Bedienung

Der folgende Text ist ein Auszug aus dem Buch „Praktische Mikrocomputertechnik“ vom M. Kramer, 1. Auflage 1987; mit freundlicher Genehmigung des Autors.

2.3. Monitorprogramm

(S.64-69)

Nach der Vorstellung der Befehle des Mikroprozessors soll nun die Programmierung in Maschinensprache folgen.

Bei der Schaltungs- oder Programmentwicklung muß der Bediener mit dem Mikrocomputer über die Peripherie (Tastatur, Bildschirm usw.) in Verbindung treten können, muß Programme eingeben, im Speicher kontrollieren, starten und verändern. Dazu ist ein sogenanntes Monitorprogramm erforderlich, welches meist fest im ROM des Mikrocomputers untergebracht ist und beim Einschalten als erstes arbeitet. Dieses Monitorprogramm, oft auch als «der Monitor» bezeichnet, besteht im wesentlichen aus zwei Teilen:

- Programme für den jeweils vorhandenen Computer bezüglich Speichergröße und -ausführung sowie Peripherie (z. B. Anzahl der Tasten oder Bildschirmzeilen).
- Programme, die sich auf den eingesetzten Prozessor beziehen und von der Konfiguration unabhängig sind.

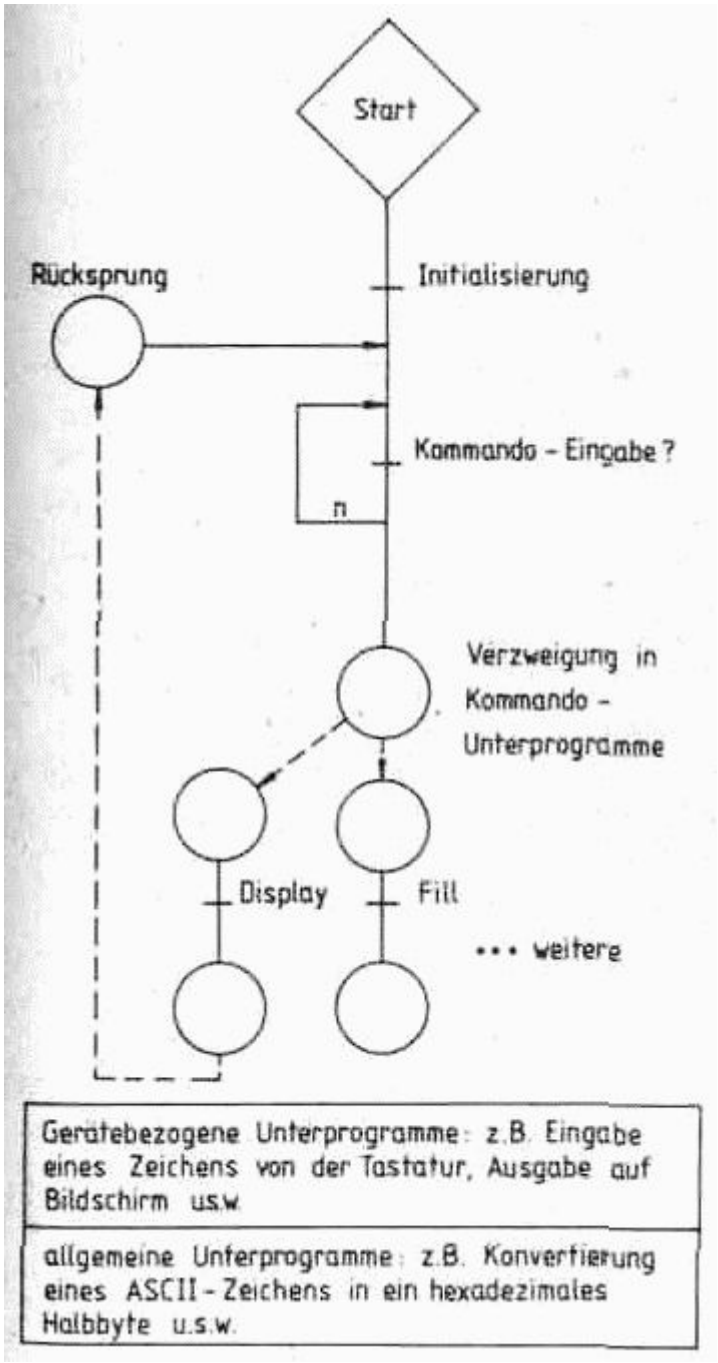
Für beides hat sich inzwischen eine Art unverbindlicher Standard herausgebildet, der in der Literatur auch ausführlich beschrieben ist und vor allem eine optimale Ausnutzung der Speicherkapazität des Monitorprogramms erlaubt.

