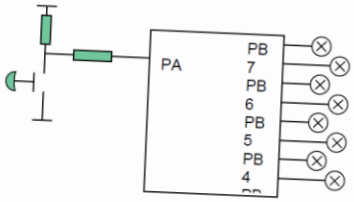
1DT301, Computer Technology I

Lecture #4,
Wednesday, September 11, 2019

- Preview of lab 2
- Switching between programs
- Random number generator
- Calling subroutines with arguments
- Using Register Pairs
- Masking bits
- Subroutine call, rcall
- Use of Stack
- Push and Pop

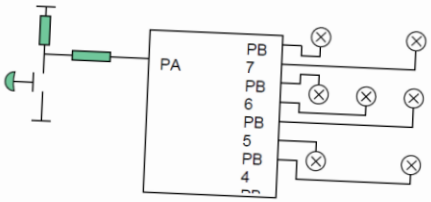
Task 1:
Switch – Ring counter / Johnson counter

Write a program which switch between Ring counter and Johnson counter. You should not use Interrupt in this lab. The pushbutton must be checked frequently, so there is no delay between the button is pressed and the change between Ring/Johnson. Use SW0 (PA0) for the button. Each time you press the button, the program should change counter.



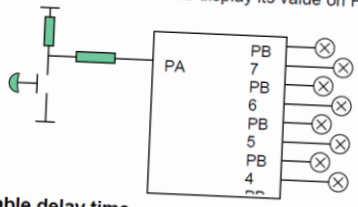
Task 2:
Electronic dice

You should create an electronic dice. Think of the LEDs placed as in the picture below. The number 1 to 6 should be generated randomly. You could use the fact that the time you press the button varies in length.



Task 3:
Change counter

Write a program that is able to count the number of changes on a switch. As a change we count when the switch SW0 goes from 0 to 1 and from 1 to 0, we expect therefore positive and negative edges. We calculate the changes in a byte variable and display its value on PORTB.



Task 4:
Delay subroutine with variable delay time

Modify the program in task 5 in Lab 1 to a general delay routine that can be called from other programs. It should be named **wait_milliseconds.** The number of milliseconds should be transferred to register pair R24, R25.

Using subroutines:

When calling subroutines in a program, the return jump address will be stored on the stack. That is handled automatically by the processor. But for this to work the Stack Pointer has to be initialized in the program. It can for example be done by the following instructions in the initialization part of the program.

```
.equ    SPH     =0x3E
.equ    SPL     =0x3D
.equ    RAMEND  =0x21FF          (Atmega16:  RAMEND = 0x45F)
```

; initialize the Stack Pointer (SP) to the highest address in SRAMs (RAMEND)

```
ldi     r16, HIGH(RAMEND)    ; MSB part av address to RAMEND
out     SPH, r16             ; store in SPH
ldi     r16, LOW(RAMEND)     ; LSB part av address to RAMEND
out     SPL, r16             ; store in SPL
```

RAMEND, SPH and SPL is defined in the file m2560def.inc for the ATMega2560 CPU which can therefore be included in the program with the following line. The three .EQU states above will thus not be needed:

```
.include "m2560def.inc"          ; Define names for ATmega2560
```

You can also write the correct address instead of the SPH, SPL and RAMEND as follows:

```
ldi r20, HIGH(0x21FF)    ; R20 = high byte of RAMEND address, 0x21
out 0x3E,R20             ; SPH = high part of pointer to STACK
ldi R20, LOW(0x21FF)     ; R20 = low byte of RAMEND address, 0xFF
out 0x3D,R20             ; SPL = low part of pointer to STACK
```

or like this:

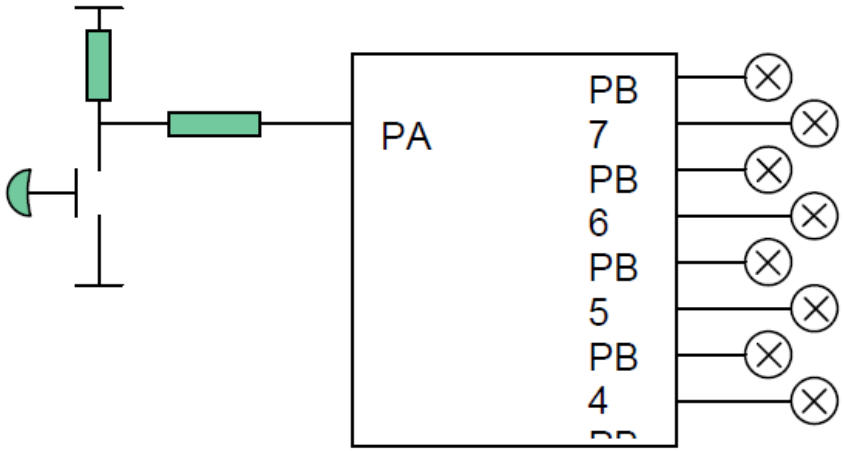
```
ldi  r16, 0x21     ; r16 = high byte of RAMEND address, 0x21
out  0x3E,r16      ; 0x3E = SPH = high part of pointer to STACK
ldi  r16,0xFF      ; r16 = low byte of RAMEND address, 0xFF
out  0x3D,r16      ; 0x3D = SPL = low part of pointer to STACK
```

Lab 2, task 1..

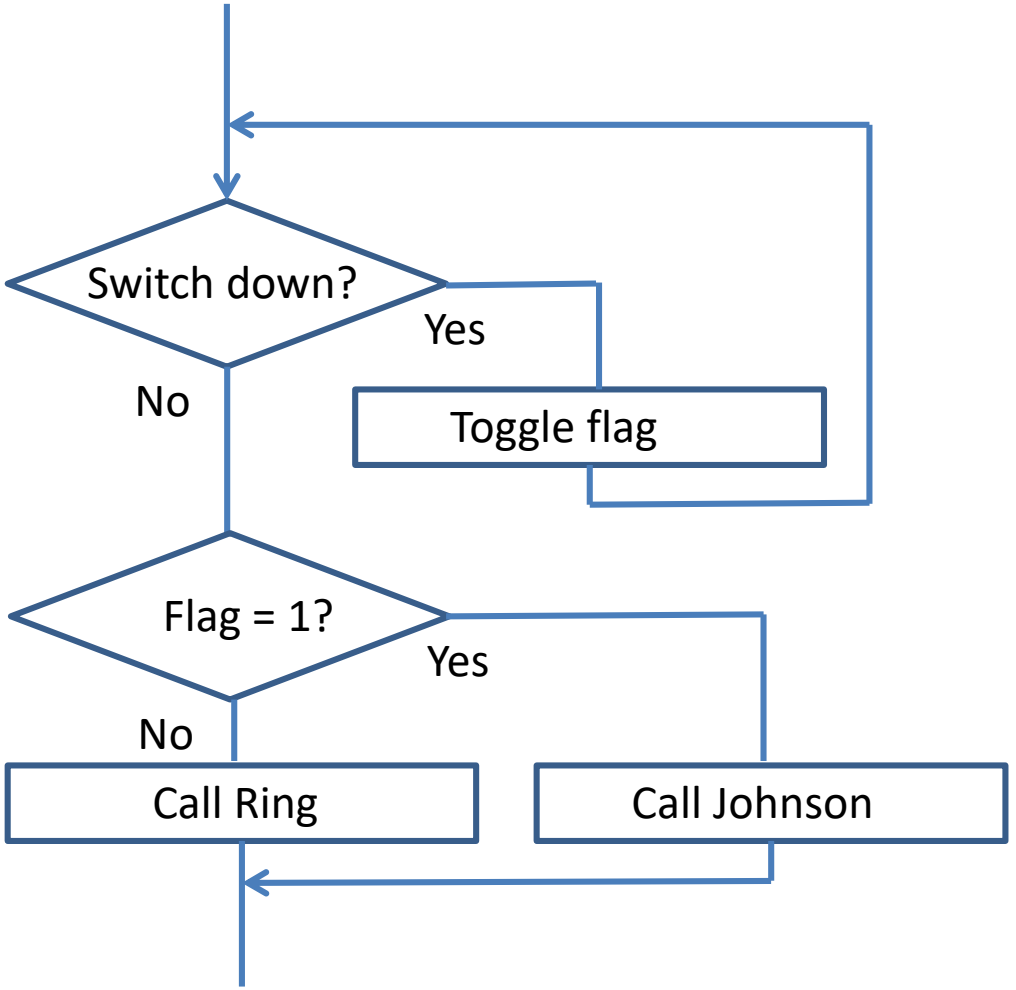
Task 1:

Switch – Ring counter / Johnson counter

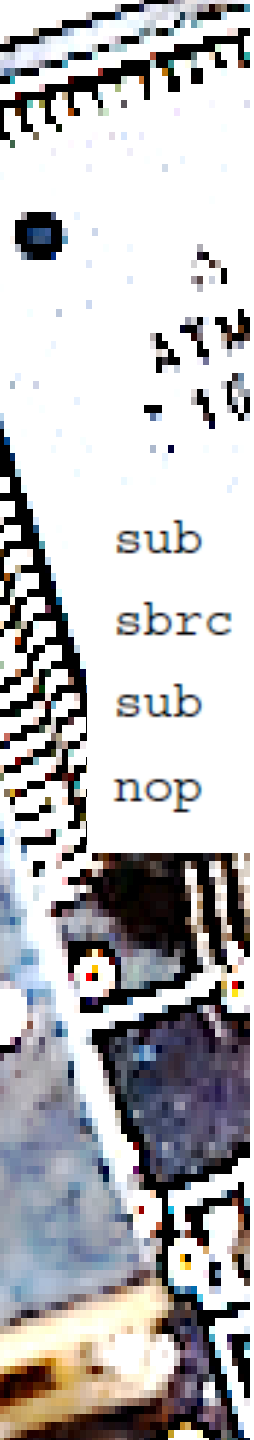
Write a program which switch between Ring counter and Johnson counter. You should not use Interrupt in this lab. The pushbutton must be checked frequently, so there is no delay between the button is pressed and the change between Ring/Johnson. Use SW0 (PA0) for the button. Each time you press the button, the program should change counter.



