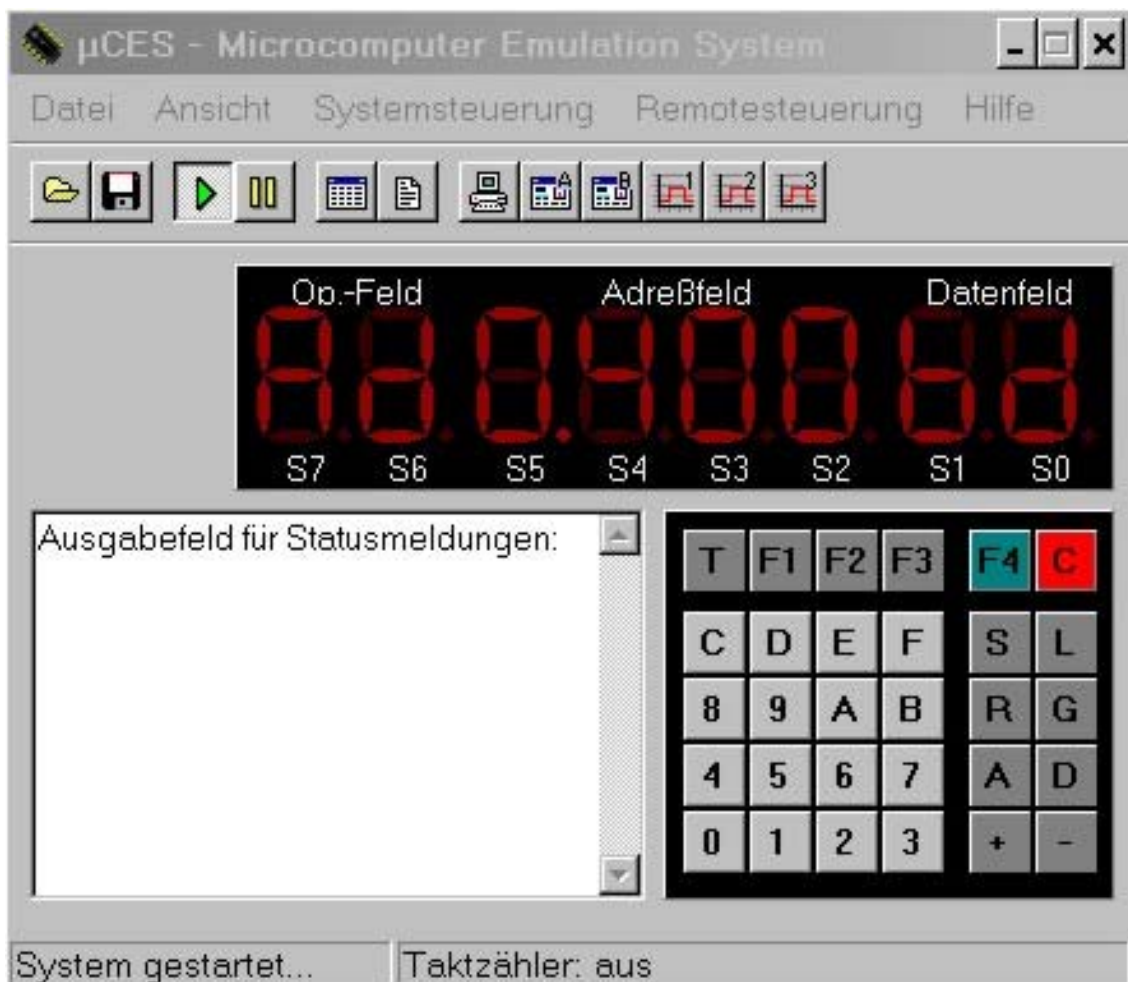


# Skript zum Mikrorechner-Praktikum

## Software-Handbuch

Autoren: Holger Kűfner und Helmut Bűhring





## Der 6809-Cross-Assembler "AS9"

Der mitgelieferte Assembler war in der Grundversion der freiverfügbare Cross-Assembler "AS9" der Firma Motorola. Der Assembler wurde für das Praktikum um einige Optionen erweitert und verändert. Er ist auf einem IBM kompatiblen PC mit DOS 3.1 oder höher lauffähig und erzeugt als Cross-Assembler Maschinencode für den Motorola 6809-Prozessor.

Der Assembler ist ein *two pass assembler*, d.h. der Übersetzungsvorgang wird in zwei Durchläufen ausgeführt. Im ersten Durchlauf wird das Quellprogramm gelesen und eine Symboltabelle aufgebaut. Im zweiten Durchlauf wird das Maschinenprogramm mit Hilfe der Symboltabelle erzeugt. Zusätzlich kann auf Anforderung (Unterabschnitte 2 und 5) eine Datei mit dem Programmlisting und der Symboltabelle generiert werden. Die Symboltabelle des "AS9" kann maximal 2000 Symbole mit acht oder weniger Zeichen verwalten. Werden mehr als acht Zeichen pro Symbol verwendet (bis zu maximal 16 Zeichen), so schrumpft entsprechend die Anzahl der maximal verwaltbaren Symbole. Ein Linker wird nicht benutzt. Das generierte Maschinenprogramm ist sofort ablauffähig.

Jede Befehlszeile des Quellprogramms wird vom Assembler vollständig bearbeitet, bevor der nächste Befehl des Quelltextes eingelesen wird. Bei jeder Befehlszeile überprüft der Assembler die Marke, den mnemonischen Operationscode und das Operandenfeld. Der mnemonische Operationscode wird mit der Zuordnungstabelle verglichen und bei Übereinstimmung der entsprechende Maschinencode in das Maschinenprogramm eingesetzt. Bei Nichtübereinstimmung wird eine Fehlermeldung generiert. Wird eine Assembler-Direktive (z.B. ORG) gefunden, wird die geforderte Aktion durch den Assembler durchgeführt. Jeden Syntaxfehler, der bei der Überprüfung entdeckt wird, zeigt der Assembler durch eine eingeschobene Zeile vor der fehlerhaften Befehlszeile im Programmlisting an. Wird kein Programmlisting generiert, wird eine Fehlermeldung am Bildschirm ausgegeben, um auf den Fehler und die nicht erfolgreiche Assemblierung hinzuweisen.

### 1. Syntax eines Quellcodeprogramms

Assemblerprogramme bestehen aus einer Folge von Befehlszeilen. Jede Befehlszeile besteht aus einer Sequenz von ASCII-Zeichen, die mit einem Zeilenende-Zeichen (*line feed*) abgeschlossen ist. Gültige Zeichen sind beim "AS9" eine Teilmenge des ASCII-Zeichensatzes und zwar

- |                                 |   |
|---------------------------------|---|
| - die Buchstaben                | A..Z, a..z,   |
| - die Ziffern                   | 0..9,   |
| - die arithmetischen Operatoren | +, -, *, /, % (Divisionsrest),  |
| - die logischen Operatoren      | &,  , ^ (exklusiv-oder) und   |
| - die Sonderzeichen             | [, ], \, _, \$, #, @, . (Punkt), : (Doppelpunkt),<br>; (Strichpunkt), ' (Hochkomma), , (Komma),<br>Leerzeichen oder Tabulator ( <i>white space</i> ). |

Als gültige Zeichen für Symbole (Namen) dürfen nur Buchstaben, Ziffern und die Sonderzeichen \$, . (Punkt) und \_ (Unterstrich) verwendet werden. Hierbei dürfen Symbole maximal 16 Zeichen lang sein und das erste Zeichen muß ein Buchstabe, ein Unterstrich oder ein Punkt sein. Alle Zeichen eines Symbols werden berücksichtigt, wobei zwischen Groß- und Kleinschreibung unterschieden wird.

Jede Befehlszeile eines Quellcodeprogramms für den Motorola "AS9" besteht aus den vier Feldern:

MARKE	OPERATOR	OPERAND	KOMMENTAR
-------	----------	---------	-----------

Innerhalb einer Befehlszeile müssen die Felder "MARKE", "OPERATOR" und "OPERAND" durch mindestens ein Leerzeichen voneinander getrennt sein. Das Feld "KOMMENTAR" wird durch einen Strichpunkt vom Operanden-Feld getrennt. Ist die Befehlszeile länger als es der verwendete Texteditor zuläßt, so kann die Befehlszeile durch das Zeichen "\" in die nächste Zeile verlängert werden und zwar bis zu maximal 256 Zeichen.

### Marken-Feld

Ein Stern (\*) oder ein Strichpunkt als erstes Zeichen in der ersten Spalte des Marken-Feldes definiert die ganze Befehlszeile als Kommentar. Eine so gekennzeichnete Befehlszeile wird vom Assembler vollständig ignoriert. Ein Leerzeichen (oder Tabulator) als erstes Zeichen zeigt dem Assembler an, daß das Marken-Feld leer ist, d.h. die Befehlszeile ist kein Kommentar und hat keine Marke.

Ist das Zeichen in der ersten Spalte der Zeile ein gültiges Symbolzeichen, so kennzeichnet dies die Definition einer Marke (Symbol). Eine Marke ist eine Folge von Zeichen, der durch die Definition ein Wert zugewiesen wird. Die Definition einer Marke muß eindeutig sein, d.h. die Marke darf in einem weiteren Markenfeld einer anderen Befehlszeile nicht mehr enthalten sein (in einem Operandenfeld kann die Marke mehrfach stehen). Mehrfachdefinitionen einer Marke werden durch den Assembler erkannt und als Fehler ausgegeben. Mit der Ausnahme von einigen Assembler-Direktiven (z.B. EQU) wird einer Marke das erste Byte der Speicheradresse der gerade assemblierten Befehlszeile zugeordnet. Die Definition von Marken kann optional mit einem Doppelpunkt abgeschlossen werden. Zum Beispiel sind die folgenden beiden Codefragmente identisch:

Start:	STA		Start	STA	
	BNE	Start		BNE	Start

Eine Marke kann auch alleine in einer Befehlszeile stehen und markiert dann die Speicheradresse des in der folgenden Zeile stehenden Maschinenbefehls.

### Operator-Feld

Das Operator-Feld muß einen gültigen mnemonischen Operationscode oder eine Assembler-Direktive enthalten. Mnemonics entsprechen direkt den zugehörigen Maschinenbefehlen und enthalten z.T. die notwendigen Registernamen, z.B. STA, CLRB. Hierbei verdeutlichen "ST" und "CLR" den Typ des Befehls und "A" und "B" sind die Register, die der Befehl benutzen soll (implizite Adressierung). Assembler-Direktiven sind spezielle Befehle, die den Übersetzungsvorgang des Assemblers steuern. Groß-

buchstaben im Operator-Feld werden vom Assembler zu Kleinbuchstaben konvertiert und danach auf ein gültiges Befehlswort überprüft. Das bedeutet, die Mnemonics "LDU", "LdU", "ldu" werden vom Assembler als ein und dasselbe Mnemonic interpretiert.

### Operanden-Feld

Die Bedeutung des Ausdruckes im Operanden-Feld ist vom Inhalt des Operator-Feldes abhängig. Das Feld kann ein Symbol, einen Ausdruck oder ein Symbol und einen Ausdruck enthalten. Hierbei kann ein Ausdruck ein Operand, eine Speicheradresse, ein Symbol, eine Marke oder ein Register sein. Durch die Anwendung von Kommata und eckigen Klammern wird die Adressierungsart, mit der der Befehl arbeiten soll, gekennzeichnet. In Tabelle 1 werden die einzelnen Operandenformate den zugehörigen Adressierungsarten des 6809 zugeordnet.

Tabelle 1: Operandenformat

Operandenformat *)	Adressierungsart
kein Ausdruck	implizit
Ausdruck	direkt, erweitert oder relativ
#Ausdruck	unmittelbar
<Ausdruck	kennzeichnet 8-bit-Offset
>Ausdruck	kennzeichnet 16-bit-Offset
[Ausdruck]	indirekt
Ausdruck,R	indiziert
<Ausdruck,R	indiziert, erzwingt 8-bit-Offset
>Ausdruck,R	indiziert, erzwingt 16-bit-Offset
[Ausdruck,R]	indirekt-indiziert
<[Ausdruck,R]	indirekt-indiziert, erzwingt 8-bit-Offset
>[Ausdruck,R]	indirekt-indiziert, erzwingt 16-bit-Offset
,Q+	Autoinkrement um 1
,Q++	Autoinkrement um 2
[,Q++]	Autoinkrement um 2, indirekt
,-Q	Autodekrement um 1
,--Q	Autodekrement um 2
[,--Q]	Autodekrement um 2, indirekt
W <sub>1</sub> , [W <sub>2</sub> , ..., W <sub>n</sub> ]	unmittelbar

\*) R ist eines der Register PC, S, U, X oder Y. Q ist eines der Register S, U, X oder Y und W<sub>i</sub> (i=1 bis n) ist eines der Symbole A, B, CC, D, DP, PC, S, U, X oder Y.

Ein Ausdruck ist eine Kombination von Symbolen, Konstanten, algebraischen und logischen Operatoren. Mit ihm kann der Wert eines Operanden oder die vom Befehl zu benutzende Adresse spezifiziert werden.

Steht zu Beginn des Operanden das Zeichen "<", so erzeugt der Assembler Maschinencode für einen 8-bit-Offset bei der Adressierung. Bei dem Zeichen ">" wird Code für eine Adressierung mit 16-bit-Offset generiert.

### **Operatoren**

Die Notation der Operatoren entspricht den Operatoren in der Programmiersprache C:

- + Addition
- Subtraktion
- \* Multiplikation
- / Division
- % Divisionsrest
- & bitweise UND-Verknüpfung
- | bitweise ODER-Verknüpfung
- ^ bitweise Exklusiv-Oder-Verknüpfung

### **Ausdrücke**

Ausdrücke werden von links nach rechts berechnet. Klammern in Ausdrücken werden als syntaktische Fehler gewertet. Arithmetische Operationen werden im Zweierkomplement mit Integer-Genauigkeit (16 bit bei IBM-kompatiblen PCs) berechnet. Besteht ein Ausdruck nur aus dem Zeichen "\*", so repräsentiert der Ausdruck den aktuellen Wert des Programmzählers, z.B. bedeutet der Befehl "NEWPC EQU \*", daß dem Symbol "NEWPC" der aktuelle Wert des Programmzählers zugeordnet wird.

### **Symbole**

Jedes Symbol repräsentiert einen 16-bit-Integerwert. Symbole können innerhalb eines Ausdrucks als Repräsentant ihres Integerwertes benutzt werden. Der Assembler ersetzt bei der Berechnung des Ausdrucks das Symbol durch den zugehörigen Integerwert.

### **Konstanten**

Konstanten repräsentieren Zahlenwerte, die während der Programmausführung nicht verändert werden können. Sie werden als 16-bit-Integerwert gespeichert. Sie können durch das Voranstellen eines Sonderzeichens in fünf verschiedenen Formaten notiert werden:

- \$ Hex
- % Binär
- @ Oktal
- ' ASCII-Zeichen
- Dezimalzahl (kein Sonderzeichen)

Wird kein Sonderzeichen verwendet, so wird die Konstante als Dezimalzahl interpretiert. Der Assembler konvertiert jedes vorgegebene Format in binären Maschinencode und stellt die Werte im Programmlisting als Hex-Zahlen dar. Im folgenden einige für den Assembler gültige und ungültige Zahlenformate:

Tabelle 2: Gültige und ungültige Zahlenformate

	Gültiges Format	Ungültiges Format	Begründung
Dezimal	12	123456	Bereichsüberschreitung
	12345	12.3	ungültiges Zeichen
Hexadezimal	\$12	ABCD	kein "\$"-Zeichen
	\$ABCD	\$G2A	ungültiges Zeichen
	\$001F	\$2F018	zu viele Ziffern
Binärformat	%00101	1010101	kein "%" -Zeichen
	%1	%10001100010101011	zu viele Ziffern
	%10100	%210101	ungültiges Zeichen
Oktalformat	@17634	@2317234	zu viele Ziffern
	@377	@277272	Bereichsüberschreitung
	@177600	@23914	ungültiges Zeichen
ASCII-Zeichen	!*	'XXY	zu viele Zeichen

Im Falle von mehr als einem ASCII-Zeichen verarbeitet der Assembler das erste Zeichen und ignoriert die restlichen Zeichen. Bei Bereichsüberschreitung oder zu vielen Ziffern werden die überzähligen Ziffern ebenfalls ignoriert. Eine Fehlermeldung wird nicht generiert.