2.3.1. Struktur

Bild 2.7 zeigt die Struktur des Monitorprogramms. Nach dem Einschalten bzw. nach dem RESET-Signal erfolgt zunächst die Initialisierung. Die entsprechend der jeweiligen Hardware-Konfiguration vorhandenen programmierbaren Peripherieschaltkreise PIO, CTC, SIO usw. programmiert man je nach ihrer Verwendung, z. B. ein PIO für eine Tastatur bei Port A auf Ausgabe und bei Port B auf Eingabe. Der Speicher für den Bildschirm sowie die unter 2.5. näher erläuterte RAM-Tabelle des Monitorprogramms, die nach dem Einschalten einen zufälligen Inhalt aufweisen, müssen einen definierten Anfangszustand erhalten. Der Bildschirm wird «gelöscht», indem der Prozessor den Bildwiederholspeicher mit Leerzeichen beschreibt. Nach der Initialisierung gelangt das Programm in die Aufrufschleife und erwartet. die Eingabe eines Monitorkommandos. Die Kommandos werden als Unterprogramme abgearbeitet, anschließend erfolgt die Rückkehr in die Aufrufschleife, und es kann eine neue Kommandoeingabe erfolgen.

Die Programmteile für die Kommandos rufen weitere Unterprogramme auf, z. B. für eine Konvertierung in eine andere Form der Darstellung oder für die Ausgabe eines Zeichens auf dem Bildschirm.

Bild 2.7 Struktur des Monitors



2.3.2. IN/OUT-Routinen

Das zuletzt genannte Unterprogramm wird außer vom Monitor auch von anderen Programmen benötigt. Da alle Programme irgendwelche Daten über Eingabekanäle erhalten und nach der Bearbeitung wieder ausgeben, sind die Unterprogramme für Peripheriegeräte wie Tastatur, Bildschirm und externe Speicher (Kassettenrecorder, Lochband usw.) bei jedem Mikrocomputer von allgemeinem Interesse, Im vorliegenden Monitor erfolgt die Verarbeitung in Form von Einzelzeichen, üblich ist auch die Verarbeitung von Datenblöcken.

2.3.3. Einsprungtabelle

Es besteht natürlich die Möglichkeit, aus anderen Programmen die IN/OUT-Routinen direkt aufzurufen (CALL-Befehl). Bei einer Änderung des Monitorprogramms müßten jedoch auch Adressen in allen anderen Programmen (z.B. BASIC-Interpreter, Editor, Assembler) geändert werden. Deshalb fügt man eine Tabelle mit Sprungbefehlen zu den IN/OUT-Routinen an einer Stelle ein, wo Änderungen nicht zu erwarten sind, z. B. hinter den Restart-Adressen am Anfang des Monitors. Im vorliegenden Monitor ist die Tabelle 2.18. vorhanden.

Anstelle der wirklichen IN/OUT-Routine kann nun eine Adresse dieser Sprungtabelle, z. B. Adresse 00E0H für eine Zeicheneingabe von der Tastatur, aufgerufen werden. Änderungen der Routine CI selbst wirken sich so auf übergeordnete Programme nicht aus.

