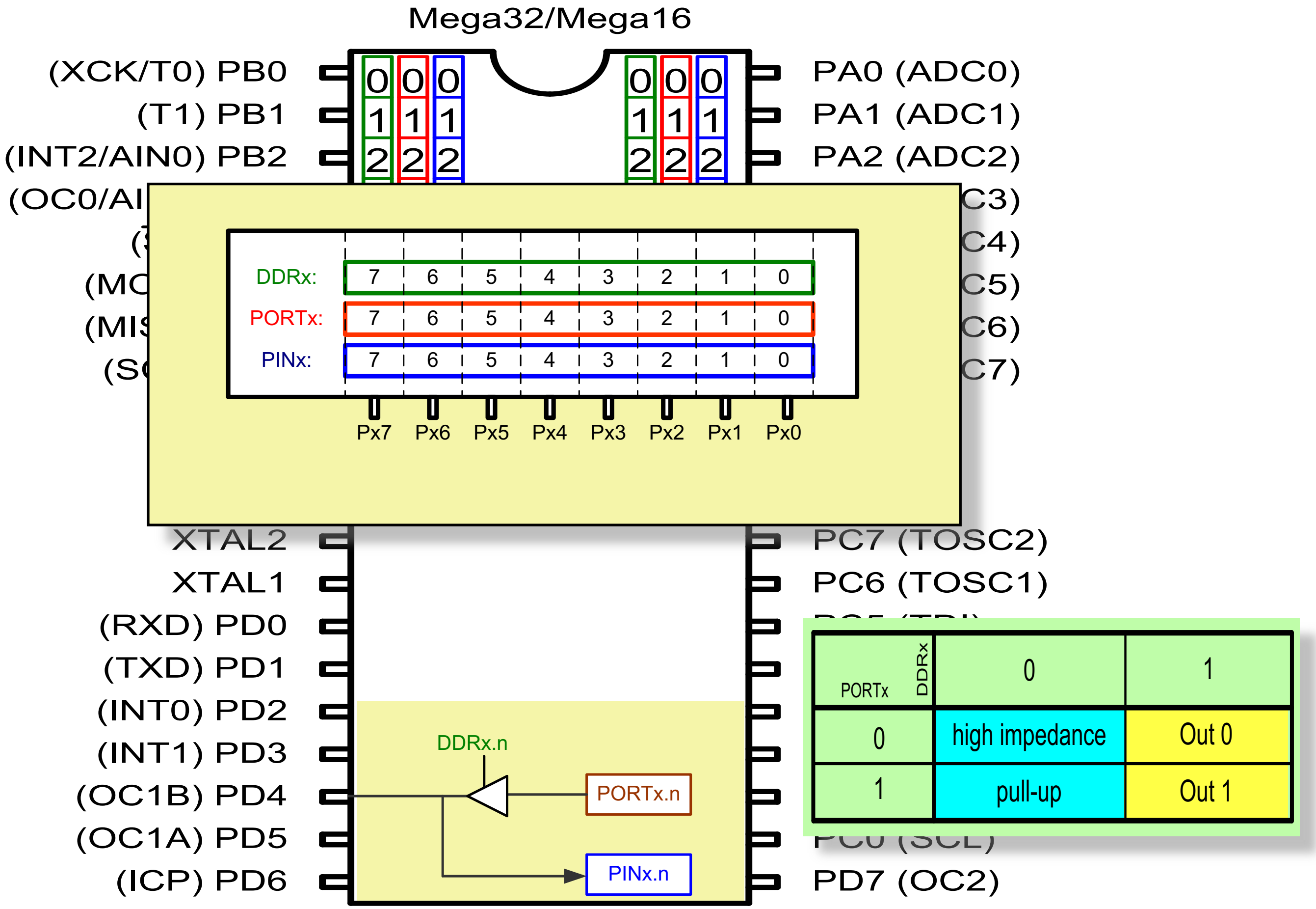
Lecture 3 content

- Ports
  - DDRx – defining direction (input/output)
  - Read from ports (PINx) and Write to ports (PORTx), Config.
- Instructions related to Ports:
  - OUT/IN
  - SBI/CBI
  - SBIC/SBIS
- Boolean instructions
  - AND/ANDI, OR/ORI
  - LSL, LSR, ASR, ROL, ROR

Ports

- I/O pins are generally grouped into ***ports*** of 8 pins, which can be accessed with a single byte access.
- Pins can either be input only, output only, or
  — most commonly,
  — bidirectional, that is, capable of both input and output. (not simultaneously!)

