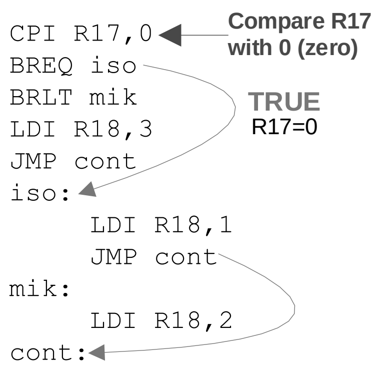
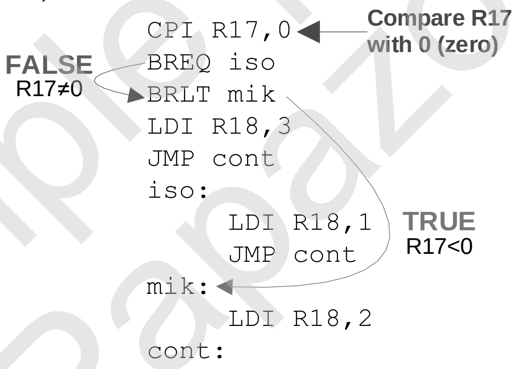


```
CPI R17,0
BREQ iso
BRLT mik
LDI R18,3
JMP cont
iso:
     LDI R18,1
     JMP cont
mik:
     LDI R18,2
cont:
```

Figure 4.8a Execution flow for R17=0

On the other hand, if the content of register `R17` is less than zero (fig. 4.8b), then the condition of the `BREQ` is FALSE and the execution flow is not driven to the label `iso`. Thus, the instruction `BRLT mik` is executed. Now, this condition is TRUE (R17<0) and the execution flow continues from the point `mik` where the value 2 is stored in R18 (`LDI R18,2`). After the execution of this instruction, the execution flow continues from the point `cont` which follows (is a wrong programming practice the insertion of a JMP instruction for going to the `cont` point because the execution flow is driven always there).



```
CPI R17,0
BREQ iso
BRLT mik
LDI R18,3
JMP cont
iso:
     LDI R18,1
     JMP cont
mik:
     LDI R18,2
cont:
```

Figure 4.8b Execution flow for R17<0

Finally, if the content of register `R17` is positive (R18>0), then the conditions of the instructions `BREQ` and `BRLT` are FALSE. Thus, the execution flow is not driven to the points `iso` or `mik` and the instruction `LDI R18,3` is executed. Initially, the condition for equivalence (`BREQ`) is checked which is FALSE. Thus, the next condition is checked which is also FALSE. As a result, the value 3 is stored in the register `R18`. After the value storage, the instruction `JMP cont` is executed for bypassing the next code sections which belong to different cases (R17=0, R17<0), as shown in figure 4.8c.

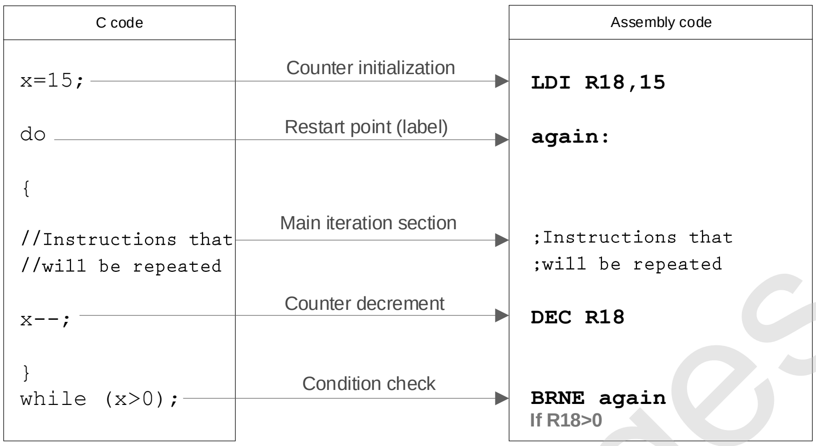


Figure 4.13 C and Assembly

As shown in figure 4.13, the counter takes values in the range [15,0], while the loop will be terminated when the counter content becomes zero (the Z bit of the SREG will be activated). Figure 4.14 shows the corresponding flow chart diagram.