Tabelle 2.18. Einsprungtabelle

Adresse	Befehl	Bemerkung
0E0H	MCI: JMP CI	Einzelzeicheneingabe Tastatur, Zeichen in Register A
0E3H	MRI: JMP RI	Einzelzeicheneingabe INPUT-Kanal, Zeichen in Register A
0E6H	MCO: JMP CO	Einzelzeichenausgabe Bildschirm, Zeichen in Register C
0E9H	MWO: JMP WO	Einzelzeichenausgabe OUTPUT-Kanal, Zeichen in Register C
0ECH	MLO: JMP LO	Einzelzeichenausgabe LIST-Kanal, Zeichen in Register C
0EFH	MCSTS: JMP CSTS	Tastaturstatus 0 = kein Zeichen, -1 = Zeichen
0F2H	MIOC: JMP IOCHK	IN/OUT-Zuweisung abfragen, Byte in Register A
0F5H	MIOS: JMP IOSET	IN/OUT-Gerät zuweisen, Byte in Register A

2.3.4. IN/OUT-Zuweisung

Die Ausgabe und Eingabe von Zeichen erfolgt meist über verschiedene Peripheriegeräte. So kann ein Programm durch den INPUT-Kanal mit der Routine MRI sowohl über die Tastatur, über ein Kassettenmagnetbandgerät oder auch von einem anderen externen Speicher in den Mikrocomputer gelangen. Gleiches gilt für die Ausgabe: Mit der Routine MLO kann z. B. auf den Bildschirm, auf einen externen Speicher, auf verschiedene Drucker oder andere Geräte aus-gegeben werden.

Die Auswahl des physisch vorhandenen Geräts «Tastatur» zur logischen Funktion «INPUT-Kanal» (Zeicheneingabe) speichert der Mikrocomputer in einem Byte. Beim Aufruf der Routine MRI wird dieses IN/OUT-Byte bei jedem Zeichen abgefragt und ausgewertet. Die Ausgabekanäle OUTPUT und LIST arbeiten in gleicher Weise. Die Änderung des IN/OUT-Bytes und der damit verbundene Wechsel der physischen Geräte erfolgt mit Monitorkommandos.

2.3.5. Debugger

Der auf den Mikroprozessor und nicht auf das Gerätekonzept zugeschnittene Teil des Monitors erlaubt vor allem die Arbeit mit dem Speicher, z. B. lesen, schreiben, ändern, Programme starten. Da diese Aktivitäten zumeist beim Test und bei der Fehlersuche an Programmen gebraucht werden, hat sich hierfür die Bezeichnung DEBUG-Programm oder DEBUGGER eingebürgert. Die Tabelle 2.19. gibt eine Übersicht der Monitorkommandos, ihren Anwendungszweck und die Form, in der die Kommandos eingegeben werden müssen. Einige bestehen nur aus einem Sprung in ein anderes Programm, das

dann nur die IN/OUT-Routinen benutzt (z. B. BASIC). Das erste Zeichen, ein Buchstabe, ist die eigentliche Auswahl des Kommandos. Meist sind dann noch weitere «Parameter», wie Speicheradressen oder Datenbytes, erforderlich.

Die Eingabe von Adressen und Daten erfolgt im Hexadezimalcode, wobei sowohl große als auch kleine Zeichen zulässig sind. Ein Beispiel soll die Bedienung des Monitors näher erläutern: Die Eingabe D0,F (CR) ergibt die Bildschirmausschrift des Inhalts der Speicherplätze 1 bis 15.

Das Komma dient als Trennzeichen, es schließt die Eingabe der ersten Adressen ab. Als weiteres mögliches Trennzeichen dient auch das Leerzeichen (Space-Taste).