### Kommentar-Feld

Das letzte Feld in der Befehlszeile eines "AS9"-Quellprogramms ist das Kommentarfeld. Es ist optional und wird ohne Veränderungen in das Programmlisting übernommen. Im Kommentarfeld kann jedes druckbare ASCII-Zeichen stehen. Ein Kommentar wird durch einen Strichpunkt eingeleitet. Steht der Kommentar allein in der Befehlszeile, so kann der Kommentar auch durch einen Stern (\*) in der ersten Spalte der Zeile gekennzeichnet werden.

## 2. Erstellen und Assemblieren eines "AS9"-Quellprogramms

Für die Erstellung eines Assembler-Quellprogramms können Sie jeden Texteditor verwenden, der es erlaubt, Ihr Programm als ASCII-Datei abzuspeichern. Hierbei werden nur der reine Text, Leerzeichen und Zeilenendezeichen in die Datei übernommen. Weitere Steuerzeichen, wie z.B. zur Formatierung des Textes oder zur Druckersteuerung werden in diesem Textformat nicht abgespeichert. Wird das Assembler-Quellprogramm mit Steuer-/Formatierungszeichen des Editors abgespeichert, so führen diese bei der Assemblierung zu Fehlern.

Das Quellprogramm ist in der Syntax des "AS9"-Assemblers zu erstellen. Hierbei ist insbesondere auf die Benutzung von Leerzeichen zur Trennung der einzelnen Felder zu achten, bzw. daß dem Kommentarfeld einen Strichpunkt voranzustellen ist. Tabelle 3 enthält ein für den "AS9" syntaktisch korrektes Programm.

Der Assembler wird von der DOS-Kommandozeile aus mit dem Aufruf

AS9    Datei1 (Datei2 . . .) ( - Option1 Option2 . . . )

gestartet. Die Inhalte der Klammern geben hierbei Optionen an, die bei Bedarf eingegeben werden können. Datei1, Datei2 usw. sind die Namen der Quellprogramme, die assembliert werden sollen. Durch die Verwendung mehrerer Quelldateien kann eine "Modularisierung" des Assemblerprogramms erreicht werden. Hierbei hat der Programmierer allerdings für eine korrekte Speicherabbildung der einzelnen Teilprogramme zu sorgen. Bei Mehrfachbelegungen von Speicheradressen durch Assemblerbefehle erzeugt der Assembler Fehlermeldungen.

Nach einem Minuszeichen (durch Leerzeichen vom letzten Dateinamen getrennt und vor der ersten Option) können optional verschiedene Steuerbefehle für die Generierung zusätzlicher Ausgaben angegeben werden. Es stehen folgende zusätzliche Ausgabemöglichkeiten zur Verfügung:

L	Ausgabe des Programmlistings
NOL	kein Programmlisting (Voreinstellung)
CRE	Ausgabe der Cross-Reference-Tabelle
S	Ausgabe der Symboltabelle
C	Ausgabe der Anzahl der Zyklen
NOC	keine Ausgabe der Zyklenanzahl (Voreinstellung)
HEX	Ausgabedatei im Intel Hex-Format (Voreinstellung)
MOT	Ausgabedatei im Motorola S-Format

Die Optionen zur Ausgabesteuerung des Assemblers (nicht HEX und MOT) können auch direkt im Quellprogramm stehen und werden durch die Verwendung der Assembler-Direktive "OPT" gekennzeichnet (Unterabschnitt 4). Diese überschreiben die eventuell auf Kommandozeilenebene eingegebenen Optionen.

Die Dateinamen müssen in der Kommandozeile vollständig, d.h. Dateiname mit Endung, angegeben werden. Die Endung ist (fast) frei wählbar und wird vom Assembler nicht überprüft. Es ist allerdings gute Programmierpraxis, wenn Sie die Endung Ihrer Quelldateien mit ".ASM" (für Assembler) benennen<sup>1</sup>. Der Assembler benennt die erzeugte Datei (Maschinenprogramm) nach dem Dateinamen der ersten angegebenen Quelldatei mit der Endung ".HEX". Wird z.B. das Quellprogramm "FIRSTPRG.ASM" assembliert, so erzeugt der Assembler als Ergebnis die Datei "FIRSTPRG.HEX". Die Endung ".HEX" deutet daraufhin, daß das generierte Maschinenprogramm im Intel HEX-Dateiformat abgespeichert wurde. Wird die Datei im Motorola S-Format mit der Option "-MOT" generiert, so erhält sie die Endung ".S19". Die Endungen ".HEX" bzw. ".S19" sind somit für die erzeugten Dateien mit den 6809-Maschinenprogrammen reserviert und sollten nicht anderweitig verwendet werden. Hat z.B. die Quelldatei die Endung ".S19", so wird diese Datei durch das erzeugte Maschinenprogramm überschrieben!

Werden Optionen in der Kommandozeile oder im Quellprogramm mit angegeben, so werden die zusätzlich erzeugten Listings auf die Standardausgabe gelenkt. Dies ist

---

<sup>1</sup> Für das Praktikum verwenden Sie bitte nur die Endung ".ASM", da alle Tools des Praktikums diese Endung erwarten.



unter DOS in der Regel der Bildschirm. Wollen Sie diese Listings in einer Datei speichern, so erweitern Sie die Kommandozeile um folgendes Konstrukt:

AS9 Datei1 (Datei2 . . .) (- Option1 Option2 . . .) > Liste

Alle zusätzlichen Informationen (auch Fehler), die ohne das Konstrukt über den Bildschirm ausgegeben werden, werden durch "> Liste" in die Datei "Liste" umgelenkt. Der Dateiname "Liste" ist dabei frei wählbar. Es hat sich jedoch bewährt, als Dateinamen für eine Liste den Dateinamen des Quellprogramms mit der Endung ".LST" zu verwenden. Die Listenformate der möglichen Optionen werden in Unterabschnitt 5 besprochen.

Tabelle 3: Das Programm "P1.ASM"

	ORG	\$0400	;Beginn des Programmbereiches
	JSR	CLRDISP	;Anzeige loeschen
	LDY	#\$0600	;Datenzeiger laden
	CLR	0,Y	;Datenbereich mit Null initialisieren
	CLR	1,Y	;
START:	LDX	#\$0000	;X loeschen
	JSR	HALTKEY	;Zeichen von Tastatur lesen
	CMPB	#\$86	;Vergleich, auf Ende der Eingabe (Taste "S")
	BEQ	ENDE	;
	CMPB	#\$09	;Test der Eingabe auf gueltige Ziffer (0..9)
	BHI	START	;bei ungueltiger Ziffer zurueck
	ADDB	0,Y	;Addition der Eingabe mit LSB der Summe
	TFR	B,A	;
	ANDA	#\$F0	;Test auf Ueberlauf durch vorherige Addition
	BNE	BCDADD	;wenn ja, Korrektur
	TFR	B,A	;
	CMPA	#\$09	;Test auf gueltige BCD-Ziffer
	BLS	CARRY2	;wenn nein, Korrektur
BCDADD:	ADDB	#\$06	;Korrektur, als Ergebnis gueltige BCD-Ziffer
CARRY2:	TFR	B,A	;
	ANDA	#\$F0	;
	BEQ	SHOW	;Test auf Ueberlauf vorh. Addition/Korrektur
	LDA	#\$01	;
	ADDA	1,Y	;naechsthoehoerwertige Ziffer um Eins erhoehen
	STA	1,Y	;naechsthoehoerwertige Ziffer abspeichern
SHOW:	ANDB	#\$0F	;eventl. Uebertrag der korrig. Ziffer loeschen
	STB	0,Y	;korrigierte Ziffer abspeichern
	JSR	SHOWT7SG	;korrigierte Ziffer in Anzeige
	LDB	1,Y	;naechsthoehoerwertige Ziffer laden
	LEAX	1,X	;Anzeigestelle nach links verschieben, X:=X+1
	JSR	SHOWT7SG	;naechsthoehoerwertige Ziffer in Anzeige
	BRA	START	;zurueck zur naechsten Eingabe
ENDE:	SWI1		
CLRDISP	EQU	\$F110	;Loeschen der Anzeige, In:-, Out:-
SHOWT7SG	EQU	\$F11C	;unteres Nibble von B in Anzeige, Position in X
HALTKEY	EQU	\$F143	;Lesen der Tastatur mit Warten, In:-, Out:B

#### Hinweis:

Alle hier benutzten oder zu entwickelnden Programme finden Sie auf der CD-ROM im Unterverzeichnis „Software“.

### Praktische Übung P2.4-1:

Assemblieren Sie das Programm "P1.ASM" durch Aufruf des Assemblers mit der Kommandozeile "AS9 P1.ASM". Rufen Sie dazu im Start-Menue unter „Alle Programme“ in der Programmgruppe „Zubehör“ die Eingabeaufforderung auf.

Assemblieren Sie das Programm "P1.ASM" unter der Verwendung verschiedener Optionen, z.B. mit der Kommandozeile "AS9 P1.ASM - L > P1.LST" und betrachten Sie die erzeugte Datei "P1.LST" mit Hilfe eines Texteditors. (Sollten Sie keinen Texteditor zur Verfügung haben, lesen Sie Abschnitt 4). Welche Aufgabe erfüllt das Programm? Wie reagiert das Programm, wenn der Wert "99" überschritten wird? Entwerfen Sie ein Flußdiagramm!

### 3. Fehlermeldungen

Ist das assemblierte Quellprogramm syntaktisch nicht korrekt, so wird vom "AS9" vor der syntaktisch falschen Befehlszeile eine Fehlermeldung in das Listing eingefügt. Es werden zwei Arten von Fehlermeldungen ausgegeben:

Befehlszeile: Beschreibung des Fehlers

Befehlszeile: *Warning* --- Beschreibung des Fehlers.

Fehler im ersten Durchlauf des Assemblers führen bei "normalen" Fehlern zu keinem Abbruch der Assemblierung, d.h. der zweite Durchlauf wird durchgeführt und bei Angabe der entsprechenden Optionen ein Listing erzeugt. Dies ist eine Erweiterung des Motorola Assemblers, der bei Fehlern im ersten *pass* die Assemblierung abgebrochen hat. Um ein Erkennen dieser Fehler in der Programmumgebung zu ermöglichen, wurde diese Erweiterung eingebaut. Allerdings können nachfolgende vom Assembler erkannte Fehler, Folgefehler des ersten Fehlers sein. Diese sollten nur unter starkem Vorbehalt (oder gar nicht) betrachtet werden, um Rückschlüsse auf den ersten Fehler zu ziehen. Nach Korrektur des ersten Fehlers ist eine nochmalige Assemblierung des Programms durchzuführen und bei Auftauchen weiterer Fehler sind diese, Fehler für Fehler, zu korrigieren. Bedingt durch die fortgesetzte Assemblierung des ersten *pass* im Fehlerfall wird auch eine Datei mit Maschinencode erzeugt. Diese ist natürlich nicht korrekt und sollte auch nicht zu Testzwecken benutzt werden. Erst im fehlerfreien Fall entsteht ein korrektes Maschinenprogramm.

Warnungen (Fehlertyp 2) führen ebenfalls zu keinem Abbruch der Assemblierung, weisen aber auf ein mögliches Zuordnungsproblem zwischen Symbolen und Adressen hin. Diese Warnungen sollten ernst genommen werden. Einige Fehler werden vom Assembler als "*fatal error*" klassifiziert und führen zu einem sofortigen Abbruch der Assemblierung. Bei diesen Fehlern kann der Assembler im allgemeinen keine temporären Dateien erzeugen und im Verzeichnis des Assemblers speichern (z.B. kein freier Speicher), oder diese Dateien wurden während der Assemblierung beschädigt.

Werden mehrere Dateien gleichzeitig assembliert, so erweitert sich die Fehlermeldung zu:

Dateiname, Befehlszeile: Beschreibung des Fehlers.

Am Ende des Listings wird die Gesamtanzahl der gefundenen Fehler ausgegeben.

#### Praktische Übung P2.4-2:

Assemblieren Sie das Programm "P2.ASM" durch Aufruf des Assemblers mit der Kommandozeile "AS9 P2.ASM -L > P2.LST".

Korrigieren Sie die gefundenen Fehler und assemblieren Sie neu. Welcher logischer Fehler ist weiterhin in dem Programm, und wie verändert er die Funktion des Programms?

#### 4. Assembler-Direktiven

Assembler-Direktiven sind Befehle, mit deren Hilfe der Assemblierungsvorgang und die Art der Assemblierung gesteuert werden kann. Die Direktiven des "AS9" sind im Gegensatz zu anderen Assemblern rudimentär, erfüllen jedoch die für das Praktikum gestellten Anforderungen. Für Ihre ersten Assemblerprogramme reicht es aus, wenn Ihnen die Direktiven "ORG" und "EQU" geläufig sind. Alle anderen Direktiven werden in der Regel erst bei größeren Programmen benutzt. Die Direktiven sind im folgenden alphabetisch aufgeführt. Sind Teile des Befehls in runde Klammern gesetzt, so sind diese optional. Der Begriff "Ausdruck" kann ein Zeichen, eine numerische Konstante, ein Symbol oder einen durch einen arithmetischen Ausdruck berechenbaren Wert repräsentieren. Ist es im folgenden nicht anders erwähnt, wird bei Direktiven der benutzten Marke der aktuelle Inhalt des Programmzählers zugeordnet.

##### BSZ

(Marke) BSZ Ausdruck (Kommentar)

Mit der Direktive BSZ (*Block Storage of Zeros*) kann eine bestimmte Anzahl von Bytes allokiert werden. Jedes Byte des Blockes wird mit Null initialisiert. Die Anzahl der zu allozierenden Bytes wird durch "Ausdruck" bestimmt. Sollte "Ausdruck" ein undefiniertes Symbol oder ein Symbol enthalten, dem erst später im Programm ein Wert zugewiesen wird (*forward reference*), wird ein Fehler (*Phasing Error*) generiert.

##### EQU

Marke EQU Ausdruck (Kommentar)

Mittels der Direktive EQU (*EQUate Symbol to a value*) wird einer Marke der Wert von "Ausdruck" zugewiesen. Dieser Wert ist *nicht* notwendigerweise der aktuelle Wert des Programmzählers. Der Marke kann innerhalb des Programms kein neuer Wert zugewiesen werden. Sollte "Ausdruck" ein undefiniertes Symbol oder ein Symbol enthalten, dem erst später ein Wert zugewiesen wird (*forward reference*), wird ein Fehler generiert.

##### FCB

(Marke) FCB Ausdruck (,Ausdruck, ..., Ausdruck) (Kommentar)

Der Direktive FCB (*Form Constant Byte*) können ein oder mehrere durch Kommata getrennte Ausdrücke folgen. Der Wert eines jeden Ausdrucks wird auf acht Bit begrenzt und wird in je einem Byte im Speicher ab dem aktuellen Wert des Programmzählers abgelegt. Bei mehreren Ausdrücken werden die einzelnen Bytes hintereinander abgespeichert.