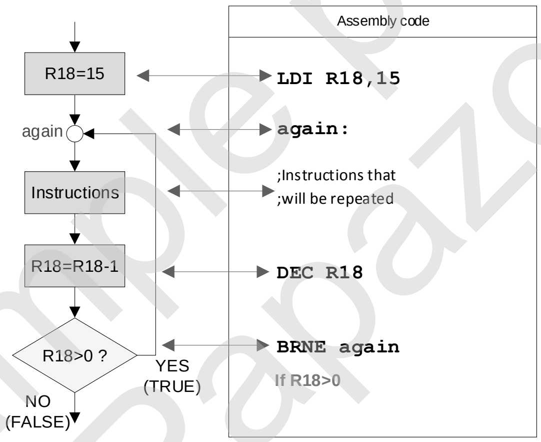
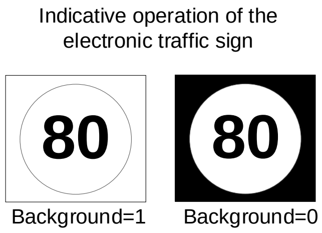


Figure 4.14 Code and flow chart diagram

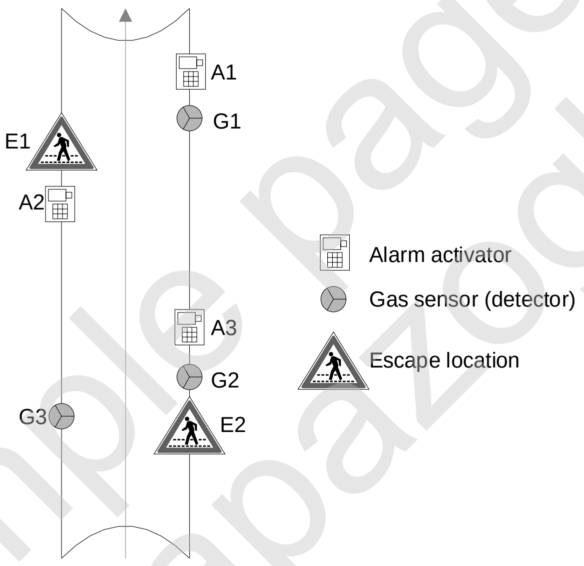
4.3.2 Iteration structure while-do

In this type of iteration, the condition check is performed at the beginning (before the main iteration section). As a result, the instruction group inside the iteration, is never executed if the condition is not TRUE at least once. The implementation of the above structure is more complex as compared to the previous (do-while). Moreover, the while-do structure can be implemented with two different ways.

Figure 4.15 shows the flow chart diagram, while the code 4.3 is the implementation of the iteration structure where the execution flow is driven in the main iteration section only if the condition check is TRUE. Otherwise, the iteration is terminating with jumping to the label `exit`, where the condition check is performed. When the condition check is FALSE, the execution flow is not driven to



14. In a tunnel of the national highway there are 3 locations of alarm activators for the drivers and 3 locations with gas detectors. Moreover, there are 2 locations of escape paths. Based on the tunnel management rules, in a case of emergency (activated alarms), the closer escape path must be lighted with a proper sound alarm for helping drivers and passengers to be driven at the emergency exit.



The logic of the tunnel management system can be described as follows:

If G1=1 or A1=1 or A2=1, then E1=1 (0=no active, 1=active)
If G2=1 or G3=1 or A3=1, then E2=1 (0=no active, 1=active)
Of course, the activation of E1 and E2 may be happened at the same time. The gas detection is performed by sensors, while the alarm can be activated by drivers using a special red button. The buttons are install in a communication device which can be used also by drivers in a case of emergency.
The register R16 represents at any time the status of the G1, G2, G3, A1, A2 and A3. The above status is mapped on specific bits as follows:

	7	6	5	4	3	2	1	0
R16	**X**	**X**	**G3**	**G2**	**G1**	**A3**	**A2**	**A1**

Note
A main difference of the LED diode as compared to classic one, is that the corresponding voltage drop across positive and negative pins is much greater. For a red LED, this voltage is 1.8 to 2.0V. Thus, it is assumed that U_D=2V, when a current calculation takes place.