Bei Eingabe von mehr als vier hexadezimalen Zeichen gelten nur die letzten vier. Man kann somit Eingabefehler durch Überschreiben berichtigen. Dieses Prinzip gilt bei allen Parametereingaben. Es sind nur die letzten HEXA-Zeichen gültig. Der Abschluß der Kommandos erfolgt mit der Taste Wagenrücklauf (CR).

Erst mit dem Druck auf diese Taste wird das Kommando ausgerührt. Bei Eingabe nicht mit einem Kommando belegter Buchstaben oder anderer fehlerhafter Eingaben antwortet der Mikrocomputer als Fehlermeldung mit einem Fragezeichen und gelangt anschließend wieder in die Aufrufschleife.

Tabelle 2.19. Monitorkommandos

Kommando	Funktion	Syntax, Parameter
A	Assembler-Aufruf	A
B	BASIC-Aufruf	B
C	COPY WO: = RI	C
D	DUMP	DAADR, EADR
E	End-File	EAADR
F	Fill	FAADR,EADR,Byte
G	GO	GAADR,EADR1,EADR2 oder GAADR oder G
H	HEX-Arithmetik	HADR,ADR
I	INPUT-Kanal zuweisen	IT oder IL oder IU
J	JUMP RAM-Tabelle	J
K	KNOW RAM-Test	KAADR,EADR
L	LIST-Zuweisung	LT oder LL oder LU
M	MOVE	MAADR,EADR,ZADR
O	OUTPUT-Kanal zuweisen	OT oder OL oder OU
P	PRINT Disassembler	PAADR,EADR
R	READ ext. Speicher	RADR
S	SUBSTITUTE	SADR,
T	TEXT Editor	T
V	VERIFY	VAADR,EADR,ZADR
W	WRITE	WAADR,EADR
X	Register anzeigen	X oder Xr
Z	Zweitregistersatz	Z oder Zr

Erläuterungen:

ADR Adresse, AADR Anfangsadresse, EADR Endeadresse, ZADR Zieladresse
r Register A, B, ...

2.3.6. RAM-Tabelle

Der Monitor benötigt für seine Aufgaben auch einige RAM-Speicherplätze. Beim Testlauf eines Programms unter Verwendung von Haltepunkten müssen die Adressen der Haltepunkte sowie deren Speicherinhalt und der Inhalt der Register des Mikroprozessors im RAM gespeichert werden. Durch Anzeige dieser Speicherplätze mit den Monitorkommandos X und Z kann man die richtige Arbeitsweise des zu testenden Programms kontrollieren.

Weitere Speicherplätze der RAM-Tabelle sind die Adressen für den jeweils aktuellen Ausgabeplatz auf den Bildschirm sowie für Peripheriegeräte des Nutzers, die den IN/OUT-Routinen zugewiesen werden können. Die aktuelle Zuweisung der Peripheriegeräte selbst steht ebenfalls in dieser Tabelle. Auch Steuerzeichen bzw. Betriebsweisen von Peripheriegeräten, Zeitkonstanten usw., die veränderlich sind, bringt man dort unter. Beim Start des Monitors wird die RAM-Tabelle in einen definierten Zustand versetzt, indem sie aus dem ROM übertragen wird. So ist der Platz des Registers A mit AA, B mit BB usw. belegt, so daß die im zu testenden Programm festgelegten Veränderungen (z.B. LD A,B) beim Haltepunkt erkennbar sind.

2.4. Programmbeispiele in Maschinensprache

2.4.1. Bedienung des Monitors

Bevor konkrete Beispiele folgen, soll zunächst die Bedienung des Monitors näher erläutert werden. Als besonders gut verständliche Zielstellung mögen wieder die Erstellung und der Testlauf eines Programms für eine Zeitverzögerung dienen. Entsprechend dem Programmablaufplan von Bild 1.1 wollen wir zunächst die innere Schleife aufbauen. Wir laden das Register B mit der maximal möglichen Zahl und vermindern den Inhalt um den Wert 1, bis das Register B leer ist (Meldung durch Z-Flag). In der Mnemonik geschrieben, ergibt sich:

```
    LD B,0FFH
M1: DEC B
    JPNZ M1
```

