Needed algo

```
Unset
.macro sevenseg_decode ; hex sevenseg to display
 andi @0, 0x0f
 cpi @0, 0
 breq sevenseg0
 cpi @0, 1
 breq sevenseg1
 cpi @0, 2
 breq sevenseg2
 cpi @0, 3
 breq sevenseg3
 cpi @0, 4
 breq sevenseg4
 cpi @0, 5
 breq sevenseg5
 cpi @0, 6
 breq sevenseg6
 cpi @0, 7
 breq sevenseg7
 cpi @0, 8
 breq sevenseg8
 cpi @0, 9
 breq sevenseg9
 cpi @0, 10
 breq sevensegA
 cpi @0, 11
 breq sevensegB
 cpi @0, 12
 breq sevensegC
 cpi @0, 13
 breq sevensegD
 cpi @0, 14
 breq sevensegE
 cpi @0, 15
 breq sevensegF

sevenseg0: ldi @1, 0b00111111 rjmp fin
sevenseg1: ldi @1, 0b00000110 rjmp fin
sevenseg2: ldi @1, 0b01011011 rjmp fin
sevenseg3: ldi @1, 0b01001111 rjmp fin
sevenseg4: ldi @1, 0b01100110 rjmp fin
sevenseg5: ldi @1, 0b01101101 rjmp fin
sevenseg6: ldi @1, 0b01111101 rjmp fin
```

```
sevenseg7: ldi @1, 0b00000111 rjmp fin
sevenseg8: ldi @1, 0b01111111 rjmp fin
sevenseg9: ldi @1, 0b01101111 rjmp fin
sevensegA: ldi @1, 0b01110111 rjmp fin
sevensegB: ldi @1, 0b01111100 rjmp fin
sevensegC: ldi @1, 0b00111001 rjmp fin
sevensegD: ldi @1, 0b01011110 rjmp fin
sevensegE: ldi @1, 0b01111001 rjmp fin
sevensegF: ldi @1, 0b01110001 rjmp fin
fin: nop
.endmacro
```

Unset

```
.macro keepindataspace
  st Y+, @0           ; work with X, Y, Z
.endmacro
```

Unset

```
.macro loadfromZ
       lpm Z, @0
       adiw Z, 1
.endmacro
```

Unset

```
.macro masklower4
       andi @0, 0b00001111
.endmacro
```

Unset

```
.macro maskupper4
       andi @0, 0b00001111
.endmacro
```

Unset

```
.macro setlower4
       ldi r16, 0b00001111
       or @0, r16
.endmacro
```

Unset

```
.macro sethigher4
       ldi r16, 0b11110000
       or @0, r16
.endmacro
```

Unset

```
.macro togglelower4
       ldi r16, 0b00001111
       eor @0, r16
.endmacro
```

Unset

```
.macro toggleall
       com @0          ; compliment
.endmacro
```

Unset

```
.macro setRegBit                    ; @1 is for the bit that wanna set
  ldi r16, 0
  sbr r16, (1 << @1)
  or @0, r16
.endmacro
```

Unset

```
.macro delay1sec ; 1 000 000 clk = 1 sec
       ldi r16, 10
l1: ldi r17, 80
```

```
l2: ldi r18, 250
l3:
       nop
       nop
       dec r18
       brne l3
       dec r17
       brne l2
       dec r16
       brne l1
.endmacro
```

Unset

```
.macro delay8sec
       ldi r16, 32
l1: ldi r17, 200
l2: ldi r18, 250
l3:
       nop
       nop
       dec r18
       brne l3
       dec r17
       brne l2
       dec r16
       brne l1
.endmacro
```

Unset

```
.macro delay1ms
delay1sec: ; 1 000 000 clk = 1 sec
       ldi r16, 1
l1: ldi r17, 8
l2: ldi r18, 250
l3:
       nop
       nop
       dec r18
       brne l3
       dec r17
```

```
       brne l2
       dec r16
       brne l1
.endmacro
```

Unset

```
.macro getdata_2     ; swap Z to get data from the another loc
       movw r15:r14, r31:r30
       movw r31:r30, r29:r28
.endmacro
```

Unset

```
.macro getdata_1     ; get Z back from getdata_2
       movw r29:r28, r31:r30
       movw r31:r30, r15:r14
.endmacro
```

Unset

```
.macro swapreg8
       movw r16, @0
       movw @0, @1
       movw @1, r16
.endmacro

.macro swapreg16
       movw r17:r16, @0:@1
       movw @0:@1, @2:@3
       movw @2:@3, r17:r16
.endmacro
```