Figure 5.10a shows a LED diode in a simple electrical circuit. For the same circuit, the voltage source can be replaced by a microcontroller pin (fig. 5.10b). Thus, the LED can be controlled by the microcontroller based on the corresponding software. It must be noticed that, due to the small LED resistance, an additional resistor must be used for limiting the current. Otherwise, the LED and the microcontroller port can be damaged.

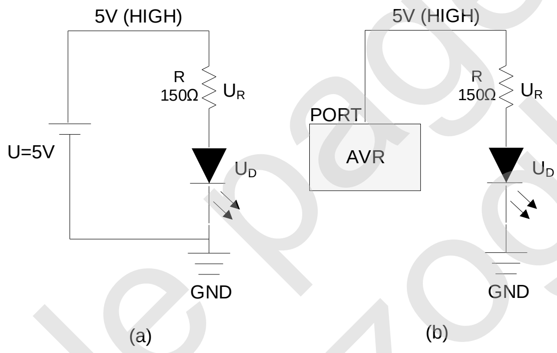


Figure 5.10 LED control from a voltage source or from a microcontroller

The current of circuit 5.10 is:

$$I = \frac{U - U_D}{R} = \frac{5V - 2V}{150\Omega} = \frac{3V}{150\Omega} = 0.02A = 20mA$$

Figure 5.11 shows how the LED can be lit by setting the microcontroller pin level (e.g. PB0) to HIGH (5V) or LOW (0V).

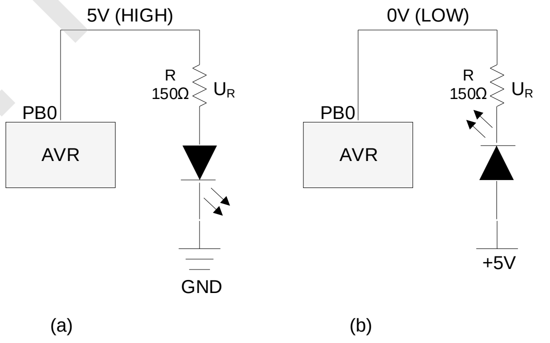


Figure 5.11 LED activation with a signal 5 or 0V

microcontroller is achieved by register PORTx, assuming that all the pins of port B have been set as outputs with the instructions `LDI R16,0xFF` and `OUT DDRB,R16`.

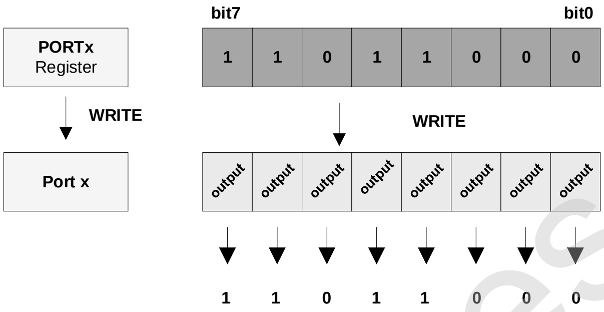


Figure 5.16 Using the register PORTx

As shown in figure 5.16, anything loaded in PORTx, is appeared as voltage level to the corresponding pins. The complete code for writing the binary number 11011000 in port B, is as follows:

Code 5.1

```
LDI R16,0xFF         ;load 11111111 in R16
OUT DDRB,R16         ;set all the pins of port B, as outputs
LDI R16,0b11011000   ;load 11011000 in R16
OUT PORTB,R16        ;write the data in the pins of port B
```

Figure 5.17 shows the data direction set for the port B through register DDRB as well as the pin voltage set (binary number 11011000) through register PORTB. **This operation is the same for all the AVR microcontrollers.** In the same figure, the pinout of the ATmega8515 and ATmega32 is used.

