Handbuch

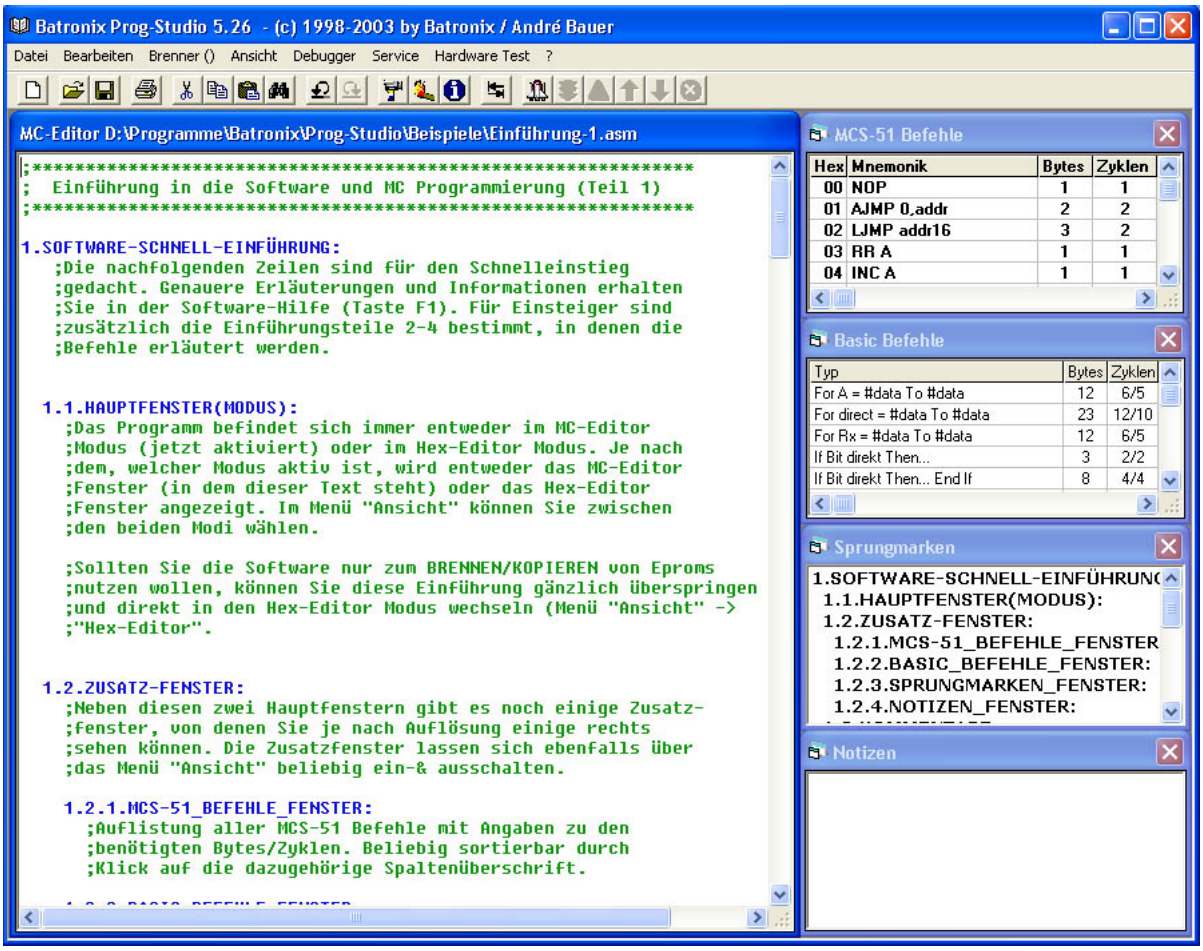
Prog-Studio Software
MC-Editor

- Allgemeines und Oberfläche -
- Assembler Programmierung -
- Basic Programmierung -
- Der Debugger -

1. Allgemeine Informationen

1.1 Was kann die Prog-Studio Software im MC-Editor Modus

Im MC-Editor Modus können Sie eigene Assembler / Basic Programme für die Mikrocontroller der MCS-51 Reihe entwickeln und mit dem eingebauten Debugger testen. Wenn dieses nicht Ihr Anwendungsbereich ist, brauchen Sie sich nur die allgemeine Anleitung und nicht diesen Spezialteil der Prog-Studio Software Anleitung anzusehen.



1.2 Erste Schritte und Beispielprogramme

Zunächst möchten wir Ihnen empfehlen, einen Blick auf die mitgelieferten Beispielprogramme zu werfen. Sie können diese im Unterverzeichnis \Beispiele der Prog-Studio Software finden. Klicken Sie im Menü „Datei“ auf „Öffnen“ und wählen Sie im Dialogfenster die Datei „Einführung-1.asm“ an.

