Handbook
part II

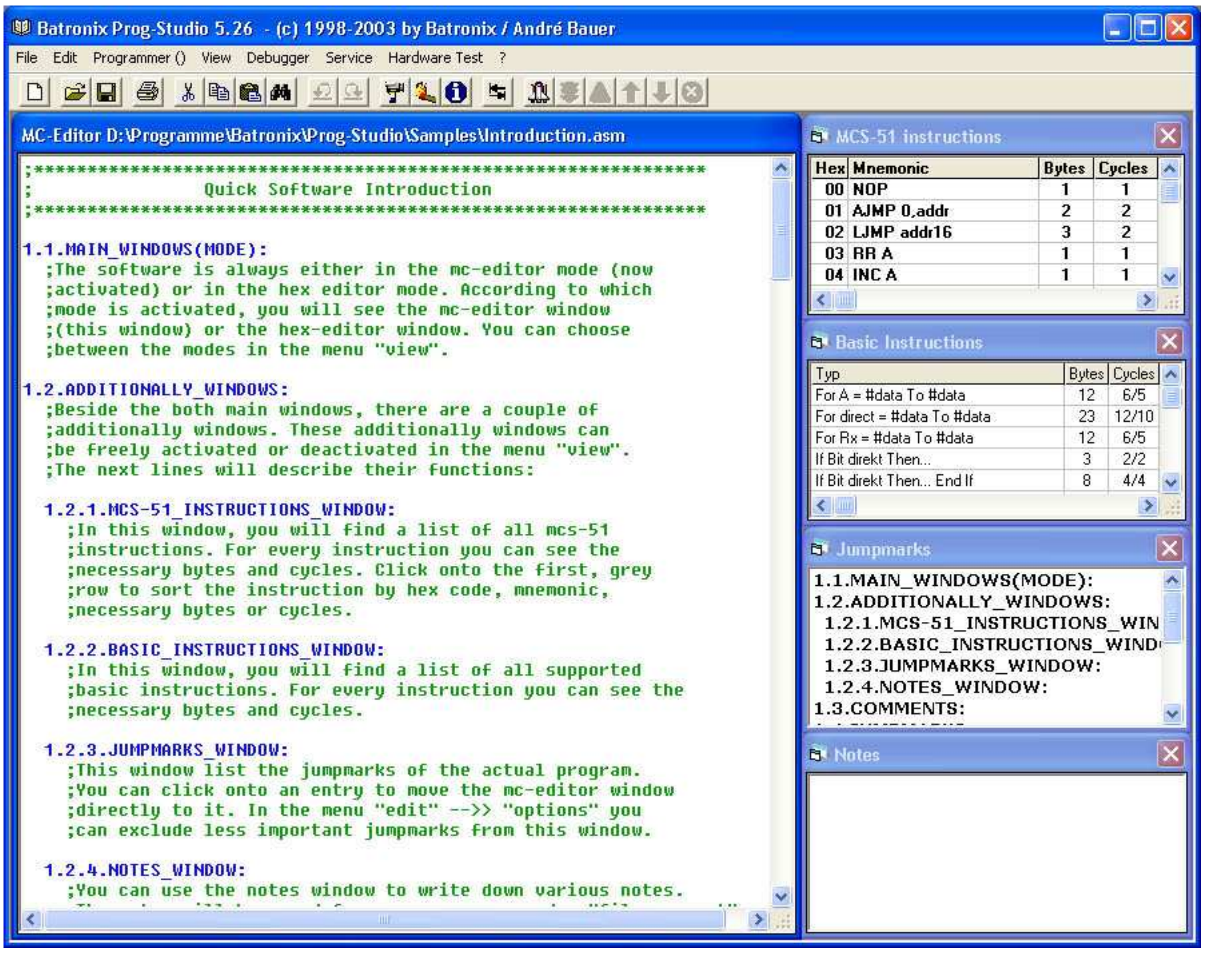
Prog-Studio Software
MC Editor

- General and Surface -
- Assembler Programming -
- Basic Programming -
- The Debugger -

1. General information

1.1 What can the Prog-Studio software do in the MC-Editor mode?

In the MC-Editor mode, you can develop separate assembler / basic programs for the microcontroller in the MCS-51 series and test them with the built-in debugger. If this is not your area of application, you only need to look at the general directions, and not at this special section of the Prog-Studio software manual.