## **FCC**

(Marke) FCC Ausdruck Begrenzer String Begrenzer (Kommentar)

Mit der Direktive FCC (*Form Constant Character string*) kann ein String (eine Folge von Zeichen) im Speicher abgelegt werden. Das erste Zeichen wird unter der Speicheradresse abgespeichert, auf die der aktuelle Inhalt des Programmzählers zeigt. Die folgenden Zeichen werden unter den nachfolgenden Speicheradressen abgelegt. Der (optionalen) Marke wird die Speicheradresse des ersten Zeichens zugewiesen. Der String kann jedes druckbare Zeichen enthalten und wird in der Befehlszeile durch zwei identische Begrenzer, die ebenfalls jedes druckbare Zeichen sein können, gekennzeichnet. Das erste Zeichen nach der Direktive FCC, das kein Leerzeichen ist, wird als Begrenzer interpretiert. Zum Beispiel wird mit der Befehlszeile

```
MESSAGE1 FCC ,Input correct,
```

der String "Input correct" der Marke "MESSAGE1" zugewiesen.

## **FDB**

(Marke) FDB Ausdruck (,Ausdruck, ...,Ausdruck) (Kommentar)

Der Direktive FDB (*Form Double Byte constant*) können ein oder mehrere durch Kommata getrennte Ausdrücke folgen. Der Wert eines jeden 16-bit-Ausdruckes wird in zwei nacheinander folgenden Bytes gespeichert. Bei mehreren Ausdrücken werden die 16-bit in je zwei hintereinander folgenden Bytes abgespeichert (Big Endian Format: High Byte unter der niedrigeren Speicheradresse, Low Byte unter der höheren).

## **FILL**

(Marke) FILL Ausdruck, Ausdruck

Mittels der Direktive FILL (*FILL memory*) kann der Assembler veranlaßt werden, Code für die Initialisierung eines Speicherbereiches mit einem konstanten Wert zu generieren. Der erste Ausdruck repräsentiert die Konstante (0 – 255), mit der der Speicherbereich aufgefüllt werden soll. Der zweite Ausdruck gibt die Anzahl der zu initialisierenden Bytes innerhalb des Speichers an.

## **OPT**

OPT option (,option, ..., option) (Kommentar)

Die Direktive OPT (*assembler output OPTions*) wird benutzt, um neben dem zu erzeugenden Maschinenprogramm zusätzliche Ausgabe-Listings zu generieren. Die Optionen sind mit den Optionen, die auf der Kommandozeilenebene eingegeben werden können, identisch. Allerdings überschreiben die Optionen des Quellprogramms die Optionen der Kommandozeilenebene. Folgende Optionen stehen zur Verfügung:

- L        Ausgabe des Programmlistings ab der aktuellen Befehlszeile
- NOL     Keine Ausgabe des Programmlistings (Voreinstellung). Kann mit der Option "L" dazu verwendet werden, nur Teile des Programmlistings auszugeben.
- CRE     Gibt die für das assemblierte Programm gültige Cross-Reference-Tabelle am Ende des Programmlistings aus. Wird diese Option im Quelltext verwendet, so muß die Option vor dem ersten Symbol des Programmtextes stehen.
- S        Gibt am Ende des Programm-Listings die Symboltabelle aus.

- C        Gibt für jeden Befehl die benötigten Prozessortakte des Befehls an. Die Taktanzahl erscheint im Programmlisting nach dem Maschinencode und vor dem Quellcode.
- NOC     Schaltet die Option "C" aus (Voreinstellung).

Die Formate der einzelnen Listings werden in Unterabschnitt 5 besprochen.

## **ORG**

ORG    Ausdruck        (Kommentar)

Die Direktive ORG (*set program counter to ORiGin*) setzt den Programmzähler auf den Wert, der von "Ausdruck" repräsentiert wird. Die folgenden Befehlszeilen werden vom Assembler an die entsprechend nachfolgenden Speicheradressen verschoben und assembliert. Ist in einem Quellprogramm keine "ORG"-Direktive gesetzt, so wird der Programmzähler mit Null initialisiert, und das Maschinenprogramm beginnt entsprechend dem Programmzähler an der Speicheradresse \$0000. Undefinierte Ausdrücke werden vom Assembler als Fehler erkannt.

## **PAGE**

PAGE                    (Kommentar)

Mit der Direktive PAGE (*top of PAGE*) kann der Assembler veranlaßt werden, das zu erzeugende Programmlisting ab der Direktive "PAGE" auf einer neuen Seite beginnen zu lassen. Wird kein Programmlisting erzeugt, so hat PAGE keine Auswirkungen.

## **RMB**

(Marke) RMB    Ausdruck        (Kommentar)

Die Direktive RMB (*Reserve Memory Bytes*) reserviert einen Speicherbereich in der Größe der Anzahl der Bytes, die "Ausdruck" repräsentiert. Der Speicherbereich wird im Gegensatz zu der Direktive "FILL" nicht mit einem konstanten Ausdruck vorinitialisiert. Diese Direktive wird in der Regel für die Reservierung von Speicher für den späteren Gebrauch von Tabellen benutzt. Wird in der Befehlszeile eine "Marke" verwendet, so erhält die "Marke" die Adresse des ersten Byte des reservierten Speicherbereiches.

## **ZMB**

(Marke) ZMB    Ausdruck        (Kommentar)

Die Direktive ZMB (*Zero Memory Bytes*) entspricht der Direktive BSZ und wird aus Kompatibilitätsgründen zu anderen Assemblern mitgeführt.

## **5. Datei-Formate**

Mit den Assembler-Optionen "-L", "-S", "-C" und "-CRE" kann der Assembler veranlaßt werden, neben dem eigentlichen Maschinencode zusätzliche Informationen in ein Listing auszugeben.

Im folgenden werden die einzelnen Ausgabeformate besprochen.

## Programmlisting

Das Assembler-Programmlisting hat das folgende Zeilenformat:

Zeilennummer	Speicheradresse	Maschinencode	([#Zyklen])	Quellcode
--------------	-----------------	---------------	-------------	-----------

Die Zeilennummer ist eine vierstellige fortlaufende Nummer, die als Verweis in der Cross-Reference-Tabelle verwendet wird. Die Speicheradresse ist eine vierstellige Hexadezimalzahl, die die Adresse des ersten Bytes des jeweiligen Maschinenbefehles im Speicher des Rechners angibt. Der Maschinencode ist der vom Assembler erzeugte Code für den 6809-Prozessor, entsprechend dem im Quellcode stehenden Assemblerbefehl. Der Maschinencode wird hexadezimal angegeben und kann mehrere Bytes betragen. Ist im Quellcode oder auf Kommandozeilenebene die Option "-C" gewählt worden, so erscheint im Programmlisting nach dem Maschinencode die Anzahl der benötigten Taktzyklen des Prozessors für den Maschinenbefehl in eckigen Klammern. Danach wird das Quellcodeprogramm mit Marken-, Operator-, Operanden- und Kommentarfeld ausgegeben.

## Symboltabelle

Die Symboltabelle hat das folgende Zeilenformat:

Symbol	Symbolwert
--------	------------

Alle Symbole, die im Markenfeld des Quellprogramms definiert wurden, werden in dieser Tabelle aufgelistet. Im ersten Feld steht der Name des jeweiligen Symbols, im zweiten Feld der Wert des Symbols in hexadezimaler Form. Der Wert des Symbols ist bei einer Marke die Speicheradresse, für die die Marke definiert wurde oder bei einer Konstantendefinition der Wert der Konstanten.

## Cross-Reference-Tabelle

Die Cross-Reference-Tabelle hat das folgende Zeilenformat:

Symbol	Symbolwert	* Zeilennummer der Def.	Zeilennummer ...
--------	------------	-------------------------	------------------

Die Cross-Reference-Tabelle ist eine Erweiterung der Symboltabelle. In den ersten beiden Spalten stimmt sie mit der Symboltabelle überein. Im dritten Feld wird nach einem Stern (\*) die Zeilennummer des Quellprogramms aufgelistet, in dem das Symbol definiert wurde. Danach werden alle Zeilennummern aufgelistet, in denen das Symbol (Marke) verwendet wird.

## Dateiformate

Wie bereits erwähnt, speichert der Assembler "AS9" das erzeugte Maschinenprogramm im Intel HEX-Format oder im Motorola S-Dateiformat ab. Das generierte Programm (im Intel HEX-Format) kann direkt in den Praktikumsrechner geladen werden (Abschnitt 2.4.4). Im folgenden werden die beiden Datenformate für Interessierte kurz vorgestellt.

## Motorola S-Dateiformat

Dieses Format wurde von Motorola entwickelt, um Programm- und Objektdateien in einer druckbaren Form darstellen zu können. Hierdurch können die Dateien mit jedem Texteditor betrachtet und ohne Probleme zwischen verschiedenen Rechnern via Modemkabel ausgetauscht werden. Das S-Format beinhaltet neben den Daten und eini-

gen Steuerbytes ein Prüfsummen-Feld, mit dem die Integrität der Datei überprüft werden kann.

Eine S-Datei kann aus mehreren S-Datensätzen gleichen oder verschiedenen Typs zusammengesetzt sein. Ein S-Datensatz besteht aus mehreren Feldern fester und variabler Länge, die den Typ des S-Datensatzes, die Anzahl der Datenbytes, die Speicheradresse, die Daten und das Prüfsummenfeld beinhalten. Um die binären Daten als druckbare ASCII-Zeichen darstellen zu können, werden sie codiert im S-Datensatz dargestellt. Jedes Byte mit "binärem Inhalt" wird in zwei Bytes, welche das "binäre Byte" im ASCII-Code darstellen, aufgeteilt. Das erste ASCII-Byte repräsentiert die oberen vier Bits der zu codierenden acht Bits, das zweite ASCII-Byte die unteren vier Bits. Ein S-Datensatz besteht aus fünf Feldern mit folgendem Format:

Tabelle 4: Motorola S-Format

Feldtyp	Bytes	Inhalt
Datensatztyp	2	Typ des S-Datensatzes: S1, S9, ...
Satzlänge	2	Anzahl der Zeichenpaare (hexadezimal) im Datensatz - ohne die Felder "Datensatztyp" und "Satzlänge"
Speicheradresse	4,6 oder 8	Speicheradresse, ab der die nachfolgenden Daten abgespeichert werden sollen, beginnend mit dem ersten Zeichenpaar des Datenfeldes
Daten	0-2n	0 - n codierte Bytes; ausführbarer Maschinencode, Daten oder Informationen
Checksumme	2	Die 8-bit-Summe der Werte, die durch die Zeichenpaare der Felder "Satzlänge", "Speicheradresse" und "Daten" repräsentiert werden; im Einerkomplement

Jeder S-Datensatz kann mit einem der Steuerzeichen CR (*carriage return* - Wagenrücklauf), LF (*line feed* - Zeilenende) oder NUL (*null character* - Nullzeichen) abgeschlossen werden.

Motorola definierte für verschiedene Anwendungen acht S-Datensatzformate, von denen der "Assembler AS9" zwei Typen benutzt:

- S1: Der Datensatz beinhaltet Maschinencode oder Daten. Die Speicheradresse ist 4 Zeichen lang.
- S9: Der Datensatz "S9" wird als letzter Datensatz zur Terminierung einer Übertragung an das Ende einer S-Datei angehängt. Das Feld "Datensatztyp" hat den Wert "S9", das Feld "Satzlänge" den Wert "03", das "Speicheradressfeld" den Wert "0000", das "Datenfeld" ist leer, und das "Prüfsummenfeld" hat den Wert "FC".

Aus der Verwendung der beiden Datensatztypen "S1" und "S9" leitet sich die Endung "S19" der generierten "AS9" Maschinencodateien ab.

### Intel Hex-Dateiformat

Das Intel Hex-Format unterscheidet sich vom Motorola S-Format im wesentlichen durch die Anordnung der einzelnen Datenfelder eines Datensatzes, durch die Firmenkennung sowie durch das Prüfsummenfeld. Wie eine Motorola S-Datei besteht eine Intel Hex-Datei ebenfalls aus mehreren Datensätzen. Die Codierung des Datenfeldes

entspricht dem Motorola-Verfahren, indem ein Byte in zwei ASCII-Zeichen aufgesplittet wird. Tabelle 5 zeigt das Format der Datensätze.

Tabelle 5: Intel Hex-Format

Feldtyp	Anzahl Zeichen	Inhalt
Kennung	1	Intel-Kennung: ":"
Satzlänge	2	Anzahl der Zeichenpaare (hexadezimal) im Datenfeld (ohne alle anderen Felder)
Speicheradresse	4	Speicheradresse, ab welcher der nachfolgende Inhalt des Datenfeldes abgespeichert werden soll, beginnend mit dem ersten Zeichenpaar des Datenfeldes
Datensatztyp	2	Typ des Intel-Datensatzes: 00, 01, ...
Daten	0-2n	0 - n codierte Bytes ausführbarer Maschinen-code, Daten oder Informationen
Checksumme	2	Die 8-bit-Summe der Werte, die durch die Zeichenpaare der Felder "Satzlänge", "Speicheradresse", "Datensatztyp" und "Daten" repräsentiert werden; im Zweierkomplement

Der Datensatztyp "00" entspricht dem Motorola Typ "S1", der Intel Typ "01" entspricht dem Motorola Typ "S9". Demnach besteht der Intel Typ "01" aus der Zeichenfolge ":00000001FF". Das Feld "Speicheradresse" enthält wie bei Motorola den Wert "0000".



## Der 6809-Disassembler "DS9"

Der im Praktikum verwendete 6809-Disassembler "DS9" "rück-übersetzt" Maschinenprogramme, die im Intel-Hex-Format gespeichert sind, in Assemblerprogramme, mit der Ihnen vom "AS9" bekannten Syntax.

Auf DOS-Ebene wird der Disassembler durch das Kommando "DS9" gestartet. Der Bildschirm wird gelöscht und es erscheint die Meldung "6809 Disassembler Vers. #". In der folgenden Bildschirmzeile wird nach der zu disassemblierenden Datei gefragt. Nach Eingabe, eventuell mit Pfadangabe und optional ohne Endung ".HEX", wird die Datei eingelesen und disassembliert. Das Programm zeigt dies durch die Meldungen "Reading Input file:", "Processing File:" und "OK" an. Wird die einzulesende Datei nicht gefunden, so bricht das Programm ohne Fehlermeldung ab. Als Ergebnis der Disassemblierung wird eine Datei erzeugt, die das eingelesene Programm in Maschinencode und in Assemblersprache auflistet. Als Dateiname wird der Name der eingelesenen Datei mit der Endung ".LST" verwendet. Die Eingabe eines anderen Dateinamens als Ausgabedatei ist nicht möglich.

Der "DS9" unterstützt auch den Kommandozeilenmodus. Beispielsweise wird durch die Eingabe "DS9 BCDADD" auf DOS-Ebene die Datei "BCDADD.HEX" eingelesen und disassembliert. Als Ausgabedatei wird die Datei "BCDADD.LST" erzeugt.

### Praktische Übung P2.4-3:

Disassemblieren Sie das Programm "P1.HEX" durch Aufruf des Disassemblers mit der Kommandozeile "DS9 P1". Kopieren Sie bitte vorher das bereits vorhandene Assemblerlisting "P1.LST" nach "P1old.LST" und vergleichen Sie nach der Disassemblierung die beiden Listings. Welche Unterschiede bestehen zwischen den beiden Listings?