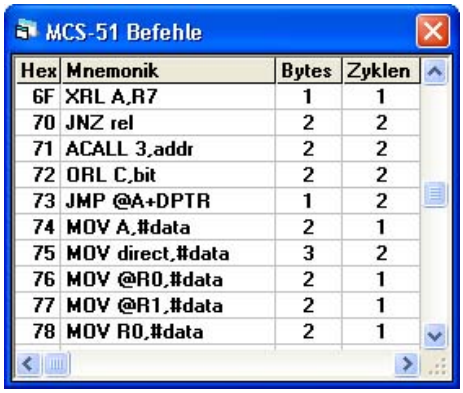
Die Einführung, die Ihnen dort als erstes Assembler Programm gezeigt wird, enthält bereits einen groben Überblick mit den Möglichkeiten der Programmierung von Assembler Programmen in der Prog-Studio Software. Sie finden auch noch weitere Einführungsteile sowie andere Beispielprogramme in diesem Verzeichnis.

1.3 Die Zusatzfenster

Über das Menü „Ansicht“ können Sie eine ganze Reihe an Zusatzfenstern anzeigen lassen, die Ihnen die Programmierung erleichtern sollen. Hier werden zunächst die vier Zusatzfenster vorgestellt, die ständig zur Verfügung stehen. Weiter unten im „Debugger“ Kapitel können Sie die Informationen zu weiteren vier Zusatzfenstern lesen, die nur bei laufendem Debugger aktiv sind.

Das Zusatzfenster MCS-51 Befehle

Dieses Zusatzfenster enthält eine Liste aller MCS51 Befehle. Zudem enthält es für jeden Befehl die Information über seinen Hex-Code und die dafür benötigten Bytes sowie Zyklen.



Hex	Mnemonik	Bytes	Zyklen
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

Sie können die Befehlsliste nach beliebigen Kriterien sortieren lassen, klicken Sie einfach auf die Bezeichnung in der obersten (grauen) Zeile.

Wenn Sie sich im Hex-Editor Modus befinden und das Fenster MCS-51 Befehle geöffnet haben, wird per Doppelklick auf ein Byte der entsprechende Befehl in der Liste markiert.

Feste Zahlenangaben werden in der Tabelle mit #data, Speicheradressen werden mit direct und der Akkumulator wird mit A bezeichnet. Die Register werden mit Rx bezeichnet. Wenn Sie die Befehle benutzen tippen Sie anstatt "data" oder "direct" natürlich die entsprechenden Werte oder auch Konstanten ein und das Rx ersetzen Sie durch das gewünschte Register (R0-R7).

Das Zusatzfenster Basic Befehle

Dieses Zusatzfenster gibt Ihnen eine kurze Übersicht über die verwendbaren Basic Befehle. Zudem enthält es für jeden Befehl die Information über die benötigten Bytes sowie Zyklen.

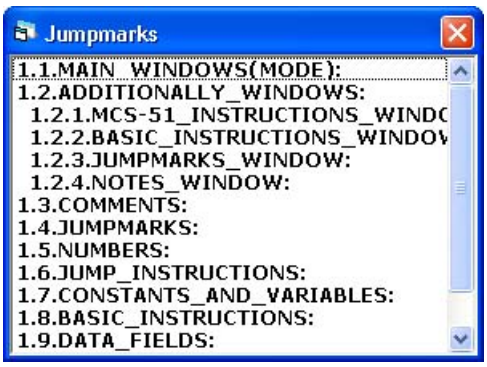
Typ	Bytes	Zyklen
For A = #data To #data	12	6/5
For direct = #data To #data	23	12/10
For Rx = #data To #data	12	6/5
If Bit direkt Then...	3	2/2
If Bit direkt Then... End If	8	4/4
If Not Bit direkt Then...	3	2/2
If Not Bit direkt Then... End If	8	4/4
If A = #data Then...	3	2/2
If A = #data Then...End If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

Für die Zyklen werden zwei Zahlen angegeben, die erste steht für die benötigten Zyklen des ersten Durchlaufes einer For-Next Schleife bzw. für die benötigten Zyklen der IF-Then Anweisung, wenn diese erfüllt ist. Die zweite Zahl steht für die benötigten Zyklen jedes weiteren Durchlaufes einer For-Next Schleife bzw. für die benötigten Zyklen einer IF-Then Anweisung, wenn diese nicht erfüllt ist.

Nähere Informationen über die Programmierung mit Basic Befehle können Sie in dem entsprechenden Kapitel weiter unten lesen.

Das Zusatzfenster Sprungmarken

In diesem Zusatzfenster können Sie sich immer schnell einen Überblick über ihr Programm beschaffen oder auch schnell und gezielt zu bestimmten Unterroutinen springen indem Sie auf den entsprechenden Eintrag klicken. Gerade in größeren Programmen ist dies sehr hilfreich und Sie sollten diese Funktion auch nutzen. Die Einträge in diesem Fenster werden so wie im Text dargestellt, eingerückte Marken werden also auch hier eingerückt gezeigt.



