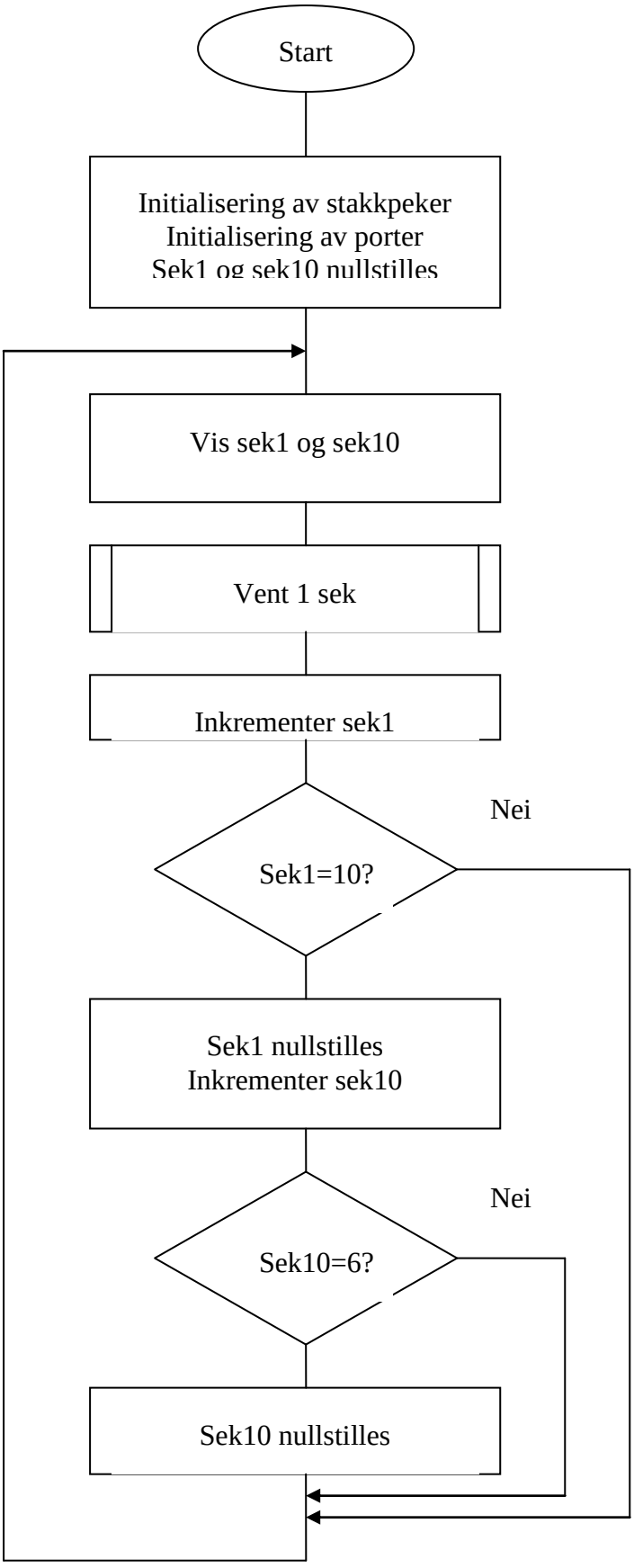
```
seg7:
        ldi zl,low(tabell*2)
        ldi zh,high(tabell*2)
        clr r22
        add zl,r16
        adc zh,r22
        lpm r17,z
        ret
```

2. Flytskjema, se neste side.

3. Det vil være naturlig å legge tabellen i Flash-minnet, men det er fullt mulig å bygge opp en tabell i SRAM som vist i nedenfor. Vi må da skrive instruksjoner som legger riktig innhold i de ulike SRAM adressene. Når må vi så bruke SRAM? Tenk på en situasjon hvor vi mottar data fra omverdenen og disse lagres i SRAM. Vi skal lese ut femte data som vi mottok. Da må vi kjenne til hvordan vi leser i SRAM. Legg også merke til hvordan en sikrer riktig innhold i x- og z-register i prosedyrene ovenfor (se på bruken av r22).