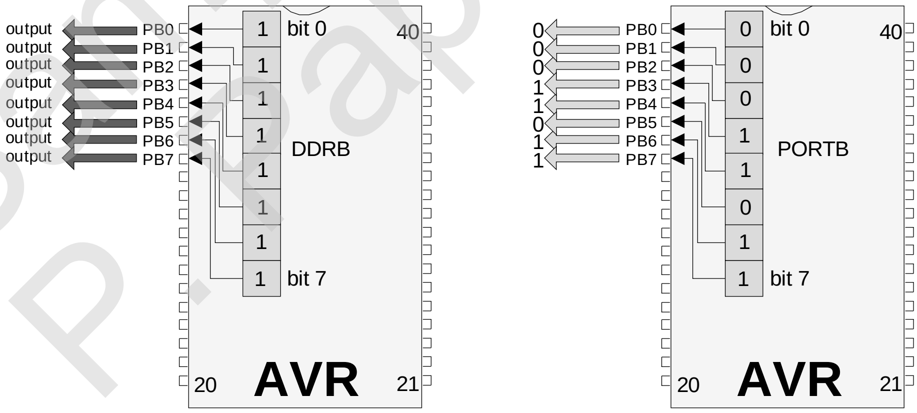

output output output output output output output output
PB0 PB1 PB2 PB3 PB4 PB5 PB6 PB7
1 bit 0
1 1 1 1 1 1 1 bit 7
DDRB
40
20 AVR 21
0 0 0 1 1 0 1 1
PB0 PB1 PB2 PB3 PB4 PB5 PB6 PB7
0 bit 0
0 0 1 1 0 1 1 bit 7
PORTB
40
20 AVR 21

Figure 5.17 Write data bits in the output pins of port B (ATmega8515/ATmega32)

IN – Read from an I/O Register

For reading from an I/O register, the following instruction is used:

```
IN Rd,IOReg
```

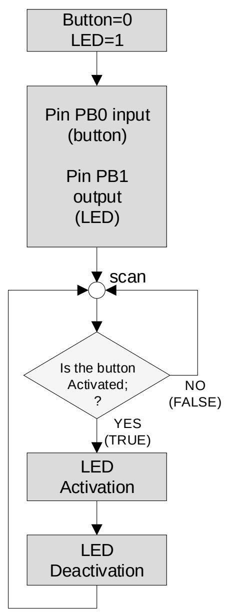
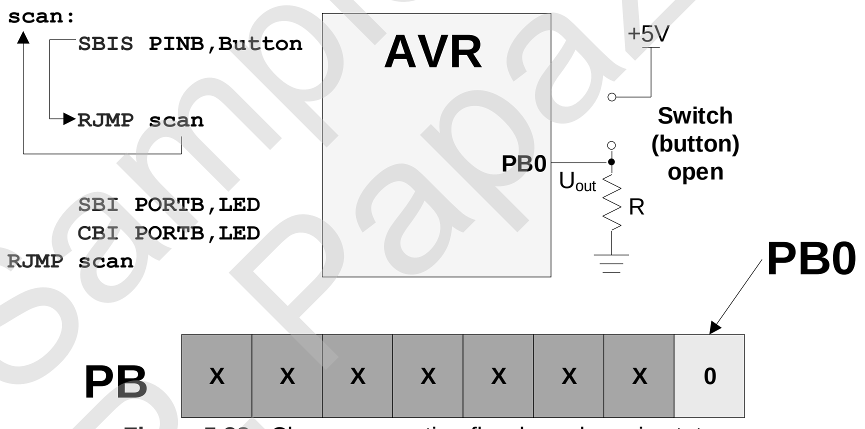


Figure 5.32 Flow chart

Of course, in such a case, a different port could be used for different data direction, but the current program supports only the above operation (the button is checked constantly for LED activation).

The `SBIS` instruction checks the input pin PB0. If this pin is activated (PB0=1), then the next instruction is not executed (the flow does not return to button check) and the LED is activated and deactivated for a moment. Using this approach, the LED is only activated while the button is pressed. After the above LED operation, the execution flow returns to the button check.

The execution flow returns also to button check while PB0=0 (checking with the SBIS instruction). Figure 5.33 shows the execution flow based on the button state.



```
scan:
        SBIS  PINB,Button
        RJMP  scan
        SBI  PORTB,LED
        CBI  PORTB,LED
RJMP  scan
```

Figure 5.33a Change execution flow based on pin status

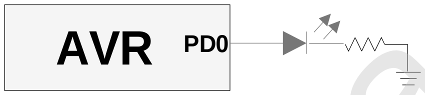
LABORATORY EXERCISE 2
Time Delay

GOAL

In this exercise a time delay will be developed and measured for controlling a LED.