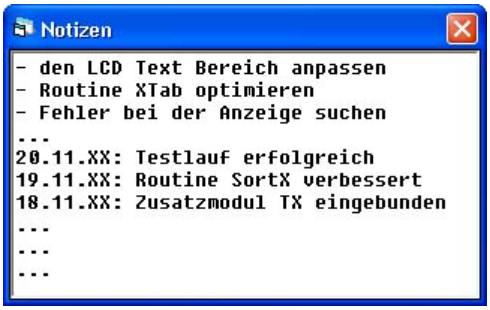
Um auch in wirklich großen Programmen noch die Übersicht zu behalten, können Sie die Unterdrückung der Anzeige von weniger wichtigen Sprungmarken aktivieren. Im Menü "Bearbeiten" -> "Optionen" befinden sich dafür die Optionen, Sprungmarken mit einem vorangehenden Tabzeichen und/oder einem vorangehenden Unterstrich "_" nicht im Sprungmarkenfenster anzuzeigen.

Sie können sich bei aktiviertem Sprungmarkenfenster mit der Betätigung einer Taste die erste Sprungmarke im Fenster anzeigen lassen, die mit dem gedrücktem Zeichen beginnt (Beispiel: Nach der Betätigung der Taste "T" wird die erste Sprungmarke angezeigt, die mit dem Buchstaben T beginnt).

Nähere Informationen über die Programmierung mit Sprungmarken können Sie in dem entsprechenden Kapitel weiter unten lesen.

Das Zusatzfenster Notizen

In diesem Fenster können Sie alles eintragen, was Ihnen notierenswert erscheint. Der Assembler beachtet die Einträge zwar nicht bei der Assemblierung, speichert aber die Notizen für jedes Ihrer Programme unter dem Programmnamen mit der Endung .not ab.



2. Assembler Programmierung

2.1 Allgemeine Hinweise

Nachdem Sie ein Mikrocontroller Programm im MC-Editor Fenster erstellt haben, muss der Assembler/Compiler Ihre Anweisungen in die nötigen Hex-Codes umsetzen, damit der Mikrocontroller oder ein Speicherchip damit beschrieben werden können. Damit der Assembler Ihr Programm richtig deuten kann, müssen Sie z.B. bestimmte Zahlenangaben kennzeichnen, Konstanten und Sprungmarken angeben, Kommentare und Datenfelder kennzeichnen u.s.w..

2.2 Befehle und grundlegende Programmierung

Wir können Ihnen an dieser Stelle leider keine ausführliche Referenz über den kompletten Befehlsumfang oder die grundlegende Assemblerprogrammierung angeben, da dieses den Rahmen der Anleitung sprengen würde. Sie finden alles Wissenswerte über die Befehle sowie den Aufbau und die Arbeitsweise der MCS-51 Mikrocontroller in diverser Literatur, sowie z.B. den Befehlsumfang mit Erläuterungen als PDF Datei auf unserer Webseite Batronix.com im Downloadbereich. Als schnelle Nachschlaghilfe eignet sich auch das Zusatzfenster „MCS-51 Befehle“ der Prog-Studio Software, das alle Befehle auflistet.

Wir gehen hier nur auf die speziellen Möglichkeiten der Prog-Studio Software ein und möchten in Bezug auf den Befehlssatz, die Programmierungstechniken sowie die Mikrocontroller Architektur auf entsprechende Literatur verweisen.

2.3 Zahlenangaben

Sie können nach Belieben führende Nullen zu den Zahlen dazufügen oder weglassen. Eine "0009" wird demnach einfach als "9" gedeutet.

Unterscheidung zwischen Zahlen und Speicheradressen

Zahlenangaben ohne vorausgehende Raute (#) werden in der Prog-Studio Software als Speicheradressen gedeutet. Möchten Sie anstatt einer Speicheradresse direkt eine Zahl angeben, müssen Sie diese mit einer vorangehenden Raute(#) kennzeichnen. Diese Kennzeichnung ist bei den Intel Mnemonics üblich und hier auch bei den Basic Befehlen notwendig, um zwischen Speicheradressen und Zahlen unterscheiden zu können.

Hexadezimale Zahlen

Es gibt drei Möglichkeiten für den Assembler, hexadezimale Zahlenangaben als solche zu erkennen. Die erste Möglichkeit (Standard) ist es, diese durch ein "h" abzuschließen. Weiterhin ist es möglich, sie mit einem vorausgehendem Dollar-Zeichen ($) zu kennzeichnen. Die dritte Möglichkeit, die Zahlen ohne Kennzeichnung als hexadezimale Zahlen zu deuten, kann im Menü "Bearbeiten" -> "Optionen" aktiviert werden. Davon möchten wir aber abraten, da es nicht dem Standard entspricht und nur noch als Überbleibsel für die Kompatibilität mit alten Versionen integriert ist.

Einige Beispiele für hexadezimale Zahlenangaben:

```
MOV A,3Dh     ;Kopiert den Wert der Speicheradresse 3D (hex) in den Akku (A)
MOV A,#3Dh    ;Kopiert die hexadezimale Zahl 3D in den Akku (A)

MOV A,$3D     ;Kopiert den Wert der Speicheradresse 3D (hex) in den Akku (A)
MOV A,#$3D    ;Kopiert die hexadezimale Zahl 3D in den Akku (A)
```

Dezimale Zahlen

Es gibt auch für dezimale Zahlenangaben drei Möglichkeiten für den Assembler, diese als solche zu erkennen. Die erste Möglichkeit (Standard) ist es, dezimale Zahlen ohne weitere Kennzeichnung anzugeben. Das funktioniert aber nur, wenn im Menü "Bearbeiten" -> "Optionen" die Deutung von Zahlen ohne Kennzeichnung als dezimale Zahlen aktiviert ist (Standardeinstellung). Weiterhin ist es möglich, sie mit einem abschließendem "d" oder einem vorausgehendem Prozentzeichen (%) zu kennzeichnen.