Grunnleggende styreprogram-struktur




Start
Initialisering av stakkpeker
Initialisering av porter
Sek1 og sek10 nullstilles
Vis sek1 og sek10
Vent 1 sek
Inkrementer sek1
Sek1=10?
Nei
Sek1 nullstilles
Inkrementer sek10
Sek10=6?
Nei
Sek10 nullstilles

;Alternativ A: SRAM inneholder tabell

```
.nolist
.include "m128def.inc"
.list
.def sek1=r20
.def sek10=r21
.def kode=r17
.def tall=r16
.def ventetid=r18
.def temp=r19
.org 0
                rjmp start
.org 0x46
start:  ldi temp,low(ramend)
                out spl,temp
                ldi temp,high(ramend)
                out sph,temp
                ldi temp,0x7e
                sts tabell,temp
                ldi temp,0x30
                sts tabell+1,temp
                ldi temp,0x6d
                sts tabell+2,temp
                ldi temp,0x79
                sts tabell+3,temp
                ldi temp,0x33
                sts tabell+4,temp
                ldi temp,0x5b
                sts tabell+5,temp
                ldi temp,0x5f
                sts tabell+6,temp
                ldi temp,0x70
                sts tabell+7,temp
                ldi temp,0x7f
                sts tabell+8,temp
                ldi temp,0x7b
                sts tabell+9,temp
                ldi temp,0xff
                out ddrb,temp
                out ddrc,temp
                clr sek1
                clr sek10
vistid:         mov tall,sek1
                rcall seg7
                out portb,kode
                mov tall,sek10
                rcall seg7
                out portc,kode
                ldi ventetid,1
                rcall vent
```

```
                inc sek1
                cpi sek1,10
                brne slutt1
                clr sek1
                inc sek10
                cpi sek10,6
                brne slutt2
                clr sek10
slutt2:
slutt1:
                rjmp vistid

seg7:
                ldi xl,low(tabell)
                ldi xh,high(tabell)
                clr r22
                add xl,tall
                adc xh,r22
                ld kode,x
                ret
vent:           ret     ; lager ikke kode for vent

.dseg
.org 0x100
tabell:  .byte 10
```

;Alternativ B: Tabellen ligger i FLASH-minnet

```
.nolist
.include "m128def.inc"
.list
.def sek1=r20
.def sek10=r21
.def kode=r17
.def tall=r16
.def ventetid=r18
.def temp=r19
.org 0
                rjmp start
.org 0x46
start:          ldi temp,low(ramend)
                out spl,temp
                ldi temp,high(ramend)
                out sph,temp
                ldi temp,0xff
                out ddrb,temp
                out ddrc,temp
                clr sek1
```

```
                clr sek10
vistid:         mov tall,sek1
                rcall seg7
                out portb,kode
                mov tall,sek10
                rcall seg7
                out portc,kode
                ldi ventetid,1
                rcall vent
                inc sek1
                cpi sek1,10
                brne slutt1
                clr sek1
                inc sek10
                cpi sek10,6
                brne slutt2
                clr sek10
slutt2:
slutt1:
                rjmp vistid
        seg7:
                ldi zl,low(tabell*2)
                ldi zh,high(tabell*2)
                clr r22
                add zl,r16
                adc zh,r22
                lpm r17,z
                ret

vent:           ret             ;lager ikke kode for vent

Tabell:         .db 0x7e,0x30,0x6d,0x79,0x33,0x5b,0x5f,0x70,0x7f,0x7b
```

Init Assembly!

```
    .equ F_osc = 4000000 #klokkefrekvens
    .equ tick = F_osc/1024 #klokkefrekvens med prescalar

    .cseg
    init:

#init stack
    .def tmp = R16 #definer tmp til register 16

    ldi tmp, low(ramend)
    out spl, tmp
    ldi tmp, high(ramend)
    out sph, tmp

#init port
    ldi tmp, 0xFF #alle bit settes til 1, PORTA som output (high, alle bit til 1)
    out DDRA, tmp
    ldi tmp, 0x0 #alle bit settes til 0, PORTB som input (low, alle bit til 0)
    out DDRB, tmp
    ldi tmp, 0xFF
    ldi tmp, &(0x02) #bit 1 settes til 0 (low), kontroll bit satt til input
    #(rest av bits er output)
    out DDRC, tmp

#init timer

#timer compare
    ldi tmp,low(tick)
    out OCR1Bl, tmp
    ldi tmp, high(tick)
    out OCR1BH, tmp

#timer counter
    ldi tmp, (1 << WGM12) | (1 << CS12) | (1 << CS10) #WGM12 = Clear on timer (CTC)
    #alternativ: ldi tmp, (1 << ICES1) #ICES1 = Input capture. Trengs | (OR) med prescaler b
    #CS12 + CS10 = 1024 prescaler bits
    #CS11 + CS10 = 64 prescaler bits
    out TCCRIB, tmp

#clear timer counter
    clr tmp
    out TCNT1L,tmp
```

```
    out TCNT1H,tmp

#enable interrupts on compare
    ldi tmp, (1 << OCIE1B) #Output compare enable
    #alternativ: ldi tmp, (1 << TICIE1) #Timer input capture enable
    out TIMSK, tmp

#enable interrupts
    sei

    .org OCR1Baddr #adresse 18 - hopp til ISR
    jmp ISR

    ISR:
    push tmp #push tmp til stack
    in tmp,sreg #hent fra sreg til tmp
    push tmp #push tmp til stack (sreg)
    ...
    ...
    pop tmp #hent tmp fra stack
    out sreg, tmp #send tmp til sreg
    pop tmp #hent tmp fra stack
    reti #return from interrupt
```

