Some Code!

How to temporarely stop ISR.

```
/******* Test on switch 1 ***********/
    if ((~PINA) & (1 << 1) )        //   then test on switch 1
    {
        if(tid_1.updated == 0)
// use the updated field to see if tid_1 is updated on this key press !
        {
            ATOMIC_BLOCK(ATOMIC_RESTORESTATE)
// This macro stores the SREG turns off global IRQ when entering the ATOMIC_BLOCK,
/and it restore the SREG when exiting the ATOMIC_BLOCK
            {
// The reason for turning off IRQ in this block,
//We do not want the ISR to update the tid when we read it!!
               tid_1.sek = tid.sek ;
               tid_1.min = tid.min ;
               tid_1.timer = tid.timer ;
            }
           tid_1.updated = 1;
// set the updated field to indicate the tim_1 is already updated on this key press
        }
    }
    else
        tid_1.updated = 0;
// reset the updated field to indicate that switch 1 is released.
```

More obs!

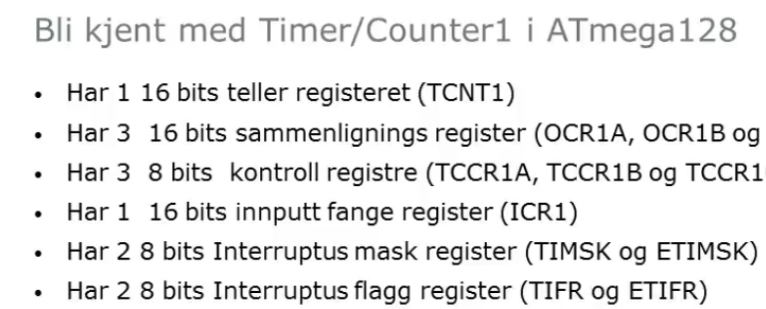

Figure 1: Timer/Counter and some

4.1.2.1 Status Register (SREG)

The Status Register or SREG contains the important information about the ALU such as the Carry Bit, Overflow Bit, and Zero Bit. These bits are set and cleared during ALU instructions. This register becomes extremely useful during branching operations. The following table details the bit assignments within the SREG.

Figure 2: SREG

Table 2: SREG Definition

Bit	Name	Description
7	I	Global Interrupt Enable
6	T	Bit Copy Storage
5	H	Half Carry Flag
4	S	Sign Bit
3	V	Twos Compliment Overflow Flag
2	N	Negative Flag
1	Z	Zero Flag
0	C	Carry Flag

As an example of using this register, look at the **BREQ** or Branch If Equal instruction. When this instruction is called, it looks at the Zero Flag in the SREG. If the Zero Flag is set, then the instruction will branch to the program address specified, otherwise it will continue on as usual.

Figure 3: SREG extended

5.1 Pre-compiler Directives

Pre-compiler directives are special instructions that are executed before the code is compiled and directs the compiler. These instructions are denoted by the preceding dot, i.e. .EQU. 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. The following sections will contain detailed information on the most commonly used directives. The following table contains an overview of the directives supported by the AVR Assembler.

Table 5: Pre-Compiler 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 words
.ENDMACRO	End macro
.EQU	Set a symbol equal to an expression
.ESEG	EEPROM segment
.EXIT	Exit from a file
.INCLUDE	Read source from another file
.LIST	Turn listfile generation on
.LISTMAC	Turn macro expression on
.MACRO	Begin Macro
.NOLIST	Turn listfile generation off
.ORG	Set program origin
.SET	Set a symbol to an expression

Figure 4: Assembler directives