The structure of IO pins (not the chip we have)

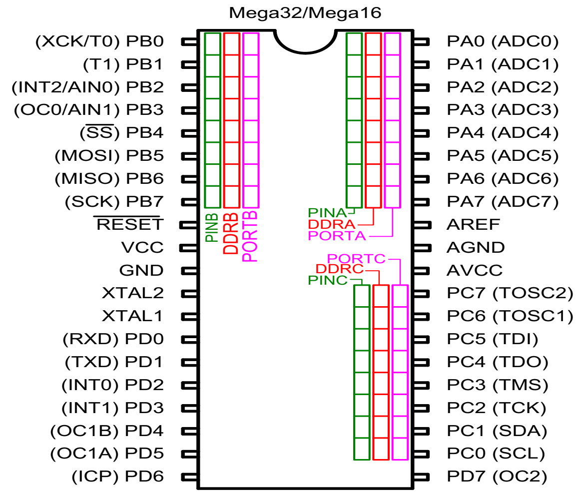


PORTx \ DDRx	0	1
0	high impedance	Out 0
1	pull-up	Out 1

Example 1

- Write a program that makes all the pins of PORTA one.

DDRA:	1	1	1	1	1	1	1	1
PORTA:	1	1	1	1	1	1	1	1




```
LDI   R20,0xFF   ;R20 = 0b11111111
OUT   DDRA,R20   ;DDRA = R20
OUT   PORTA,R20  ;PORTA = R20
```

PORTx \ DDRx	0	1
0	high impedance	Out 0
1	pull-up	Out 1

Example 2

- The following code will toggle all 8 bits of Port B forever with some time delay between "on" and "off" states:

```
    LDI     R16,0xFF      ;R16 = 0xFF = 0b11111111
    OUT     DDRB,R16      ;make Port B an output port
L1: LDI     R16,0x55      ;R16 = 0x55 = 0b01010101
    OUT     PORTB,R16     ;put 0x55 on port B pins
    CALL    DELAY
    LDI     R16,0xAA      ;R16 = 0xAA = 0b10101010
    OUT     PORTB,R16     ;put 0xAA on port B pins
    CALL    DELAY
    RJMP    L1
```

Example 5: Input

- The following code indefinitely gets the data present at the pins of port C and sends it to port B, after adding the value 5 to it:

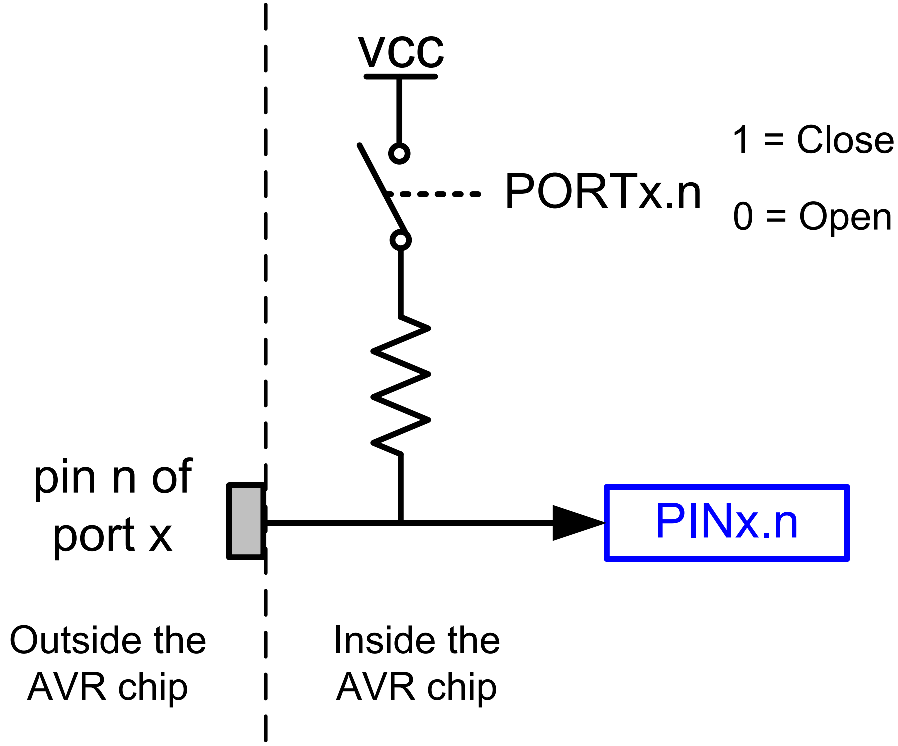
```
      LDI    R16,0x00      ;R16 = 00000000 (binary)
      OUT    DDRC,R16      ;make Port C an input port
      LDI    R16,0xFF      ;R16 = 11111111 (binary)
      OUT    DDRB,R16      ;make Port B an output port(1 for Out)
L2:   IN     R16, PINC     ;read data from Port C and put in R16
      LDI    R17,5
      ADD    R16,R17       ;add 5 to it
      OUT    PORTB,R16     ;send it to Port B
      RJMP   L2            ;continue forever
```