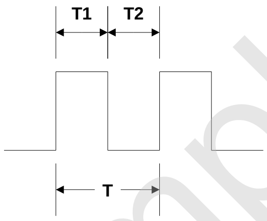
Step 1

Develop a program for inverting the LED status (on/off/on, etc) every 0.5 seconds. The LED is connected at PD0 (pin 0 of port D). For the program operation a circuit has to be implemented (see next figure). Initially, observe the program operation on the LED state.



Step 2

Fill the following signal attributes based on the expected measurements on pin PD0.



T1 =

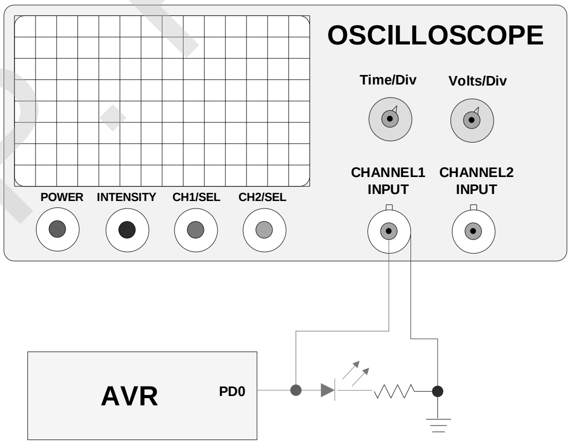
T2 =

T =

Frequency F =

Step 3

Connect one channel of the oscilloscope as described in the next figure. Set properly the time base (Time/Div) and the voltage scale (Volts/Div) on the oscilloscope in order to display the measured signal at the right size.



Note
The SSD unit of common anode is the most popular circuit implementation due to the fact that offers power independency using an external power supply. The goal of the current application is to give two different solutions for using the SSD units (common anode or common cathode) that may be chosen by the engineer.

In the circuit of figure 6.8b, each segment activation is performed with a logic signal 0 (LOW), and the power supply is based on an external source. For supporting the above operation, the transistor BC557 (PNP type) is used instead of the BC547 (NPN type).

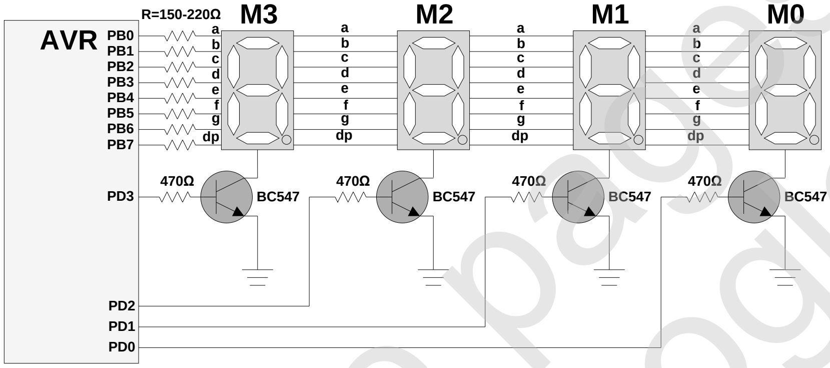


Figure 6.8a Circuit with four common cathode SSD units

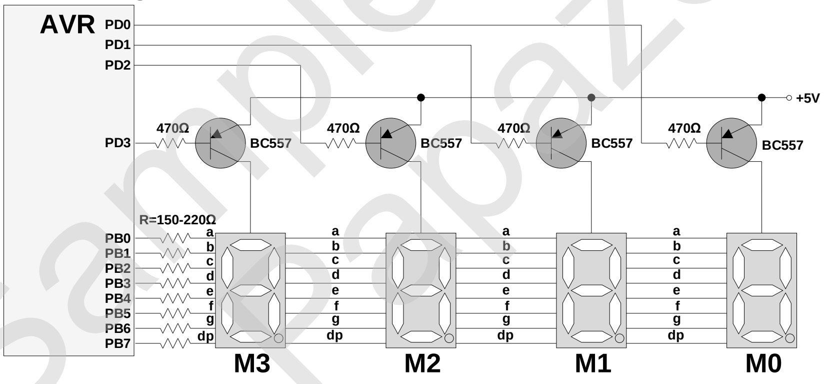


Figure 6.8b Circuit with four common anode SSD units

Now, more pins of port D are used (fig. 6.8a,b). Table 6.3 shows the pin values (PD3 to PD0) in order to display the digit ‘1’ in all the SSD units (M3 to M0).