Check if switch is pressed.

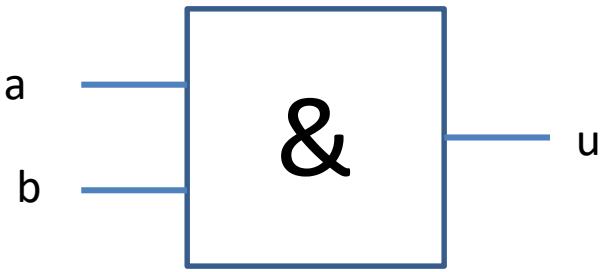


Example:



```
sub   r0,r1     ; Subtract r1 from r0
sbrc  r0,7      ; Skip if bit 7 in r0 cleared
sub   r0,r1     ; Only executed if bit 7 in r0 not cleared
nop             ; Continue (do nothing)
```

Example: ANDI r16, 0b0001000



a	b	u
0	0	0
0	1	0
1	0	0
1	1	1

R16	1	1	0	1	1	0	0	1
K	0	0	0	1	0	0	0	0
Result:	0	0	0	1	0	0	0	0

AND – Logical AND

Description:

Performs the logical AND between the contents of register Rd and register Rr and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd • Rr

	Syntax:	Operands:	Program Counter:
(i)	AND Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0010	00rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S: N ⊕ V, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7}$ •$\overline{R6}$ •$\overline{R5}$ •$\overline{R4}$ •$\overline{R3}$• $\overline{R2}$ •$\overline{R1}$ •$\overline{R0}$
Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

ANDI – Logical AND with Immediate

Description:

Performs the logical AND between the contents of register Rd and a constant and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd • K

	Syntax:	Operands:	Program Counter:
(i)	ANDI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

0111	KKKK	dddd	KKKK

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	0	⇔	⇔	–

S: N ⊕ V, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

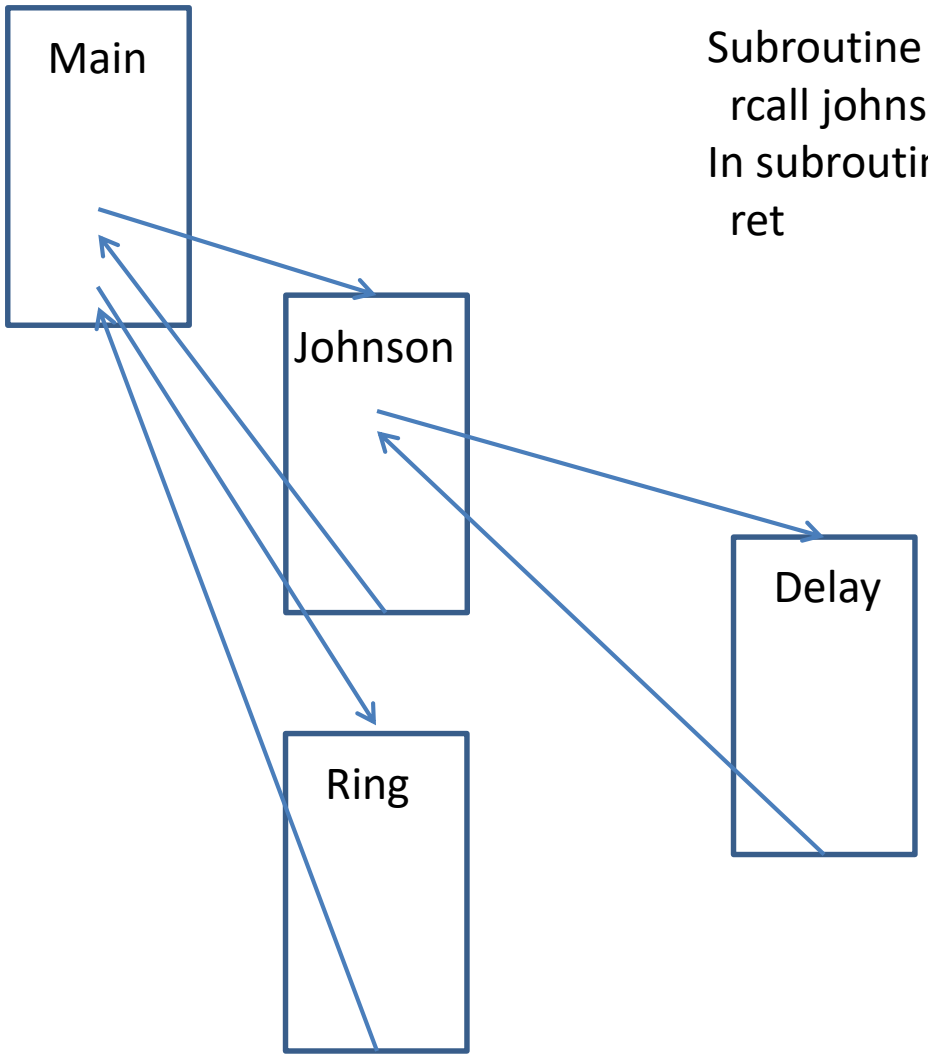
R (Result) equals Rd after the operation.

Example:

```
andi  r17,$0F   ; Clear upper nibble of r17
andi  r18,$10   ; Isolate bit 4 in r18
andi  r19,$AA   ; Clear odd bits of r19
```

Words: 1 (2 bytes)

Program structure, example.



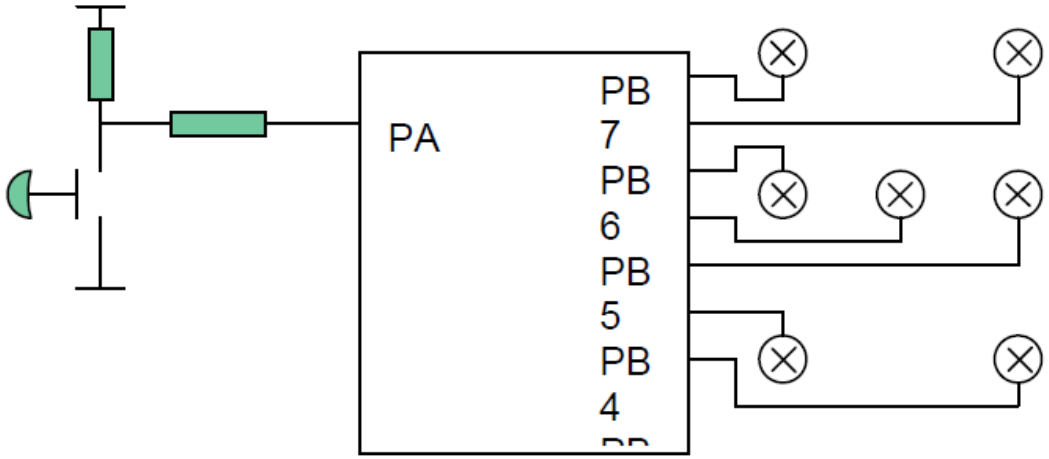
Subroutine call, example:
rcall johnsson
In subroutine:
ret

Question:
Where to check the switch?