Einige Beispiele für dezimale Zahlenangaben:

```
MOV A,212     ;Schreibt den Wert der Speicheradresse 212 in den Akku A)
MOV A,#212    ;Schreibt die dezimale Zahl 212 in den Akku (A)

MOV A,212d    ;Schreibt den Wert der Speicheradresse 212 in den Akku (A)
MOV A,#212d ;Schreibt die dezimale Zahl 212 in den Akku (A)

MOV A,%212    ;Schreibt den Wert der Speicheradresse 212 in den Akku (A)
MOV A,#%212 ;Schreibt die dezimale Zahl 212 in den Akku (A)
```

Binäre Zahlen

Binäre Zahlenangaben können durch ein abschließendes "b" (Standard) oder durch ein führendes Ausrufezeichen (!) deklariert werden.

Einige Beispiele für binäre Zahlenangaben:

```
MOV A,10101010b    'Schreibt den Wert der Speicheradresse 10101010 (binär) in den Akku
MOV A,#10101010b   'Schreibt die binäre Zahl 10101010 in den Akku

MOV A,!10101010    'Schreibt den Wert der Speicheradresse 10101010 (binär) in den Akku
MOV A,#!10101010   'Schreibt die binäre Zahl 10101010 in den Akku
```

ASCII Zeichen

ASCII Zeichenketten müssen in Anführungsstriche (") gesetzt werden.

Einige Beispiele für ASCII Zeichenketten:

MOV A,#"H" ;Schreibt den ASCII Wert des Zeichens "H"(=72dez) in den Akku

DB "Ein String für Ihre LCD-Ansteuerung" ;Ein Datenfeld mit den ASCII Werten des Satzes

Variablen und Konstanten

Sie können alle deklarierten Variablen und Konstanten wie Zahlen verwenden. Bei der Assemblierung werden diese dann mit den dazugehörigen Werten ersetzt. Für die Deklarierung von Variablen und Konstanten stehen mehrere Befehle zur Verfügung. Mehr zur Deklaration von Variablen und Konstanten können Sie weiter unten in dem entsprechenden Kapitel lesen.

Beispiele:
MOV A,Port1 ;Schreibt den Wert der Speicheradresse "Port1" (=90h) in den Akku(A)
MOV Merker,A ;Schreibt den Inhalt des Akku's (A) in die Adresse, die mit der Bezeichnung Merker definiert wurde.

2.4 Kommentare

Kommentare werden durch ein vorangehendes Semikolon(;) oder Hochkomma(') gekennzeichnet. Da der Assembler prinzipiell jeden Text als Befehl oder Sprungmarke zu deuten versucht, müssen Kommentare gekennzeichnet werden.

Nach dem Verlassen einer Zeile mit dem Cursor werden Kommentare zur besseren Übersicht grün eingefärbt.

Beispiele:
;Diese Zeile ist ein Kommentar
MOV A,#10h ;Das hier ist ein Kommentar

2.5 Variablen und Konstanten (EQU/BIT/DATA)

Sie können im MC-Editor Fenster für frei wählbare Bezeichnungen Werte festgelegen, so dass beim Assemblieren/Kompilieren diese Bezeichnungen gedeutet und durch die entsprechenden Werte ersetzt werden. Dafür stehen der EQU, der BIT und der DATA Befehl zur Verfügung.

Haben Sie z.B. die Betriebs LED Ihrer MC-Schaltung an einen bestimmten Portpin angeschlossen, so können Sie den Begriff "Betriebs_LED" definieren. Schreiben Sie in den MC-Editor "Betriebs_LED EQU P1.3" dann wird der Assembler wissen, dass er den Ausdruck "Betriebs_LED" mit der Bitangabe 93 (=Port 1, Pin3) deuten muss.

Für jeden Mikrocontroller gibt es eine ganze Reihe an festen Bezeichnungen für bestimmte Register-Adressen, so wie z.B. dass obige P1.3 für Port 1 Pin 3 (=Bit 93) steht. Um für den Assembler/Compiler den verwendeten Mikrocontroller und damit alle dazugehörigen Bezeichnungen festzulegen, wird dieser mit dem INCLUDE Befehl im MC-Editor Fenster eingebunden. Mehr Informationen über den INCLUDE Befehl können Sie in einem folgendem Kaptitel lesen.

Mit der Anweisung BIT können Sie festlegen, dass es sich bei dem Wert der Bezeichnung um eine Bitadresse handelt. Wenn Sie danach die Bezeichnung mit z.B. INC BEZEICHUNG fälschlicherweise als Byte-Wert behandeln, warnt Sie die Software beim Assemblieren davor. Die Anweisung DATA verhält sich ähnlich, nur dass Sie den Wert der Bezeichnung als Byte deklariert.