Dabei stellt die Marke M1 die symbolische Form der Adresse des Decrementbefehls dar. Da wir Startadresse und Länge der Befehle bisher nicht wissen, können wir die Sprungadresse noch nicht anders angeben. Das vollständige Unterprogramm sieht dann wie folgt aus:

```
    LD A,0FFH
M2: LD B,0FFH
M1: DEC B
    JPNZ M1
    DEC A
    JPNZ M2
    RET
```

Wir legen die Startadresse an den Beginn des RAM-Bereiches im Grundgerät, entnehmen die hexadezimale Codierung den Tabellen und stellen alles noch einmal übersichtlich dar:

```
Adresse Daten   Mnemonik
0C00    3E    LD A,0FFH
0C01    FF
0C02    06    LD B,0FFH
0C03    FF
0C04    05    DEC B
0C05    C2    JPNZ 0C04
0C06    04
0C07    0C
0C08    3D    DEC A
0C09    C2    JPNZ 0C02
0C0A    02
0C0B    0C
0C0C    C9    RET
```

Die Eingabe des Programms in den Computer erfolgt mit dem Kommando S (Substitution). Nach der Eingabe der Zeichenfolge S0C00 und Leertaste gibt der Computer den augenblicklichen Speicherinhalt und danach einen Bindestrich aus. Nach der Eingabe von 3E und Leertaste überschreibt er damit den alten Speicherinhalt der Adresse 0C00H und zeigt den Inhalt der nächsten auf dem Bildschirm an. Nach Eintippen des ganzen Programms wird mit CR abgeschlossen. Mit dem P-Kommando kontrollieren wir es noch einmal auf Fehler. Der Disassembler wandelt den Speicherinhalt wieder in die Mnemonik um. Die Eingabe lautet P0C00, 0C0C (CR). Wenn alles richtig eingetippt wurde, kann mit dem GO-Kommando der Testlauf beginnen. Am Ende des Programms befindet sich ein RET-Befehl. Es steht jedoch keine Rückkehradresse im Keller. Deshalb setzen wir an die Stelle des RET einen Haltepunkt im Testlauf: G0C00, 0C0C (CR). Nach der Verzögerungszeit unserer Zeitschleife meldet der Computer die Adresse des Haltepunkts. Mit dem Kommando X (CR) sehen wir uns die Register an und stellen fest, daß die Register A und B nicht mehr den bei der Initialisierung eingetragenen Wert besitzen, sondern auf 00 stehen. Somit ist die ordnungsgemäße Funktion des Programms erwiesen. Es ist jedoch noch verbesserungswürdig, denn die «Speicherplatzverschwendung» bei den Sprungbefehlen stört. Mit relativen Sprüngen, bei denen nur der Abstand des Ziels vom Absprungort als Argument und keine vollständige Adresse erforderlich ist, kann man jeweils 1 Byte einsparen. Außerdem läuft das Programm dann adressenunabhängig. Jetzt enthält es beim 1. Sprungbefehl die konkrete Adresse 0C04. Würde das Programm auf der Adresse 0D00 im Speicher stehen, erfolgte der Sprung trotzdem auf 0C04, wo aber dann nichts Sinnvolles steht. Der relative Sprung 20FD bedeutet nur «3 Speicherplätze zurückspringen, wenn das Z-Flag nicht gesetzt ist», und das Programm läuft damit auf beliebigen Adressen.

