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)
    #alternativ: ser tmp
    out DDRA, tmp
    ldi tmp, 0x0 #alle bit settes til 0, PORTB som input (low, alle bit til 0)
    out DDRB, tmp
    ser tmp #setter alle bits til high (1)
    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
    ser tmp #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
```

Alternativ løsning Vår 2014 eksamen med BCD

```
do_bcd:

    ldi r20, 0x74 #putter et stort tall i r20
    mov r18, r20 #lager en kopi av tallet til r18

    lsr r18 #right shift bits i r18, fire ganger
```

```
lsr r18 #flytter inn 4 nuller fra venstre
lsr r18 #dytter ut de til høyre
lsr r18 #ex: 1010 0111 blir til 0000 1010

mov r19,r20 #lager en ny kopi av tallet fra r20 i r19

andi r19, 0b00001111 #AND imitiade slik at kun de siste bits til høyre blir igjen

ldi r21, 10 #setter inn 10 i r21
mul r18, r21 #ganger ta r18 med r21
#OBS! mul sender svar til r0

add r0,r19 #addere r19 med svar fra mul i r0

stop:
rjmp stop
```