Vår 2014 eksamen

```
    .equ F_osc = 15 000 000
    .equ tick = F_osc/256

    .def tmp = r16
    .def rc = r17
    .def rd = r18
    .dseg
    .org 0x100

    sekunder: .byte 1 ;0-59
    minutter: .byte 1 ; 0-99

    .cseg
    .org 0x00
    rjmp init
    #mellomrom fra 0 og 46 finnes interrupt vektors
    .org 0x46 #her kan koden starte, viktig!
    init:

#init porter
    ldi tmp, 0xFF #setter PORTC, og PORTD som output
    out DDRC, tmp
    out DDRD, tmp

    clr tmp
    out DDRA, tmp #setter PORTA som input
    out PORTA, tmp #clear PORTA, B, og C
    out PORTC, tmp
    out PORTD, tmp

#init stack

    ldi tmp, low(ramend)
    out spl, tmp
    ldi tmp, high(ramend)
    out sph, tmp

    call init_timer #call subroutine, denne pusher PC til stack
    #forventer at init_timer har "ret", for return

    sei
```

```
#subrutiner

    nullstill:
    clr tmp
    sts sekunder, tmp #store direct to dataspace/sram (sts)
    sts minutter, tmp
    rjmp loop #jump back to loop

    min:
    clr rd #clear rd
    lds rd, minutter #hent minutter fra sram

    dec:
    cpi rd, 0x0A #compare rd with 10
    brlt vis #break if compare result is lower* rd<10
    inc rc #inkrementer rc
    sub rd, 0x0A #subtract rd with 10
    rjmp dec #jump back to dec, and loop

    vis:
    out PORTC, rc
    OUT PORTD, rd

    loop:
    sbis PINA, 0x00 #while bit 0 in PINA is 1, skip next step
    call nullstill #if bit 0 in PINA is 0, jump to nullstill
    sbis PINA, 0x07 #while bit 7 in PINA is 1, skip next step
    rjmp min #if bit 7 in PINA is 0, jump to min

    sek:
    clr rd
    clr rc
    lds rd, sekunder #hent sekunder fra sram
    rjmp dec
```

ATmega128 Timers with interrupts in C

Set up tick like this (This is with a prescaler of 64, duh):

```
#define tick F_osc/64
```

OCR1A - Used like:

```
TIMSK = (1 << OCIE1A)
```

To send an interrupt when timer 1 matches the value of OCR1A (Set to the tick you want).
The interrupt vector for this is

```
ISR(TIMER1_COMPA_vect)
```

Use it as a function doing code that will happen when the compare matches.

If TCCR1B was initialized with (1 << WGM12), the timer will be reset when the comparison matches.
Initialize TCCR1B with prescaler and reset on match like this (example):

```
TCCR1B = (1 << CS10) | (1 << CS11) | (1 << WGM12)
```

Timer prescaler is set with CS10, CS11 and CS12 like this:

Table 62. Clock Select Bit Description

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

Figure 1: Prescaling

Set value of timer like this (set to zero before using):

```
TCNT1 = 0x0;
```

For other timers, replace 1 with the corresponding timer you want to use.