Für die innere Schleife bietet sich der Befehl DJNZ an, der den Inhalt von Register B um 1 vermindert («dekrementiert») und einen Sprung um die Distanz e-2 von der Adresse des Befehls ausführt. Der Operand des Befehls ist also keine konkrete Adresse, sondern wird als Zahl bewertet. Um sowohl rückwärts als auch vorwärts im Programm springen zu können, dient das Bit 7 dieses Bytes als Vorzeichen. Nicht gesetzt, bedeutet positive Zahl und damit Sprung vorwärts. Daraus ergibt sich eine maximale Sprungdistanz von. +127 bis -128 Speicherplätzen. Im Beispiel soll der Sprung zum DJNZ-Befehl erfolgen, also um -2. Damit ergibt sich der hexadezimale Wert FE als Operand. Das Argument des 2. relativen Sprunges berechnen wir mit Hilfe der hexadezimalen Arithmetik des Monitors. Entsprechend dem allgemeinen Ausdruck

H Ziel, Quelle +2
erhalten wir im Beispiel den Wert F9H. Die Summe der Adressen, die das H-Kommando liefert, benötigen wir bei Vorwärtssprüngen. Es ergibt sich nun:

```
Adresse Daten    Mnemonik
0C00    3E    LD A,0FFH
0C01    FF
0C02    06    LD B,0FFH
0C03    FF
0C04    10    DJNZ 0C04
0C05    FE
0C06    3D    DEC A
0C07    20    JRNZ 0C02
0C08    F9
0C09    C9    RET
```

Mit der Anweisung M0C00, C09, D00 verschiebt man das Programm auf die Adressen D00 bis D0A und kann es, wie bereits beschrieben, mit dem GO-Kommando prüfen. Mit der Eingabe OT weist man als Ausgabekanal den Bildschirm zu und kann sich nach Eingabe der Folge WD00,D09 (CR) ansehen, wie der Computer dieses Programm auf einen externen Speicher (z.B. Kassettenrecorder) ausgibt. Diese Form ist für langsame externe Speicher international üblich und soll deshalb ausführlicher erläutert werden.

Die Zeile
:0A0D00003EFF06FF10FE3D20F9C97A
läßt nur schwer unser Programm erkennen.
Der Doppelpunkt ist das Kennzeichen für den Beginn eines Datenblocks. Falls man beim Einlesen mitten in einen Block schaltet oder wenn ein anderer Fehler auftritt, kann der nächste Datenblock daran erkannt werden. Sonst treten ja nur hexadezimale Zahlen auf. Die nächsten beiden Ziffern geben die Blocklänge an. Bei Ausgabe eines größeren Speicherbereichs (siehe Anhang) haben die Blöcke eine «Nutzlänge» von 10H (16dezimal) Bytes. Üblich sind auch Blocklängen von 80H, jedoch lassen sie sich schlechter auf Bildschirm oder Drucker darstellen. Die folgenden 4 hexadezimalen Ziffern bezeichnen die Anfangsadresse des Datenblocks, im Beispiel 0D00. Es folgen 2 Bytes, die das vorliegende Monitorprogramm nicht auswertet. Bei größeren Programmsystemen dienen sie mitunter zur Bezeichnung von Plätzen im Datenblock, auf denen Adressen stehen.

Nach den Datenbytes unseres Programms, folgen am Schluß 2 hexadezimale Ziffern zur Kontrolle. Sie errechnen sich durch Addition aller Daten des Blocks ohne Übertrag. Beim Einlesen berechnet der Computer diese Prüfsumme erneut. Nur wenn sie mit der gelesenen Summe übereinstimmt, erkennt er den Datenblock als fehlerfrei an und überträgt ihn aus dem Zwischenspeicher des Einleseprogramms auf die zugehörige Adresse. Ist als Eingabekanal kein externes Gerät zugewiesen, so können in dieser Form mit dem R-Kommando des Monitors auch Programme (z. B. aus dem Anhang des Buches) über die Tastatur eingegeben werden. Gegenüber dem S-Kommando besteht eine höhere Sicherheit gegen Tippfehler. Das R-Kommando erwartet als Abschluß einen Datenblock, der die Startadresse des Programms angibt. In unserem Beispiel wird er im Anschluß an das W-Kommando mit E0D00(CR) erzeugt. Das Einleseprogramm erlaubt es auch, Datenblöcke mit einem Adressenversatz einzulesen. Das Programm läuft zwar in der Regel nur auf den Adressen, für die es geschrieben ist, jedoch kann man die Verschiebung für die Programmierung von EPROMs nutzen. Soll ein EPROM später auf Adresse 8000H laufen, wie z. B. der BASIC-Interpreter, so kann dieses Programm zunächst mit R9000(CR) in einen RAM-Bereich ab Adresse 1000H eingegeben werden, von wo man es auf das EPROM programmiert. Nach dem gerade beschriebenen Muster kann man mit dem