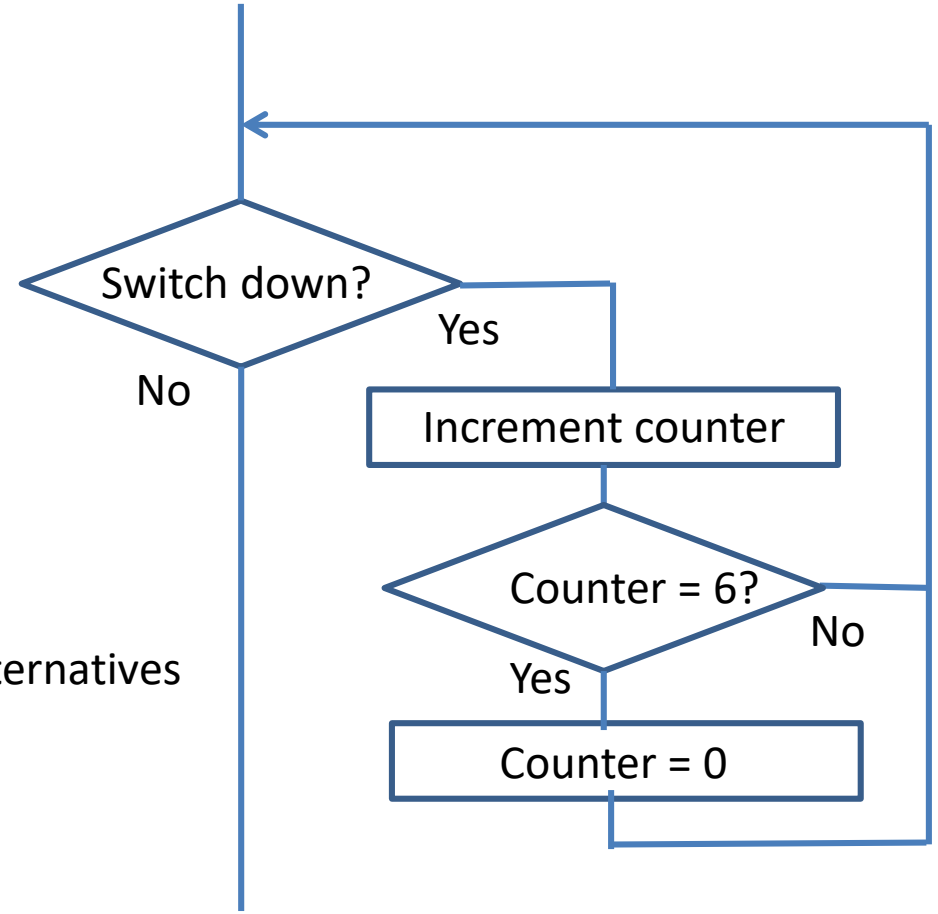
Lab 2, task 2.

Task 2:
Electronic dice

You should create an electronic dice. Think of the LEDs placed as in the picture below. The number 1 to 6 should be generated randomly. You could use the fact that the time you press the button varies in length.

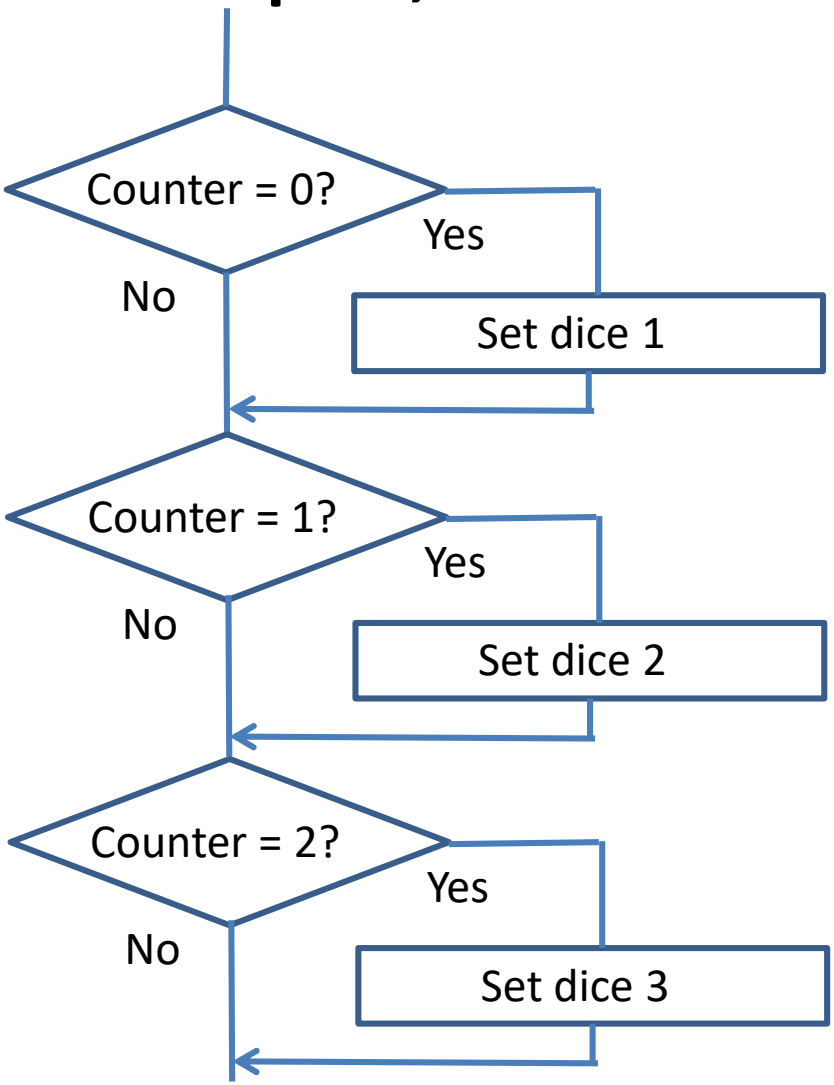


Example, random generator.



Important:
Check that all alternatives are equal.

Example, dice.



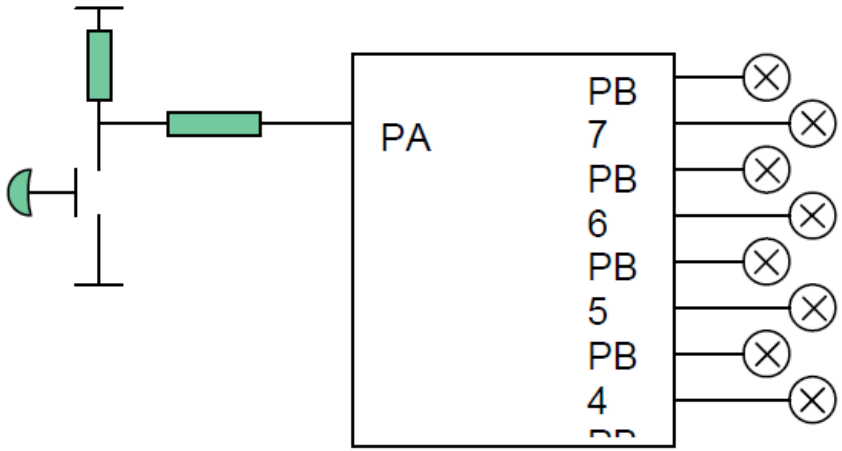
Und so weiter…

Lab 2, task 3.

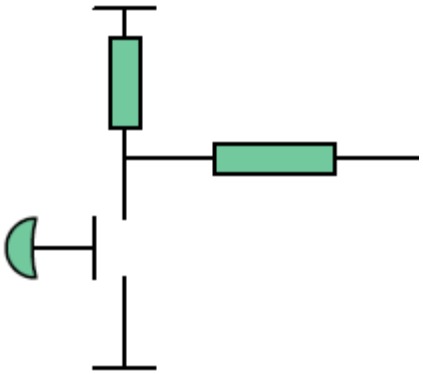
Task 3:
Change counter

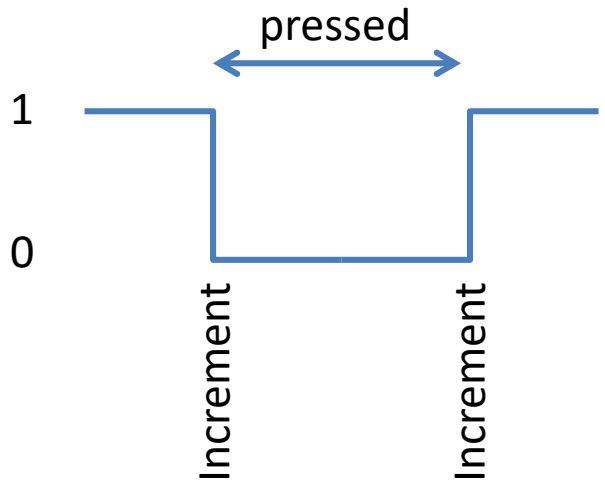
Skriv ett program som klarar av att beräkna förändringarna på en strömställare. Som förändring räknar vi när strömställaren SW0 går från 0 till 1 respektive 1 till 0, vi räknar alltså positiva och negativa flanker. Vi räknar förändringarna i en byte variabel och lägger ut dess värde på PORTB.

Write a program that is able to calculate the changes on a switch. As a change we expect when the switch SW0 goes from 0 to 1 and from 1 to 0, we expect therefore positive and negative edges. We calculate the changes in a byte variable and puts out its value on PORTB.

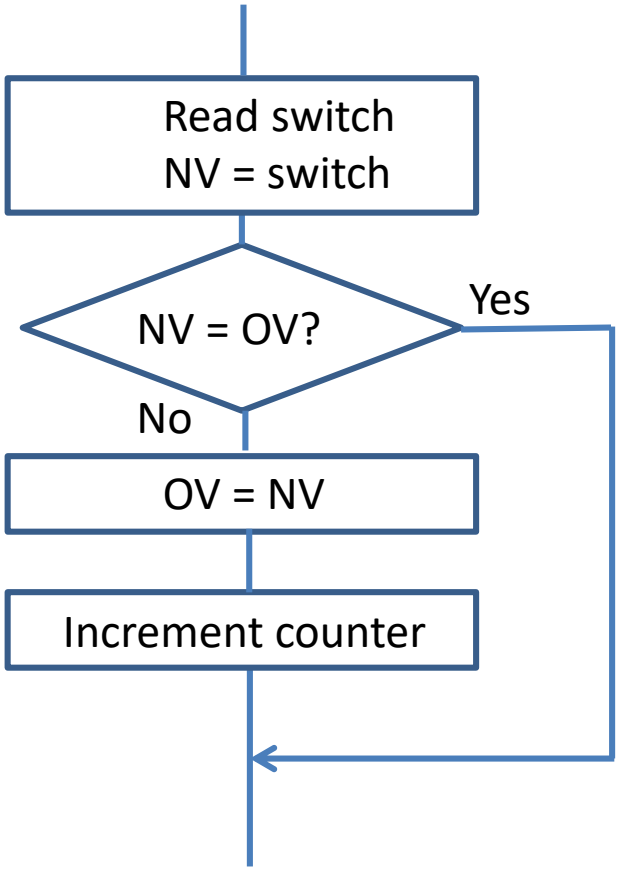


Switch.