Bitwise operations in C

The AND (&) operator

bit a	bit b	a & b (a AND b)
0	0	0
0	1	0
1	0	0
1	1	1

Figure 2: Examples of result of &

The OR (|) operator

bit a	bit b	a | b (a OR b)
0	0	0
0	1	1
1	0	1
1	1	1

Figure 3: Examples of result of |

The NOT (~) operator

bit a	~a (complement of a)
0	1
1	0

Figure 4: Examples of result of ~

The XOR (^) operator

bit a	bit b	a ^ b (a XOR b)
0	0	0
0	1	1
1	0	1
1	1	0

Figure 5: Examples of result of ^

Bit-shifting

Using << (left-shift) will shift the current bits to the left.
Like this: 0b00001010 → 0b00010100

>> will do the opposite.
Like this: 0b00010100 → 0b00001010

Time in C

```c
#include <avr/io.h>
#include <avr/interrupt.h>

#define F_CPU 4000000
#define tick = F_CPU/64

type_tid sw = {0,0}; //allokerer plass til allokeringen trenger 2 bytes, de settes til 0
type_tid* tider_p[256];
//lager en array; allokerer 510 bytes (adresseområde) - 0 området tells ikke ergo 256
uint8_t trykk = 0;
uint8_t antallTider = 0;

static void initTimer(void)
{
    OCR1A = tick; //output compare med tick
    TCCR1B = (1 << cs10)|(1 << cs11); //sett prescaler til 64
    TCNT1 = 0; //setter timer counter til 0
    TIMSK = (1 << OCIE1A); //interupt on compare (OCR1A)
    sei(); //start ISR routines
}

static void initPort(void)
{
    DDRD = ~0x04; //setter PORTD bit 4 til inngang (0)
}

ISR(TIMER1_COMPA_vector)
{
    incTid(&sw); //& finner adresse til sw og caller incTid med den adressen til sw
}

static void incTid(type_tid* tid_p)
//inc tid tar imot en peker av type_tid og i sin scope refferer til den som navnet: tid_p
{
    if (tid_p->sekunder < 59) //sekunder skal økes til den er 58
        tid_p->sekunder++;
    else //når den er blitt 59 vil den nå nulle og øke minutter
    {
        tid_p->sekunder=0;
        tid_p->min++;
    }
```

```
    //alternativ løsning med modulus:
    if ++ tid_p->sekunder % 60 == 0
    //øker sekunder, også sjekker om sekunder modulus 60 blir 0
    {
        tid_p->sekunder = 0;
        tid_p->minutter++;
    }
}

static int main(void)
{
    initPort();
    initTimer();

    while(antallTider < 255) //kjør til listen blir full
    {
        if(~PIND & 0x04) //hvis PIND er trykt (0)
        {
            if(!trykk) //hvis trykk variabelen == 0
            {
                type_tid* nyTid = (type_tid*)malloc(sizeof(type_tid));
                //caster type (type_tid*), malloc reserverer minne med
                //samme størrelse som type_tid
                nyTid->sekunder = sw.sekunder; //lagrer fra sw sekunder til nyTid sekunder
                nyTid->minutter = sw.minutter; //lagrer fra sw minutter til nyTid minutter
                tider_p[antallTider++]=nyTid; //lagrer nyTid var. til antallTider array
                trykk = 1;
            }
            else
                trykk = 0;
        }
    }
    return 0;
}
```