Grundgerät Programme erstellen und betreiben. Trotz der Ausführlichkeit der letzten Darstellung wurden längst nicht alle Möglichkeiten der Arbeit mit dem Monitor und dem Grundgerät gezeigt. Der Autor empfiehlt jedem Amateur, zunächst alle Monitorkommandos und danach alle Befehle des Mikroprozessors in Maschinensprache «durchzuspielen».

2.5. Programmierung mit Assembler

(S. 72-76)

Die in Abschnitt 2.4. ausführlich beschriebene Programmierung in Maschinensprache ist im Vergleich zu BASIC recht mühsam und fehlerträchtig. Die schematische Übersetzung der Mnemonik in den Maschinencode, insbesondere die Berechnung von Argumenten der relativen Sprungbefehle, kann jedoch auch der Computer mit einem speziellen Programm, dem Assembler, ausführen. Der Assembler erwartet die Eingabe eines Programms als sogenannten Quelltext in der Mnemonik, wobei Sprungziele und Operanden symbolisch mit Marken oder Namen bezeichnet werden können. Er liefert daraus den Maschinencode für den Computer und für den Programmierer eine Liste, in der Quelle und Maschinencode übersichtlich zusammen enthalten sind.

Da der Assembler das Quellprogramm schematisch übersetzt, muß man bei der Schreibweise der Mnemonik sehr korrekt vorgehen. Bedingt durch die stürmische Entwicklung der Mikrocomputertechnik und andere historische Voraussetzungen, gibt es keine einheitliche Festlegung für die Mnemonik der Befehle.

Heute sind in der DDR drei unterschiedliche Typen für den Prozessor UB 880 D üblich, die sich jedoch nur bei einigen. Befehlen unterscheiden. International am weitesten verbreitet ist die Schreibweise, die in den Tabellen für die Programmierung in Maschinensprache enthalten ist. In der Literatur findet man jedoch auch häufig eine Mnemonik, die vom Prozessor 8080 übernommen wurde. Steht in einem Quellprogramm anstelle von LD A,B der entsprechende Befehl MOV A,B, so «versteht» der Assembler nur eine Schreibweise, die andere bewertet er als Fehler. Der im Anhang aufgeführte Assembler arbeitet mit der ROBOTRON-Mnemonik. Die korrekte Schreibweise der Befehle kann man in den Programmlisten im Anhang nachlesen.

2.5.1. Editor

Die Erstellung des Quellprogramms erfolgt ebenfalls mit einem Hilfsprogramm. Mit dem Editor können Texte über die Tastatur oder von einem externen Gerät in den Speicher des Computers eingegeben und komfortabel bearbeitet werden. Man kann Zeichen und sogar ganze Zeilen einfügen oder streichen, und der übrige Text verschiebt sich entsprechend im Speicher. In Verbindung mit einem Drucker macht der Editor den Computer zur komfortablen Schreibmaschine. Der im Abschnitt 5.2.6. aufgeführte Editor entstand nicht nur, um damit Programme zu erstellen, sondern diente auch zum Schreiben dieses Buches. Der relativ geringe Umfang des Programms bedingt natürlich auch Grenzen der Leistungsfähigkeit und konnte nur durch direkten Zugriff auf Unterprogramme aus dem Monitor realisiert werden. Trotzdem ist der Effekt erstaunlich. Einige Einzelheiten dienen auch der Zusammenarbeit mit anderen vorhandenen Programmen. Die meisten Kommandos zur Bedienung des Editors (Tabelle 2.20) entsprechen den vergleichbaren des Monitors; nur sind die Argumente hexadezimale Zeilennummem und lediglich bei der Textpufferzuweisung Adressen im Speicher.