Ein Beispiel für den Einsatz von Variablen/Konstanten:

```
INCLUDE 8051.mc      ;Lädt das Prozessorfile und legt die Register Bezeichnungen fest

Eingang EQU P3       ;Legt den Wert B0h (Adresse von P3) für die Bezeichnung Eingang fest
SOLL_WERT EQU 125;Legt den Wert 125 für die Bezeichnung SOLL_WERT fest

Signal_LED BIT 90h   ;'Legt den Wert 90h für die Bezeichnung Signal_LED fest (typ = Bit)
Anzeige DATA 90h     ;Legt den Wert 90h für die Bezeichnung Anzeige fest (typ = Byte)

MOV A,Eingang        ;Kopiert den Inhalt der Adresse B0h (=P3=Eingang) in den Akku (A)
CJNE A,#SOLL_WERT,Ungleich       ;Vergleicht den Akkuinhalt mit dem Wert SOLL_WERT
SETB Signal_LED      ;Setzt das Bit 90h (=Signal_LED)

Ungleich:
```

2.6 Sprungmarken (Label)

Sie können so genannte Label oder feste Adressen für Ihre Sprungmarken verwenden. Für die Label wird beim Assemblieren automatisch die passende Adresse gewählt. Zunächst ein Beispiel für einen Label:

```
Start:
```

Diese Stelle können Sie nun von jeder beliebigen Stelle im Programm durch z.B. ein "LJMP Start" oder "LCALL Start" anspringen. Der Befehl, der direkt nach dem bzw. neben dem Label steht wird dann als nächstes ausgeführt. Sie können die Label natürlich auch bei den anderen Sprungbefehlen benutzen. An einigen Stellen sind feste Sprungmarken nötig, z.B. wenn es um die Interrupt Behandlung geht. Hier ein Beispiel für eine feste Sprungadresse:

```
(0011h):
```

Damit der Assembler die festen Sprungadressen von den Labeln unterscheiden kann, müssen Sie in Klammern gesetzt werden. Der Befehl, der direkt danach bzw. daneben steht, steht nun an der Adresse 0011h des Programmspeichers. Alle Sprungmarken werden immer mit einem Doppelpunkt abgeschlossen. Es gelten bei den festen Sprungmarken die gleichen Konventionen wie bei den Befehlen.

Nach dem Verlassen einer Zeile mit dem Cursor werden Sprungmarken zur besseren Übersicht blau eingefärbt.

2.7 Datenfelder (DB, DW)

Es gibt mehrere Sorten von Datenfeldern, wobei als Standard die "DB" Datenfelder eingesetzt werden. In diesen Datenfeldern können Sie beliebige Zahlenangaben, Konstanten und ASCII Zeichen verwenden, wobei die einzelnen Zahlen durch ein Komma getrennt werden müssen (ASCII Zeichen werden in Anführungsstriche gesetzt). Neben den DB-Datenfeldern (DB=DataByte), die für Werte von 0-255 (1 Byte) verwendet werden, können Sie auch DW-Datenfelder (DW=DataWord) verwenden, die für Werte von 0-65535 (2 Byte) genutzt werden.

Damit das Datenfeld adressierbar ist, wird eine davor stehende, variable oder feste Sprungmarke verwendet.

Hier einige Beispiele:

'Lese den ersten Wert aus dem ersten variablen Datenfeld und schreibe diesen in den Akku:
CLR A
MOV DPTR,#Datenfeld1
MOVC A,@A+DPTR

'Lese den ersten Wert aus dem zweitem variablen Datenfeld und schreibe diesen in den Akku:
CLR A
MOV DPTR,#Datenfeld2
MOVC A,@A+DPTR

'Lese den ersten Wert aus dem Datenfeld an Adresse 0700h und schreibe diesen in den Akku:
CLR A
MOV DPTR,#0700h
MOVC A,@A+DPTR

Datenfeld1:
DB A0h,B7h,30h,02h,FFh,10h
Datenfeld2:
DB 22,255,12,0,55

(0700):
DB "Gemischtes Datenfeld...", 20h, 30h, 220, !10101010

DW 20h, 130h, 2220, !1010101010101010

2.8 Include Anweisungen

Mit Hilfe der INCLUDE Anweisung können Sie an einer beliebigen Stelle Ihres Programms den Inhalt einer Datei einbinden. Die einzubindende Datei wird zunächst im Verzeichnis der gerade im MC-Editor stehenden Datei, dann in dem Verzeichnis \INCLUDE und zuletzt in dem Verzeichnis \MC der Prog-Studio Software gesucht.

Bei Bedarf können Sie den Pfad der einzubindenden Datei aber auch direkt angeben. Setzen Sie dafür den Pfad in Anführungsstriche. Beispiel: INCLUDE "C:\Windows\Test\TestProz.mc"

In den einzubindenden Dateien sind alle Anweisungen erlaubt, Sie können also im vollem Maße modular programmieren. Alle einmal definierten Bezeichnungen gelten auch für nachfolgend eingebundenen Dateien und Sie können z.B. auch beliebig zwischen Routinen des Programms und der Module mit Sprungbefehlen hin- und her springen. Im Prinzip treten also an die Stelle des Include Befehls beim Assemblieren die Zeilen in der einzubindenden Datei.