Lights in C

```
#include <avr/io.h>
#include <avr/interrupt.h>

#define F_CPU 4000000
#define tickA F_CPU/64
#define tickB tickA/2


static volatile uint8_t counter_A;
static volatile uint8_t counter_B;

static void initTimers(void);
static void initPort(void);

static void initPort(void)
{
    DDRA = 0xFF; // Init PORTA as output
    DDRB = 0xFF; // Init PORTB as output
    PORTA = 0x0;
    PORTB = 0x0;
    DDRC = ~0x01; // Init PORTC as output, bit 0 is input
    PORTC = 0x00;
};

static void initTimers(void)
{
    //Timer 1
    TCNT1 = 0x0;
    // Prescaler 64 in control register for timer 1, WGM12: Set CTC
    // (Clear timer on compare match)
    TCCR1B = (1 << CS10)|(1 << CS11)|(1 << WGM12);
    OCR1A = tickA;
    // Interrupt (corresponding vector, TIMER1_COMPA_vect) on timer match
    // with tickA (62500)
    TIMSK = (1 << OCIE1A);

    //Timer 3
    TCNT3 = 0x0;
    // Prescaler 64 in control register for timer 3, WGM12: Set CTC
    // (Clear timer on compare match)
    TCCR3B = (1 << CS10)|(1 << CS11)|(1 << WGM12);
    OCR3A = tickB;
```

```
    // Interrupt (corresponding vector, TIMER3_COMPA_vect) on timer match
    // with tickB (31250)
    ETIMSK = (1 << OCR3A);

    sei(); // Enable global interrupts
};

ISR(TIMER1_COMPA_vect)
{
    counter_A++;
};

ISR(TIMER3_COMPA_vect)
{
    counter_B++;
};

int main(void)
{
    initPort();
    initTimers();

    static update = 0;

    while(1)
    {
        // Run while bit 0 in PINC is 0 (Invert PINC then AND with 1, this will
        // return 1 if PINC0 was originally 0) (Check if button is pressed)
        while (~PINC & 1)
        {
            if (!update) {
                cli(); // Disable global interrupts
                PORTA = 0xFF;
                PORTB = 0xFF;
                update = 1; // Has been updated
            }
        }
        // If button is not pressed, continue blinking lights
        if (update)
        {
            update = 0;
            sei(); // Enable global interrupts
        }

        if (counter_A > 6) counter_A = 0;
        // Bitshifting shenanigans, shifts one up. Ex: 0b00000010 -> 0b00000100
```

```
        PORTA = (1 << counter_A++);

        if (counter_B > 6) counter_B = 0;
        // Bitshifting shenanigans, shifts one up. Ex: 0b00000010 -> 0b00000100
        PORTB = (1 << counter_B++);
    };

    return 0;
}
```