Example, counter.



Lab 2, task 4.

Task 4:

Delay subroutine with variable delay time

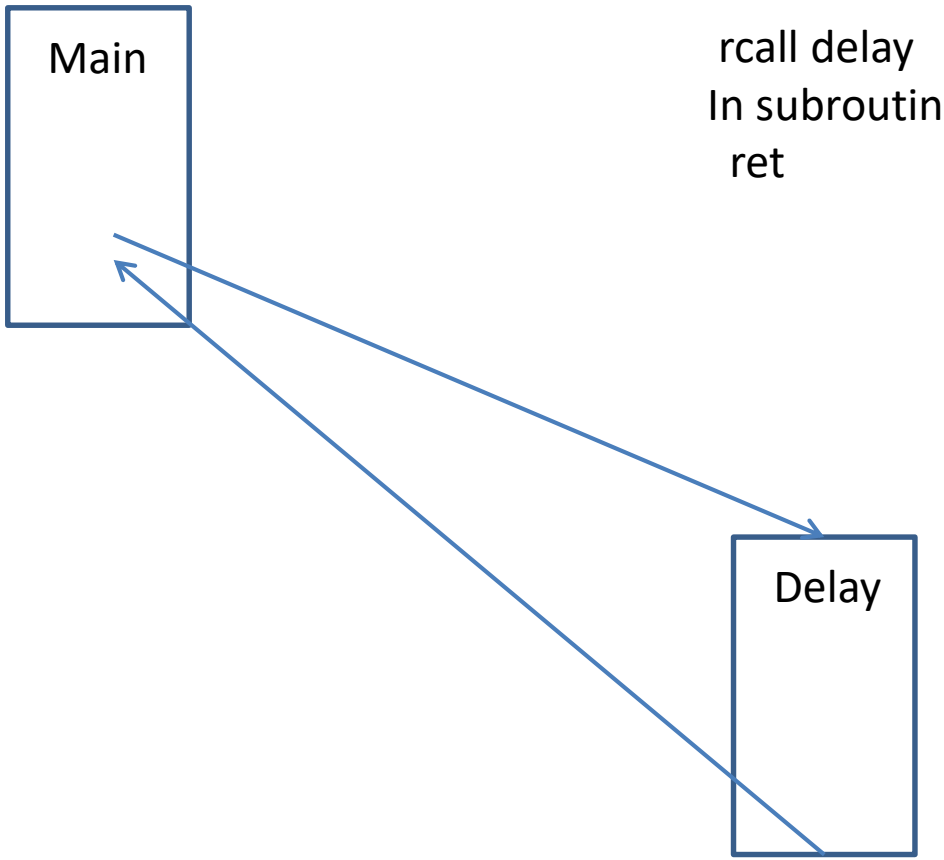
Modify the program in task 5 in Lab 1 to a general delay routine that can be called from other programs. It should be named **wait_milliseconds.** The number of milliseconds should be transferred to register pair R24, R25.

Example, C language:

num_day = function3(number);

function3() is a routine or function, *number* is a variable transferred to the subroutine.
The variable *num_day* will get the return value from the subroutine *function3().*

Program structure, example.



rcall delay
In subroutine:
ret

Question:
Where to check the switch?

RCALL – Relative Call to Subroutine

Description:

Relative call to an address within PC - 2K + 1 and PC + 2K (words). The return address (the instruction after the RCALL) is stored onto the Stack. (See also CALL). In the assembler, labels are used instead of relative operands. For AVR microcontrollers with Program memory not exceeding 4K words (8K bytes) this instruction can address the entire memory from every address location. The Stack Pointer uses a post-decrement scheme during RCALL.

Operation:

(i) PC ← PC + k + 1 Devices with 16 bits PC, 128K bytes Program memory maximum.
(ii) PC ← PC + k + 1 Devices with 22 bits PC, 8M bytes Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:
(i)	RCALL k	-2K ≤ k < 2K	PC ← PC + k + 1	STACK ← PC + 1 SP ← SP - 2 (2 bytes, 16 bits)
(ii)	RCALL k	-2K ≤ k < 2K	PC ← PC + k + 1	STACK ← PC + 1 SP ← SP - 3 (3 bytes, 22 bits)

16-bit Opcode:

1101	kkkk	kkkk	kkkk

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
            rcall   routine     ; Call subroutine
            ...
routine:    push    r14         ; Save r14 on the Stack
            ...
```

RET – Return from Subroutine

Description:

Returns from subroutine. The return address is loaded from the STACK. The Stack Pointer uses a pre-increment schem during RET.

Operation:

(i) PC(15:0) ← STACKDevices with 16 bits PC, 128K bytes Program memory maximum.

(ii) PC(21:0) ← STACKDevices with 22 bits PC, 8M bytes Program memory maximum.

	Syntax:	Operands:	Program Counter:	Stack:
(i)	RET	None	See Operation	SP←SP + 2, (2bytes,16 bits)
(ii)	RET	None	See Operation	SP←SP + 3, (3bytes,22 bits)

16-bit Opcode:

1001	0101	0000	1000

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
                call    routine     ; Call subroutine
                ...
routine:        push    r14         ; Save r14 on the Stack
                ...
                pop     r14         ; Restore r14
                ret                 ; Return from subroutine
```

PUSH – Push Register on Stack

Description:

This instruction stores the contents of register Rr on the STACK. The Stack Pointer is post-decremented by 1 after the PUSH.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) STACK ← Rr

	Syntax:	Operands:
(i)	PUSH Rr	0 ≤ r ≤ 31

Program Counter: PC ← PC + 1

Stack: SP ← SP - 1

16-bit Opcode:

1001	001d	dddd	1111

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
            call    routine ; Call subroutine
            ...
routine:    push    r14     ; Save r14 on the Stack
            push    r13     ; Save r13 on the Stack
            ...
            pop     r13     ; Restore r13
            pop     r14     ; Restore r14
            ret             ; Return from subroutine
```

Words: 1 (2 bytes)

POP – Pop Register from Stack

Description:

This instruction loads register Rd with a byte from the STACK. The Stack Pointer is pre-incremented by 1 before the POP.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

	Operation:	Syntax:	Operands:	Program Counter:	Stack:
(i)	Rd ← STACK	POP Rd	0 ≤ d ≤ 31	PC ← PC + 1	SP ← SP + 1

16-bit Opcode:

1001	000d	dddd	1111

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
                call    routine   ; Call subroutine
                ...
routine:        push    r14       ; Save r14 on the Stack
                push    r13       ; Save r13 on the Stack
                ...
                pop     r13       ; Restore r13
                pop     r14       ; Restore r14
                ret               ; Return from subroutine
```

Words: 1 (2 bytes)

Instruction Set Nomenclature

Status Register (SREG)

SREG:	Status Register
C:	Carry Flag
Z:	Zero Flag
N:	Negative Flag
V:	Two's complement overflow indicator
S:	N ⊕ V, For signed tests
H:	Half Carry Flag
T:	Transfer bit used by BLD and BST instructions
I:	Global Interrupt Enable/Disable Flag

Registers and Operands

Rd:	Destination (and source) register in the Register File
Rr:	Source register in the Register File
R:	Result after instruction is executed
K:	Constant data
k:	Constant address
b:	Bit in the Register File or I/O Register (3-bit)
s:	Bit in the Status Register (3-bit)
X,Y,Z:	Indirect Address Register
	(X=R27:R26, Y=R29:R28 and Z=R31:R30)
A:	I/O location address
q:	Displacement for direct addressing (6-bit)

ATMEL®

8-bit AVR® Instruction Set

LDI – Load Immediate

Description:

Loads an 8 bit constant directly to register 16 to 31.