Sie sollten die Include Anweisung am Anfang jedes Ihrer Programme nutzen, um damit den verwendeten Mikrocontroller festzulegen. Dafür liegen der Prog-Studio Software im Verzeichnis \MC eine ganze Reihe an so genannten Prozessorfiles bei, die Sie z.B. per "Include 8051.mc" einbinden können. In diesen Dateien sind alle Register Bezeichnungen und die dazugehörigen Registeradressen des jeweiligen Mikrocontrollers festgelegt.

Beachten Sie aber bitte, dass die Datei an der Stelle eingebunden wird, an der Sie die Include Anweisung setzen. Sie sollten also z.B. die Datei MATH.ASM (liegt im Verzeichnis \include der Prog-Studio Software bei) nicht gleich in den ersten Zeilen Ihres Programms einbinden, da die enthaltenen Routinen dann auch die ersten Programmspeicher Adressen belegen würden. Es empfiehlt sich in dem Falle also entweder diese Datei am Ende einzubinden oder am Anfang mit einem vorausgehendem, festgelegtem Label. Hier einige Beispiele:

```
INCLUDE 8051.mc

'Ihr Programm...

INCLUDE MATH.ASM
```

3. Basic Programmierung

3.1 Allgemeine Hinweise

Sie können in dem MC-Editor Fenster auch einige Basic Befehle nutzen, die vom Programm in die nötigen Mikrocontroller Befehle kompiliert/umgesetzt werden. Je nach Ausführung eines Basic Befehls sind dafür eine Vielzahl von MC Befehlen nötig, da bei einigen Operationen z.B. das Akkumulator Register benötigt wird und dieses dann vorher erst auf den Stack gerettet und nachher wieder hergestellt werden muss u.s.w..

Die Basic Befehle entsprechen nicht ganz dem Basic Standart, sondern wurden an die Mikrocontroller Programmierung angepasst. Feste Zahlen müssen mit einer vorangehenden Raute (#) gekennzeichnet werden, sonst werden Sie als Speicheradressen gedeutet.

Der ursprüngliche Basic-Befehl "FOR A = 1 TO 10" muss also hier so geschrieben werden: "FOR A = #1 to #10".

Sie finden eine komplette Liste aller unterstützten Basic Befehle im Zusatzfenster „Basic Befehle“.

3.2 If Then ... End If

Je nach Ausführung der „If Then“ Anweisung, muss diese von der Software durch andere Mikrocontroller Befehle ersetzt werden. Deswegen benötigen die einzelnen Varianten unterschiedlich lange und benötigen unterschiedlich viele Bytes an Programmspeicher.

Wenn Sie nach einem "If Then" Befehl mehrere Befehle setzen möchten, müssen Sie diese jeweils in eine neue Zeile setzen und den Block dann mit einem "End If" abschließen. In diesem Fall können Sie auch den ELSE Befehl verwenden. Die Verwendung des Else Befehl erhöht die Anzahl der benötigten Bytes um 3 Byte und die Maschinentaktanzahl bei Erfüllung der Bedingung um 2 Takte. Der ELSE Zweig ist optional.

Hier einige Beispiele:

Bei einem Befehl nach „If Then“:

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

Bei mehreren Befehlen zwischen „If Then“ und „End If“:

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

3.3 If Then ... End If für einzelne Bits

Für die Anwendung des Befehles auf einzelne Bits gibt es eine spezielle Syntax:

Bei einem Befehl nach „If Then“:

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

Bei mehreren Befehlen zwischen „If BIT Then“ und „End If“:

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

3.4 For ... Next

Je nach Ausführung der „For ... Next“ Anweisung muss diese von der Software durch andere Mikrocontroller Befehle ersetzt werden. Deswegen benötigen die einzelnen Varianten unterschiedlich lange und benötigen unterschiedlich viele Bytes an Programmspeicher.

Ein Beispiel für eine For ... Next Anweisung:

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

3.5 Mathematische Anweisungen

Sie können in dem MC-Editor Fenster auch direkt mathematische Operationen verwenden, wobei diese gleich bei der Assemblierung berechnet werden. Sie können die Addition(+), Subtraktion(-), Multiplikation(*) und Division(/) sowie die Potenzfunktion(^) nutzen. Hier gleich ein Beispiel

```
Pulsdauer data 20h
Pausendauer data 30h
Periodendauer equ Pulsdauer + Pausendauer

MOV A,#Periodendauer
```

Beispiel 2:

```
Wartezeit equ 8h
Mov Port3, Wartezeit ^ 2 + 5
```

Die Rechenoperationen werden in der Reihenfolge abgearbeitet, in der Sie auftreten. Es gelten also NICHT Regeln der Reihenfolge der Mathematik. Es können beliebige Zahlen und Bezeichnungen gemischt werden und die Prog-Studio Software warnt vor dem Überschreiten der für den jeweiligen Befehl zulässigen Bereiche.