Unset

```
.macro keepinflash          ;X here points to loc that we wanna keep
       swapreg16 ZH, ZL, XH, XL
       ldi ZH, high(@0)
       ldi ZL, low(@0)
       movw r1:r0, @1:@2
```

```
        spm
        adiw Z, 1
        swapreg16 ZH, ZL, XH, XL
.endmacro
```

Unset

```
.macro setpullup     ; 0->rA / 1->ddrx1 / 2->ddrx2 / 3->portx1
        ldi @0, 0x00
        out @1, @0
        ldi @0, 0xff
        out @2, @0
        out @1,@0
.endmacro
```

Unset

```
.macro toIdxZero
  ldi xh, high(0x0100)
  ldi xl, low(0x0100)
.endmacro

.macro toIdxFromReg
  toIdxZero
  ldi r16, 255
iterate:
  inc r16
  cp @0, r16
  breq endIterate
  adiw x, 1
  rjmp iterate
endIterate:
.endmacro

.macro storeArrayIdx
  toIdxFromReg @1
  st x, @0
.endmacro

.macro loadArrayIdx
  toIdxFromReg @1
  ld @0, x
```

```
.endmacro
```

Unset

```
; use case of array
toIdxZero
ldi r21, 0x01
ldi r17, 0
storeArrayIdx r21, r17
ldi r21, 0x13
ldi r17, 1
storeArrayIdx r21, r17
ldi r21, 0x55
ldi r17, 3
storeArrayIdx r21, r17
ldi r21, 0x61
ldi r17, 2
storeArrayIdx r21, r17

ldi r17, 1
loadArrayIdx r21, r17
```

Unset

```
.macro fibfunction
; fibonucci
       ldi r16, 0x00
       out ddrb, r16
       ldi r16, 0xff
       out ddrd, r16

       ldi r16, low(0)                    ; a
       ldi r17, high(0)
       ldi r18, low(1)                    ; b
       ldi r19, high(1)

fib:
       add16bit r16, r17, r18, r19
       movw r31:r30, r17:r16              ; store result
       swapregist16bit r17:r16, r19:r18
       rjmp fib
```

```
.endmacro
```

Get bit @1: (immediate value) of @0: (register) and store in @2: (register)

```
Unset
.macro getRegBit
       ldi r16, 0
       ldi @2, 0
  sbr r16, (1 << @1)
       and r16, @0
       breq regBitIsZero
       ldi @2, 1
regBitIsZero:
.endmacro
```

```
Unset
; Dabble address
.macro dabbleAddr
       lds r17, @0
       dabble r17
       sts @0, r17
.endmacro

; Dabble part from doubleDabble algorithm
.macro dabble
       sts 0x0316, r16
       ldi r16, 0x0f
       and r16, @0
       subi r16, 0x05
       brlo skipDabble1
       ldi r16, 0x03
       add @0, r16
skipDabble1:
       ldi r16, 0xf0
       and r16, @0
       subi r16, 0x50
       brlo skipDabble2
```

```
        ldi r16, 0x30
        add @0, r16
skipDabble2:
        lds r16, 0x0316
.endmacro

; 0x0400 0x0401 will be for storing bcd value
; 0x0402 0x0403 will be for original binary waiting to be shifted
.macro doubleDabbleTick
        sts 0x0416, r16
        dabbleAddr 0x0400
        dabbleAddr 0x0401
        getRegBitAddr 0x0401, 7, r17
        shiftLeftInputAddr 0x0400, r17
        getRegBitAddr 0x0402, 7, r17
        shiftLeftInputAddr 0x0401, r17
        getRegBitAddr 0x0403, 7, r17
        shiftLeftInputAddr 0x0402, r17
        getRegBitAddr 0x0404, 7, r17
        shiftLeftInputAddr 0x0403, r17
        lds r16, 0x0416
.endmacro

.macro doubleDabble
        ldi r16, 0x00
        sts 0x0400, r16
        sts 0x0401, r16
        sts 0x0402, @0
        sts 0x0403, @1
        ldi r16, 16
for_loop:
        doubleDabbleTick
        dec r16
        breq doubleDabbleFinish
        jmp for_loop
doubleDabbleFinish:
.endmacro
```

Unset

```
.macro swapregist16bit
        movw r21:r20, @0
        movw @0, @1
```

```
        movw @1, r21:r20
.endmacro
```