Batronix Prog-Studio 5.26 - (c) 1998-2003 by Batronix / André Bauer
File Edit Programmer () View Debugger Service Hardware Test ?
MC-Editor D:\Programme\Batronix\Prog-Studio\Samples\Introduction.asm
MCS-51 instructions
Basic Instructions
Jumpmarks
Notes

```
;*****************************************************************
;                  Quick Software Introduction
;*****************************************************************

1.1.MAIN_WINDOWS(MODE):
  ;The software is always either in the mc-editor mode (now
  ;activated) or in the hex editor mode. According to which
  ;mode is activated, you will see the mc-editor window
  ;(this window) or the hex-editor window. You can choose
  ;between the modes in the menu "view".

1.2.ADDITIONALLY_WINDOWS:
  ;Beside the both main windows, there are a couple of
  ;additionally windows. These additionally windows can
  ;be freely activated or deactivated in the menu "view".
  ;The next lines will describe their functions:

  1.2.1.MCS-51_INSTRUCTIONS_WINDOW:
    ;In this window, you will find a list of all mcs-51
    ;instructions. For every instruction you can see the
    ;necessary bytes and cycles. Click onto the first, grey
    ;row to sort the instruction by hex code, mnemonic,
    ;necessary bytes or cycles.

  1.2.2.BASIC_INSTRUCTIONS_WINDOW:
    ;In this window, you will find a list of all supported
    ;basic instructions. For every instruction you can see the
    ;necessary bytes and cycles.

  1.2.3.JUMPMARKS_WINDOW:
    ;This window list the jumpmarks of the actual program.
    ;You can click onto an entry to move the mc-editor window
    ;directly to it. In the menu "edit" -->> "options" you
    ;can exclude less important jumpmarks from this window.

  1.2.4.NOTES_WINDOW:
    ;You can use the notes window to write down various notes.
```

Hex	Mnemonic	Bytes	Cycles
00	NOP	1	1
01	AJMP 0,addr	2	2
02	LJMP addr16	3	2
03	RR A	1	1
04	INC A	1	1

Typ	Bytes	Cycles
For A = #data To #data	12	6/5
For direct = #data To #data	23	12/10
For Rx = #data To #data	12	6/5
If Bit direkt Then...	3	2/2
If Bit direkt Then... End If	8	4/4

```
1.1.MAIN_WINDOWS(MODE):
1.2.ADDITIONALLY_WINDOWS:
 1.2.1.MCS-51_INSTRUCTIONS_WIN
 1.2.2.BASIC_INSTRUCTIONS_WIND
 1.2.3.JUMPMARKS_WINDOW:
 1.2.4.NOTES_WINDOW:
1.3.COMMENTS:
```

1.2 First steps and example programs

First, we would like to recommend that you take a look at the supplied sample programs. You can find them in the subdirectory \Examples of the Prog-Studio software. In the file menu, click on „open“ and choose the file “introdution-1.asm” in the dialogue window.

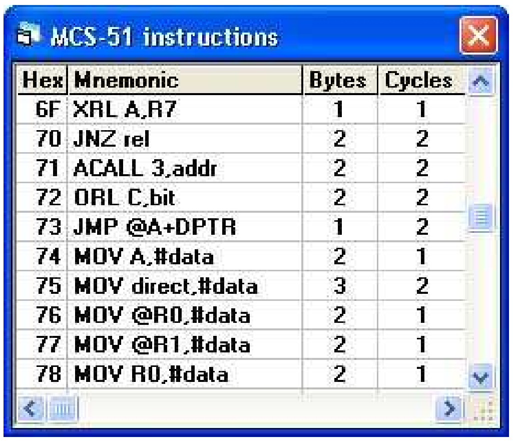
This introduction which is shown to you here as a first assembler program already contains a rough overview of the options for assembler programs in the Prog-Studio software. There are also other introductory components as well as example programs in this directory.

1.3 The additional windows

In the View menu, you can have a whole set of additional windows shown. These can make programming easier for you. Here, we will initially show the four additional windows which are constantly available. Further down in the chapter „Debuggers“, you can find the information on another four additional windows which are only active when the debugger is running.

The additional window MCS-51 instructions

This additional window contains a list of all MCS51 commands. It also contains the hex code and required bytes / cycles for each command.



MCS-51 instructions

Hex	Mnemonic	Bytes	Cycles
6F	XRL A,R7	1	1
70	JNZ rel	2	2
71	ACALL 3,addr	2	2
72	ORL C,bit	2	2
73	JMP @A+DPTR	1	2
74	MOV A,#data	2	1
75	MOV direct,#data	3	2
76	MOV @R0,#data	2	1
77	MOV @R1,#data	2	1
78	MOV R0,#data	2	1

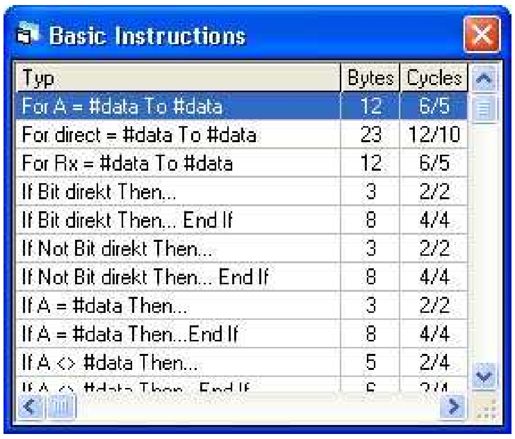
You can sort the command list by any desired criteria. Simply click on the description in the first (gray) line.

If you are in the Hex-Editor mode and the window MCS-51 Commands, double clicking on the byte will mark the corresponding command in the list.

Fixed number specifications are marked in the table with #data, memory addresses with “direct” and the accumulator with A. The registers are marked with Rx. When using the commands, type in the corresponding values or constants instead of "data" or "direct", and replace Rx with the desired register (R0-R7).