Bitte beachten Sie, dass die Berechnung bereits bei der Assemblierung und nicht erst im Mikrocontroller erfolgt! Der Befehl "MOV A,10+11" führt also nicht dazu, dass der Inhalt von Adresse 10 plus den Inhalt von Adresse 11 in den Akku geschrieben wird, sondern das der Inhalt von Adresse 21 in den Akku geschrieben wird! Aus dem Befehl MOV A,10+11 wird bei der Assemblierung der Befehl MOV A,21.

4. Der Debugger

4.1 Arbeitsweise

Der integrierte Debugger der Prog-Studio Software soll Ihnen helfen, Ihre Programme auf Fehler und Laufzeiten zu überprüfen und ggf. Fehler zu beseitigen und Laufzeiten zu verbessern. Dafür stehen Funktionen wie Durchlauf, Einzelschritt und Stoppunkte zur Verfügung während die RAM und SFR Inhalte (damit auch z.B. die Zustände der Ports) sowie die Maschinenzyklen und die Programmadresse ständig aktualisiert in den Zusatzfenstern angezeigt werden.

Außerdem eignet sich der Debugger bestens als Hilfe für Einsteiger, die den Umgang und die Programmierung der Mikrocontroller erlernen möchten. Die Wirkung eines Befehls direkt am Monitor zu sehen ist wesentlich angenehmer als sich diese durch trockene Lektüre anzueignen und gerade die vielen Anfängerfehler, die wohl jedem von uns passiert sind bzw. noch passieren werden, lassen sich schnell entdecken und beseitigen.

Bevor der Debugger ans Werk gehen kann, muss er das Programm assemblieren, die Register und RAM Adressen einrichten, u.s.w.. Im Menü "Debugger" und in der Toolbar finden Sie den Eintrag "Debugger Daten vorbereiten". Mit einem Klick darauf werden die oben genannten Vorgänge erledigt und danach die Funktionen des Debuggers freigeschaltet (Zuvor sind diese deaktiviert/ausgeblendet).

Mit einem Klick auf "Start" bzw. "Stop" (im Menü Debugger, im Fenster Debugger Control oder die Ampel in der Toolbar) können Sie den Durchlauf starten bzw. stoppen. Dabei wird ein Befehl nach dem anderen durchgeführt, bis entweder Sie mit einem Klick auf "Stop" den Durchlauf abbrechen oder der Durchlauf auf den Befehl "STOP" im MC-Editor trifft. Zusätzlich können Sie mit der Option "Ablaufverzögerung" im Debugger Control Zusatzfenster den Ablauf so verzögern, dass Sie dem Lauf besser folgen können.

Bitte beachten: Der Debugger unterstützt in der aktuellen Version noch nicht die Simulierung des seriellen Ports und auch noch nicht alle besonderen SFR-Zusatzfunktion spezieller Mitglieder der MCS-51 Familie.

4.2 Stoppunkte

Mit einem Klick auf "Start" bzw. "Stop" (im Menü Debugger, im Fenster Debugger Control oder die Ampel in der Toolbar) können Sie den Durchlauf starten bzw. stoppen. Dabei wird ein Befehl nach dem anderen durchgeführt, bis entweder Sie mit einem Klick auf "Stop" den Durchlauf abbrechen oder der Durchlauf auf den Befehl "STOP" im MC-Editor trifft.

Der "Stop" Befehl kann wie ein "normaler" Befehl im MC-Editor benutzt werden. Bei der Assemblierung wird er nur beachtet, wenn diese für den Debuglauf erfolgt. Er muss nicht vor dem Brennen des Programmes auf einen MC/Eprom entfernt werden.

Ein Beispiel für den Einsatz:

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

Der Durchlauf wird bei dem obigen Beispiel nur gestoppt, wenn die Bedingung A > #222 erfüllt ist. Der Stop Befehl kann aber natürlich auch einzeln eingesetzt werden.

4.3 Bedingte Assemblierung von bestimmten Bereichen

Bereiche des Programms können bedingt assembliert/kompiliert werden, je nach dem ob der Code für den Debugger oder für den MC erstellt wird. Diese Erweiterung ermöglicht es nun die Routinen, die bereits sicher funktionieren und keinen wichtigen Einfluss auf andere Routinen ausüben (z.B. Anzeige, Pausenroutinen), für den Debuglauf auszuschließen. Bei dem folgendem Beispiel werden die Zeilen zwischen den Anweisungen "#IF NOT DEBUGGING" und "#END IF" nur assembliert, wenn die Assemblierung nicht für den Debugger geschieht:

```
#IF NOT DEBUGGING
  LCALL Anzeige_refresh
  LCALL Pausenroutine
#END IF
```

In dem nächsten Beispiel werden die Zeilen nur für den Debugger assembliert:

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

4.4 Die Zusatzfenster im Debuglauf

Über das Menü „Ansicht“ können Sie eine ganze Reihe an Zusatzfenster anzeigen lassen, die Ihnen die Programmierung erleichtern sollen. Hier werden die vier Zusatzfenster vorgestellt, die nur bei gestartetem Debugger zur Verfügung stehen.