Table 6.3 SSD unit activation

PD3	PD2	PD1	PD0	Unit activation
1	0	0	0	M3
0	1	0	0	M2
0	0	1	0	M1
0	0	0	1	M0

R_1	C_2	SW_6
R_1	C_3	SW_7
R_2	C_0	SW_8
R_2	C_1	SW_9
R_2	C_2	SW_A
R_2	C_3	SW_B
R_3	C_0	SW_C
R_3	C_1	SW_D
R_3	C_2	SW_E
R_3	C_3	SW_F

Figure 7.2 shows the «active-electric path» based on the SW_0 activation. Initially, the desired line is activated (e.g. R_0) and all the columns are scanned. Thus, by activating the line R_0 and button SW0, only the column C_0 gives 5V signal. With the line-column combination, the activated button is known. For finding any activated button, the lines R_0 to R_3 are connected to output pins of the microcontroller, while the columns C_0 to C_3 are connected to input pins of the microcontroller. In other words, when a line is activated through a microcontroller pin (output), then the column signals (column outputs) are read through the microcontroller pins (input).

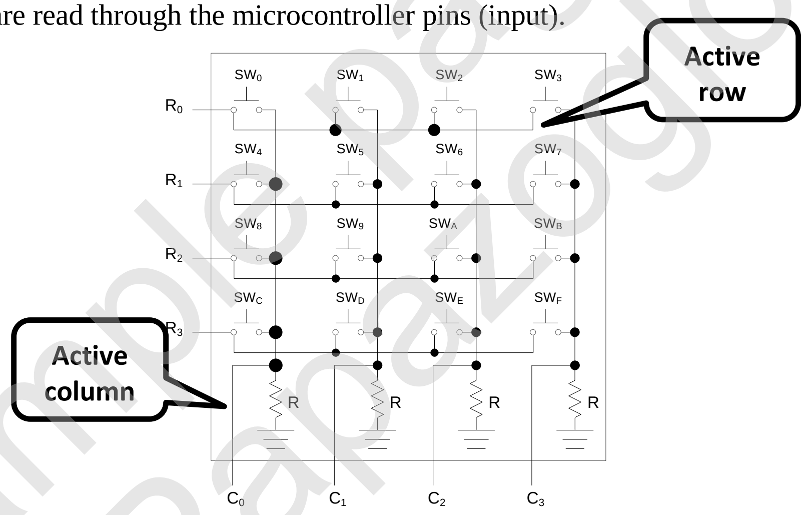


Figure 7.2 SW_0 is activated

The following algorithm describes the scanning operation in a circuit with N lines and M columns (NxM) for finding the pressed button.

```
BEGIN
For A=1 to N (lines)
   {
      Line A activation
      For B=1 to M (columns)
        {
          If column B=active then
               the button (A,B) is pressed,
               perform the corresponding operations
          End-If
        }
   }
END
```

For verifying the keyboard operation as well as the corresponding code operation, the SSD units will be used for displaying the keyboard symbols. Figure 7.5a shows the full circuit which contains the input keyboard as well as the SSD units. Alternatively, only one SSD unit can be directly used (fig. 7.5b). The display circuit is based on the digital multiplexing of multiple SSD units. In this application, only the right side SSD unit (M0) is used for displaying the symbols. Thus the other SSD units (M1 to M3) will remain unconnected. This happens for showing the full circuit and the capability to display more digits later by using more SSD units.