The additional window Basic instructions

This additional window gives you a short overview of the available basic commands. It also contains the information about the required bytes / cycles for each command.



Basic Instructions

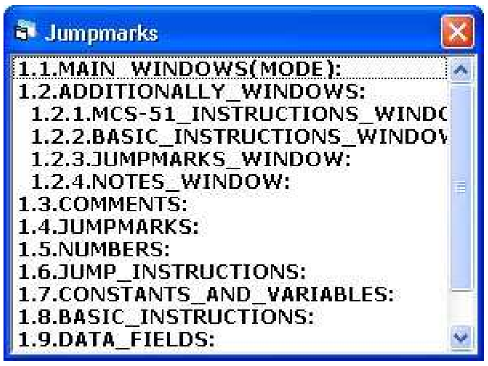
Typ	Bytes	Cycles
For A = #data To #data	12	6/5
For direct = #data To #data	23	12/10
For Rx = #data To #data	12	6/5
If Bit direkt Then...	3	2/2
If Bit direkt Then... End If	8	4/4
If Not Bit direkt Then...	3	2/2
If Not Bit direkt Then... End If	8	4/4
If A = #data Then...	3	2/2
If A = #data Then...End If	8	4/4
If A <> #data Then...	5	2/4

For the cycles, two numbers are shown. The first states the required cycles for the first run of a For-Next loop / the required cycles for the IF-Then direction, if the latter is met. The second number stands for the required cycles for each additional run of a For-Next loop / the required cycles of an IF-Then direction, if this is not met.

You can find further information on programming with Basic Commands in the corresponding chapter further below.

The additional window jumpmarks

In this additional window, you can quickly gain an overview of your program or move quickly and directly to certain subroutines by clicking on the corresponding entry. This is very helpful particularly in larger programs, and you should make use of this function. The entries in this window are shown as in the text. Indented marks are shown with indents here as well.



```
1.1.MAIN_WINDOWS(MODE):
1.2.ADDITIONALLY_WINDOWS:
  1.2.1.MCS-51_INSTRUCTIONS_WINDC
  1.2.2.BASIC_INSTRUCTIONS_WINDOV
  1.2.3.JUMPMARKS_WINDOW:
  1.2.4.NOTES_WINDOW:
1.3.COMMENTS:
1.4.JUMPMARKS:
1.5.NUMBERS:
1.6.JUMP_INSTRUCTIONS:
1.7.CONSTANTS_AND_VARIABLES:
1.8.BASIC_INSTRUCTIONS:
1.9.DATA_FIELDS:
```

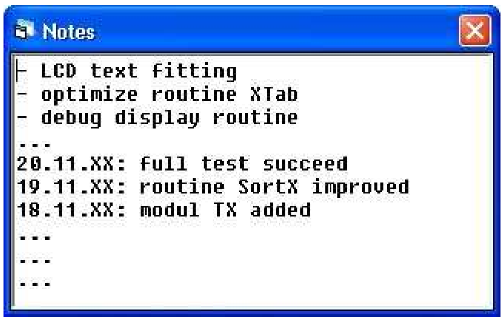
In order to keep an overview in really large programs as well, you can suppress the display of less important jump marks. In the menu Process -> Options, there is the option of not showing jump marks with a preceding tab symbol and/or a preceding underline "_" in the jumpmark window.

When the jumpmark window is activated, you can use a key to show the first jump mark in the window which starts with the symbol you have pressed (example: After pressing the "T" key, the first jump mark which begins with the letter T is shown).

Further information on programming with jump marks can be found in the corresponding chapter further below.

The additional window notes

You can use this window to enter everything which seems important to you. The assembler does not include the entries in an assembly, but it does save the notes for each of your programs under the program name with the ending .not.



```
- LCD text fitting
- optimize routine XTab
- debug display routine
...
20.11.XX: full test succeed
19.11.XX: routine SortX improved
18.11.XX: modul TX added
...
...
...
```

2. Assembler Programming

2.1 General Notes

When you have created a microcontroller program in the MC-Editor window, the assembler/compiler must transform your directions into the required hex codes so that the microcontroller or a memory chip can be described correctly. In order for the assembler to interpret your program correctly, you must, for instance, mark certain number specifications, state constants and skip marks, mark comments and data fields etc..

2.2 Commands and basic programming

Unfortunately, we cannot provide you with a comprehensive reference on the complete extent of the commands or on basic assembler programming, since it would outgrow the manual. You can find everything which you may want to know about commands, as well as the construction and use of the MCS-51 microcontroller, in various literature. Additionally, the extent of the commands with explanations can be found in a PDF file in the download area of our website: Batronix.com. The additional window „MCS-51 Commands“ in the Prog-Studio software lists all commands and is suitable as a quick look-up reference.

We only deal with the special options of the Prog-Studio software here and refer you to suitable literature for the command set, programming techniques as well as microcontroller architecture.

2.3 Number specifications

You can add leading zeros to the numbers where desired, or leave them out. A "0009" is simply interpreted as a "9".