Time in C

```
/*
 * Mappeinnlevering.c
 *
 * Created: 23.03.2015 12:05:45
 *  Author: michae14
 */

#include <avr/io.h>
#include <avr/interrupt.h>
#include <stdint.h>
#include <stdlib.h>
#include <util/atomic.h>

#define F_CPU 4000000
#define tick F_CPU/64

typedef struct
{
    volatile uint8_t centiseconds;
    volatile uint16_t seconds;
} type_tid;

typedef struct
{
    volatile uint8_t loperNummer;
    volatile type_tid* startTid;
    volatile type_tid* sluttTid;
    volatile type_tid* lopeTid;
} type_loper;

typedef struct
{
    volatile uint8_t loperNummer;
    volatile type_tid tid;
    volatile uint8_t handled;
} type_event;

type_tid tid = { 0, 0 };
type_event loper_event;

uint16_t snittTid;
type_loper* besteLoper_p;
```

```
type_loper* lopere_p[255];

static void initTimers(void);
static void initPort(void);
static void simulerNyLoper(void);
static void handleEvent(type_event* ev);
static void incTime(type_tid* tid_p);
static void stopTimers(void);
static type_tid* finnLopeTid(volatile type_tid* start_p, volatile type_tid* slutt_p);
static type_loper* finnBesteLoper(type_loper* first_p, type_loper* second_p);
static void visBesteLoper(void);

static void initTimers(void)
{
    TCCR1B = ((1 << WGM12) | (1 << ICNC1));
    // Setter timeren i CTC modus (Clear Timer on Compare) og Capture på PORTD4
    TCNT1  = 0;                             // Resetter teller
    OCR1A  = tick;                          // Setter output compare register
    TIMSK  = (1 << OCIE1A | 1 << TICIE1);   // Setter interrupt til compare match

    TCCR3B = (1 << WGM12);  // Setter timeren i CTC modus (Clear Timer on Compare)
    TCNT3  = 0;             // Reetter timer-telleren
    OCR3A  = tick;          // Setter compare-registeret til v?r tick verdi
    ETIMSK = (1 << OCIE3A); // Setter (extended) interrupt flagget til compare match

    sei();                  // Enabler  global interrupt
}

/*
 * Starter timerene ved å sette prescalar til 64
 */
static void startTimers(void)
{
    TCCR1B |= ((1 << CS10) | (1 << CS11));
    TCCR3B |= ((1 << CS10) | (1 << CS11));
}

/*
 * Stopper timerene ved å fjerne prescalar mask
 */
static void stopTimers(void)
{
    TCCR1B &= ~((1 << CS10 ) | (1 << CS11)); // Stopp timer 1
    TCCR3B &= ~((1 << CS10 ) | (1 << CS11)); // Stopp timer 3
}
```

```
/*
 * PORTE leser l?pers nummer (8-bit)
 * PORTD4 leser n?r l?per passerer start/m?l
 * PORTD0 leser n?r l?pet starter (et trykk) eller stopper (et til trykk)
 * PORTA sender l?per med beste tid sitt nummer (8-bit)
 * PORTC sender l?pere til resultattavla (8-bit)
 * PORTB bit 0 og 1 er kontrollbits for overf?ring av l?pernummer til resultattavla
 */
static void initPort(void)
{
    DDRE = DDRD = 0x0;
    DDRA = DDRC = 0xFF;
    DDRB = 0x1;

    PORTA = 0x0;
}

/*
 * Interrupt p? Timer 3:
 * Dette er simuleringstimeren, som sender ut l?pere med 2 sekunders intervall
 */
ISR(TIMER3_COMPA_vect)
{
    simulerNyLoper();
}

/*
 * Interrupt p? Timer 1:
 * Dette er simuleringstimeren som blir kallet hvert sekund, og cleara (CTC)
 */
ISR(TIMER1_COMPA_vect)
{
    incTime(&tid);
}

/*
 * Interrupt p? PORTD4:
 * Det kommer et interrupt p? Timer1 n?r PORTD4 g?r lav
 */
ISR(TIMER1_CAPT_vect)
{
    tid.centiseconds = ((s1 * 100)/tick); // Leser hundredeler fra Input Control Register

    loper_event.loperNummer = PINE;
    loper_event.tid = tid;
    loper_event.handled = 0;
```

```
}

/*
 * Denne funksjonen sender ut ny l?per for simulering av l?p
 * Start- og m?lpasseringsdetektoren gir l?pers nummer p? PORTE og signal p? PORTD4
 * PORTD4 High = L?per passerer start eller m?l
 */
static void simulerNyLoper(void)
{
    static uint8_t teller = 0;
    static uint8_t loperId = 1;
    static uint8_t antallLopere = 15;

    if (teller < 30 && loperId == antallLopere)
    {
        teller++;
        return;
    }

    PORTE = loperId * 10;
    if (PORTD & (1 << 4))
    {
        PORTD = ~(1 << 4);
        if (teller == 30)
            loperId--;
        else
            loperId++;

        if (loperId == 0)
            PORTD = (1 << 0);
            // L?pet stoppes n?r alle l?perne har passert (simuler trykk p? PORTD0)
    }
    else
    {
        PORTD = 0xFF;
    }
}

/* Her prossesseres informasjonen fra interrupt.
 * Vi har flytta den ut av interrupt rutina for ? ikke blokkere nye interrupts
 */
static void handleEvent(type_event* ev)
{
    static uint8_t loperId = 0;

    type_loper* loper_p = NULL;
```

```
    for (uint8_t i = 0; i < 255; i++)
    {
        if (lopere_p[i] == NULL) continue;
        if (lopere_p[i]->loperNummer == ev->loperNummer)
        { // Vi kjenner allerede til l?peren, dermed er l?peren n? i m?l
            loper_p = lopere_p[i];
            loper_p->sluttTid = &ev->tid;
            loper_p->lopeTid = finnLopeTid(loper_p->startTid, loper_p->sluttTid);

            if (besteLoper_p == NULL) // Antakeligvis f?rstemann i m?l, dermed forel?pig bes