Pull-up resistor

Note: You can also read this value:

IN R16, PORTC

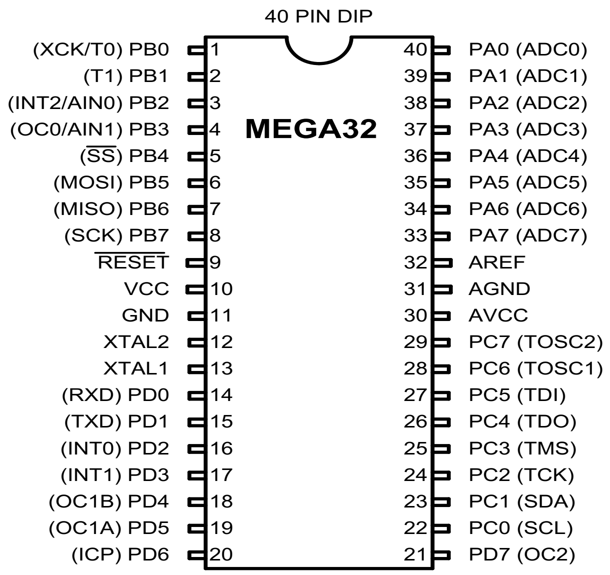
- A bug, if not intentional!




```
LDI    R16, 0x00        ;
OUT    DDRC, R16        ; Make all pins input
LDI    R16, 0xFF        ;
OUT    PORTC, R16       ; enable pull-up on all pins
……
IN     R16, PINC        ; read pins
……
```

Example 7

- Write a program that reads from port A and writes it to port B.




```
    LDI  R20,0x0    ;R20 = 00000000 (binary)
    OUT  DDRA,R20   ;DDRA = R20
    LDI  R20,0xFF   ;R20 = 11111111 (binary)
    OUT  DDRB,R20   ;DDRB = R20
L1: IN   R20,PINA   ;R20 = PINA
    OUT  PORTB,R20  ;PORTB = R20
    RJMP L1
```

PORTx \ DDRx	0	1
0	high impedance	Out 0
1	pull-up	Out 1

SBI and CBI instructions

- SBI (Set Bit in IO register)
  - SBI ioReg, bit ;ioReg.bit = 1
  - Examples:
    - SBI PORTD,0 ;PORTD.0 = 1
    - SBI DDRC,5 ;DDRC.5 = 1

- CBI (Clear Bit in IO register)
  - CBI ioReg, bit ;ioReg.bit = 0
  - Examples:
    - CBI PORTD,0 ;PORTD.0 = 0
    - CBI DDRC,5 ;DDRC.5 = 0

Example

- Write a program that toggles PORTA.4 continuously.

```
     SBI   DDRA,4    ;PA4 output
L1:  SBI   PORTA,4   ;PA4=1
     CBI   PORTA,4   ;PA4=0
     RJMP  L1
```

Example

An LED is connected to each pin of Port D. Write a program to turn on each LED from pin D0 to pin D7. Call a delay module before turning on the next LED.

```
        LDI     R20, 0xFF
        OUT     DDRD, R20           ;make PORTD
                                    ;an output
                                    ;port
        SBI     PORTD,0             ;set bit PD0
        CALL    DELAY               ;delay
                                    ;before next
        SBI     PORTD,1             ;turn on PD1
        CALL    DELAY               ;delay
                                    ;before next
        SBI     PORTD,2             ;turn on PD2
        CALL    DELAY
        SBI     PORTD,3
        CALL    DELAY
        SBI     PORTD,4
        CALL    DELAY
        SBI     PORTD,5
        CALL    DELAY
        SBI     PORTD,6
        CALL    DELAY
        SBI     PORTD,7
        CALL    DELAY
```