Differentiation between numbers and memory addresses

Number specifications without a preceding number sign (#) are interpreted by the Prog-Studio software as memory addresses. If you would like to enter a number instead of a memory address, you must precede it with the number sign (#). This marking is common with the Intel Mnemonics and is required here for the basic commands also in order to be able to differentiate between memory addresses and numbers.

Hexadecimal numbers

There are three ways in which the assembler can recognize hexadecimal number specifications. The first method (standard) is to end it with an „h“. It can also be marked by preceding it with a dollar sign ($). The third method, of having the numbers interpreted as hexadecimal numbers by marking them, can be activated in the menu Process -> Options. We not recommend this, since it does not correspond to the standard and only remains integrated as a leftover for compatibility with old versions.

Some examples for hexadecimal number specifications are:

```
MOV A, 3Dh   ; Copies the value of the memory address 3D (hex) into the accumulator (A)
MOV A,#3Dh   ; Copies the hexadecimal number 3D into the accumulator (A)

MOV A,$3D    ;Copies the value of the memory address 3D (hex) into the accumulator (A)
MOV A,#$3D   ;Copies the hexadecimal number 3D into the accumulator (A)
```

Decimal numbers

The assembler also has three options for recognizing decimal number specifications as such. The first option (standard) is to state decimal numbers without marking them. However, this only works if the interpretation of numbers without markings as decimal numbers is activated in the menu Process -> Options (standard setting). It is also possible to mark them by ending them with a "d" or a preceding percent symbol (%).

Some examples of decimal number specifications are:

```
MOV A,212     ;Writes the value of the memory address 212 into the accumulator A)
MOV A,#212    ;Writes the decimal number 212 into the accumulator (A)

MOV A,212d    ;Writes the value of the memory address 212 into the  accumulator (A)
MOV A,#212d   ;Writes the decimal number 212 into the Accumulator (A)

MOV A,%212    ;Writes the value of the memory address 212 into the accumulator (A)
MOV A,#%212   ;Writes the decimal number 212 into the accumulator (A)
```

Binary numbers

Binary number specifications can be marked by ending them with a "b" (standard) or beginning them with an exclamation mark (!).

Some examples of binary number specifications are:

```
MOV A,10101010b    'Writes the value of the memory address 10101010 (binary) into the
        accumulator
MOV A,#10101010b   'Writes the binary number 10101010 into the accumulator

MOV A,!10101010    'Writes the value of the memory address 10101010 (binary) into the
        accumulator
MOV A,#!10101010   'Writes the binary number 10101010 into the accumulator
```

ASCII symbols

ASCII symbol chains must be placed in parentheses (“).

Some examples of ASCII symbol chains are

MOV A,#"H"   ;Writes the ASCII value of the s symbol "H"(=72dec) into the accumulator

DB "a string for your LCD" ;a data field with the ASCII values of the sentence

Variables and constants

You can use all declared variables and constants like numbers. In the assembly, these are then replaced by the applicable values. Several commands are available for the declaration of variables and constants . You can find more information on the declaration of variables and constants in the respective chapter further below.

Examples:
MOV A,Port1   ;Writes the value of the memory address "Port1" (=90h) into the accumulator(A)
MOV marker,A;Writes the content of the accumulator's (A) into the address which was defined with the identification marker.

2.4 Comments

Comments are marked by preceding them with a semicolon (;) or high comma ('). Since the assembler principally tries to interpret all text as a command or skip mark, comments must be marked.

After leaving a line with the cursor, comments are colored green for a better overview.

Examples:
;This line is a comment
MOV A,#10h ;This is a comment

2.5 Variables and constants (EQU/BIT/DATA)

In the MC-Editor window for freely selectable identifications, you can set values so that these identifications are interpreted during assembly / compilation and replaced with the respective values. The EQU, the BIT and the DATA commands are available for this purpose.

If, for instance you have connected the operating LED of your MC switch to a certain port pin, you can define the term "Operating_LED". If you write "Operating_LED   EQU   P1.3" into the MC-Editor, the assembler knows that it must interpret the identification "Operating_LED" with the bit specification 93 (=Port 1, Pin 3).

In the case of each microcontroller, there is a series of fixed descriptions for certain register addresses, for instance that the above P1.3 stands for Port 1 Pin 3 (=Bit 93). To set the utilized microcontroller and therefore all related descriptions for the assembler / compiler, this is tied into the MC-Editor window with the INCLUDE Command. You can find more information on the INCLUDE Command in one of the following chapters.

You can use the direction BIT to determine that the value of a description is a bit address. If, after this, for instance, you mistakenly treat the description INC DESCRIPTION as a byte value, the software warns you of this during the assembly. The direction DATA behaves similarly, except it declares the value of the description as a byte.