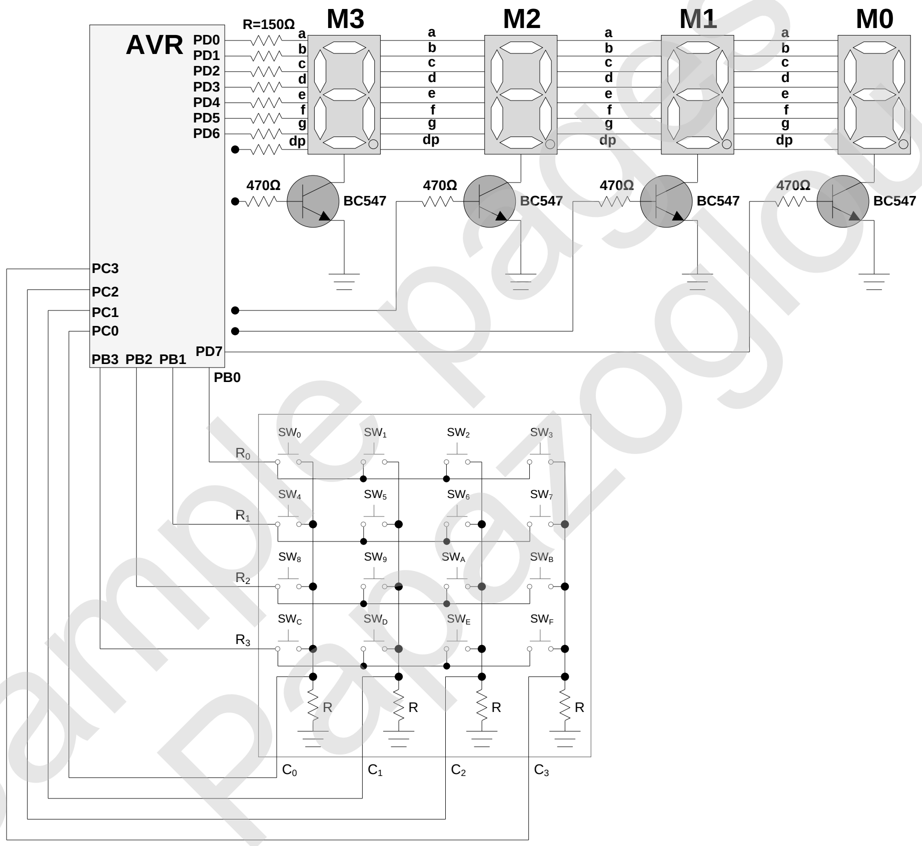


Figure 7.5a Test circuit for the 4x4 keyboard

As shown in the circuit of figure 7.5a, only the control line for the last digit is used (SSD unit M0), while the floating point led (dp) is also not used. Thus, the first seven pins of the port D are used for the segments a,b,c,d,e,f and g, while the last pin is used for controlling the SSD unit through the transistor. Due to the fact that the SSD unit is always active, the pin PD7 will be always 1 (fig. 7.6). The same connection method is also used in the circuit of figure 7.5b. Moreover, eight bits will be written to port D and thus the corresponding hexadecimal numbers for every digit will be different. Table 7.2 shows the hexadecimal numbers that correspond to the segments activation