Ein Disassembler kann in der Regel nicht zwischen Programmcode und Daten unterscheiden<sup>2</sup>. Von komfortablen Disassemblern werden einige Hilfestellungen zur Erkennung von Unterprogrammrouinen und der automatischen Generierung von Marken (*Labels*) gegeben. Dennoch ist immer ein "Nach-Disassemblieren" per Hand notwendig, um Marken und Unterprogrammen aussagekräftige Namen zu geben, Schnittstellen zwischen Unterprogrammrouinen zu erkennen und um festzustellen, ob es sich bei dem vorliegenden Speicherbereich um Programmcode, einen Stack oder einfach Daten handelt.

Disassemblierung in Verbindung mit einem Debugger ist auch heute noch in der Zeit der objektorientierten und logischen Programmiersprachen, der Client/Server-Architekturen und der graphischen Benutzeroberflächen, unter Umständen das letzte Hilfsmittel, einen Fehler in einem Programm zu finden. Programmcode, der auf Hochsprachenebene in mehreren Durchläufen debuggt wurde, - und für richtig befunden wurde -, kann durch optimierende Compiler oder auch durch schlichtweg falsch übersetzende Compiler und Interpreter zu unverständlichen Fehlern führen. In diesen Fällen hilft nur ein Debuggen und Disassemblieren auf Maschinensprachebene.

---

<sup>2</sup> Außer Programmcode und Daten werden in verschiedenen Speichersegmenten gehalten.

## Die 6809-Entwicklungsumgebung "IDE9"

### 1. Allgemeine Informationen

Die Entwicklungsumgebung "IDE9" (*Integrated Development Environment*) ist auf einem IBM-kompatiblen Rechner unter DOS ablauffähig. Eine Bedienung mit der Maus ist möglich. Die Entwicklungsumgebung besitzt eine Dateiverwaltung, einen vollständigen Texteditor, Menüpunkte zum Assemblieren und Disassemblieren von erstellten oder geladenen 6809-Programmen sowie verschiedene Möglichkeiten zur Datenübertragung zwischen dem PC und einem „realen“ Praktikumsrechner. Diese Übertragungen werden bei der ausschließlichen Benutzung des Simulators/Emulators nicht benötigt.

Nach Aufruf des Programms auf DOS-Kommandozeilenebene mit "IDE9" erscheint ein Fenster, in dem man den seriellen Port des PCs für die Übertragung zwischen Praktikumsrechner und PC wählen kann (Voreinstellung "COM2", falls vorhanden). Nach Betätigen der Return-Taste schließt das Fenster und Sie befinden sich im Hauptfenster der Entwicklungsumgebung.

In der obersten Zeile des Fensters befindet sich die Menüleiste, mit der Sie die verschiedenen Untermenüs zur Versionskennung ( $\equiv$ ), zur Dateiverwaltung (*File*), zum Texteditor (*Edit*), zur Datenübertragung (*Transmit*), zum Assemblieren und Disassemblieren von 6809-Programmen (*Projekt*) oder zur Fensterverwaltung (*Window*) erreichen können. Die einzelnen Untermenüs erreichen Sie entweder mit einem Mausklick, oder über die F10-Taste mit nachfolgender Auswahl durch die Cursortasten, oder über die Tastenkombination "<ALT> + rot hervorgehobener Buchstabe".

In der untersten Zeile des Hauptfensters befindet sich eine Statuszeile, in der die wichtigsten HotKeys aufgelistet sind. Ganz rechts in der Zeile wird der noch zur freien Verfügung stehende Arbeitsspeicher angezeigt.

Für die einzelnen Untermenüs erscheint jeweils ein weiteres Fenster, in dem die zu dem Unterpunkt passenden Funktionen ausgewählt werden können. Bei einigen Unterpunkten öffnen sich Dialogboxen, um zwischen verschiedenen Einstellungen auszuwählen oder auch Parameter einzugeben. In einer Dialogbox kann mittels der TAB-Taste durch die verschiedenen Funktionsgruppen durchgeschaltet werden (z.B. Dateimaske, Dateiauswahl, Buttons, usw.). Ein Dialog wird in einer Box mit der Return-Taste abgeschlossen oder mit der ESC-Taste abgebrochen. Einige Funktionen können in den Boxen auch direkt durch die HotKeys "<ALT> + hervorgehobener Buchstabe" angesprochen werden.

Die Dateiverwaltung erlaubt neben dem Laden und Abspeichern von Dateien auch das Speichern einer Datei unter einem anderen Namen sowie einen Verzeichnis- oder Laufwerkswechsel. Der Texteditor entspricht in der Handhabung dem üblichen Standard und besitzt neben Undo-, Cut-, Copy-, Paste- und Delete-Funktionen auch die Möglichkeit, eine Textstelle zu suchen bzw. zu suchen und zu ersetzen.

Im folgenden wird die Menüstruktur der Entwicklungsumgebung aufgelistet, anschließend ein Assemblierungsvorgang erläutert und danach die Datenübertragungsmöglichkeiten zwischen PC und Praktikumsrechner besprochen.

## 2. Menüstruktur

Da die Menüstruktur im wesentlichen selbsterklärend ist, werden die einzelnen Menüunterpunkte nur kurz beschrieben. Sollte Ihnen ein Programmpunkt unklar sein, so probieren Sie die Funktion einfach aus.

### FILE

- New erzeugt eine neue zu editierende Textdatei "Untitled".
- Open (F3) öffnet eine existierende Textdatei.
- Save (F2) speichert die aktive Datei.
- Save as speichert die aktive Datei unter einem anderen Namen.
- Format öffnet eine Dialogbox zur Festlegung des Dateiformates.
- Change dir wechselt in ein anderes Verzeichnis oder Laufwerk.
- Exit (Alt-X) beendet das Programm.

### EDIT

(Textbereiche werden mit der Shift-Cursortaste oder der Maus markiert.)

- Undo (Alt-U) macht die letzte Aktion im Editor rückgängig.
- Cut (Shift-Entf) schneidet den markierten Bereich aus und legt ihn im Clipboard ab.
- Copy (Strg-Einf) kopiert den markierten Bereich ins Clipboard.
- Paste (Shift-Einf) fügt den Inhalt des Clipboards ein.
- Show Clipboard zeigt den Inhalt des Clipboards an.
- Delete (Strg-Entf) löscht den markierten Bereich.
- Find öffnet eine Dialogbox für die Textsuche.
- Replace öffnet eine Dialogbox für den Textaustausch.
- F&R next (Strg-L) wiederholt die letzte Find- oder Replace-Aktion.

### TRANSMIT

- Send (Alt-S) sendet die aktive Datei, falls das eingestellte Dateiformat mit der Dateiendung übereinstimmt. Öffnet sonst eine Dialogbox zur Dateiauswahl.
- Receive (Alt-R) empfängt eine Datei im eingestellten Dateiformat.
- Port öffnet eine Dialogbox zur Auswahl des Kommunikationsports.
- Baud öffnet eine Dialogbox zur Einstellung der Baudrate.

### PROJECT

- Assemble (F9) assembliert die aktive Datei, falls die Dateiendung "ASM" lautet. Öffnet sonst ein Dateiauswahlfenster. Treten bei der Assemblierung Fehler auf, oder ist einer der Assembler-Parameter gesetzt, wird ein Listing erzeugt und angezeigt.
- Parameter öffnet eine Dialogbox zur Einstellung der Assemblerparameter.
- Disassemble (Shift F9) ruft den Disassembler auf, falls die Dateiendung der aktiven Datei ".HEX" lautet. Öffnet sonst ein Dateiauswahlfenster zur Auswahl der zu disassemblierenden Datei.

## WINDOW

- List listet alle geöffneten Fenster der Entwicklungsumgebung auf.
- Size/Move (Strg-F5) bietet die Möglichkeit, Fensterposition und -größe per Tastatur zu verändern. Die Cursortasten verändern die Position und die <Shift-Taste> + Cursortasten verändern die Größe des Fensters.
- Zoom (F5) zoomt ein Fenster auf Maximalgröße und wieder zurück.
- Tile ordnet alle Fenster nicht überlappend an.
- Cascade ordnet alle Fenster überlappend an.
- Next (F6) schaltet auf das nächste Fenster.
- Previous (Shift-F6) schaltet auf das vorherige Fenster.
- Close (Alt-F3) schließt das aktive Fenster.

### 3. Assemblierung und Disassemblierung

Editieren Sie in Ihrem aktiven Fenster eine Datei mit der Endung "ASM", so können Sie diese sofort assemblieren, in dem Sie entweder die Funktionstaste F9 betätigen oder im Untermenü "*PROJECT*" die Funktion "*ASSEMBLE*" anwählen. Sollten Sie keine Datei mit der Endung "ASM" bearbeiten, öffnet sich bei anwählen der Funktion "*ASSEMBLE*" (oder der Taste F9) ein Dateiauswahlfenster. Nach dem Laden der ausgewählten Datei (mit Endung "ASM") wird diese durch den Assembler "AS9" in ein Maschinenprogramm übersetzt. Das Maschinenprogramm erhält das Dateiformat, das im Unterpunkt "*FORMAT*" vereinbart wurde (Voreinstellung "HEX").

Treten bei der Übersetzung Fehler auf, oder sind Optionen für den Assembler im Unterpunkt "*PARAMETER*" gesetzt worden, so wird ein Listing mit dem Dateinamen der assemblierten Datei mit der Endung "LST" erzeugt und nach der Übersetzung in einem neuen Fenster angezeigt. Vor dem Assemblieren werden alle Dateien, die den gleichen Dateinamen wie die zu übersetzende Datei und eine der Endungen "LST", "HEX" oder "S19" besitzen auf Festplatte zurückgespeichert und die entsprechenden Fenster geschlossen.

Durch die Tastenkombination <Shift-F9> oder durch die Funktion "*DISASSEMBLE*" im Untermenü "*PROJECT*" kann eine Datei mit der Endung "HEX" disassembliert werden. Hat die gerade aktive Datei die Endung "HEX", so wird diese disassembliert und das Ergebnis in einem Fenster angezeigt. Ansonsten öffnet sich ein Dateiauswahlfenster, durch das die gewünschte Datei selektiert werden kann.

Alle Fenster, die den gleichen Namen wie die zu disassemblierende Datei, aber die Endung "LST" haben, werden geschlossen. Der Disassembler erzeugt eine Datei mit dem Namen der zu disassemblierenden Datei und der Endung "LST", die in einem neuen Textfenster angezeigt wird. Achtung: eine "alte" Datei mit dem Dateinamen der aktiven Datei und der Endung "LST", die zum Beispiel vorher durch den Assembler erzeugt wurde, wird überschrieben. Sollte diese Datei nach einer Disassemblierung noch zur Verfügung stehen, können Sie der "alten" Datei durch die Funktion "*SAVE AS*" im Untermenü "*FILE*" einen neuen Dateinamen geben.

#### **Praktische Übung P2.4-4:**

Assemblieren Sie das Programm "P1.ASM" durch Aufruf des Assemblers in der Entwicklungsumgebung "IDE9" mit verschiedenen Assembleroptionen. Lesen Sie das erzeugte Listing P1.LST und laden Sie das erzeugte Maschinenprogramm P1.HEX in ein Textfenster. Wählen Sie im Untermenü "FORMAT" das S19-Dateiformat und assemblieren Sie erneut. Laden Sie die Datei P1.S19 in ein Textfenster und vergleichen Sie das S19-Dateiformat mit dem Intel HEX-Format.

### **Der 6809-Simulator<sup>3</sup> "EM9"**

Der im Praktikum für den 6809-Prozessor entwickelte Simulator wurde unter der graphischen Benutzeroberfläche "GEM" programmiert. Der Simulator bildet das Verhalten jeder Komponente des Praktikumsrechners auf dem PC nach. Damit alle Funktionen des Simulators verfügbar sind, ist ein freier Hauptspeicher von ca. 540 kByte unterhalb der 1-MB-Speichergrenze notwendig. Der Simulator ist auch im DOS-Fenster von Windows ablauffähig. Sie starten den Simulator, indem Sie in Ihrem jeweiligen Unterverzeichnis auf Kommandozeilenebene "EM9" eingeben.

Der 6809-Simulator erlaubt das Laden und Speichern von Programmen auf Festplatte. Eine Datenübertragung zum bzw. vom Praktikumsrechner über die serielle Schnittstelle des PCs ist ebenfalls möglich. Die Tastatur und die Anzeige des Praktikumsrechners werden in einem Fenster "wie gewohnt" dargestellt. Der Simulator benutzt für die Nachbildung des Verhaltens des Praktikumsrechners exakt die gleichen Monitorroutinen, die auch im EPROM Ihres Praktikumsrechners vorhanden sind. Der Speicherbereich des Praktikumsrechners kann in einem Editor-Fenster angezeigt und editiert werden. Mittels eines integrierten Disassemblers kann der aktuelle Speicherinhalt als Assemblerprogramm dargestellt werden. Die Registerwerte des Prozessors können in einem gesonderten Fenster angezeigt werden. Weitere Fenster simulieren das Verhalten des Timers, der ACIA und der parallelen Schnittstelle.

Die folgende Beschreibung erläutert die für die Anwendung des Simulator-Programms wesentlichen Bedienschritte. Weitergehende Anleitungen zu den einzelnen Funktionen finden Sie als "Online"-Hilfe im Simulator unter dem Menüpunkt "Anleitung" bzw. durch Anklicken von Fragezeichen (?) mit der linken und gleichzeitigem Drücken der rechten Maustaste. Beschreibungen zur Bedienung des Simulators für die Rechnerkomponenten Timer, parallele Schnittstelle und V.24-Schnittstelle finden Sie in den entsprechenden Kapitel 4, 5.

---

<sup>3</sup> synonym auch mit Emulator bezeichnet.

## 1. Allgemeine Informationen zur graphischen Oberfläche

Nach dem Aufruf des Simulators wird Ihnen zunächst Hilfe zur Bedienung des Programms angeboten. Falls Sie diese akzeptieren, erscheint das im Bild 1 dargestellte Fenster. In diesem Bild finden Sie die Bezeichnungen der einzelnen Komponenten, die zur Bedienung erforderlich sind. Die Bezeichnungen werden in englischer und deutscher Sprache vorgestellt, um Ihnen eine möglichst universell anwendbare Einführung in die Begrifflichkeit dieses Themas zu bieten.