One example for the use of variables/constants:

```
INCLUDE 8051.mc      ;loads the processor file and determines the register descriptions

Input EQU P3         ;sets the value B0h (Address of P3) for the description Input
EXP_VALUE EQU 125; Sets the value 125 for the description EXP_VALUE

Signal_LED BIT 90h   ;'Sets the value 90h for the Description Signal_LED (type = Bit)
Display DATA 90h     ;Sets the value 90h for the Description Display (type = Byte)

MOV A,Input          ;Copies the content of the address B0h (=Input) into the accumulator (A)
CJNE A,#EXP_WERT,Unequal  ;compares the accumulator content with the value EXP_WERT
SETB Signal_LED      ;Sets the Bit 90h (=Signal_LED)

Unequal:
```

2.6 Jump marks (Label)

You can use so-called labels or fixed addresses for your skip marks. The right address is automatically chosen for the labels during assembly. First, an example of a label:

Start:

You can now reach this position at any place in the program through, e.g., an "LJMP Start" or "LCALL Start". The command, which is directly after or beside the label is then carried out next. Of course you can use the labels for other skip commands as well. Fixed skip marks are required in some places, e.g. where interrupt handling is concerned. This is an example of a fixed skip address:

(0011h):

In order to allow the assembler to differentiate between the fixed skip addresses of the labels, they must be placed in brackets. The command which is directly after or beside it is now in the place of address 0011h of the program memory. All skip marks are always ended with a double period. The same conventions as for commands also apply to fixed skip marks.

After leaving a line with the cursor, skip marks are colored blue for a better overview.

2.7 Data fields (DB, DW)

There are several types of data fields, of which the „DB“ data fields are used as the standard. Within these data fields, you may use any desired number specifications, constants and ASCII symbols. The individual numbers must be separated by a comma (ASCII symbols are placed in parentheses). Aside from the DB-Data fields (DB=DataByte), which are used for values of 0-255 (1 byte), you can also use DW-Data fields (DW=DataWord), which are used for values of 0-65535 (2 bytes).

To allow addressing of the data fields, they are preceded by a variable or fixed skip mark.

Here are some examples:

```
'Read the first value from the first variables data field and write this into the
accumulator:
CLR A
MOV DPTR,#Datafield1
MOVC A,@A+DPTR

'Read the first value from the second variable data field and write this into the
accumulator:
CLR A
MOV DPTR,#Datafield2
MOVC A,@A+DPTR

' Read the first value from the data field at address 0700h and write this into
the accumulator:
CLR A
MOV DPTR,#0700h
MOVC A,@A+DPTR

Datafield1:
DB A0h,B7h,30h,02h,FFh,10h
Datafield2:
DB 22,255,12,0,55
```

(0700):
DB "Mixed data field...", 20h, 30h, 220, !10101010

DW 20h, 130h, 2220, !1010101010101010

2.8 Include directions

With the aid of the INCLUDE direction, you can tie the contents of a file into any place in your program. The file to be linked is initially searched for in the lists of the file which is currently in the MC-Editor, and then in the lists of \INCLUDE, and last in the list \MC of the Prog-Studio software.

When required you can also state the path of the file to be linked directly. Name the path in parentheses, for example: INCLUDE "C:\Windows\Test\TestProz.mc"

All directions are permitted in the files to be linked, therefore you can do modular programming to the fullest degree. All descriptions which have been defined once also apply to subsequently linked files, and you can, for instance, also switch back and forth between routines of the program and the module with skip commands. In principle, the lines in the file to be linked replace the Include commands during assembly.

You should make use of the Include direction at the beginning of each of your programs to set the microcontroller which is to be used. The Prog-Studio software contains, within the file list \MCa series of so-called processor files which you can link, for instance, with "Include 8051.mc" . All register descriptions and the respective register addresses of the respective microcontrollers are set in these files.

However, please note that the file is tied in at the location at which you place the Include direction. Therefore you should, for instance, not tie the file MATH.ASM (in the file list \include within the Prog-Studio software) into the first few lines of your program, since the routines which it contains would then also occupy the first program memory addresses. In such a case, it is recommended to either tie the file in at the end, or tie it in at the beginning with a preceding, fixed label. Here are a few examples:

```
INCLUDE 8051.mc

'Your program...

INCLUDE MATH.ASM
```

3. Basic Programming

3.1 General notes

In the MC-Editor window, you can also use some basic commands which the program compiles / changes into the required microcontroller commands. Depending on the execution of a basic command, a large number of MC commands may be necessary for this, since the operations require, for instance, the accumulator register, which must then first be saved to the stack and restored again afterwards, etc..

The basic commands do not quite correspond to the basic standard, but were adapted to microcontroller programming. Fixed numbers must be marked with a preceding number sign (#) , otherwise they will be interpreted as memory addresses.

The original basic command "FOR A = 1 TO 10" must therefore be written here as follows: "FOR A = #1 to #10".

You can find a complete list of all basic commands in the additional window „Basic Commands".

3.2 If Then ... End If

Depending on the execution of the „If Then" direction, it must be replaced within the software by other microcontroller commands. For this reason, the individual variants take differing amounts of time and require different numbers of bytes within the program memory.

If you want to place several commands after an "If Then" command, you must place each command in a new line and then end the block with an "End If" . In this case, you can also use the ELSE command. The use of the Else command increases the number of the required bytes by 3 bytes, and the machine cycle number at completion of the condition by 2 cycles. The ELSE branch is optional.