SBIC and SBIS

- SBIC (Skip if Bit in IO register Cleared)
  - SBIC ioReg, bit ; if (ioReg.bit = 0) skip next instr
  - Example:

```
SBIC   PIND,0   ;skip next instr if PIND.0=0
INC    R20
LDI    R19,0x23
```

- SBIS (Skip if Bit in IO register Set)

  SBIS ioReg, bit ; if (ioReg.bit = 1) skip next instr
  - Example:

```
SBIS   PIND,0   ;skip next instr if PIND.0=1
INC    R20
LDI    R19,0x23
```

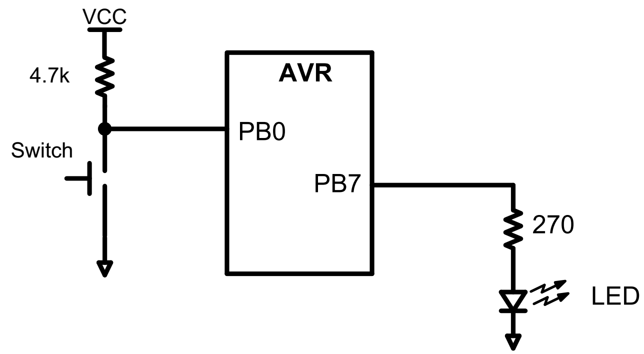
Example

- Write a program to perform the following:
- (a) Keep monitoring the PB2 bit until it becomes HIGH;
- (b) When PB2 becomes HIGH, write value $45 to Port C, and also send a HIGH-to-LOW pulse to PD3.

```
        CBI  DDRB, 2    ;make PB2 an input
        SBI  PORTB,2    ;enable pull-up on PB2
        LDI  R16, 0xFF
        OUT  DDRC, R16  ;make Port C an output port
        SBI  DDRD, 3    ;make PD3 an output
AGAIN:  SBIS PINB, 2    ;Skip if Bit PB2 is HIGH
        RJMP AGAIN      ;keep checking if LOW
        LDI  R16, 0x45  ;$45 = 0x45!
        OUT  PORTC, R16 ;write 0x45 to port C
        SBI  PORTD, 3   ;set bit PD3 high (H-to-L)
        CBI  PORTD, 3   ;set bit PD3 low again (clear PD3)
HERE:   RJMP HERE
```

Example

A switch is connected to pin PB0 and an LED to pin PB7. Write a program to get the status of Switch and send it to the LED.




```
        CBI   DDRB,0        ;make PB0 an input
        SBI   DDRB,7        ;make PB7 an output
AGAIN:  SBIS  PINB,0        ;skip next if PB0 is =1
        RJMP  OVER          ;if PB7 = 0, jump to OVER
        CBI   PORTB,7       ;LED = OFF
        RJMP  AGAIN
OVER:   SBI   PORTB,7       ;LED = ON
        RJMP  AGAIN
```

Logic instructions - Setting and Clearing bits

```
AND    Rd,Rr    ;Rd = Rd AND Rr
ANDI   RD, k    ;Rd = Rd AND k
OR     Rd,Rr    ;Rd = Rd OR Rr
ORI    Rd, k    ;Rd = Rd OR k
EOR    Rd,Rr    ;Rd = Rd XOR Rr ( immediate is not supported)
COM    Rd       ;Rd = 1’ Complement of Rd (11111111 – Rd)
NEG    Rd       ;Rd = 2’ Complement of Rd (100000000 – Rd)
SWAP   Rd       ;Swap the 2 nibbles (e.g 0x0F -> 0xF0)
```

- ***AND is used to clear an specific bit/s of a byte***

```
LDI    R20, 0x35    ; R20 =    0011 0101
ANDI   R20, 0x0F    ; AND      0000 1111
                    ; R20 =    0000 0101
```

Logic instructions - Setting and Clearing bits

AND	Rd,Rr	;Rd = Rd AND Rr
ANDI	RD, k	;Rd = Rd AND k
OR	Rd,Rr	;Rd = Rd OR Rr
ORI	Rd, k	;Rd = Rd OR k
EOR	Rd,Rr	;Rd = Rd XOR Rr (immediate is not supported)
COM	Rd	;Rd = 1' Complement of Rd (11111111 – Rd)
NEG	Rd	;Rd = 2' Complement of Rd (100000000 – Rd)
SWAP	Rd	;Swap the 2 nibbles (e.g 0x0F -> 0xF0)