Logical shift left of @0: (register) with input @1: (register)

```
Unset
.macro shiftLeftInput
        lsl @0
        ldi r16, 0b00000001
        andi r16, @1
        breq inputBitIsZero
        setRegBit @0, 0
inputBitIsZero:
.endmacro
```

array

```
Unset
; Go to array index zero
.macro toIdxZero
        ldi xh, high(0x0100)
        ldi xl, low(0x0100)
.endmacro

; Go to array index @0: (register)
.macro toIdxFromReg
        toIdxZero
        ldi r16, 255
iterate:
        inc r16
        cp @0, r16
        breq endIterate
        adiw x, 1
        rjmp iterate
endIterate:
.endmacro

; Store value from @0: (register) to array at index @1: (register)
.macro storeArrayIdx
        toIdxFromReg @1
        st x, @0
```

```
.endmacro

; Load value to @0: (register) from array at index @1: (register)
.macro loadArrayIdx
	toIdxFromReg @1
	ld @0, x
.endmacro
```

Double dabble

```
Unset
; Dabble address
.macro dabbleAddr
	lds r17, @0
	dabble r17
	sts @0, r17
.endmacro

; Dabble part from doubleDabble algorithm
.macro dabble
	sts 0x0316, r16
	ldi r16, 0x0f
	and r16, @0
	subi r16, 0x05
	brlo skipDabble1
	ldi r16, 0x03
	add @0, r16
skipDabble1:
	ldi r16, 0xf0
	and r16, @0
	subi r16, 0x50
	brlo skipDabble2
	ldi r16, 0x30
	add @0, r16
skipDabble2:
	lds r16, 0x0316
.endmacro

; 0x0400 0x0401 will be for storing bcd value
; 0x0402 0x0403 will be for original binary waiting to be shifted
.macro doubleDabbleTick
	sts 0x0416, r16
	dabbleAddr 0x0400
```

```
        dabbleAddr 0x0401
        getRegBitAddr 0x0401, 7, r17
        shiftLeftInputAddr 0x0400, r17
        getRegBitAddr 0x0402, 7, r17
        shiftLeftInputAddr 0x0401, r17
        getRegBitAddr 0x0403, 7, r17
        shiftLeftInputAddr 0x0402, r17
        getRegBitAddr 0x0404, 7, r17
        shiftLeftInputAddr 0x0403, r17
        lds r16, 0x0416
.endmacro

.macro doubleDabble
        ldi r16, 0x00
        sts 0x0400, r16
        sts 0x0401, r16
        sts 0x0402, @0
        sts 0x0403, @1
        ldi r16, 16
for_loop:
        doubleDabbleTick
        dec r16
        breq doubleDabbleFinish
        jmp for_loop
doubleDabbleFinish:
.endmacro
```

Fib to 7 segment

```
Unset
ldi r16, 0xff
out ddrd, r16
out ddrb, r16

reset:
  ldi16 r21, r22, 0 ; Number #1
  ldi16 r23, r24, 1 ; Number #2

while:
  add16 r21, r22, r23, r24
  swapReg16 r21, r22, r23, r24
```

```
        ldi r16, 0x00
        sts 0x0400, r16
        sts 0x0401, r16
        sts 0x0402, r21
        sts 0x0403, r22

        doubleDabble r21, r22
        lds r17, 0x0401
        sevenSeg r25, r17
        sts 0x0203, r25
        lsr r17 lsr r17 lsr r17 lsr r17
        sevenSeg r25, r17
        sts 0x0202, r25
        lds r17, 0x0400
        sevenSeg r25, r17
        sts 0x0201, r25
        lsr r17 lsr r17 lsr r17 lsr r17
        sevenSeg r25, r17
        sts 0x0200, r25
        rcall waitWithDisplay


        ldi r16, 0x1A
        sub r16, r21
        breq resetActivated
   jmp while
resetActivated:
        jmp reset


displaySevenSeg:
        ldi r16, 0b00000001
        out portD, r16
        lds r16, 0x0200
        out portB, r16
        rcall delay

        ldi r16, 0b00000010
        out portD, r16
        lds r16, 0x0201
        out portB, r16
        rcall delay

        ldi r16, 0b00000100
```

```
        out portD, r16
        lds r16, 0x0202
        out portB, r16
        rcall delay

        ldi r16, 0b00001000
        out portD, r16
        lds r16, 0x0203
        out portB, r16
        rcall delay
        ret

waitWithDisplay:
              sts 0x318, r18
              sts 0x319, r19
              sts 0x320, r20
              ldi r18, 2
wd1:    ldi r19, 2
wd2:    ldi r20, 10
wd3:    nop
        rcall displaySevenSeg
        dec r20
        brne wd3
        dec r19
        brne wd2
        dec r18
        brne wd1
        ret

wait:
              sts 0x318, r18
              sts 0x319, r19
              sts 0x320, r20
              ldi r18, 4
w1:           ldi r19, 200
w2:           ldi r20, 250
w3:           nop
        nop
        dec r20
        brne w3
        dec r19
        brne w2
        dec r18
        brne w1
```

```
       lds r18, 0x318
       lds r19, 0x319
       lds r20, 0x320
       ret

delay:
              sts 0x318, r18
              sts 0x319, r19
              sts 0x320, r20
              ldi r18, 4
d1:           ldi r19, 20
d2:           ldi r20, 25
d3:           nop
       dec r20
       brne d3
       dec r19
       brne d2
       dec r18
       brne d1
       lds r18, 0x318
       lds r19, 0x319
       lds r20, 0x320
       ret
```