                besteLoper_p = loper_p;
            else
                besteLoper_p = finnBesteLoper(besteLoper_p, loper_p);
// Vi kaller en funksjon for ? sjekke om denne l?peren er raskere enn n?v?rende innehaver

            visBesteLoper(); // Oppdater PORTA med lopernummer til raskeste l?per

// Vi har gjort det vi kom hit for ? gj?re. Un?dvendig ? forsette loopen og NULL-sjekken.
            return;
        }
    }
    if (loper_p == NULL)
    { // L?peren var ikke i lista, dermed har han nettopp passert start og m? legges til
        loper_p = (type_loper*)malloc(sizeof(type_loper));
        loper_p->loperNummer = ev->loperNummer;
        loper_p->startTid = &ev->tid;

        lopere_p[++loperId] = loper_p;
    }
}

static void incTime(type_tid* tid_p)
{
    tid_p->seconds++;
}

/*
 * Her regnes differansen ut mellom to tider
 * Brukes til ? regne ut hvor lang tid en l?per har brukt
 */
static type_tid* finnLopeTid(volatile type_tid* start_p, volatile type_tid* slutt_p)
{
    type_tid* diff_p = (type_tid*)malloc(sizeof(type_tid));

    diff_p->seconds = slutt_p->seconds - start_p->seconds; // Trekk fra sekunder
    if (start_p->centiseconds < slutt_p->centiseconds)
```

```
    // Pass p? at start-hundredeler o stopp-hundredeler ikke ender som et minustall.
    {
        diff_p->seconds--;
        diff_p->centiseconds = 100-(start_p->centiseconds-slutt_p->centiseconds);
    }
    else
        diff_p->centiseconds = slutt_p->centiseconds - start_p->centiseconds;

    return diff_p;
}

/*
 * Her sammenlignes to spilleres lopetider og pekeren til den raskeste returneres
 */
static type_loper* finnBesteLoper(type_loper* first_p, type_loper* second_p)
{
    if (first_p == NULL)    // Om den f?rste er NULL er den andre den beste
        return second_p;
    if (second_p == NULL)   // Om den ander er NULL en den f?rste den beste
        return first_p;

    if (first_p->lopeTid->seconds < second_p->lopeTid->seconds) // Forste loper var raskere
        return first_p;
    if (first_p->lopeTid->seconds == second_p->lopeTid->seconds)
    // Loperne lop paa like mange sekunder, sjekker hundredeler
    {
        if (first_p->lopeTid->centiseconds < second_p->lopeTid->centiseconds)
            return first_p;
        else
            return second_p;
    }

    return second_p;
}
// Ytterl?kka bestemmer ny plassering
// Raskeste løper kommer først
static void sorterLopere(void)
{
    uint8_t i;
    uint8_t j;
    uint8_t flyttFra = 0;
    type_loper* flytt = NULL;

    for (i = 0; i < 255; i++)
    {
        for (j = i; j < 255; j++)
```

```
        {
            if (lopere_p[j] == NULL) continue;

            flytt = finnBesteLoper(flytt, lopere_p[j]);
            if (flytt == lopere_p[j])
                flyttFra = j;
        }

        lopere_p[flyttFra] = lopere_p[i];
        lopere_p[i] = flytt;
        flytt = NULL;
    }
}

/*
 * Oppdaterer PORTA med nummeret til den raskeste l?peren.
 * Blir kalt hver gang en l?per kommer i m?l
 */
static void visBesteLoper(void)
{
    if (besteLoper_p == NULL)
        PORTA = 0x0;
    else
        PORTA = besteLoper_p->loperNummer;
}

/*
 * Regner ut snitt-tiden til løperne
 * Enkelt når man kun holder styr på sekunder
 */
static void regnGjennomsnittstid(void)
{
    uint16_t sumSekund = 0;
    uint8_t sumHundredeler = 0;
    uint8_t antall = 0;
    uint8_t i = 0;

    for (i = 0; i < 255; i++)
    {
        if (lopere_p[i] == NULL) continue;

        sumSekund += lopere_p[i]->lopeTid->seconds;
        sumHundredeler += lopere_p[i]->lopeTid->centiseconds;
        sumSekund += sumHundredeler / 100;
        sumHundredeler = sumHundredeler % 100;
```

```
        antall++;
    }

    sumSekund = sumSekund / antall;
    sumHundredeler = (sumHundredeler * 100) / antall;
}

int main(void)
{
    uint8_t buttonUp = 0;

    loper_event.handled = 1;

    initPort();         // Sett opp portene
    initTimers();       // Sett opp timerene
    while (!(PIND & (1 << 0))); // Vent til PIND0 har f?tt 1 trykk (starter l?pet)
    startTimers();      // Starter timerene etter første trykk
    while (1)
    {
        if (!loper_event.handled) // Vi behandler kun et event en gang
        {
            ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
// Stopper midlertidig for nye interrupts slik at den nåværende dataen ikke blir overskreve
            {
                handleEvent(&loper_event);
                loper_event.handled = 1;
            }
        }

        if (PIND & ~(1 << 0))
// Vi sjekker om det f?rste trykket er sluppet, og lagrer en tilstand for det
            buttonUp = 1;
        if (buttonUp && PIND & (1 << 0)) // Vi har f?tt et nytt trykk, stopper l?pet
            break;

    }
    // Vi kommer ut av loopen etter andre trykk (dvs. når klokka stopper)
    stopTimers();       // Stopp timerene (alle er i mål)
    sorterLopere();     // Sorter
    regnGjennomsnittstid(); // Regn snitttid
}
```