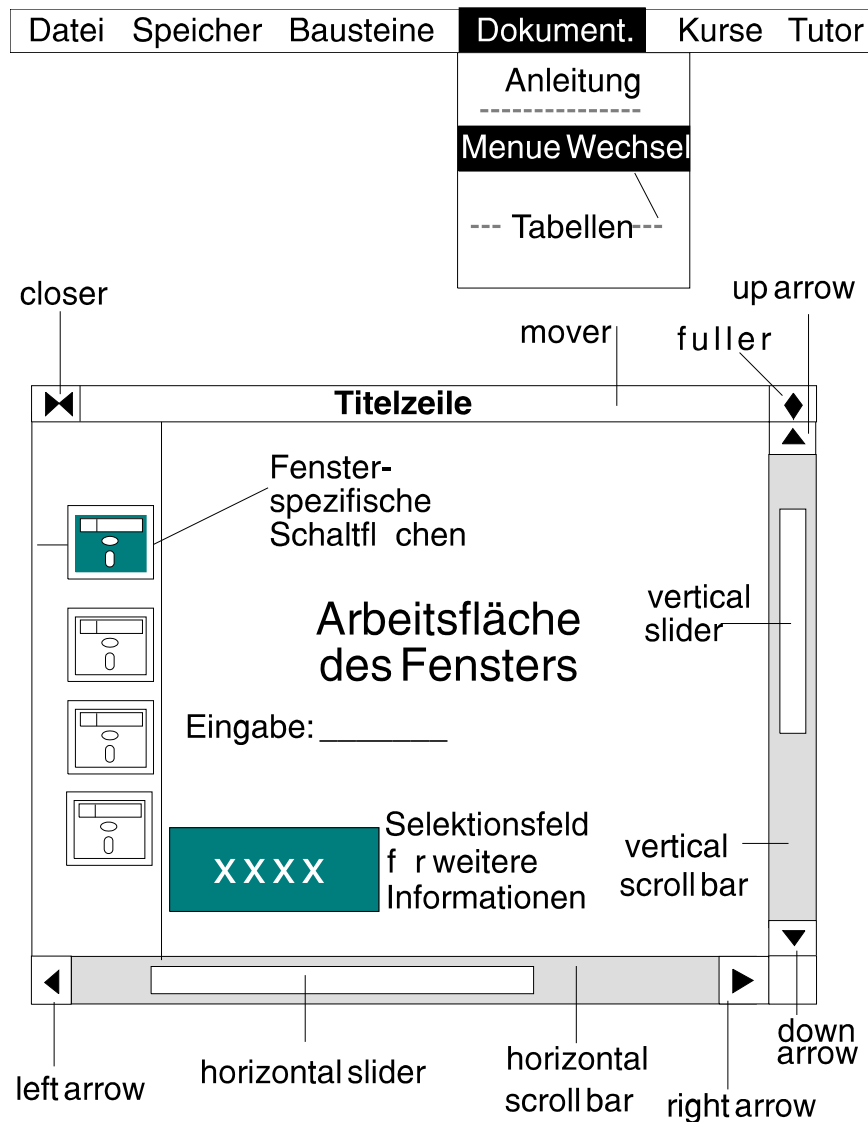


Bild 1: Aufbau eines Bildschirmfensters

Um einen im Fenster dargestellten Text komplett lesen zu können, ist es u.U. erforderlich, entweder den senkrechten Schieber (*vertical slider*) mit der linken Maustaste anzuklicken, festzuhalten und innerhalb des Verschiebebalkens (*vertical scrollbar*) nach unten zu verschieben, so daß ein nachfolgender Textausschnitt dargestellt wird, oder Sie klicken den nach unten gerichteten Pfeil (*down arrow*) mit der linken Maustaste an, um den Text nach unten zu rollen.

Es ist möglich, während der Benutzung des Simulators auf den im Bild 1 dargestellten Informationstext über den Menüpunkt **Datei**(Anleitung) **Programm**<sup>4</sup> zuzugreifen, indem Sie mit dem Mauszeiger auf den Menüeintrag "Datei" fahren und somit die zu diesem Eintrag gehörenden Unterpunkte anwählbar machen. Jetzt können Sie den Eintrag "(Anleitung) Programm" anwählen.

Verlassen können Sie diese Information durch Anklicken des Schließers (*closer*), der sich in der linken oberen Ecke eines Fensters befindet, mittels der linken Maustaste.

### **Gliederung dieses Unterabschnittes:**

- I. Benutzung der Objekte, die zur Bedienung der Fenster und das Öffnen und Schließen derselben zur Verfügung stehen.
- II. Fensterbedienung, Eingabe von Daten in aktiven Fenstern, Anwendung der Iconen
- III. Die zwei Menü-Ebenen des Simulators: Erläuterungen zum Haupt- und zum Dokumentationsmenü
- IV. Bedeutung der Statusanzeige: Die Statusanzeige ist die helle Zeile am unteren Bildrand, die während der Simulatorbenutzung Informationen zum aktuellen Prozessorstatus enthält.

## **I. Fenster-Rahmen-Objekte**

### **I.1 Schieber (*slider*)**

Es existieren sowohl senkrechte als auch waagrechte Schieber (*vertical / horizontal slider*). Diese ermöglichen in dem Fall, in dem die Bildinformation in der Waag- oder Senkrechten das Fensterformat übersteigt, den Überblick über die gesamte Information des angesprochenen Themas. Hierbei stellt der weiße Bereich innerhalb des weiß-grauen Balkens die relative Größe des Textes oder Bildausschnittes dar, der sich zur Zeit auf dem Bildschirm befindet. Wie oben bereits kurz erwähnt, ist dieser weiße Abschnitt innerhalb des Balkens durch Anklicken mit der linken Maustaste, Festhalten der Taste und gleichzeitiges Verschieben des Abschnittes nach oben oder unten zu bewegen. Mit dieser Aktion wird auch der Text- bzw. Bildinhalt in die gleiche Richtung verschoben.

Es ist auch möglich, an einer geeigneten Stelle des grauen Balkens zu klicken, um einen gewünschten Bereich in einem Text anzuwählen.

### **I.2 Pfeile (*arrows*)**

Durch Anklicken der Pfeile mit der linken Maustaste, die nach links, rechts, oben bzw. unten weisen (also *left-, right-, up-* oder *down-arrow*), setzt sich der Fensterinhalt in die angewählte Richtung in Bewegung. Dies ermöglicht gegenüber der Benutzung der Verschieber (*slider*) eine kontinuierliche Bewegung in dem Text oder Bild in kleinen Schritten.

### **I.3 Fenster-Füllraute (*fuller*)**

Als *fuller* bezeichnet man die Raute, die in der rechten oberen Ecke des Fensters zu erkennen ist. Durch Anklicken des *fullers* mit der linken Maustaste wird der Bildinhalt

---

<sup>4</sup> Menü- und Untermenüpunkte werden im weiteren in der Form ....\....\.... angegeben.

der linken Bildhälfte (hier: Graphik) vergrößert, so daß er den kompletten Bildschirm ausfüllt. Wenn Sie den alten Zustand wiederherstellen möchten, müssen Sie nur den *fuller* erneut anklicken.

#### **I.4 Schließ-Knopf (*closer*)**

In der linken oberen Ecke eines jeden Fensters befindet sich der sogenannte Schließer (*closer*). Durch Anklicken dieses Symbols mittels linker Maustaste wird, wie oben bereits erwähnt, das aktuelle Fenster geschlossen.

#### **I.5 Verschiebepfeil (*mover*)**

Das aktuelle Fenster läßt sich komplett verschieben, indem man die Titelleiste mit der linken Maustaste anklickt, die Taste gedrückt hält und nun das Fenster in die gewünschte Position zieht. Hat man diese erreicht, so kann man durch Freigeben der linken Maustaste das Fenster an dieser Position fixieren.

## **II. Benutzung der Ikonen, Fensterverwaltung und Dateneingabe**

### **II.1 Fensterbedienung**

Unter der benutzten Oberfläche ist es möglich, mehrere Fenster gleichzeitig geöffnet zu haben. Hat man z.B. die Anleitung zum Zeitgeber-/Zählerbaustein (*Timer*) und den Timer selber als Fenster nacheinander geöffnet, so trägt das jeweils aktive Fenster eine schwarze Kennung in der Titelleiste, und die inaktiven Fenster tragen eine hellgraue Kennung. Durch einfaches Anklicken eines inaktiven Fensters mit der linken Maustaste wird dieses aktiviert und, sofern es vorher von anderen Fenstern verdeckt war, in seiner ganzen Größe in den Vordergrund gerückt. Ist in dem geschilderten Fall z.B. die Anleitung zur Benutzung des Timers aktiv gewesen, so kommt durch einfaches Klicken mit der linken Maustaste auf die Fläche des Praktikumsrechners dieser wieder in den Vordergrund und ist zu verwenden, ohne daß ein anderes Fenster geschlossen werden müßte.

### **II.2 Dateneingabe in aktiven Fenstern**

In einigen Fenstern (z.B. "Bausteine \ ACIA einschalten" oder "Datei \ Speicherbereich laden") ist eine Dateneingabe über die Tastatur in das aktive Fenster möglich. Die aktuelle Position des Cursors wird über einen senkrechten Balken oder durch das Zeichen " | " angezeigt. In den Fenstern, die erscheinen, sobald die Menüpunkte "Speicherbereich laden" oder "... speichern" angewählt werden, kann man die Zeile, die man verändern, löschen oder ergänzen möchte, durch Anklicken mit der linken Maustaste aktivieren.

### **II.3 Handhabung der Ikonen**

Ihnen werden während der Benutzung des Emulationsprogramms verschiedene Arten von Ikonen begegnen:

- a.) Diskettensymbole als Schaltflächen zum Laden bzw. Löschen von Dateien auf Diskette oder Festplatte
- b.) In den Dokumentationen werden grüne Schaltflächen mit weißer Schrift dazu benutzt, die Bereiche darzustellen, die durch Anwählen mit der linken Maustaste



weitergehende Informationen zu dem dargestellten Thema liefern<sup>5</sup>.

c.) Die bei der Bedienung des Simulators auftretenden Fragezeichen sind wie folgt zu bedienen:

- Die großen Fragezeichen dienen zur Anwahl eines Hilfe-Status. Das Anklicken dieser Quadrate hat zur Folge, daß das Hilfesystem zu dem entsprechenden Thema entweder an- oder ausgeschaltet wird.
- Der aktive oder ruhende Hilfe-Status ist daran zu erkennen, daß in der betreffenden Darstellung kleine Fragezeichen unmittelbar neben den Objekten erscheinen, zu denen ein Hilfetext nach Anwahl des Fragezeichens erscheint. Ein kleines Fragezeichen wird angewählt, indem der Mauszeiger auf das Fragezeichen geführt, die rechte Maustaste gedrückt und gehalten und dann die linke Maustaste gedrückt wird.

Tiefergehende Informationen zu diesem Thema können Sie durch die Anwahl der Symbole des in Bild 1 dargestellten Fensters erhalten.

### **III. Die zwei Menü-Ebenen des Simulators**

Wie diese Überschrift schon andeutet, gibt es während der Benutzung des Emulators die Möglichkeit, zwischen zwei Menüebenen zu wechseln. Dies geschieht über folgende Menüeinträge:

- Befindet man sich in dem Menü, dessen Eintrag in der linken oberen Ecke "Datei" lautet (im folgenden als Hauptmenü bezeichnet), so kann man die zweite Menüzeile mit dem Eintrag "System" in der linken oberen Ecke (im folgenden als Dokumentationsmenü bezeichnet) über den Menüpunkt "Dokument. \ Menue-Wechsel" oder über die Tastenkombination Ctrl+M (bzw. Strg+M oder ^M) erreichen.
- Befindet man sich im Dokumentationsmenü<sup>6</sup>, so ist ein Sprung in das Hauptmenü über den Menüeintrag "System \ Menue-Wechsel" sowie ebenfalls mit der Tastenkombination Ctrl+M (bzw. Strg+M oder ^M) möglich.

Um die zwei Menüebenen kurz zu skizzieren, seien deren wesentliche Inhalte genannt:

- Das Hauptmenü beinhaltet vorwiegend Funktionen und Hilfsmittel, um den Simulator als solchen zu benutzen. Hierzu gehören u. a. die Darstellungsmöglichkeiten der Schnittstellen und des Timers, der Registerinhalte und Speicherbereiche sowie die Organisation der Datenein- und Datenausgabe.
- Das zweite Dokumentationsmenü liefert über seine verschiedenen Einträge wesentliche Informationen zum System, zur CPU, zu weiterer Hardware und zum Monitor.

Zu den meisten Menü-Oberpunkten existiert der Unterpunkt "Anleitung". Unter dem Oberpunkt "Datei" gibt es sogar drei Anleitungen: "Programm", also dieser Text, den Sie gerade lesen, "Emulator" und "Mikrorechner". Andere Anleitungen, in denen wesentliche Eigenschaften einzelner Teile des Simulators vorgestellt werden, sind als Menüunterpunkte zu den einzelnen Themen aufgeführt.

---

<sup>5</sup> In der Praktikumsversion nicht implementiert.

<sup>6</sup> In der Praktikumsversion nicht implementiert.

#### IV. Die Statusanzeige

Am unteren Bildrand der normalen Benutzeroberfläche befindet sich die sogenannte Statusanzeige. Sie enthält verschiedene Informationen zu wichtigen Vorgängen im Programm. So wird zum Beispiel beim Ablauf eines Programms, welches nicht definierte Befehle enthält, die Ursache für die unter diesen Umständen auftretenden Programmstörungen als "*illegal instruction*" in der Statuszeile gemeldet.

In der Statusanzeige wird auch zur Anzeige gebracht, wenn man von der parallelen Schnittstelle (Ports PA bzw. PB) oder der seriellen Schnittstelle (ACIA) Daten einliest oder abspeichert. Dieser Vorgang wird durch ein Diskettensymbol verdeutlicht, wobei bei Datenabspeicherung PA, PB und/oder ACIA links von dem Pfeil steht bzw. stehen, der auf das Diskettensymbol zeigt. Im Falle des Einlesens einer Datei steht der jeweilige Empfänger rechts von dem Diskettensymbol. Ein Pfeil, der von dem Diskettensymbol auf den Empfänger weist, verdeutlicht schon optisch in eindeutiger Weise die Datenübertragungsrichtung.

#### 2. Allgemeine Informationen zur Benutzung des Simulators

Die Bedienung des Simulators soll mit Hilfe der Erläuterungen zu den einzelnen Menüpunkten nähergebracht werden. Um hierbei eine möglichst gute Übersicht zu gewährleisten, werden die jeweiligen Funktionen der einzelnen Menüpunkte in der Folge ihres Auftretens erläutert. Zuerst erfolgt eine Auflistung der Menüpunkte mit einer kurzen Funktionsbeschreibung. Im Anschluß daran wird eine genauere Beschreibung der einzelnen Punkte, soweit nötig, gegeben.

Datei	Hauptmenü-Eintrag für grundsätzliche Systemfunktionen
Anleitung Programm Emulator Mikrorechner	Unter diesen Menüpunkten finden Sie allg. Informationen zu: - der Bedienung der graphischen Oberfläche - der Bedienung des Simulators - der Bedienung des simulierten Mikrorechners
Speicherbereich laden  speichern	Diese Menüpunkte erlauben: - ein auf der Festplatte gespeichertes Programm oder einen Datenbereich (über eine Dateiauswahlbox) einzulagern - ein Programm oder einen Datenbereich (über eine Dateiauswahlbox) auf der Festplatte abzuspeichern
Format	Auswahl zwischen zwei Speicherformaten: *.HEX, *.PTI Endung wirkt als Dateifilter für die Auswahlboxen
Übertragung  → Mikrorechner ← Mikrorechner	Übertragung von Daten zwischen Simulator und Praktikumsrechner: - Datenübertragung vom Praktikumsrechner zum Simulator - Datenübertragung vom Simulator zum Praktikumsrechner
Definitionen Schnittstellen  ROM\RAM- Erweiterungen	Hier werden Systembedingungen festgelegt: - Zuweisung der simulierten seriellen und parallelen Schnittstellen zu den Hardware-Schnittstellen des PCs - Erweiterung des simulierten Arbeitsspeichers von 16 auf max. 32 kbyte
Ende	Beenden des Simulators (auch über die Tastenkombination "Strg+Q" möglich)

Sie können eine allgemeine Einführung in die Benutzung des Programms und in die wesentlichen Bedienschritte über den Menüpunkt **Datei \ (Anleitung) Programm** erhalten. Die in einigen Menüpunkten und auf der Benutzeroberfläche dargestellten Fragezeichen liefern nach Anklicken der linken und unmittelbar danach der rechten Maustaste Hilfetexte zum angewählten Thema.