Short 7seg mod 16 counter

Unset

```
.macro sevenseg
       lpm @0, z+
.endmacro

.org 0x0000
       ldi r16, 0xff
       out ddrb, r16
       out ddrd, r16

reset:
       ldi zl, low(0x420)
    ldi zh, high(0x420)
while:
       sevenseg r21
       rcall waitanddis
```

```
       sevenseg r22
       rcall waitanddis
       sevenseg r23
       rcall waitanddis
       sevenseg r24
       rcall waitanddis
       cpi zl, 0x002f
       brlo while
       jmp reset

display7seg:

       ldi r16, 0b00000001
       out portb, r16
    out portd, r21
    call delay

       ldi r16, 0b00000010
       out portb, r16
    out portd, r22
    call delay

       ldi r16, 0b00000100
       out portb, r16
    out portd, r23
    call delay

       ldi r16, 0b00001000
       out portb, r16
    out portd, r24
    call delay
       ret


waitanddis:
             sts 0x318, r18
             sts 0x319, r19
             sts 0x320, r20
             ldi r18, 2
wd1:   ldi r19, 2
wd2:   ldi r20, 10
wd3:   nop
       rcall display7seg
       dec r20
```

```
       brne wd3
       dec r19
       brne wd2
       dec r18
       brne wd1
       ret

delay:
              sts 0x318, r18
              sts 0x319, r19
              sts 0x320, r20
              ldi r18, 4
d1:           ldi r19, 20
d2:           ldi r20, 25
d3:           nop
       dec r20
       brne d3
       dec r19
       brne d2
       dec r18
       brne d1
       lds r18, 0x318
       lds r19, 0x319
       lds r20, 0x320
       ret

.org 0x210
segmentTable:
    .db 0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f, 0x77, 0x7c,
0x39, 0x5e, 0x79, 0x71
```

Mod ff counter to 7 segment

```
Unset
.macro sevenseg
       lpm @0, z+
.endmacro

.macro swapreg16
       movw r25:r24, @0:@1
       movw @0:@1, @2:@3
       movw @2:@3, r25:r24
.endmacro
```

```
.org 0x0000
       ldi r16, 0xff
       out ddrb, r16
       out ddrd, r16

       ldi xl, low(0x0420)
       ldi xh, high(0x0420)
reset:
       ldi zl, low(0x0420)
       ldi zh, high(0x0420)
while:
       sevenseg r21
       swapreg16 zh, zl, xh, xl
       lpm r22, z
       swapreg16 zh, zl, xh, xl
       rcall waitanddis
       cpi zl, 0x0030
       brne while
       inc xl
       cpi xl, 0x0030
       breq clr22
       jmp reset


clr22:
       ldi xl, low(0x0420)
       jmp reset


display7seg:
       ldi r16, 0b00000010
       out portb, r16
       out portd, r21
       call delay

       ldi r16, 0b00000001
       out portb, r16
       out portd, r22
       call delay
       ret

waitanddis:
       sts 0x318, r18
       sts 0x319, r19
       sts 0x320, r20
```

```
       ldi r18, 2
wd1: ldi r19, 2
wd2: ldi r20, 10
wd3: nop
       rcall display7seg
       dec r20
       brne wd3
       dec r19
       brne wd2
       dec r18
       brne wd1
       ret

delay:
       sts 0x318, r18
       sts 0x319, r19
       sts 0x320, r20
       ldi r18, 4
d1: ldi r19, 21
d2: ldi r20, 25
d3: nop
       dec r20
       brne d3
       dec r19
       brne d2
       dec r18
       brne d1
       lds r18, 0x318
       lds r19, 0x319
       lds r20, 0x320
       ret

.org 0x210
segmentTable:
       .db 0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f, 0x6f, 0x77, 0x7c,
0x39, 0x5e, 0x79, 0x71
```