	Operation:
(i)	Rd ← K

	Syntax:	Operands:	Program Counter:
(i)	LDI Rd,K	16 ≤ d ≤ 31, 0 ≤ K ≤ 255	PC ← PC + 1

16-bit Opcode:

1110	KKKK	dddd	KKKK

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
clr   r31       ; Clear Z high byte
ldi   r30,$F0   ; Set Z low byte to $F0
lpm             ; Load constant from Program
                ; memory pointed to by Z
```

Words: 1 (2 bytes)
Cycles: 1

S: N ⊕ V, For signed tests.

V: 0
Cleared

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet R6 \bullet \overline{R5} \bullet \overline{R4} \bullet R3 \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

R (Result) equals Rd after the operation.

Example:

```
andi  r17,$0F   ; Clear upper nibble of r17
andi  r18,$10   ; Isolate bit 4 in r18
andi  r19,$AA   ; Clear odd bits of r19
```

Words: 1 (2 bytes)
Cycles: 1

BLD – Bit Load from the T Flag in SREG to a Bit in Register

Description:

Copies the T Flag in the SREG (Status Register) to bit b in register Rd.

Operation:

(i) Rd(b) ← T

	Syntax:	Operands:	Program Counter:
(i)	BLD Rd,b	0 ≤ d ≤ 31, 0 ≤ b ≤ 7	PC ← PC + 1

16 bit Opcode:

1111	100d	dddd	0bbb

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
                ; Copy bit
    bst   r1,2  ; Store bit 2 of r1 in T Flag
    bld   r0,4  ; Load T Flag into bit 4 of r0
```

Words: 1 (2 bytes)
Cycles: 1

BRCC – Branch if Carry Cleared

Description:

Conditional relative branch. Tests the Carry Flag (C) and branches relatively to PC if C is cleared. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 0,k).

Operation:

(i) If C = 0 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRCC k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k000

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
            add   r22,r23     ; Add r23 to r22
            brcc  nocarry     ; Branch if carry cleared
            ...
nocarry:    nop               ; Branch destination (do nothing)
```

Words: 1 (2 bytes)

Cycles: 1 if condition is false
2 if condition is true

BRCS – Branch if Carry Set

Description:

Conditional relative branch. Tests the Carry Flag (C) and branches relatively to PC if C is set. This instr atively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC in two's complement form. (Equivalent to instruction BRBS 0,k).

Operation:

(i) If C = 1 then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRCS k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	00kk	kkkk	k000

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
          cpi   r26,$56   ; Compare r26 with $56
          brcs  carry     ; Branch if carry set
          ...
carry:    nop             ; Branch destination (do nothing)
```

Words: 1 (2 bytes)
Cycles: 1 if condition is false
2 if condition is true

BREQ – Branch if Equal

Description:

Conditional relative branch. Tests the Zero Flag (Z) and branches relatively to PC if Z is set. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the unsigned or signed binary number represented in Rd was equal to the unsigned or signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBS 1,k).

Operation:

(i) If Rd = Rr (Z = 1) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BREQ k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	00kk	kkkk	k001

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

```
          cp    r1,r0     ; Compare registers r1 and r0
          breq  equal     ; Branch if registers equal
          ...
equal:    nop             ; Branch destination (do nothing)
```

BRGE – Branch if Greater or Equal (Signed)

Description:

Conditional relative branch. Tests the Signed Flag (S) and branches relatively to PC if S is cleared. If the instruction is executed immediately after any of the instructions CP, CPI, SUB or SUBI, the branch will occur if and only if the signed binary number represented in Rd was greater than or equal to the signed binary number represented in Rr. This instruction branches relatively to PC in either direction (PC - 63 ≤ destination ≤ PC + 64). The parameter k is the offset from PC and is represented in two's complement form. (Equivalent to instruction BRBC 4,k).

Operation:

(i) If Rd ≥ Rr (N ⊕ V = 0) then PC ← PC + k + 1, else PC ← PC + 1

	Syntax:	Operands:	Program Counter:
(i)	BRGE k	-64 ≤ k ≤ +63	PC ← PC + k + 1 PC ← PC + 1, if condition is false

16-bit Opcode:

1111	01kk	kkkk	k100

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	–	–	–	–	–

Example:

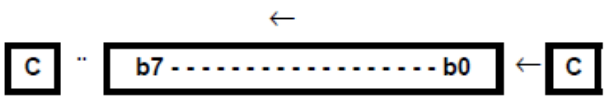
```
            cp    r11,r12    ; Compare registers r11 and r12
            brge  greateq    ; Branch if r11 ≥ r12 (signed)
            ...
greateq:    nop              ; Branch destination (do nothing)
```

ROL – Rotate Left trough Carry

Description:

Shifts all bits in Rd one place to the left. The C Flag is shifted into bit 0 of Rd. Bit 7 is shifted into the C Flag. This operation, combined with LSL, effectively multiplies multi-byte signed and unsigned values by two.

Operation:



	Syntax:	Operands:	Program Counter:
(i)	ROL Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode: (see ADC Rd,Rd)

0001	11dd	dddd	dddd

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: Rd3

S: N ⊕ V, For signed tests.

V: N ⊕ C (For N and C after the shift)

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7}$• $\overline{R6}$ •$\overline{R5}$• $\overline{R4}$• $\overline{R3}$ •$\overline{R2}$• $\overline{R1}$• $\overline{R0}$
Set if the result is $00; cleared otherwise.

C: Rd7
Set if, before the shift, the MSB of Rd was set; cleared otherwise.

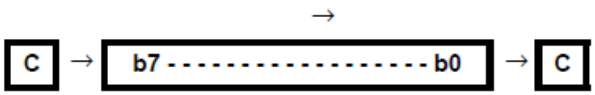
R (Result) equals Rd after the operation.

ROR – Rotate Right through Carry

Description:

Shifts all bits in Rd one place to the right. The C Flag is shifted into bit 7 of Rd. Bit 0 is shifted into the C Flag. This operation, combined with ASR, effectively divides multi-byte signed values by two. Combined with LSR it effectively divides multi-byte unsigned values by two. The Carry Flag can be used to round the result.

Operation:



	Syntax:	Operands:	Program Counter:
(i)	ROR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0111

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S: N ⊕ V, For signed tests.

V: N ⊕ C (For N and C after the shift)

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: Rd0
Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

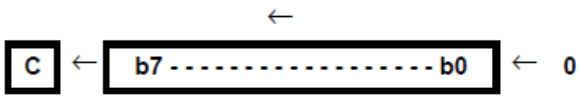
LSL – Logical Shift Left

Description:

Shifts all bits in Rd one place to the left. Bit 0 is cleared. Bit 7 is loaded into the C Flag of the SREG. This operation effectively multiplies signed and unsigned values by two.

Operation:

(i)



	Syntax:	Operands:	Program Counter:
(i)	LSL Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode: (see ADD Rd,Rd)

0000	11dd	dddd	dddd

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: Rd3

S: N ⊕ V, For signed tests.

V: N ⊕ C (For N and C after the shift)

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

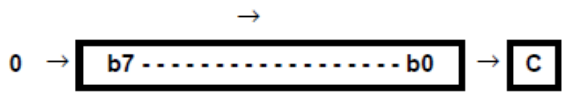
C: Rd7
Set if, before the shift, the MSB of Rd was set; cleared otherwise.

LSR – Logical Shift Right

Description:

Shifts all bits in Rd one place to the right. Bit 7 is cleared. Bit 0 is loaded into the C Flag of the SREG. This operation effectively divides an unsigned value by two. The C Flag can be used to round the result.

Operation:



	Syntax:	Operands:	Program Counter:
(i)	LSR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0110

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	0	⇔	⇔

S: N ⊕ V, For signed tests.

V: N ⊕ C (For N and C after the shift)

N: 0

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: Rd0
Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

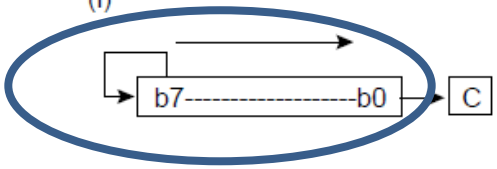
ASR – Arithmetic Shift Right

Description:

Shifts all bits in Rd one place to the right. Bit 7 is held constant. Bit 0 is loaded into the C Flag of the SREG. This operation effectively divides a signed value by two without changing its sign. The Carry Flag can be used to round the result.

Operation:

(i)



	Syntax:	Operands:	Program Counter:
(i)	ASR Rd	0 ≤ d ≤ 31	PC ← PC + 1

16-bit Opcode:

1001	010d	dddd	0101

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S: N ⊕ V, For signed tests.

V: N ⊕ C (For N and C after the shift)

N: R7
Set if MSB of the result is set; cleared otherwise.

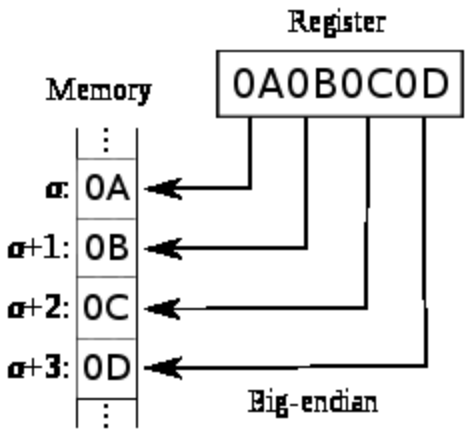
Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

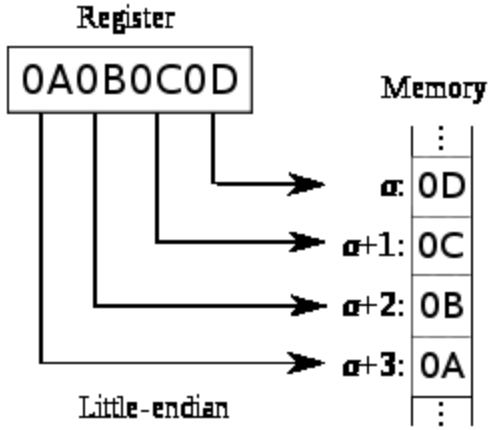
C: Rd0
Set if, before the shift, the LSB of Rd was set; cleared otherwise.