<b>Speicher</b>	Hauptmenü-Eintrag zur Speicher- und Registerdarstellung
anzeigen	Eröffnet ein Fenster, in dem der Speicherinhalt in hexa-dezimaler Form dargestellt wird und verändert werden kann
Disassembler	Eröffnet ein Fenster, in dem der aktuelle Speicherinhalt disassembliert dargestellt wird
Register	Die Registerinhalte (nach einem Trace-Trap) werden in einem separaten Fenster dargestellt
Assembler	
Anleitung	Anleitung zur Benutzung des Assemblers (vgl. Abschnitt 2.2)
Aufruf	Aufruf des Assemblers, nach Verlassen des Simulators !

<b>Bausteine</b>	Hauptmenü-Eintrag zur Darstellung und Bedienung der Rechnerkomponenten
Timer 6840	Darstellung und Bedienung der 3 Timer des Zeitgeber-/Zählerbausteins (s. Kapitel 5)
Anleitung	Unter diesem Punkt kann eine Anleitung zur Benutzung der Timer aufgerufen werden
Timer #1	Es wird für jeden Timer ein eigenes Fenster geöffnet, in dem der momentane Zustand des Timers dargestellt wird
Timer #2	
Timer #3	
PIA 6821	Darstellung und Bedienung des Parallelschnittstellen-Bausteins MC6821 (s. Kapitel 4)
Anleitung	Unter diesem Punkt kann eine Anleitung zur Benutzung der zwei Ports des Bausteins MC6821 aufgerufen werden
Port PA	Für jeden Port wird ein Fenster geöffnet, in dem der Schaltzustand der Ein-/Ausgangsleitungen zu erkennen ist; deren Zustände können auch durch Betätigen der im Fenster dargestellten Schalter verändert werden
Port PB	
ACIA R6551	Darstellung und Bedienung des seriellen Schnittstellen-Bausteins R6551 (s. Kapitel 4)
Anleitung	Unter diesem Punkt kann eine Anleitung zur Benutzung des R6551 aufgerufen werden
ACIA einschalten	In einem separaten Fenster kann die aktuelle Konfiguration des R6551 und die serielle Datenübertragung beobachtet werden

<b>Dokumentation</b>	Hauptmenü-Eintrag zur Bedienung des Dokumentationssystems
Anleitung	Hier werden weitere Informationen zur Benutzung des Dokumentationssystems gegeben
Menue-Wechsel	Wechsel vom Hauptmenü zum Dokumentationsmenü <sup>7</sup>
Tabellen	Anzeige von Tabellen zur Programmierung des simulierten Prozessors (vgl. Kapitel 1):
CPU-Befehls-satz	Tabellen B-1 und B-2 aus Anhang von Kapitel 1
Befehlssatz alphanumerisch	Tabellen C-1 und C-2 aus Anhang von Kapitel 1
Branch-Befehle	Tabelle B-3 aus Anhang von Kapitel 1
relative Adressierung	Tabelle 1.3-5 aus Kapitel 1
Postbyte	Bild A-1 aus Anhang von Kapitel 1
Hilfsroutinen	Tabelle D-1 aus Anhang von Kapitel 1

<b>Kurse</b>	Hauptmenü-Eintrag zur Bearbeitung der im Simulator implementierten Kurse <sup>8</sup>
Anleitung	Anzeige von allgemeinen Informationen zur Benutzung der Kurse:
CPU-Hardware	- Kurs zur Prozessorhardware
CPU-Software	- Kurs zur Prozessorsoftware
Peripherie	- Kurs zur Bedienung der Peripheriebausteine

<b>Tutor</b>	Hauptmenü-Eintrag zur Bearbeitung des im Simulator implementierten Tutors <sup>9</sup>
Anleitung	Anzeige von allg. Informationen zur Benutzung des Tutors:
Hardware	- Tutor zur Rechnerhardware
Software	- Tutor zur Rechnersoftware

### 3. Bedienung der simulierten Mikrorechner-Tastatur

In der linken oberen Ecke des Rechner-Bildes ist ein großes Fragezeichen dargestellt. Klickt man dieses mit der linken Maustaste an, so werden in der Umgebung der Tasten kleine Fragezeichen sichtbar und in dem Display werden den jeweiligen Stellen der

<sup>7</sup> In der vorliegenden Praktikumsversion nicht implementiert.

<sup>8</sup> In der vorliegenden Praktikumsversion nicht implementiert.

<sup>9</sup> In der vorliegenden Praktikumsversion nicht implementiert.

Anzeige die zugehörigen Funktionen zugewiesen. Wünscht man nun eine Erklärung zu den einzelnen Tastenfunktionen, so erscheint nach Drücken der rechten und der linken Maustaste (zuerst die rechte Taste drücken, festhalten, dann Drücken der linken Maustaste) ein separates Hilfefenster mit Erläuterungen zu der Funktion der jeweiligen Taste.

Die Anwahl der Tasten zur Eingabe der Zahlenwerte sowie der mit festen Funktionen vorbelegten Tasten kann auf zwei Arten erfolgen:

- Die erste Möglichkeit ist die Anwahl der Tasten mit der Maus, wobei hierfür die linke Maustaste zu benutzen ist.
- Die zweite Möglichkeit besteht in der Anwahl der Tasten über die PC-Tastatur in folgender Weise:
  - Die Anwahl der Tasten für die Eingabe von hexadezimalen Zahlen sowie der "+" bzw. "-" Tasten und der Funktionstasten F1 bis F4 erfolgt durch direkte Benutzung der gleichnamigen Tasten der PC-Tastatur.
  - Die Tasten mit fest vorbelegten Funktionen, also die Tasten der Tastengruppen drei und vier (siehe hierzu den entsprechenden Hilfetext), sind über die Tastenkombination <Strg>+<Taste> bzw. <Ctrl>+<Taste> anzuwählen.

Hinweis:

In einigen Bedienfunktionen des Mikrorechners muß eine Taste über längere Zeit "gehalten" werden. Dies wird im Simulator durch Benutzung der Umschalt- (*Shift*-) Taste der PC-Tastatur ermöglicht. Drücken Sie hierbei die Umschalttaste und selektieren Sie mit der Maus die zu haltende Taste.

## Praktische Übungen zur Assemblerprogrammierung

Im diesem Abschnitt sollen die Unterprogrammtechniken an praktischen Beispielen vertieft und erweitert werden.

### "Taschenrechner"

Das Ihnen bekannte Assemblerprogramm "P1.ASM" ist nur für eine Zahlenbreite von zwei Ziffern einsetzbar. Wünschenswert wäre ein Programm, das abhängig von einer Konstanten eine veränderbare Ziffernbreite zuläßt. Hierfür muß der Kern des Programms "P1.ASM", die BCD-Addition, aus dem Hauptprogramm herausgenommen werden und in einer eigenen Unterprogrammroutine abgelegt werden. Tabelle 6 zeigt eine mögliche Implementierung dieses Unterprogramms.

Im Unterschied zum ersten Programm, soll nun bei einem Überlauf ein Symbol (kleines Rechteck) in der Anzeige erscheinen. Wird danach die Taste "S" gedrückt, so soll das Programm beendet werden. Im Falle einer gültigen BCD-Ziffer soll das Programm von vorne beginnen.

Tabelle 6: Unterprogrammroutine UP\_ADD zur BCD-Addition

```
;=====
; Unterprogramm UP_ADD: Addition von zwei BCD-Ziffern
; In:   Y: Speicherstelle der ersten BCD-Ziffer (liegt im unteren Nib)
;       B: zweite BCD-Ziffer (liegt im unteren Nibble)
; Out:  A: gleich Null: kein Uebertrag, sonst Uebertrag fuer naechsth.
;       B: korrigierte Ziffer, vorbereitet zur Anzeige
;
;       ORG    $0500    ;Beginn Unterprogramm

UP_ADD:  ADDB   0,Y      ;Addition der Ziffer
        TFR    B,A      ;
        ANDA   #$F0     ;Test auf Ueberlauf durch vorherige Addition
        BNE    BCDADD   ;wenn ja, Korrektur
        TFR    B,A      ;
        CMPA   #$09     ;Test auf gueltige BCD-Ziffer
        BLS    CARRY2   ;wenn ja, Korrektur ueberspringen
BCDADD:  ADDB   #$06     ;Korrektur, als Ergebnis gueltige BCD-Ziffer
CARRY2:  TFR    B,A      ;
        ANDA   #$F0     ;Vorbereitung zum Test auf Ueberlauf (Hauptp.)
        ANDB   #$0F     ;eventl. Uebertrag der korrig. Ziffer loeschen
        STB    0,Y      ;korrigierte Ziffer abspeichern
        RTS         ;Ruecksprung in das Hauptprogramm
```

(Das dargestellte Unterprogramm finden Sie auf der CD-ROM als Teil des Programms P1a.asm.)

**Praktische Übung P2.5-1:**

Analysieren Sie das in Tabelle 6 gezeigte Unterprogramm UP\_ADD und entwerfen Sie ein Flußdiagramm. Wie reagiert die Routine auf einen eventuellen Übertrag auf die nächsthöhere Ziffer?

**Praktische Übung P2.5-2:**

Entwerfen Sie ein Flußdiagramm für das zugehörige Hauptprogramm zu dem Unterprogramm UP\_ADD von Tabelle 6. Im Hauptprogramm soll die verarbeitete Ziffernbreite mittels einer Konstantendefinition "einstellbar" sein. Im Falle eines Überlaufes soll ein Symbol (kleines Rechteck) in der Anzeige erscheinen. Wird danach die Taste "S" gedrückt, so soll das Programm beendet werden. Im Falle der Eingabe einer gültigen BCD-Ziffer soll das Programm von vorne beginnen.

Implementieren Sie Ihr Flußdiagramm und testen Sie die Programme auf Ihrem Praktikumsrechner aus.

Werden für das Programm weitere arithmetische Operationen implementiert, so erhält man einen kleinen Taschenrechner für Dezimalzahlen, der mit variabler Ziffernbreite arbeiten kann.

**"Vierstufen-Zähler"**

Das folgende Beispiel verdeutlicht die verschiedenen Möglichkeiten zur Zwischenspeicherung von Registerinhalten in Unterprogrammen.

Es ist ein Programm zu implementieren, das einen vierstufigen Zähler repräsentiert. Die einzelnen Zählerstufen sollen über Unterprogramme realisiert werden. Der Einfachheit halber soll jede Zählstufe von \$00 bis \$FF zählen. Die Zähler sollen hierarchisch angeordnet sein, so daß jeder Zähler seinen darunterliegenden nächsten Zähler in jedem Zählschritt einmal aufruft. Dem höchstwertigen Zähler sind die Anzeigestellen S7,S6 zugeordnet, dem niederwertigsten Zähler die Anzeigestellen S1,S0.

Die folgende Tabelle 7 zeigt das Hauptprogramm des Zählers. Die Register X und B werden während der Abarbeitung des Unterprogramms über Speicherbereiche zwischengespeichert.

Tabelle 7: Hauptprogramm des Zählers

```
; P3.ASM
;
; Zaehlprogramm ueber drei Unterprogramme zur Darstellung der ver-
; schiedenen Moeglichkeiten zur Zwischenspeicherung sowie der ver-
; schachtelten Progr. von Unterprogrammen.
;   Zaehler des Hauptprogramms erscheint in den Anzeigestellen S7,S6
;   Register B und X werden ueber Speicherbereiche gerettet
;
```

ORG	\$0400	;Beginn des Programmbereiches	
	JSR	CLRDISP	;Anzeige loeschen
	LDX	#\$0006	;Vorbereitung der Anzeige (S7,S6)
	LDB	#\$FF	;Zaehler initialisieren
NEXT0:	INCB		;Zaehler inkrementieren
	JSR	SHOWB7SG	;Zaehlerstand anzeigen
	STB	MEM1	;Zaehler und Anzeigeposition ueber
	STX	MEM2	;Speicherbereich retten
	JSR	UP1	;Aufruf des ersten UP
	LDB	MEM1	;Zaehler und Anzeigeposition einlesen
	LDX	MEM2	;
	CMPB	#\$FF	;Abbruchbedingung erfuehlt?
	BNE	NEXT0	;wenn nein, zurueck zu NEXT0
	SWI		;Programmende
MEM1	EQU	\$0500	;Speicherbereiche zum Zwischenspeichern
MEM2	EQU	\$0502	;von Daten
CLRDISP	EQU	\$F110	;Anzeige des Praktikumsrechners loeschen
SHOWB7SG	EQU	\$F120	;Anzeige der Hexzahl, die in B steht; Pos. in X

### Praktische Übung P2.5-3:

Implementieren Sie für obiges Hauptprogramm die drei geforderten Unterprogramme. Im ersten Unterprogramm sollen die Register X und B über den Stack gerettet werden. In dem zweiten Unterprogramm sollen sie über Prozessorregister zwischengespeichert werden.

Testen Sie das Programm auf Ihrem Praktikumsrechner aus.

### "Interrupt-Routinen"

Interrupt-Routinen sind eine spezielle Art von Unterprogrammen, die ausschließlich durch Hardware- und Software-Interrupts aktiviert werden. An dieser Stelle soll die grundsätzliche Struktur von Interruptroutinen besprochen werden.

Interruptbehandlung durch den Monitor:

Wird ein Interrupt vom 6809-Prozessor erkannt, so rettet er abhängig vom Interrupttyp einige (mindestens PC und CC)<sup>10</sup> Register oder alle Prozessorregister auf den Systemstack und ruft die Interruptbehandlungsroutine des Monitors auf. Diese ermittelt die Interruptquelle und setzt den Programmzähler auf die Einsprungadresse der jeweiligen Behandlungsroutine der Interruptquelle. Die gültige Einsprungadresse lädt die Monitorroutine aus der sogenannten Interruptvektortabelle, in der für alle Interruptquellen eines Systems die entsprechenden Einsprungadressen hinterlegt sind. Beispielsweise ist im Falle des Praktikumsrechners die Interrupt-Vektoradresse für den SWI3 (Software-Interrupt3) die 16bit-Adresse \$004C, \$004D.

<sup>10</sup> Vgl. Kapitel 1



## Struktur einer Interrupt-Routine

Eine Interruptroutine ist ein in sich abgeschlossenes Programm, daß mindestens einen RTI-Befehl (*return from interrupt*) besitzen muß. Das Hauptprogramm integriert die Interruptroutine in das Gesamtprogramm, in dem es die Einsprungadresse der Interruptroutine an der entsprechenden Adresse in der Interruptvektortabelle hinterlegt. Tritt im nachfolgenden Programmfluß ein Interrupt auf, so wird durch die Monitorroutinen automatisch zu der zugehörigen Interruptbehandlungsroutine verzweigt. Am Ende des Kontrollflusses der Interruptroutine wird durch den RTI-Befehl die Programmflußkontrolle wieder an das verdrängte Programm zurückgegeben und Stackinhalte zurückgeschrieben.

Wie Sie bereits aus Kapitel 1 wissen, unterstützt der 6809-Prozessor drei Hardware-Interruptleitungen (NMI, FIRQ, IRQ) sowie drei Software-Interrupts (SWI1, SWI2, SWI3). Da in den Kapiteln 4 und 5 die Interruptmechanismen für die verschiedenen Peripheriekomponenten des Praktikumsrechners beschrieben werden, beschränken wir uns im folgenden Beispiel auf den Software-Interrupt SWI3.