Here are a few examples:

In a command after „If Then":

```
If A = #122 then INC R7
```

With several commands between „If Then" and „End If":

```
If A = #122 then
  SETB P1.1
Else
  CLR P1.1
End If
```

3.3 If Then ... End If for individual bits

To apply the commands to individual bits, there is a special syntax:

For a command after „If Then“:

```
 If BIT P1.1 then INC R7
```

For several commands between „If BIT Then“ and „End If“:

```
 If NOT BIT 0 then
   SETB P1.1
 Else
   CLR P1.1
 End If
```

3.4 For ... Next

Depending on the execution of the „For ... Next“ direction, this must be replaced in the software by other microcontroller commands. For this reason, the individual variants take differing amounts of time and require different numbers of bytes of program memory.

An example of a For ... Next direction:

```
For A = #112 to #240
 If A = Port3 then
   SETB P1.1
 End If
Next A
```

3.5 Mathematical directions

Within the MC-Editor window, you can also directly use mathematical operations, which are calculated immediately during assembly. You can use the Addition(+), Subtraction(-), Multiplication(*) and Division(/) as well as the Potency function(^). Here is an example:

```
Pulse duration data 20h
Pause duration data 30h
Period duration equ Pulse duration + Pause duration

MOV A,#Period duration
```

Example 2:

```
Waiting time equ 8h
Mov Port3, Waiting time ^ 2 + 5
```

The mathematical operations are processed in the order in which they occur. The rules of mathematical order do NOT apply. Any desired numbers and descriptions can be mixed, and the Prog-Studio software warns of exceeding the areas which are allowed for the respective commands.

Please note that the calculation already takes place during assembly and not in the microcontroller! The command "MOV A,10+11" does not mean that the content of Address 10 plus the content of Address 11 is written into the accumulator, but rather, the content of Address 21 is written into the accumulator! The command MOV A,10+11 becomes the command MOV A,21 during assembly.

4. The Debugger

4.1 Manner of proceeding

The integrated debugger in the Prog-Studio software helps you to check your programs for errors and runtimes and, where required, remove errors and improve runtimes. Functions such as run-through, individual step and stop points are available for this purpose, while the RAM and SFR contents (therefore, for instance, also the conditions for the ports) as well as the machine cycles and the program address are constantly displayed and updated in the additional windows.

The debugger is also an excellent aid for beginners who would like to learn the use and programming of the microcontroller. To see the effect of a command directly on the monitor is much better than learning it through reading; and the many beginners' errors which many of us have made and probably are still making can be quickly discovered and removed.

Before the debugger can begin to work, it must assemble the program, arrange the register and RAM addresses, etc. In the menu "Debugger" and in the toolbar, you will find the entry "prepare debugger data". Clicking on it will take care of the above named processes, after which the functions of the debugger are cleared for use (previously, they were deactivated/tuned out).

Clicking on "Start" / "Stop" (in the debugger menu, in the Window Debugger Control or on the traffic light in the toolbar), you can start or stop the run-through. One command after the other is carried out until you either click "Stop" to interrupt the run-through, or the run-through meets the command "STOP" in the MC-Editor. You can also use the option "Delay process" in the Debugger Control additional window to delay the process so that it is easier for you to follow.

Please note: in the current version, the debugger does not yet support the simulation of the serial ports and not all special SFR additional functions of special members of the MCS-51 family.

4.2 Stop points

Clicking on "Start" / "Stop" (in the debugger menu, in the Window Debugger Control or the traffic light in the toolbar) allows you to start or stop the run-through. One command after the other is carried out until you either end the run-through by clicking on "Stop" or the run-through its the command "STOP" in the MC-Editor.

The "Stop" command can be used in the MC Editor like a "normal" command. During assembly, it is only applied if the assembly takes place for a debugging run. It does not need to be removed prior to burning onto a MC/Eprom.

An example for use:

```
'[...]
INC A
MOV A,@R0
IF A > #222 then STOP
LCALL Anywhere
```

In the above example, the run-through is only stopped when the condition A > #222 is fulfilled. However, the Stop command can, of course, be applied individually as well.

4.3 Conditional Assembly of certain Areas

Areas of the program can be assembled / compiled conditionally depending on whether the code is being created for the debugger or for the MC. This expansion enables the routines which are already functioning safely and have no influence on other routines (e.g. display, pause routines) to be excluded for the debugging run. In the following example, the lines between the directions "#IF NOT DEBUGGING" and "#END IF" are only assembled if the assembly is not taking place for the debugger:

```
#IF NOT DEBUGGING
  LCALL Display_refresh
  LCALL Pause routine
#END IF
```

In the next example, the lines are only assembled for the debugger:

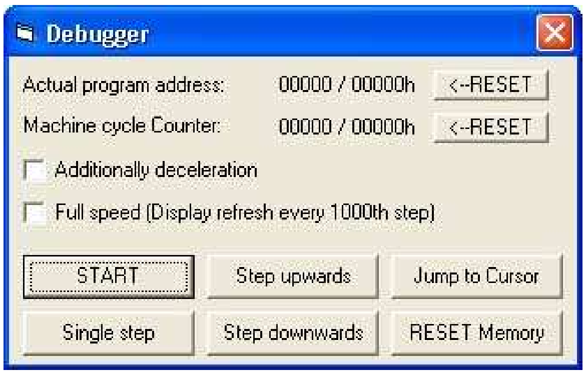
```
#IF DEBUGGING
  LCALL Test_routine
  MOV A,#0
#END IF
```

4.4 The additional windows in the debugging run

In the menu „View,“ you can choose to display a number of additional windows which will make programming easier for you. Here we introduce the four additional windows which are only available when the debugger has been started.