R (Result) equals Rd after the operation.

Big-Endian – Little-Endian

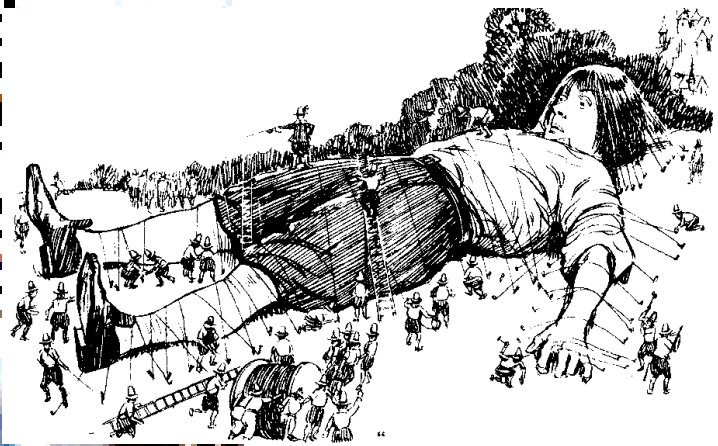
(Pictures from Wikipedia)

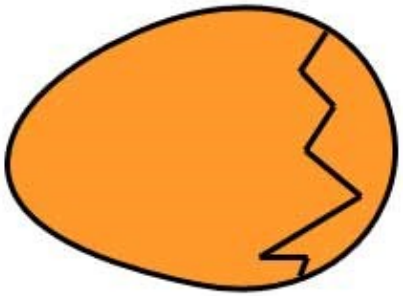




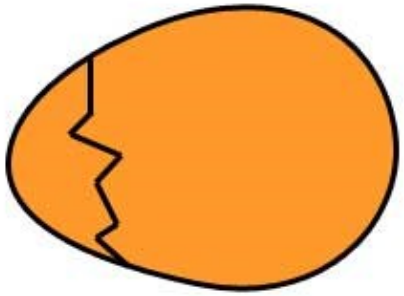
Big Endian – Little Endian

The term big endian originally comes from Jonathan Swift's satirical novel Gulliver's Travels by way of Danny Cohen in 1980.[1]





BIG ENDIAN - The way people always broke their eggs in the Lilliput land



LITTLE ENDIAN - The way the king then ordered the people to break their eggs

Picture from: http://flickeringtubelight.net/blog/2004/05/big-endian-and-little-endian-storage-schemes-how-to-remember/

- **Atmel AVR Endianess Cheat Sheet**:

Example Value: 0x0A0B

All 8-bt AVRs: Little Endian

in Memory 0x0B 0x0A (*adr: 0x0B, *(adr+1): 0x0A)

(AVR32: Big Endian in Memory 0x0A 0x0B)

SBIW – Subtract Immediate from Word

Description:

Subtracts an immediate value (0-63) from a register pair and places the result in the register pair. This instruction operate on the upper four register pairs, and is well suited for operations on the Pointer Registers.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) Rd+1:Rd ← Rd+1:Rd - K

	Syntax:	Operands:	Program Counter:
(i)	SBIW Rd+1:Rd,K	d ∈ {24,26,28,30}, 0 ≤ K ≤ 63	PC ← PC + 1

16-bit Opcode:

1001	0111	KKdd	KKKK

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S: N ⊕ V, For signed tests.

V: Rdh7 •$\overline{R15}$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R15
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R15}$• $\overline{R14}$ •$\overline{R13}$ •$\overline{R12}$ •$\overline{R11}$• $\overline{R10}$• $\overline{R9}$• $\overline{R8}$• $\overline{R7}$• $\overline{R6}$ •$\overline{R5}$• $\overline{R4}$• $\overline{R3}$ •$\overline{R2}$• $\overline{R1}$• $\overline{R0}$
Set if the result is $0000; cleared otherwise.

ADIW – Add Immediate to Word

Description:

Adds an immediate value (0 - 63) to a register pair and places the result in the register pair. This instruction operates on upper four register pairs, and is well suited for operations on the pointer registers.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) Rd+1:Rd ← Rd+1:Rd + K

	Syntax:	Operands:	Program Counter:
(i)	ADIW Rd+1:Rd,K	d ∈ {24,26,28,30}, 0 ≤ K ≤ 63	PC ← PC + 1

16-bit Opcode:

1001	0110	KKdd	KKKK

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S: N ⊕ V, For signed tests.

V: $\overline{Rdh7} \bullet R15$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R15
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $0000; cleared otherwise.

C: $\overline{R15} \bullet Rdh7$
Set if there was carry from the MSB of the result; cleared otherwise.

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S: $N \oplus V$, For signed tests.

V: $\overline{Rdh7} \bullet R15$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R15
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $0000; cleared otherwise.

C: $\overline{R15} \bullet Rdh7$
Set if there was carry from the MSB of the result; cleared otherwise.

R (Result) equals Rdh:Rdl after the operation (Rdh7-Rdh0 = R15-R8, Rdl7-Rdl0=R7-R0).

Example:

```
adiw  r25:24,1  ; Add 1 to r25:r24
adiw  ZH:ZL,63  ; Add 63 to the Z-pointer(r31:r30)
```

Words: 1 (2 bytes)
Cycles: 2

SBIW – Subtract Immediate from Word

Description:

Subtracts an immediate value (0-63) from a register pair and places the result in the register pair. This instruction operate on the upper four register pairs, and is well suited for operations on the Pointer Registers.

This instruction is not available in all devices. Refer to the device specific instruction set summary.

Operation:

(i) Rd+1:Rd ← Rd+1:Rd - K

	Syntax:	Operands:	Program Counter:
(i)	SBIW Rd+1:Rd,K	d ∈ {24,26,28,30}, 0 ≤ K ≤ 63	PC ← PC + 1

16-bit Opcode:

1001	0111	KKdd	KKKK

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S: N ⊕ V, For signed tests.

V: Rdh7 •$\overline{R15}$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R15
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R15}$• $\overline{R14}$ •$\overline{R13}$ •$\overline{R12}$ •$\overline{R11}$• $\overline{R10}$• $\overline{R9}$• $\overline{R8}$• $\overline{R7}$• $\overline{R6}$ •$\overline{R5}$• $\overline{R4}$• $\overline{R3}$ •$\overline{R2}$• $\overline{R1}$• $\overline{R0}$
Set if the result is $0000; cleared otherwise.

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	–	⇔	⇔	⇔	⇔	⇔

S: $N \oplus V$, For signed tests.

V: $Rdh7 \bullet \overline{R15}$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R15
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R15} \bullet \overline{R14} \bullet \overline{R13} \bullet \overline{R12} \bullet \overline{R11} \bullet \overline{R10} \bullet \overline{R9} \bullet \overline{R8} \bullet \overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $0000; cleared otherwise.

C: $R15 \bullet \overline{Rdh7}$
Set if the absolute value of K is larger than the absolute value of Rd; cleared otherwise.

R (Result) equals Rdh:Rdl after the operation (Rdh7-Rdh0 = R15-R8, Rdl7-Rdl0=R7-R0).

Example:

```
sbiw    r25:r24,1   ; Subtract 1 from r25:r24
sbiw    YH:YL,63    ; Subtract 63 from the Y-pointer(r29:r28)
```

Words: 1 (2 bytes)
Cycles: 2

ADD – Add without Carry

Description:

Adds two registers without the C Flag and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd + Rr

	Syntax:	Operands:	Program Counter:
(i)	ADD Rd,Rr	$0 \le d \le 31$, $0 \le r \le 31$	PC ← PC + 1

16-bit Opcode:

0000	11rd	dddd	rrrr

Status Register (SREG) and Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: $Rd3 \bullet Rr3 + Rr3 \bullet \overline{R3} + \overline{R3} \bullet Rd3$
Set if there was a carry from bit 3; cleared otherwise

S: N ⊕ V, For signed tests.

V: $Rd7 \bullet Rr7 \bullet \overline{R7} + \overline{Rd7} \bullet \overline{Rr7} \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: $Rd7 \bullet Rr7 + Rr7 \bullet \overline{R7} + \overline{R7} \bullet Rd7$
Set if there was carry from the MSB of the result; cleared otherwise.

R (Result) equals Rd after the operation.

ADC – Add with Carry

Description:

Adds two registers and the contents of the C Flag and places the result in the destination register Rd.

Operation:

(i) Rd ← Rd + Rr + C

	Syntax:	Operands:	Program Counter:
(i)	ADC Rd,Rr	0 ≤ d ≤ 31, 0 ≤ r ≤ 31	PC ← PC + 1

16-bit Opcode:

0001	11rd	dddd	rrrr

Status Register (SREG) Boolean Formula:

I	T	H	S	V	N	Z	C
–	–	⇔	⇔	⇔	⇔	⇔	⇔

H: $Rd3 \bullet Rr3 + Rr3 \bullet \overline{R3} + \overline{R3} \bullet Rd3$
Set if there was a carry from bit 3; cleared otherwise

S: N ⊕ V, For signed tests.