Zu beachten ist, daß der 6809-Prozessor bei allen drei Software-Interrupts alle Register auf den Stack rettet. Werden an die Interruptroutine Übergabeparameter mittels Register übertragen, so liegen diese beim Einsprung in die Interruptroutine auf dem Stack. Eine oft genutzte Möglichkeit diese Werte in den Prozessor zu laden ist die Benutzung der Stack-relativen Adressierung. Hierbei werden die zu lesenden Werte relativ zum aktuellen Stackpointer adressiert und geladen. Rückgabeparameter an das aufrufende Programm können in der gleichen Weise (Stack-relativ !) auf den Stack geschrieben werden. Bei Beendigung der Interruptroutine werden diese durch den RTI-Befehl automatisch in die zugehörigen Prozessor-Register übertragen. Die Reihenfolge in der die einzelnen Registerwerte auf den Stack gelegt werden, ist in Kapitel 1 bei der Beschreibung des RTI-Befehls angegeben.

### Praktische Übung P2.5-4:

Schreiben Sie ein Programm, daß die Betriebssystemschnittstelle eines PCs auf dem Praktikumsrechner beispielhaft nachbildet. Hierzu werden alle Monitorroutinen nicht mehr direkt angesprochen, sondern durch ein sogenanntes Gate<sup>11</sup> (Interruptroutine mit SWI3) aktiviert. Dem Gate wird als Übergabeparameter die Nummer der zu aktivieren- den Monitorroutine nach folgender Tabelle in Register X übergeben:

<sup>11</sup> Durch die Implementierung von (Call-) Gates ist es möglich, unberechtigte Betriebssystemzugriffe durch Anwendungsprogramme zu verhindern. Moderne Betriebssysteme arbeiten hierzu in völlig gekapselten Adreßbereichen und Ebenen (z.B. bei Intel-Prozessoren im Protected Mode Level 0).

Monitorroutine	codierte Nummer
SHOWT7SG, Anzeigestelle S0	0
SHOWT7SG, Anzeigestelle S1	1
SHOWT7SG, Anzeigestelle S2	2
SHOWT7SG, Anzeigestelle S3	3
SHOWT7SG, Anzeigestelle S4	4
SHOWT7SG, Anzeigestelle S5	5
SHOWT7SG, Anzeigestelle S6	6
SHOWT7SG, Anzeigestelle S7	7
CLRDISP	8
HALTKEY	9

Im Register B soll abhängig von der gewünschten Monitorroutine der zu verarbeitende Wert übergeben werden, oder der zu erwartende Wert zurückgegeben werden (im Falle von HALTKEY).

Das Hauptprogramm soll einen Ziffernzähler für die Ziffern "0" und "1" realisieren. Dazu sollen die eingelesenen Ziffern in der Anzeigestelle S0 dargestellt werden und die Anzahl der bereits eingegebenen Ziffern in den Anzeigestellen S7 (Ziffer 0) und S6 (Ziffer 1) ausgegeben werden. Bei Betätigen der "+"-Taste soll das Programm die beiden Zähler zurücksetzen und erneut die Auftrittshäufigkeit der beiden Ziffern "0" und "1" zählen und darstellen. Andere Tasten sollen keine Auswirkungen haben.

**Hinweis:**

- > Die Aktivierung von Monitorroutinen durch das Hauptprogramm soll nur durch das obengenannte Gate mit dem Software-Interrupt 3 stattfinden.
- > Die Interrupt-Vektoradresse für den SWI3 (Software-Interrupt3) ist die 16bit-Adresse \$004C, \$004D.

# Lösungsvorschläge zu den Praktischen Übungen

## Zu P2.4-1:

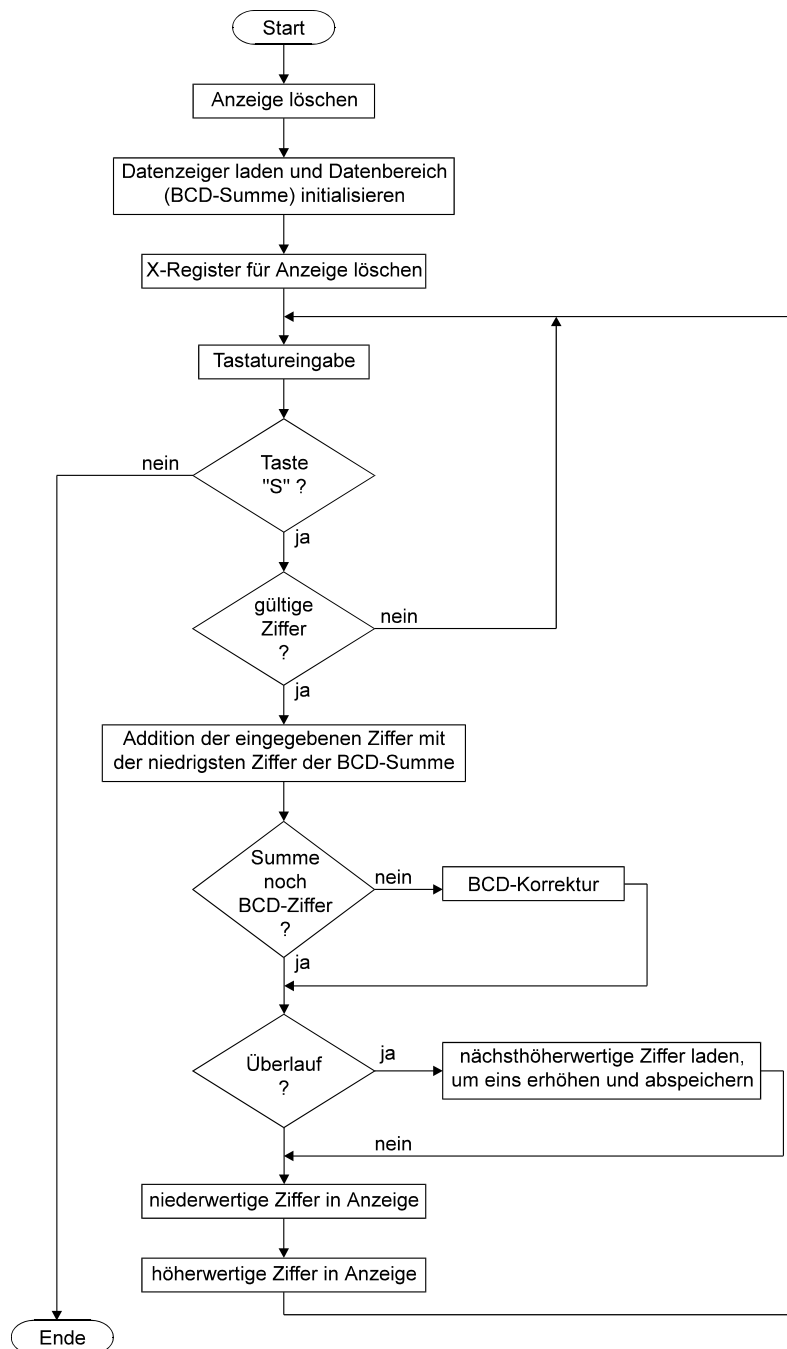
1. Wird das Programm mit der Option "-L" assembliert, ergibt sich das folgende Listing. Ganz links steht die Zeilennummer des Listings. Rechts daneben steht die Speicheradresse mit dem OP-Code. Weiter rechts folgt das Quellcodeprogramm.

```
0001                      *****
0002                      * P1.ASM
0003                      *
0004                      * Programm zur Addition von BCD-Zahlen ohne Verwen
0005                      * des DAA-Befehls
0006                      *
0007 0400                ORG    $0400    ;Beginn des Programmbereichs
0008
0009 0400 bd f1 10        JSR    CLRDISP ;Anzeige loeschen
0010 0403 10 8e 06 00    LDY    #$0600 ;Datenzeiger laden
0011 0407 6f a4          CLR    0,Y    ;Datenbereich mit Null initialisieren
0012 0409 6f 21          CLR    1,Y    ;
0013 040b 8e 00 00      START: LDX    #$0000 ;X loeschen
0014 040e bd f1 43      JSR    HALTKEY ;Zeichen von Tastatur lesen
0015 0411 c1 86          CMPB   #$86    ;Vergleich, auf Ende der Eingabe "S"
0016 0413 27 30          BEQ    ENDE    ;
0017 0415 c1 09          CMPB   #$09    ;Test der Eingabe auf gueltige Ziffer
0018 0417 22 f2          BHI     START  ;bei ungueltiger Ziffer zurueck
0019 0419 eb a4          ADDB   0,Y    ;Addition der Eingabe mit LSB der Sum
0020 041b 1f 98          TFR     B,A    ;
0021 041d 84 f0          ANDA   #$F0    ;Test auf Ueberlauf durch vorherige A
0022 041f 26 06          BNE     BCDADD ;wenn ja, Korrektur
0023 0421 1f 98          TFR     B,A    ;
0024 0423 81 09          CMPA   #$09    ;Test auf gueltige BCD-Ziffer
0025 0425 23 02          BLS     CARRY2 ;wenn nein, Korrektur
0026 0427 cb 06          BCDADD: ADDB  #$06 ;Korrektur, als Ergebnis gueltige BCD
0027 0429 1f 98          CARRY2: TFR   B,A ;
0028 042b 84 f0          ANDA   #$F0    ;
0029 042d 27 06          BEQ     SHOW   ;Test auf Ueberlauf durch vorherige A
0030 042f 86 01          LDA     #$01    ;
0031 0431 ab 21          ADDA   1,Y    ;naechsthoeherwertige Ziffer um Eins
0032 0433 a7 21          STA     1,Y    ;naechsthoeherwertige Ziffer abspeich
0033 0435 c4 0f          SHOW:  ANDB  #$0F ;eventl. Uebertrag der korrigierten Z
0034 0437 e7 a4          STB     0,Y    ;korrigierte Ziffer abspeichern
0035 0439 bd f1 1c      JSR    SHOWT7SG;korrigierte Ziffer in Anzeige
0036 043c e6 21          LDB     1,Y    ;naechsthoeherwertige Ziffer laden
0037 043e 30 01          LEAX   1,X    ;Anzeigestelle nach links verschieben
0038 0440 bd f1 1c      JSR    SHOWT7SG;naechsthoeherwertige Ziffer in Anz
0039 0443 20 c6          BRA     START  ;zurueck zur naechsten Eingabe
0040 0445 3f          ENDE:  SWI1
0041
0042 f110                CLRDISP EQU   $F110 ;Loeschen der Anzeige, In:-,
0043 f11c                SHOWT7SG EQU   $F11C ;unteres Nibble von B in Anzeige
0044 f143                HALTKEY  EQU   $F143 ;Lesen der Tastatur mit Warten,
```

Number of errors 0

2. Das Programm addiert die eingegebene BCD-Ziffer mit der niederstwertigen Ziffer der Summe. Die BCD-Addition wird ohne den DAA-Befehl durchgeführt. Überschreitet die Summe den Wert "99", so wird hexadezimal in der zweiten Stelle weiterge-

zählt. D.h. eine Überprüfung auf eine gültige BCD-Ziffer in der zweiten Stelle findet nicht statt. Beispielsweise ergibt  $99 + 1 = A0$ .  
Das folgende Flußdiagramm zeigt die Problematik:



#### Zu P2.4-2:

1. In dem Programm sind drei syntaktische Fehler vorhanden:  
Für Zeile 29 gibt der Assembler die Fehlermeldung "29: *Branch out of Range*". Hier fehlt das Sprungziel "SHOW" im Verzweigungsbefehl.  
Für Zeile 34 meldet er den Fehler "34: *Unrecognized Mnemonic*". Es fehlt beim *Store*-Befehl das Quellregister "B".

Für Zeile 37 meldet der Assembler: "37: Illegal Register for Indexed". In diesem Fall ist bei den Operanden des "LEAX"-Befehls das erste Komma zuviel.

2. Der logische Fehler des Programms besteht aus einer falschen Wahl eines Verzweigungsbefehls. Statt einen vorzeichenlosen Verzweigungsbefehl zu benutzen, wurde in Zeile 18 der "BGT"-Befehl benutzt. Hierdurch werden die Tasten des Funktionsblockes akzeptiert, so daß die Addition zu falschen Ergebnissen führt.

### Zu P2.4-3:

Der "DS9" erzeugt aus der Datei "P1.HEX" das folgende Listing:

```
;                *** 6809 - Disassembler Ver. 1.3 ***

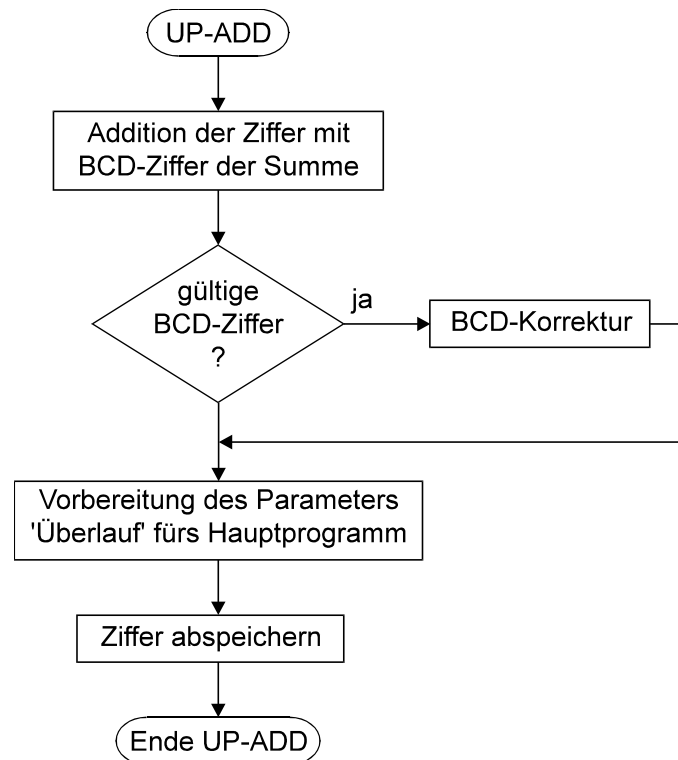
0400: BD F1 10      JSR    $F110
0403: 10 8E 06 00    LDY    #$0600
0407: 6F A4          CLR    ,Y
0409: 6F 21          CLR    $01,Y
040B: 8E 00 00      LDX    #$0000
040E: BD F1 43      JSR    $F143
0411: C1 86          CMPB   #$86
0413: 27 30          BEQ    $30    ;$0445
0415: C1 09          CMPB   #$09
0417: 22 F2          BHI    $F2    ;$040B
0419: EB A4          ADDB   ,Y
041B: 1F 98          TFR    B,A
041D: 84 F0          ANDA   #$F0
041F: 26 06          BNE    $06    ;$0427
0421: 1F 98          TFR    B,A
0423: 81 09          CMPA   #$09
0425: 23 02          BLS    $02    ;$0429
0427: CB 06          ADDB   #$06
0429: 1F 98          TFR    B,A
042B: 84 F0          ANDA   #$F0
042D: 27 06          BEQ    $06    ;$0435
042F: 86 01          LDA    #$01
0431: AB 21          ADDA   $01,Y
0433: A7 21          STA    $01,Y
0435: C4 0F          ANDB   #$0F
0437: E7 A4          STB    ,Y
0439: BD F1 1C      JSR    $F11C
043C: E6 21          LDB    $01,Y
043E: 30 01          LEAX   $01,X
0440: BD F1 1C      JSR    $F11C
0443: 20 C6          BRA    $C6    ;$040B
0445: 3F            SWI (1)
```

Symbolische Adressen für Sprungziele und Unterprogramme können bei einer Disassemblierung nicht berücksichtigt werden. Ein "Nach-Disassemblieren" von Hand ist notwendig. Der generierte Quellcode stimmt ansonsten mit dem "Original"-Code überein.

Beachten Sie bitte, daß der "DS9" bei Verzweigungen nicht nur den Offset zum Sprungziel angibt, sondern auch die absolute Adresse des Sprungzieles.