The additional window Debugger: Control

This additional window contains some buttons for guiding the debugger. You can run the debugger, carry out individual steps, and skip up or down lines without carrying out the commands in them.



Debugger

Actual program address: 00000 / 00000h <--RESET

Machine cycle Counter: 00000 / 00000h <--RESET

[ ] Additionally deceleration

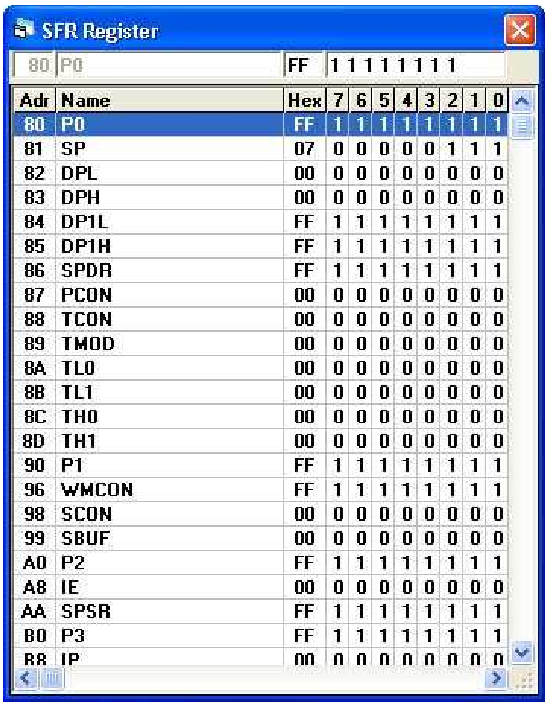
[ ] Full speed (Display refresh every 1000th step)

You can use the reset buttons to reset the program address, the machine cycle counter, or the memory.

Through the checkboxes, you can additionally slow down the debugger (in order to be better able to follow certain processes) or speed them up.

The additional window Debugger: SFR

Here you can view the contents of the Special Functions Register (SFR) of the microcontroller. In the text fields above the window, you can directly access/ alter the contents. You can also change the bits in the field to the respective opposite condition by double-clicking on them. The descriptions (names) of the individual registers are taken out of the process file. Only those registers which are present in the selected microcontroller are shown.



SFR Register

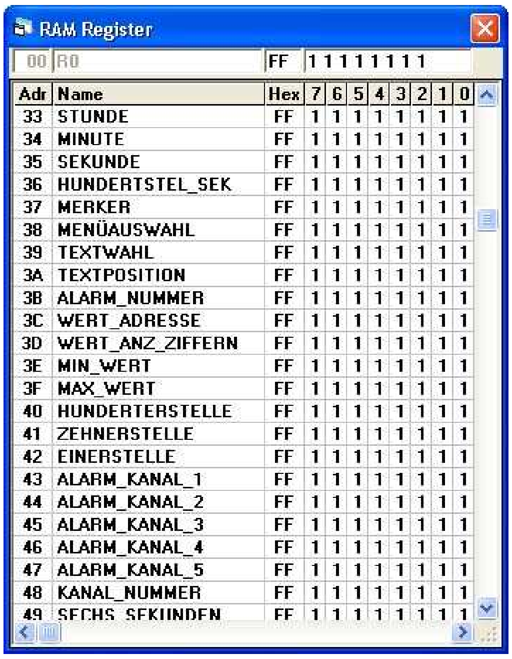
80 P0 FF 11111111

Adr	Name	Hex	7	6	5	4	3	2	1	0
80	P0	FF	1	1	1	1	1	1	1	1
81	SP	07	0	0	0	0	0	1	1	1
82	DPL	00	0	0	0	0	0	0	0	0
83	DPH	00	0	0	0	0	0	0	0	0
84	DP1L	FF	1	1	1	1	1	1	1	1
85	DP1H	FF	1	1	1	1	1	1	1	1
86	SPDR	FF	1	1	1	1	1	1	1	1
87	PCON	00	0	0	0	0	0	0	0	0
88	TCON	00	0	0	0	0	0	0	0	0
89	TMOD	00	0	0	0	0	0	0	0	0
8A	TL0	00	0	0	0	0	0	0	0	0
8B	TL1	00	0	0	0	0	0	0	0	0
8C	TH0	00	0	0	0	0	0	0	0	0
8D	TH1	00	0	0	0	0	0	0	0	0
90	P1	FF	1	1	1	1	1	1	1	1
96	WMCON	FF	1	1	1	1	1	1	1	1
98	SCON	00	0	0	0	0	0	0	0	0
99	SBUF	00	0	0	0	0	0	0	0	0
A0	P2	FF	1	1	1	1	1	1	1	1
A8	IE	00	0	0	0	0	0	0	0	0
AA	SPSR	FF	1	1	1	1	1	1	1	1
B0	P3	FF	1	1	1	1	1	1	1	1
B8	IP	00	0	0	0	0	0	0	0	0