Das Zusatzfenster Debugger: Control

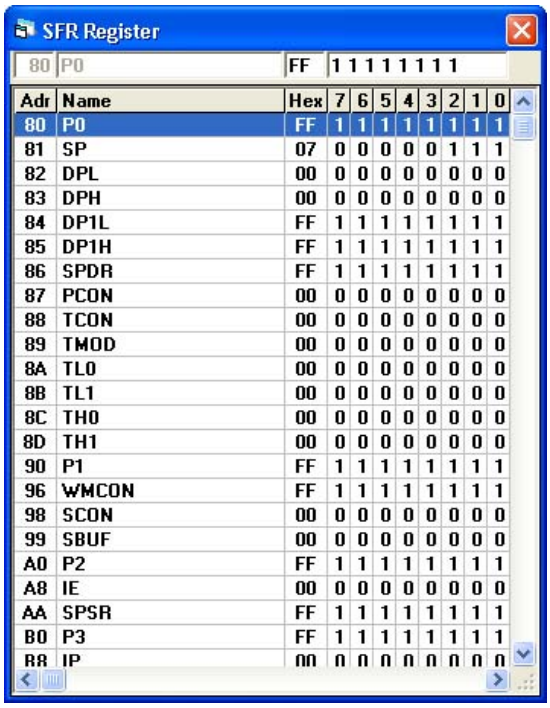
Dieses Zusatzfenster enthält einige Button zur Ansteuerung des Debuggers. Sie können den Debugger laufen lassen, Einzelschritte durchführen sowie Zeilen nach oben oder unten springen ohne die dort stehenden Befehle auszuführen.

Mit den Reset-Buttons können Sie die Programmadresse, den Maschinenzykluszähler oder den Speicher zurücksetzen.

Über die Checkboxen können Sie den Lauf des Debuggers zusätzlich abbremsen (um bestimmte Vorgänge besser verfolgen zu können) oder diesen weiter beschleunigen.

Das Zusatzfenster Debugger: SFR

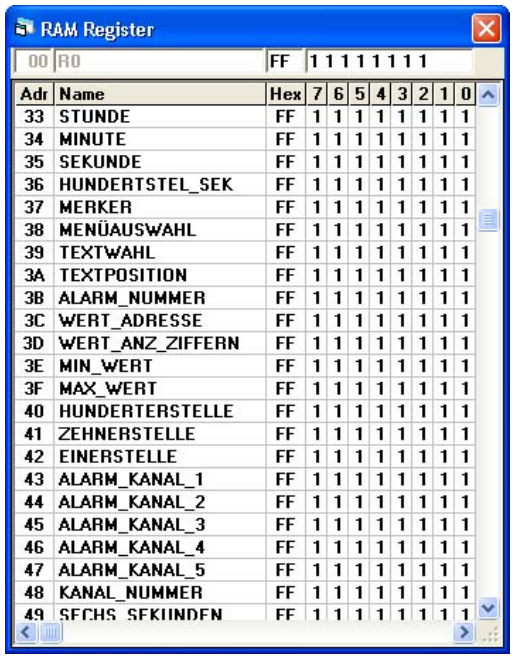
Hier können Sie die Inhalte der Spezial-Funktions-Register (SFR) des Mikrocontrollers einsehen. Über die oben im Fenster angebrachten Textfelder kann direkt auf die Inhalte zugegriffen bzw. diese verändert werden. Außerdem können die Bits im Feld durch doppeltes Anklicken in den jeweils entgegen gesetzten Zustand versetzt werden. Die Bezeichnungen (Namen) der einzelnen Register werden aus dem Prozessorfile entnommen. Es werden nur die Register angezeigt, die bei dem gewählten Mikrocontroller vorhanden sind.



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

Das Zusatzfenster Debugger: RAM

Anzeige der Inhalte des integrierten RAM's des Mikrocontrollers. Über die oben im Fenster angebrachten Textfelder kann direkt auf die Inhalte zugegriffen bzw. diese verändert werden. Außerdem können die Bits im Feld durch doppeltes Anklicken in den jeweils entgegen gesetzten Zustand versetzt werden. Die Bezeichnungen (Namen) der einzelnen Register werden aus dem EQU/BYTE Anweisungen entnommen (falls vorhanden). Wenn Sie also z.B. mit der Anweisung "Pulszeit EQU 20h" im MC-Editor Fenster eine Variable/Konstante deklariert haben, wird dieser Name im RAM an der Adresse 20h angezeigt. Zudem werden die Registerbezeichnungen R0-R7 an den Adressen der aktuellen Registerbank angezeigt.



Beachten Sie bitte, dass nicht jeder MCS-51 Mikrocontroller 256 Byte integriertes RAM besitzt, etliche haben nur 128 Byte! Da dieses aber nicht durch die Prozessorfiles definiert wird, zeigt das Debugger RAM Fenster immer die ganzen 256 Byte an.

Das Zusatzfenster Debugger: EXRAM

Dieses Fenster zeigt die Inhalte eines externen RAM Speichers der an einem Mikrocontroller angeschlossen werden kann. Über die oben im Fenster angebrachten Textfelder kann direkt auf die Inhalte zugegriffen werden.