### Zu P2.5-1:

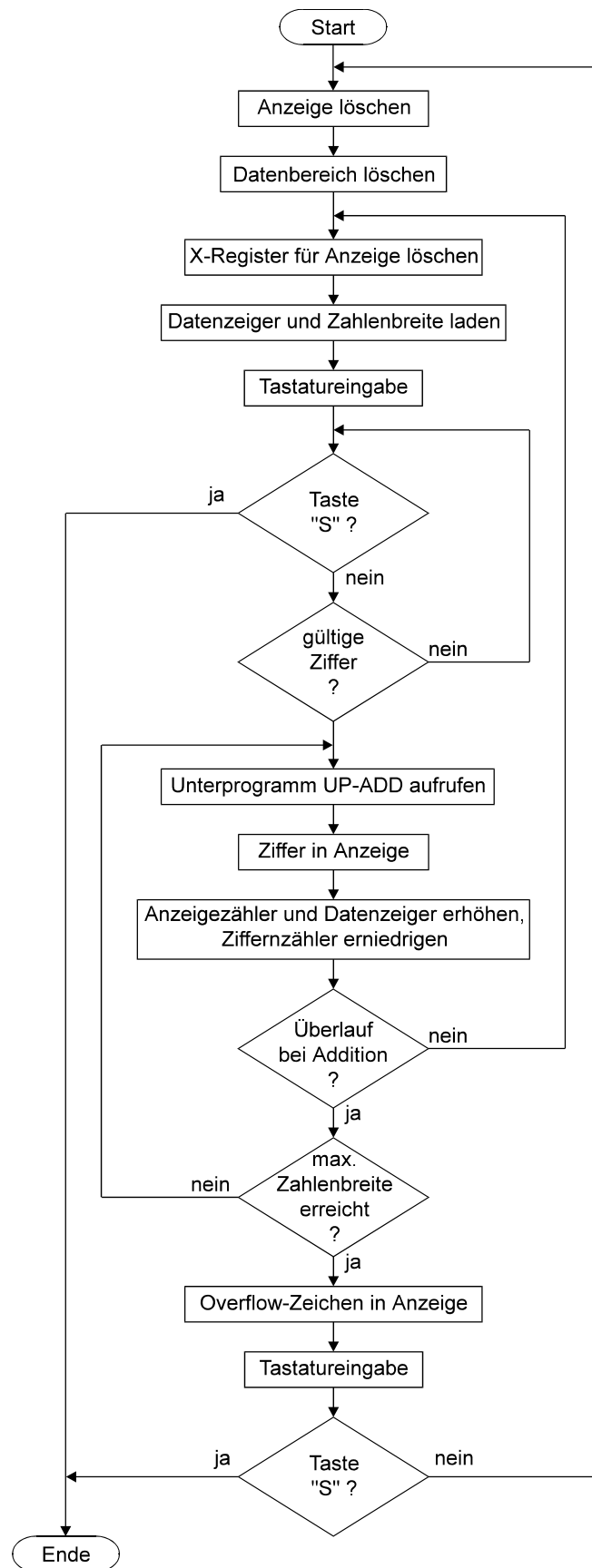
Für das Unterprogramm von Tabelle 6 ergibt sich das folgende Flußdiagramm:



Im Falle eines Übertrages auf die nächsthöhere Ziffer wird das Register A auf einen Wert ungleich Null gesetzt und dem Hauptprogramm übergeben. Die berechnete Ziffer wird abgespeichert. Die Übergabe der zugehörigen Speicheradresse erfolgt durch das Register Y.

### Zu P2.5-2:

Das Flußdiagramm für das Hauptprogramm ist auf der folgenden Seite abgebildet. Erzeugt die BCD-Addition einen Übertrag, so wird das Unterprogramm nochmals aufgerufen, um den Übertrag zur nächsthöheren Ziffer der Summe zu addieren. Im *worst case* muß über die gesamte Zahlenbreite eine Übertragsaddition erfolgen. Wird ein Überlauf erkannt, so wird das Overflow-Symbol zur Anzeige gebracht. Wird danach die "S"-Taste gedrückt, so wird das Programm beendet, ansonsten bei Eingabe einer gültigen Ziffer mit einer neuen Addition begonnen.



Eine mögliche Implementierung für das Hauptprogramm folgt auf der nächsten Seite.

```

;*****
; Pla.ASM
;
; Programm zur Addition von BCD-Zahlen mit maximal 7 Ziffern ohne Verwendung
; des 6809-Befehls DAA. Der Befehl wird durch das Unterprogramm UP_ADD er-
; setzt.
; Erweiterung der Wortbreite ohne Probleme moeglich
; (Praktikumsrechner hat allerdings max. 8 Stellen).
; Bei Ueberlauf Aktivierung der Anzeige.
; Abbruch der Eingabe mit der Taste "S" des Praktikumsrechners.
;
        ORG     $0400          ;Beginn des Programmbereiches

NEW:    JSR     CLRDISP        ;Anzeige loeschen
        CLRA                    ;A loeschen
        CLRB                    ;B loeschen
        STD     >$0600        ;Datenbereich $0600 - $0607
        STD     >$0602        ;mit Null initialisieren
        STD     >$0604        ;
        STD     >$0606        ;
START:   LDX     #$0000        ;X loeschen
        LDY     #$0600        ;Datenzeiger laden
        LDU     #ZWIDTH       ;Zahlenbreite laden
        JSR     HALTKEY        ;Zeichen von Tastatur lesen
        CMPB    #$86          ;Vergleich, auf Ende der Eingabe (Taste "S")
        BEQ     END           ;
        CMPB    #$09          ;Test der Eingabe auf gueltige Ziffer (0..9)
        BHI     START         ;bei ungueltiger Ziffer zurueck zur Eingabe
CARRY:   JSR     UP_ADD        ;Aufruf UP zur Addition von zwei BCD-Ziffern
        JSR     SHOWT7SG      ;korrigierte Ziffer in Anzeige
        LEAX    1,X           ;X:=X+1
        LEAY    1,Y           ;Y:=Y+1
        LEAU    -1,U          ;U:=U-1
        LDB     #$01          ;Reg. B fuer Addition des Uebertragsbits auf 1
        CMPA    #$00          ;Vergleich auf Ueberlauf
        BEQ     START        ;wenn nein, zurueck zur naechsten Eingabe
        CMPU    #$0000        ;Vergleich auf max. Zahlenbreite
        BEQ     OVERFL       ;Overflow, Uebertrag ignorieren; zur OV-Behandlung
        BRA     CARRY        ;Uebertrag addieren und anzeigen
OVERFL:  LDA     #$5C         ;
        JSR     SHOWA         ;Overflow-Symbol in Anzeigestelle S7
        JSR     HALTKEY        ;Zeichen einlesen
        CMPB    #$86          ;Programmende?
        BNE     NEW          ;nein, neuer Lauf
END:     SWI                  ;Programmende

ZWIDTH   EQU     $0007 ;MaximaleZiffernanzahl einer zu speichernden BCD-Zahl

;-----
; Hier beginnen die Definitionen der Monitorroutinen und der Adressen der
; Hardware-Register.
;
;                               Monitorroutinen
;
CLRDISP   EQU     $F110        ;Loeschen der Anzeige, In:-, Out:-
SHOWT7SG  EQU     $F11C        ;unteres Nibble von B in Siebensegmentcode wandeln
;und anzeigen, Position in X
SHOWA     EQU     $F113        ;Direkte Ansteuerung der Segmente der Anzeige-
;stelle durch das Byte in A, Position in X
HALTKEY    EQU     $F143        ;Lesen der Tastatur mit Warten, In:-, Out:B

```



### Zu P2.5-3:

Die folgenden drei Unterprogramme repräsentieren die geforderten Zähler. In "UP1" werden die Register über den Stack gerettet, während in "UP2" nicht benötigte Register verwendet werden. (S. Programm P3.ASM auf der CD-ROM.)

```
;=====
;Unterprogramm UP1: Zaehler, für die Anzeigestellen S5,S4
;
;           Register B und X werden ueber den Stack gerettet.
;
UP1:  LDX    #$0004    ;Anzeigeposition laden (S5,S4)
      LDB    #$FF      ;Zaehler initialisieren
NEXT1: INCB          ;Zaehler inkrementieren
      JSR    SHOWB7SG  ;Zaehlerstand anzeigen
      PSHS   B         ;Zaehler und Anzeigeposition ueber
      PSHS   X         ;den Stack retten
      JSR    UP2       ;Aufruf des zweiten UP
      PULS   X         ;Zaehler und Anzeigeposition einlesen
      PULS   B         ;
      CMPB   #$FF      ;Abbruchbedingung erfuehlt?
      BNE    NEXT1     ;wenn nein, zurueck zu NEXT1
      RTS             ;Ende Unterprogramm

;=====
;Unterprogramm UP2: Zaehler, für die Anzeigestellen S3,S2
;
;           Register B und X werden ueber Register gerettet.
;
UP2:  LDX    #$0002    ;Anzeigeposition laden (S3,S2)
      LDB    #$FF      ;Zaehler initialisieren
NEXT2: INCB          ;Zaehler inkrementieren
      JSR    SHOWB7SG  ;Zaehlerstand anzeigen
      TFR    B,A       ;Zaehler und Anzeigeposition ueber
      TFR    X,Y       ;andere Prozessorregister retten
      JSR    UP3       ;Aufruf des dritten (letzten) UP
      TFR    A,B       ;Zaehler und Anzeigeposition wieder
      TFR    Y,X       ;zurueckschreiben
      CMPB   #$FF      ;Abbruchbedingung erfuehlt?
      BNE    NEXT2     ;wenn nein, zurueck zu NEXT2
      RTS             ;Ende Unterprogramm

;=====
;Unterprogramm UP3: Zaehler, für die Anzeigestellen S1,S0
;
UP3:  LDX    #$0000    ;Anzeigeposition laden (S1,S0)
      LDB    #$FF      ;Zaehler initialisieren
NEXT3: INCB          ;Zaehler inkrementieren
      JSR    SHOWB7SG  ;Zaehlerstand anzeigen
      CMPB   #$FF      ;Abbruchbedingung erfuehlt?
      BNE    NEXT3     ;wenn nein, zurueck zu NEXT3
      RTS             ;Ende Unterprogramm
```

Im Unterprogramm "UP2" werden je zwei Stackoperationen benutzt, um die Registerinhalte auf den Stack zu schreiben bzw. einzulesen. Es ist auch möglich, die Stackoperationen mit jeweils einem Befehl auszuführen. Hierzu ist entweder der numerische Wert "immediate" anzugeben; oder die einzelnen Register durch Kommata getrennt aufzuzählen.

#### Zu P2.5-4:

Eine mögliche Implementierung des Hauptprogramms und der Interruptroutine zeigt das folgende Listing:

```
;*****
; P4.ASM
;
; Ziffern-Zaehlprogramm mittels einer Software-Interruptroutine.
; Software-Interrupt wird als Monitor- (Betriebssystem-) Gate benutzt,
; um Funktionen des Monitors zu benutzen. Eingegebener Wert wird in S0
; dargestellt, Anzahl der eingegebenen Werte fuer "0" in S7, fuer "1" in
; S6. Uebergabeparameter: X-Reg.: call-by-value, B-Reg.: call-by-refr.
; Parameter werden in der Interruptroutine Stack-relativ geladen.
;
        ORG    $0400    ;Beginn des Programmbereiches

        LDX    #MRGATE ;Interruptroutine in der Vektortabelle
        STX    $004C    ;an Adresse $004C, $004D eintragen
NEW:     LDX    #$0008    ;Uebergabeparameter nach A (Anzeige loeschen)
        SWI3                ;Software-Interrupt ausfuehren
        CLRB                ;Zaehler in den Speicherstellen
        LDX    #$0006    ;$06 und $07 in der Zero-Page
        STB    ,X          ;mit Null initialisieren
        SWI3                ;und jeweiliger Wert (0)
        LEAX   1,X         ;in den Anzeigestellen
        STB    ,X          ;S6 und S7 anzeigen
        SWI3                ;
INPUT:   LDX    #$0009    ;Uebergabeparameter fuer Tastatur nach X und
        SWI3                ;zur Interruptroutine verzweigen
        CMPB   #$01        ;Wurde Taste "1" betaetigt?
        BNE    INPUT0      ;Wenn nein, vielleicht "0"
        LDX    #$0000    ;Wert in Anzeigestelle
        SWI3                ;S0 anzeigen
        LEAX   6,X         ;Zaehler fuer den
        LDB    ,X          ;Wert "1" laden,
        INCB                ;inkrementieren und wieder
        STB    ,X          ;abspeichern
        SWI3                ;In S6 anzeigen
        BRA    INPUT      ;zurueck zur naechsten Eingabe
INPUT0:  CMPB   #$00        ;Vergleich auf Taste "0"
        BNE    INPUT+      ;wenn nein, vielleicht "+"-Taste
        LDX    #$0000    ;Wert in Anzeigestelle
        SWI3                ;S0 anzeigen
        LEAX   7,X         ;Zaehler fuer "0"
        LDB    ,X          ;laden,
        INCB                ;inkrementieren und
        STB    ,X          ;abspeichern
        SWI3                ;In S7 anzeigen
        BRA    INPUT      ;zurueck zur naechsten Eingabe
INPUT+:  CMPB   #$80        ;Vergleich auf "+"-Taste
        BEQ    NEW         ;Wenn ja, Zaehler neu initialisieren
        BRA    INPUT      ;ungueltige Taste, zurueck zur Eingabe

; Interruptroutine
MRGATE:  LDX    4,S         ;X-Register vom Stack holen (Stack-relativ)
```

```

        CMPX #$0008 ;Routine fuer Anzeige gefordert ?
        BHS DISP ;Wenn nein, vielleicht Anzeige loeschen
        JSR SHOWT7SG;Anzeige des unteren Nibbles aus B
        BRA SUBEND ;zum RTI
DISP:    CMPX #$0008 ;Routine zum loeschen der Anzeige gefordert?
        BNE TAST ;Wenn nein, vielleicht Routine zur Tastatureing.
        JSR CLRDISP ;Anzeige loeschen
        BRA SUBEND ;zum RTI
TAST:    CMPX #$0009 ;Routine zur Tastatureingabe gefordert?
        BNE SUBEND ;Wenn nein, falscher Uebergabeparameter; zurueck
        JSR HALTKEY ;auf Eingabe von Tastatur warten
        STB 2,S ;B-Register auf den Stack legen (Stack relativ)
SUBEND:  RTI ;Ruecksprung zum Hauptprogramm

CLRDISP EQU $F110 ;Loeschen der Anzeige, In:-, Out:-
SHOWT7SG EQU $F11C ;unteres Nibble von B in Anzeige, Position in X
HALTKEY EQU $F143 ;Lesen der Tastatur mit Warten, In:-, Out:B

```

Es hat sich als gute Programmierpraxis erwiesen, in Unterprogrammroutinen, und insbesondere in Interruptroutinen, nur eine Aussprungstelle zu programmieren. In der obigen Interruptroutine wäre es möglich nach den einzelnen Abfragen nicht mit einem BRA-Befehl zum Ende der Routine zu verzweigen und erst dann einen RTI-Befehl durchzuführen, sondern statt eines BRA-Befehls sofort einen RTI-Befehl zu programmieren. In Bezug auf ein gutes Software-Engineering und einfache Wartungs- und Pflegeoptionen hat sich herausgestellt, daß sich eine Beschränkung auf eine einzige Aussprungstelle positiv auf die Programmierfehlerrate auswirkt.

Sie finden das Assemblerprogramm in dem Unterverzeichnis „Software“ auf der CD-ROM in der Datei „P4.ASM“.