The additional window Debugger: RAM

Display of the contents of the integrated RAM in the microcontroller. In the text fields above the window, you can directly access / alter the contents. Also, the bits in the field can be changed to the respective opposite condition by double-clicking on them. The descriptions (names) of the individual registers are taken from the EQU/BYTE directions (if present). For instance, if you have declared a variable/constant with the direction "Pulse time EQU 20h" in the MC-Editor window, this name is shown in the RAM at the address 20h. Furthermore, the register descriptions R0-R7 are shown in the addresses of the current register bank.

Please note that not every MCS-51 microcontroller has 256 bytes of integrated RAM; many only have 128 bytes! Since this is not, however, defined in the processor files, the debugger RAM window always shows the entire 256 bytes.

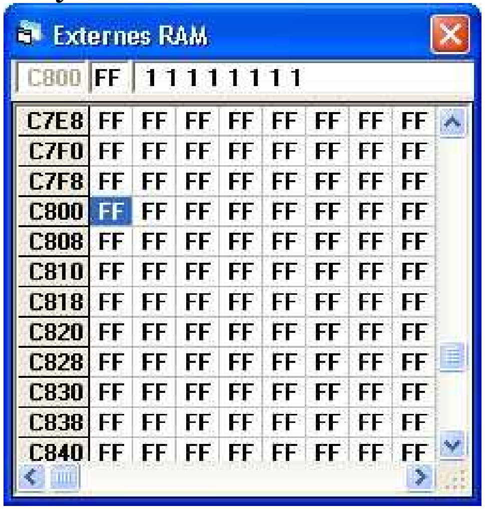


00 R0 FF 11111111

Adr	Name	Hex	7	6	5	4	3	2	1	0
33	STUNDE	FF	1	1	1	1	1	1	1	1
34	MINUTE	FF	1	1	1	1	1	1	1	1
35	SEKUNDE	FF	1	1	1	1	1	1	1	1
36	HUNDERTSTEL_SEK	FF	1	1	1	1	1	1	1	1
37	MERKER	FF	1	1	1	1	1	1	1	1
38	MENÜAUSWAHL	FF	1	1	1	1	1	1	1	1
39	TEXTWAHL	FF	1	1	1	1	1	1	1	1
3A	TEXTPOSITION	FF	1	1	1	1	1	1	1	1
3B	ALARM_NUMMER	FF	1	1	1	1	1	1	1	1
3C	WERT_ADRESSE	FF	1	1	1	1	1	1	1	1
3D	WERT_ANZ_ZIFFERN	FF	1	1	1	1	1	1	1	1
3E	MIN_WERT	FF	1	1	1	1	1	1	1	1
3F	MAX_WERT	FF	1	1	1	1	1	1	1	1
40	HUNDERTERSTELLE	FF	1	1	1	1	1	1	1	1
41	ZEHNERSTELLE	FF	1	1	1	1	1	1	1	1
42	EINERSTELLE	FF	1	1	1	1	1	1	1	1
43	ALARM_KANAL_1	FF	1	1	1	1	1	1	1	1
44	ALARM_KANAL_2	FF	1	1	1	1	1	1	1	1
45	ALARM_KANAL_3	FF	1	1	1	1	1	1	1	1
46	ALARM_KANAL_4	FF	1	1	1	1	1	1	1	1
47	ALARM_KANAL_5	FF	1	1	1	1	1	1	1	1
48	KANAL_NUMMER	FF	1	1	1	1	1	1	1	1
49	SECHS_SEKUNDEN	FF	1	1	1	1	1	1	1	1

The additional window Debugger: EXRAM

This window shows the contents of an external RAM memory which can be connected to a microcontroller. In the text fields above the window, you can access the contents directly.



C800 FF 11111111

C7E8	FF	FF	FF	FF	FF	FF	FF	FF
C7F0	FF	FF	FF	FF	FF	FF	FF	FF
C7F8	FF	FF	FF	FF	FF	FF	FF	FF
C800	FF	FF	FF	FF	FF	FF	FF	FF
C808	FF	FF	FF	FF	FF	FF	FF	FF
C810	FF	FF	FF	FF	FF	FF	FF	FF
C818	FF	FF	FF	FF	FF	FF	FF	FF
C820	FF	FF	FF	FF	FF	FF	FF	FF
C828	FF	FF	FF	FF	FF	FF	FF	FF
C830	FF	FF	FF	FF	FF	FF	FF	FF
C838	FF	FF	FF	FF	FF	FF	FF	FF
C840	FF	FF	FF	FF	FF	FF	FF	FF