Der 6809-Cross-Assembler "AS9"

Der mit dem Praktikumsrechner auf Diskette mitgelieferte Assembler[1] 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 Übersetzungvorgang 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), ;(Strichpunkt), ' (Hochkomma), , (Komma), Leerzeichen oder Tabulator (*white space*).

[1] Bei der Wahl des Assemblers wurde versucht, einen freiverfügbaren 6809-Assembler zu finden, der zum einen einfach zu bedienen ist, zum anderen aber auch einen gewissen Komfort bietet.

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. Großbuchstaben 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 2.4-1 werden die einzelnen Operandenformate den zugehörigen Adressierungsarten des 6809 zugeordnet.

Tabelle 2.4-1: 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_1, [W_2, ..., W_n]	unmittelbar

*) R ist eines der Register PC, S, U, X oder Y. Q ist eines der Register S, U, X oder Y und W_i (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 Ausdruckes als Repräsentant ihres Integerwertes benutzt werden. Der Assembler ersetzt bei der Berechnung des Ausdruckes 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
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 2.4-2: 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	%1000110001010101	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 2.4-3 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[2]. 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 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 2.4-3: Das Programm "P24-1.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            ;
```

[2] Für das Praktikum verwenden Sie bitte nur die Endung ".ASM", da alle Tools des Praktikums diese Endung erwarten.

```
                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         ;naechsthoeherwertige Ziffer um Eins
                                  ;erhoehen
                STA   1,Y         ;naechsthoeherwertige 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         ;naechsthoeherwertige Ziffer laden
                LEAX  1,X         ;Anzeigestelle nach links verschieben, X:=X+1
                JSR   SHOWT7SG    ;naechsthoeherwertige 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:
Zur Veranschaulichung kann das assemblierte Programm auch direkt in den Praktikumsrechner geladen werden. Abschnitt 2.4.5 erläutert hierzu verschiedene Möglichkeiten.

Praktische Übung P2.4-1:

Assemblieren Sie das Programm "P24-1.ASM" durch Aufruf des Assemblers mit der Kommandozeile "AS9 P24-1.ASM". Rufen Sie dazu im Start-Menue unter „Alle Programme“ in der Programmgruppe „Zubehör“ die Eingabeaufforderung auf.

Assemblieren Sie das Programm "P24-1.ASM" unter der Verwendung verschiedener Optionen, z.B. mit der Kommandozeile "AS9 P24-1.ASM - L > P24-1.LST" und betrachten Sie die erzeugte Datei "P24-1.LST" mit Hilfe eines Texteditors. (Sollten Sie keinen Texteditor zur Verfügung haben, lesen Sie Abschnitt 2.4.4). Welche Aufgabe erfüllt das Programm? Wie reagiert das Programm, wenn der Wert "99" überschritten wird? Entwerfen Sie ein Flußdiagramm!

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 "P24-2.ASM" durch Aufruf des Assemblers mit der Kommandozeile "AS9 P24-2.ASM -L > P24-2.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 allokierenden 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 Ausdruckes 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.

PAG

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 Zeilennumer 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 einigen 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 2.4-4: 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" Maschinencodedateien ab.

Intel Hex-Dateiformat

Das Intel Hex-Format unterscheidet sich vom Motorola S-Format im wesentlichendurch die Anordnung der einzelnen Datenfelder eines Datensatzes, durch die Firmen-Kennung 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 2.4-5 zeigt das Format der Datensätze.

Tabelle 2.4-5: 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 Maschinencode, 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".