V: $Rd7 \bullet Rr7 \bullet \overline{R7} + \overline{Rd7} \bullet \overline{Rr7} \bullet R7$
Set if two's complement overflow resulted from the operation; cleared otherwise.

N: R7
Set if MSB of the result is set; cleared otherwise.

Z: $\overline{R7} \bullet \overline{R6} \bullet \overline{R5} \bullet \overline{R4} \bullet \overline{R3} \bullet \overline{R2} \bullet \overline{R1} \bullet \overline{R0}$
Set if the result is $00; cleared otherwise.

C: $Rd7 \bullet Rr7 + Rr7 \bullet \overline{R7} + \overline{R7} \bullet Rd7$
Set if there was carry from the MSB of the result; cleared otherwise.

Addition, register pair

Example:

```
                        ; Add R1:R0 to R3:R2
    add   r2,r0         ; Add low byte
    adc   r3,r1         ; Add with carry high byte
```

Words: 1 (2 bytes)

Cycles: 1

Section 4

AVR Assembler User Guide

4.1 Introduction

Welcome to the Atmel *AVR* Assembler. This manual describes the usage of the Assembler. The Assembler covers the whole range of microcontrollers in the AT90S family.

The Assembler translates assembly source code into object code. The generated object code can be used as input to a simulator or an emulator such as the Atmel *AVR* In-Circuit Emulator. The Assembler also generates a PROMable code and an optional EEPROM file which can be programmed directly into the program memory and EEPROM memory of an *AVR* microcontroller.

The Assembler generates fixed code allocations, consequently no linking is necessary.

The Assembler runs under Microsoft Windows 3.11, Microsoft Windows95 and Microsoft Windows NT. In addition, there is an MS-DOS command line version. The Windows version of the program contains an on-line help function covering most of this document.

The instruction set of the *AVR* family of microcontrollers is only briefly described, refer to the *AVR* Data Book (also available on CD-ROM) in order to get more detailed knowledge of the instruction set for the different microcontrollers.

To get quickly started, the Quick-Start Tutorial is an easy way to get familiar with the Atmel *AVR* Assembler.

4.5 Assembler directives

The Assembler supports a number of directives. The directives are not translated directly into opcodes. Instead, they are used to adjust the location of the program in memory, define macros, initialize memory and so on. An overview of the directives is given in the following table.

Summary of directives:

Directive	Description
BYTE	Reserve byte to a variable
CSEG	Code Segment
DB	Define constant byte(s)
DEF	Define a symbolic name on a register
DEVICE	Define which device to assemble for
DSEG	Data Segment
DW	Define constant word(s)
ENDMACRO	End macro
EQU	Set a symbol equal to an expression
ESEG	EEPROM Segment
EXIT	Exit from file
INCLUDE	Read source from another file
LIST	Turn listfile generation on
LISTMAC	Turn macro expansion on
MACRO	Begin macro
NOLIST	Turn listfile generation off
ORG	Set program origin
SET	Set a symbol to an expression

Note: *All directives must be preceded by a period.*

.EQU

4.5.9 EQU - Set a symbol equal to an expression

The EQU directive assigns a value to a label. This label can then be used in later expressions. A label assigned to a value by the EQU directive is a constant and can not be changed or redefined.

Syntax:

```
.EQU label = expression
```

Example:

```
.EQU io_offset = 0x23
.EQU porta = io_offset + 2
.CSEG                           ; Start code segment
        clr     r2              ; Clear register 2
        out     porta,r2        ; Write to Port A
```

.CSEG

4.5.2 CSEG - Code Segment

The CSEG directive defines the start of a Code Segment. An Assembler file can consist of several Code Segments, which are concatenated into one Code Segment when assembled. The BYTE directive can not be used within a Code Segment. The default segment type is Code. The Code Segments have their own location counter which is a word counter. The ORG directive (see description later in this document) can be used to place code and constants at specific locations in the Program memory. The directive does not take any parameters.

Syntax:

```
.CSEG
```

Example:

```
.DSEG                           ; Start data segment
        vartab: .BYTE 4         ; Reserve 4 bytes in SRAM
.CSEG                           ; Start code segment
        const:  .DW 2           ; Write 0x0002 in prog.mem.
        mov r1,r0               ; Do something
```

.DEF

4.5.4 DEF - Set a symbolic name on a register

The DEF directive allows the registers to be referred to through symbols. A defined symbol can be used in the rest of the program to refer to the register it is assigned to. A register can have several symbolic names attached to it. A symbol can be redefined later in the program.

Syntax:

```
.DEF Symbol=Register
```

Example:

```
.DEF temp=R16
.DEF ior=R0
.CSEG
        ldi     temp,0xf0       ; Load 0xf0 into temp register
        in      ior,0x3f        ; Read SREG into ior register
        eor     temp,ior        ; Exclusive or temp and ior
```

Assembler commands:

- Example:

```
.def Temp = r16
.equ PINA = 0x00
.equ DDRA = 0x01
.cseg
```

4.5 Assembler directives

The Assembler supports a number of directives. The directives are not translated directly into opcodes. Instead, they are used to adjust the location of the program in memory, define macros, initialize memory and so on. An overview of the directives is given in the following table.

Summary of directives:

Directive	Description
BYTE	Reserve byte to a variable
CSEG	Code Segment
DB	Define constant byte(s)
DEF	Define a symbolic name on a register
DEVICE	Define which device to assemble for
DSEG	Data Segment
DW	Define constant word(s)
ENDMACRO	End macro
EQU	Set a symbol equal to an expression
ESEG	EEPROM Segment
EXIT	Exit from file
INCLUDE	Read source from another file
LIST	Turn listfile generation on
LISTMAC	Turn macro expansion on
MACRO	Begin macro
NOLIST	Turn listfile generation off
ORG	Set program origin
SET	Set a symbol to an expression

Note: *All directives must be preceded by a period.*

Address	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Page
0x1A (0x3A)	TIFR5	-	-	ICF5	-	OCF5C	OCF5B	OCF5A	TOV5	166
0x19 (0x39)	TIFR4	-	-	ICF4	-	OCF4C	OCF4B	OCF4A	TOV4	167
0x18 (0x38)	TIFR3	-	-	ICF3	-	OCF3C	OCF3B	OCF3A	TOV3	167
0x17 (0x37)	TIFR2	-	-	-	-	-	OCF2B	OCF2A	TOV2	193
0x16 (0x36)	TIFR1	-	-	ICF1	-	OCF1C	OCF1B	OCF1A	TOV1	167
0x15 (0x35)	TIFR0	-	-	-	-	-	OCF0B	OCF0A	TOV0	134
0x14 (0x34)	PORTG	-	-	PORTG5	PORTG4	PORTG3	PORTG2	PORTG1	PORTG0	102
0x13 (0x33)	DDRG	-	-	DDG5	DDG4	DDG3	DDG2	DDG1	DDG0	102
0x12 (0x32)	PING	-	-	PING5	PING4	PING3	PING2	PING1	PING0	102
0x11 (0x31)	PORTF	PORTF7	PORTF6	PORTF5	PORTF4	PORTF3	PORTF2	PORTF1	PORTF0	101
0x10 (0x30)	DDRF	DDF7	DDF6	DDF5	DDF4	DDF3	DDF2	DDF1	DDF0	102
0x0F (0x2F)	PINF	PINF7	PINF6	PINF5	PINF4	PINF3	PINF2	PINF1	PINF0	102
0x0E (0x2E)	PORTE	PORTE7	PORTE6	PORTE5	PORTE4	PORTE3	PORTE2	PORTE1	PORTE0	101
0x0D (0x2D)	DDRE	DDE7	DDE6	DDE5	DDE4	DDE3	DDE2	DDE1	DDE0	101
0x0C (0x2C)	PINE	PINE7	PINE6	PINE5	PINE4	PINE3	PINE2	PINE1	PINE0	102
0x0B (0x2B)	PORTD	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	101
0x0A (0x2A)	DDRD	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	101
0x09 (0x29)	PIND	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	101
0x08 (0x28)	PORTC	PORTC7	PORTC6	PORTC5	PORTC4	PORTC3	PORTC2	PORTC1	PORTC0	101
0x07 (0x27)	DDRC	DDC7	DDC6	DDC5	DDC4	DDC3	DDC2	DDC1	DDC0	101
0x06 (0x26)	PINC	PINC7	PINC6	PINC5	PINC4	PINC3	PINC2	PINC1	PINC0	101
0x05 (0x25)	PORTB	PORTB7	PORTB6	PORTB5	PORTB4	PORTB3	PORTB2	PORTB1	PORTB0	100
0x04 (0x24)	DDRB	DDB7	DDB6	DDB5	DDB4	DDB3	DDB2	DDB1	DDB0	100
0x03 (0x23)	PINB	PINB7	PINB6	PINB5	PINB4	PINB3	PINB2	PINB1	PINB0	100
0x02 (0x22)	PORTA	PORTA7	PORTA6	PORTA5	PORTA4	PORTA3	PORTA2	PORTA1	PORTA0	100
0x01 (0x21)	DDRA	DDA7	DDA6	DDA5	DDA4	DDA3	DDA2	DDA1	DDA0	100
0x00 (0x20)	PINA	PINA7	PINA6	PINA5	PINA4	PINA3	PINA2	PINA1	PINA0	100

Notes:

1. For compatibility with future devices, reserved bits should be written to zero if accessed. Reserved I/O memory addresses should never be written.
2. I/O registers within the address range $00 - $1F are directly bit-accessible using the SBI and CBI instructions. In these registers, the value of single bits can be checked by using the SBIS and SBIC instructions.
3. Some of the status flags are cleared by writing a logical one to them. Note that the CBI and SBI instructions will operate on all bits in the I/O register, writing a one back into any flag read as set, thus clearing the flag. The CBI and SBI instructions work with registers 0x00 to 0x1F only.
4. When using the I/O specific commands IN and OUT, the I/O addresses $00 - $3F must be used. When addressing I/O registers as data space using LD and ST instructions, $20 must be added to these addresses. The ATmega640/1280/1281/2560/2561 is a complex microcontroller with more peripheral units than can be supported within the 64 location reserved in Opcode for the IN and OUT instructions. For the Extended I/O space from $60 - $1FF in SRAM, only the ST/STS/STD and LD/LDS/LDD instructions can be used.