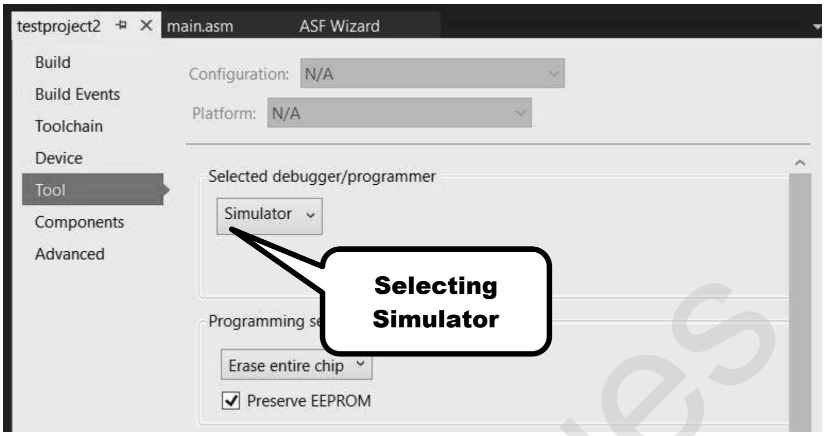
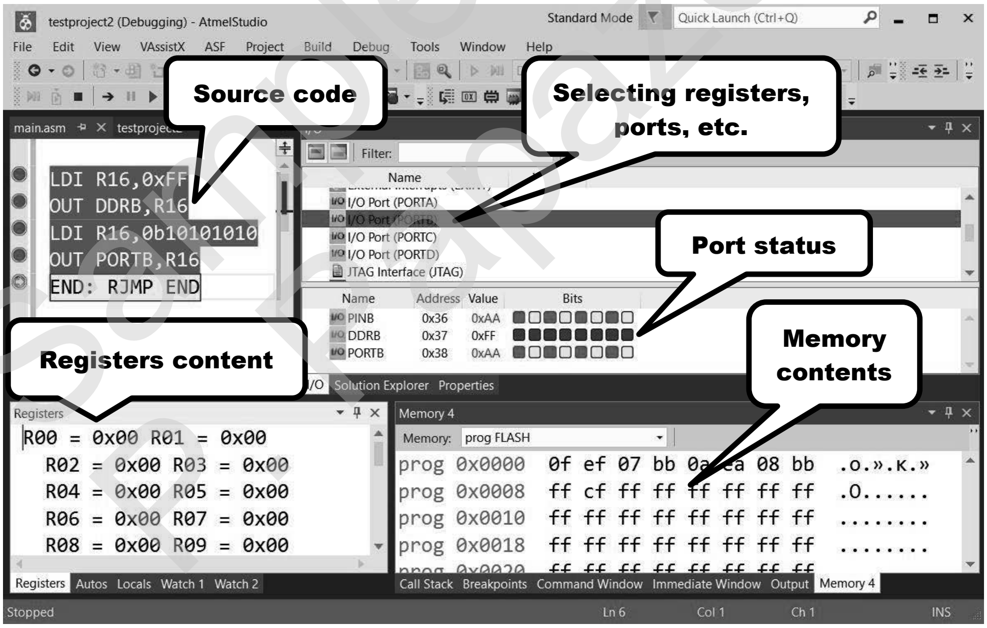


Figure A.11 Selecting the simulator tool

STEP 10
Run the simulation step by step

Now the execution process can be started. **Press successively the play button** in order to view the corresponding results in the microcontroller. Figure A.12 shows the information which is displayed during the step by step execution. From the menu **Debug**, select **Windows** to activate the information window type.


testproject2 (Debugging) - AtmelStudio
File Edit View VAssistX ASF Project Build Debug Tools Window Help
Source code
Selecting registers, ports, etc.
Port status
Registers content
Memory contents
I/O Port (PORTA)
I/O Port (PORTB)
I/O Port (PORTC)
I/O Port (PORTD)
JTAG Interface (JTAG)
Solution Explorer Properties
Registers Autos Locals Watch 1 Watch 2
Call Stack Breakpoints Command Window Immediate Window Output Memory 4
Stopped Ln 6 Col 1 Ch 1 INS

```
LDI R16,0xFF
OUT DDRB,R16
LDI R16,0b10101010
OUT PORTB,R16
END: RJMP END
```

Name	Address	Value	Bits
PINB	0x36	0xAA	■□■□■□■□
DDRB	0x37	0xFF	■■■■■■■■
PORTB	0x38	0xAA	■□■□■□■□

```
R00 = 0x00 R01 = 0x00
  R02 = 0x00 R03 = 0x00
  R04 = 0x00 R05 = 0x00
  R06 = 0x00 R07 = 0x00
  R08 = 0x00 R09 = 0x00
```

Memory: prog FLASH

```
prog 0x0000  0f ef 07 bb 0a ea 08 bb  .o.».к.»
prog 0x0008  ff cf ff ff ff ff ff ff  .O......
prog 0x0010  ff ff ff ff ff ff ff ff  ........
prog 0x0018  ff ff ff ff ff ff ff ff  ........
prog 0x0020  ff ff ff ff ff ff ff ff
```

Figure A.12 Simulation process