- ***OR is used to set an specific bit/s of a byte***

```
LDI    R20, 0x35    ; R20 =     0011 0101
ORI    R20, 0xC0    ; OR        1100 0000
                    ; R20 =     1111 0101
```

Setting only a few bits in DDRx (1/3)

Sometimes we only want to use a few bits in a Port. We then need to set those bits in the DDRx. 3 ways:

1. Assume that the remaining bits are not used.
   1. Either set them to output (but what happens if something is connected to that bit?)
   2. Or, set them to input (without pullup, which means they are high impedance)

   Example:

```
LDI     R16, 0b00010000     ; C4 out, C5 in
OUT     DDRC, R16
```

Setting only a few bits in DDRx (2/3)

2. Read current value of DDRx, adjust the interesting bits and write back the updated values. Other bits are not affected. Example:

```
LDS     R16, DDRC          ; read DDRC
ORI     R16, 0b00010000    ; bit 4 = 1. PC4 is output
STS     DDRC , R16         ; store DDRC

LDS     R16, DDRC          ; read DDRC
ANDI    R16, 0b11011111    ; bit 5 = 0. PC5 is input
STS     DDRC, R16          ; store DDRC
```

Setting only a few bits in DDRx (3/3)

Alternative:

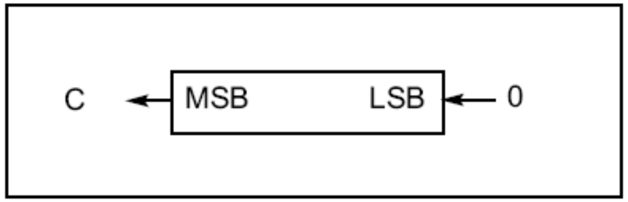
```
LDS     R16, DDRC          ; read DDRC
ANDI    R16, 0b11011111    ; bit 5 = 0. PC5 is input
OUT     DDRC, R16
```

OUT is equivalent to STS!

3. Use the instructions SBI/CBI to set the individual bits in the DDRx. Example:

```
SBI DDRC, 4        ; PC4 is defined as output
CBI DDRC, 5        ; PC5 is defined as input
SBI PORTC, 5       ; set pullup on PC5
```

LSL instruction



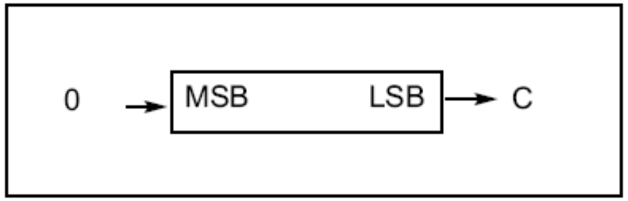
LSL Rd ;logical shift left

In LSL, as bits are shifted from right to left, 0 enters the LSB and the MSB exits to the carry flag. In other words, **in LSL 0 is moved to the LSB, and the MSB is moved to the C.** **This instruction multiplies content of the register by 2 assuming that after LSL the carry flag is not set.**

In the next code you can see what happens to 00100110 after running 3 LSL instructions.

```
CLC            ;make C = 0 (carry is 0 )
LDI R20 , 0x26 ;R20 = 0010 0110(38 – dec…) C = 0
LSL R20        ;R20 = 0100 1100(76) C = 0
LSL R20        ;R20 = 1001 1000(152) C = 0
LSL R20        ;R20 = 0011 0000(48) C = 1
               ;content of R20 is not multiplied by 2
```

LSR instruction



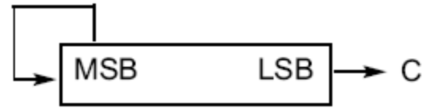
LSR   Rd   ;Logical shift right

In LSR, as bits are shifted from left to right, 0 enters the MSB and the LSB exits to the carry flag. In other words, **in LSR 0 is moved to the MSB, and the LSB is moved to the C.** **This instruction divides content of the register by 2 and carry flag contains the remainder of division.**

In the next code you can see what happens to 0010 0110 after running 3 LSL instructions.

```
LDI R20,0x26        ;R20 = 0010 0110 (38)
LSR R20             ;R20 = 0001 0011 (19) C = 0
LSR R20             ;R20 = 0000 1001 (9) C = 1
LSR R20             ;R20 = 0000 0100 (4) C = 1
```