0x11 (0x31)	PORTF	PORTF7	POR
0x10 (0x30)	DDRF	DDF7	DDI
0x0F (0x2F)	PINF	PINF7	PIN
0x0E (0x2E)	PORTE	PORTE7	POR
0x0D (0x2D)	DDRE	DDE7	DDI
0x0C (0x2C)	PINE	PINE7	PIN
0x0B (0x2B)	PORTD	PORTD7	POR
0x0A (0x2A)	DDRD	DDD7	DDI
0x09 (0x29)	PIND	PIND7	PIN
0x08 (0x28)	PORTC	PORTC7	POR
0x07 (0x27)	DDRC	DDC7	DD(
0x06 (0x26)	PINC	PINC7	PIN(
0x05 (0x25)	PORTB	PORTB7	POR
0x04 (0x24)	DDRB	DDB7	DDI
0x03 (0x23)	PINB	PINB7	PIN
0x02 (0x22)	PORTA	PORTA7	POR
0x01 (0x21)	DDRA	DDA7	DDA
0x00 (0x20)	PINA	PINA7	PIN

I/O View

PORTB

Name	Value
AD_CONVERTER	
ANALOG_COMPARATOR	
BOOT_LOAD	
CPU	
EEPROM	
EXTERNAL_INTERRUPT	
JTAG	
PORTA	
PORTB	
Port B Data Register	0x02
Port B Data Direction Register	0xFF
Port B Input Pins	0x02
PORTC	
PORTD	
Port D Data Register	0x02
Port D Data Direction Register	0xFF
Port D Input Pins	0x02
PORTE	

Name	Address	Value	Bits
DDRB	0x04 (0x24)	0xFF	■■■■■■■■
PINB	0x03 (0x23)	0x02	□□□□□□■□
PORTB	0x05 (0x25)	0x02	□□□□□□■□

C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_...

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;    1DT301, Computer Technology I
;    Date: 2015-09-09
;    Author:
;        Anders Haggren
;    Function:
;        Set output
;
;        Lecture example 2015-09-09, lecture #3
;        Relay card connected to PORTD
;        LEDs connected to PORTB
;        Switches connected to POPTA
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

.equ DDRB = 0x04
.equ DDRD = 0x0A

.equ PORTB = 0x05
.equ PORTD = 0x0B

.def Temp = r16

ldi Temp, 0xFF
out DDRB, Temp          ; DDRB = 0x04
out DDRD, Temp          ; DDRD = 0x0A

ldi Temp, 0x02
```

m2560def

m2560def

```
;***** THIS IS A MACHINE GENERATED FILE - DO NOT EDIT ********************
;***** Created: 2011-08-25 20:59 ******* Source: ATmega2560.xml **********
;*************************************************************************
;* A P P L I C A T I O N   N O T E   F O R   T H E   A V R   F A M I L Y
;*
;* Number           : AVR000
;* File Name        : "m2560def.inc"
;* Title            : Register/Bit Definitions for the ATmega2560
;* Date             : 2011-08-25
;* Version          : 2.35
;* Support E-mail   : avr@atmel.com
;* Target MCU       : ATmega2560
;*
;* DESCRIPTION
;* When including this file in the assembly program file, all I/O register
;* names and I/O register bit names appearing in the data book can be used.
;* In addition, the six registers forming the three data pointers X, Y and
;* Z have been assigned names XL - ZH. Highest RAM address for Internal
;* SRAM is also defined
;*
;* The Register names are represented by their hexadecimal address.
;*
;* The Register Bit names are represented by their bit number (0-7).
;*
;* Please observe the difference in using the bit names with instructions
;* such as "sbr"/"cbr" (set/clear bit in register) and "sbrs"/"sbrc"
;* (skip if bit in register set/cleared). The following example illustrates
;* this:
;*
;* in    r16,PORTB             ;read PORTB latch
;* sbr   r16,(1<<PB6)+(1<<PB5) ;set PB6 and PB5 (use masks, not bit#)
;* out   PORTB,r16             ;output to PORTB
;*
```

m2560def

```
.equ    PORTF   = 0x11
.equ    DDRF    = 0x10
.equ    PINF    = 0x0f
.equ    PORTE   = 0x0e
.equ    DDRE    = 0x0d
.equ    PINE    = 0x0c
.equ    PORTD   = 0x0b
.equ    DDRD    = 0x0a
.equ    PIND    = 0x09
.equ    PORTC   = 0x08
.equ    DDRC    = 0x07
.equ    PINC    = 0x06
.equ    PORTB   = 0x05
.equ    DDRB    = 0x04
.equ    PINB    = 0x03
.equ    PORTA   = 0x02
.equ    DDRA    = 0x01
.equ    PINA    = 0x00
```

Example with .equ

```
; Program example, Monday, September 9, 2013
; STK600, CPU ATmega2560
; Addresses, see page 415 in doc2549_ATmega2560.pdf


.equ DDRA = 0x01
.equ PORTA = 0x02

.equ PINB = 0x03
.equ DDRB = 0x04

ldi r16, 0b11111111
out DDRA, r16           ; All one's to DDRA, outputs

ldi r16, 0b00000000
out DDRB, r16           ; All one's to DDRB, inputs

my_loop:
in r16, pinb            ; read PINB
; com r16                   ; one's complement each bit
out porta, r16          ; write in PORTA
rjmp my_loop
```

Example with .include ”m2560def.inc”

```
; Program example, Monday, September 9, 2013
; STK600, CPU ATmega2560
; Addresses, see page 415 in doc2549_ATmega2560.pdf

.include "m2560def.inc" ; include definition file for ATmega2560
; .include "m16def.inc" ; include definition file for ATmega16

ldi r16, 0b11111111
out DDRA, r16           ; All one's to DDRA, outputs


ldi r16, 0b00000000
out DDRB, r16           ; All one's to DDRB, inputs

my_loop:
in r16, PINB            ; read PINB
com r16                 ; one's complement each bit
out PORTA, r16          ; write in PORTA
rjmp my_loop
```

C:\Document__Kurser\D2_E2\Datorteknik\HT_2014\Test_...

```
;>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>
;    1DT301, Computer Technology I
;    Date: 2015-09-09
;    Author:
;        Anders Haggren
;    Function:
;        Set output
;
;        Lecture example 2015-09-09, lecture #3
;        Relay card connected to PORTD
;        LEDs connected to PORTB
;        Switches connected to POPTA
;<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

.include "m2560def.inc"

.def Temp = r16

ldi Temp, 0xFF
out DDRB, Temp          ; Set Port B to outputs
out DDRD, Temp          ; Set Port D to outputs

ldi Temp, 0x02

out PORTB, Temp         ; Set pin 2 in Port B to high
out PORTD, Temp         ; Set pin 2 in Port D to high

loop_1:
rjmp loop_1
```

Example with .include ”m2560def.inc”

C:\Document__Kurser\D2_E2\Datorteknik\HT_2013\Test_program\Lesson_...

```
; Program example, Wednesday, September 4, 2013
; STK600, CPU ATmega2560
; Addresses, see page 415 in doc2549_ATmega2560.pdf

.include "m2560def.inc" ; include definition file for ATmega256
; .include "m16def.inc" ; include definition file for ATmega16

ldi r16, 0b11111111
out DDRA, r16           ; All one's to DDRA, outputs


ldi r16, 0b00000000
out DDRB, r16           ; All one's to DDRB, inputs

my_loop:
in r16, PINB            ; read PINB
com r16                 ; one's complement each bit
out PORTA, r16          ; write in PORTA
rjmp my_loop
```

Example from lecture #2:

```
; Program example, Wednesday, September 4, 2013
; STK600, CPU ATmega2560
; Addresses, see page 415 in doc2549_ATmega2560.pdf

ldi r16, 0b11111111
out 0x01, r16                    ; All one's to DDRA, outputs


ldi r16, 0b00000000
out 0x04, r16                    ; All one's to DDRB, inputs

my_loop:
in r16, 0x03                     ; read PINB
com r16                          ; one's complement each bit
out 0x02, r16                    ; write in PORTA
rjmp my_loop
```