ASR Instruction



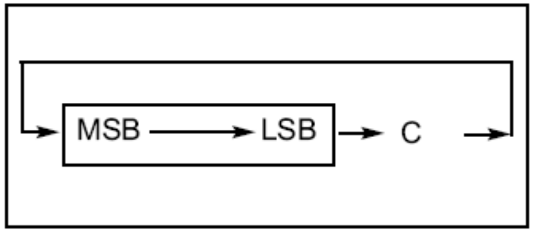
ASR Rd ;Arithmetic shift right

The ASR instruction can divide a signed number by 2. In ASR, as bits are shifted from left to right, MSB is held constant and the LSB exits to the carry flag. In other words **MSB is not changed but is copied to D6, D6 is moved to D5, D5 is moved to D4 and so on.**

In the next code you can see what happens to 0010 0110 after running 5 ASL instructions.

```
LDI   R20 , 0xD60        ;R20 = 1101 0000(-48) c = 0
ASR   R20                ;R20 = 1110 1000(-24) C = 0
ASR   R20                ;R20 = 1111 0100(-12) C = 0
ASR   R20                ;R20 = 1111 1010(-6) C = 0
ASR   R20                ;R20 = 1111 1101(-3) C = 0
ASR   R20                ;R20 = 1111 1110(-2!) C = 1
```

ROR instruction



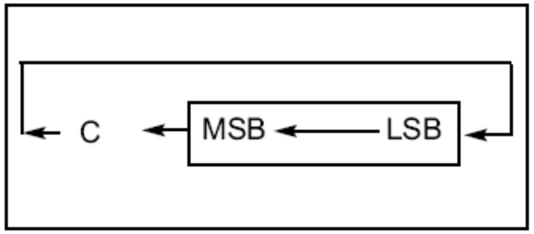
ROR Rd ;Rotate right

In ROR, as bits are rotated from left to right, the carry flag enters the MSB and the LSB exits to the carry flag. In other words, **in ROR the C is moved to the MSB, and the LSB is moved to the C.**

See what happens to 0010 0110 after running 3 ROR instructions:

```
CLC              ;CLear Carry, make C = 0 (carry is 0 )
LDI   R20 , 0x26 ;R20 = 0010 0110
ROR   R20        ;R20 = 0001 0011 C = 0
ROR   R20        ;R20 = 0000 1001 C = 1
ROR   R20        ;R20 = 1000 0100 C = 1
```

ROL instruction



ROL Rd ;Rotate left

In ROL, as bits are shifted from right to left, the carry flag enters the LSB and the MSB exits to the carry flag. In other words, **in ROL the C is moved to the LSB, and the MSB is moved to the C.**

See what happens to 0001 0101 after running 4 ROL instructions:

```
SEC              ;Set Carry, make C = 1 (carry is 1)
LDI R20,0x15     ;R20 = 0001 0101
ROL R20          ;R20 = 0010 1011 C = 0
ROL R20          ;R20 = 0101 0110 C = 0
ROL R20          ;R20 = 1010 1100 C = 0
ROL R20          ;R20 = 0101 1000 C = 1
```

Some other instructions…

…that appear in Lab 1 and that you may need to understand (we will cover this in Lecture 4):

```
…
CPI     R18, 12     ; Compare content of R18
                    ; to the value 12 (decimal)
BRNE    scan_key    ; if not equal, jump to
                    ; label ”scan_key”
…                   ; if equal, continue here
```

Assembly coding practice

It is a good practice to organize your code in four columns as shown in the code below. Your program being easier to read/debug by you and others.

```
;Col_1          Col_2          Col_3          Col_4
                ADD            R16, R17       ; Comment
                DEC            R17            // Comment
                MOV            R18, R16       ;
END:            RJMP           END            /* Comment
                                              */
```

- **Column 1 (Col_1)** is used for labels.
- **Column 2 (Col_2)** is used for the microcontroller instructions.
- **Column 3 (Col_3)** is used for arguments operated on
- **Column 4 (Col_4)** is used for comments.

Column width is not equal or fixed, but be consistent in the program!

”Include-files”

- When programs grow, it is practical to divide the program into separate files, where each file contains the implementation of a single logical part of the program. (= SW engineering…)
- Many programming languages (e.g. C) support this concept (we will cover to <file>.h and <file>.c later)
- Assembler does not!
- However, AVR Assembler support the concept of ”Include-files”!
- The syntax is .INCLUDE <file-name>
- NOTE! The pre-processor includes all content from the included file at the position of the .INCLUDE statement