Erfolgt keine ausdrückliche Pufferzuweisung, so legt der Editor den Text im Adreßbereich 4000H bis 7FFFH ab. Mit dem Kommando Q kann man den Editor verlassen und mit dem Monitor oder über diesen mit dem Assembler weiterarbeiten. Stellt sich beim Testlauf des Programms ein Fehler heraus, geht man in den Editor zurück und bearbeitet den Quelltext erneut. Beim Aufruf des Editors erfolgt deshalb zunächst eine Abfrage, ob ein neuer Text bearbeitet werden soll oder ob sich bereits etwas im Textpuffer befindet. Der Buchstabe Y bewirkt die Neuzuweisung des Textpuffers, alle anderen Zeichen stellen den Zeilenzähler nur auf Null und erlauben weiterhin den Zugriff auf den Text im Puffer.

Das Kommando E ist das Kernstück des Editors, es erlaubt Eingabe und Bearbeitung von Text über die Tastatur. Um den Computer auch für Schreibarbeiten nutzen zu können, erfolgt dabei eine Umschaltung des Zeichenvorrates. Die großen Buchstaben erscheinen nur noch mit der Taste Shift. Die letzte Bildschirmzeile dient der Bearbeitung. Alle Zeichen, die sich in dieser Zeile links vom Cursor befinden, gelangen nach Betätigung der Tasten CR, LF oder Control Z in den Speicher des Textpuffers.

Tabelle 2.21. zeigt die Steuerzeichen des Editors, Bild 2.8 die Arbeitsweise als Programmablaufplan. Tabelle 2.22. erläutert die Fehlermeldungen.

Tabelle 2.20. Editor-Kommandos

Kommando	Funktion	Syntax, Parameter
A(sign)	Textspeicherzuweisung	A nnnn (CR) nnnn (CR)
D(isplay)	Darstellen einer Anzahl von n Zeilen mit Weiterstellen des Zeilenzählers	Dn (CR) oder D(Leerz.)...(CR)
E(dit)	Texteingabe-Modus	En (CR)
G(o)	Weiterstellen des Zeilenzählers vorwärts um n Zeilen	Gn (CR)
I(nput)	Zuweisung Eingabekanal	IT, IL oder IU
L(ist)	Zuweisung Druckerkanal	LT, LL oder LU
0(utput)	Zuweisung Ausgabekanal	OT, OL oder OU
P(rint)	Ausgabe aller Zeilen auf Listkanal	P
Q(uit)	Rücksprung in den Monitor	Q
R(ead)	Lesen des Eingabekanals bis zum ETX-Zeichen	R
T(op)	Rückstellen des Zeilenzählers an den Anfang	T
U(p)	Rückstellen des Zeilenzählers um n Zeilen	Un (CR)
W(rite)	Ausgabe einer Anzahl oder aller Zeilen	Wn(CR) oder W (CR)

Erläuterungen:
n: Hexa-Zahl, max. vierstellig
(CR): Wagenrücklauf-Taste

Tabelle 2.21. Editor-Steuerzeichen

Taste	Funktion
BS	Cursor nach links, Korrektur durch Überschreiben möglich
DEL	Inhalt der Anzeigezeile löschen
LP	Zeile in Textpuffer übernehmen, nächste Zeile aus dem Puffer anzeigen

Taste	Funktion
CR	Zeile in Puffer übernehmen, E-Kommando verlassen
Control D	vorletzte Zeile in Arbeitszeile doppeln
Control Z	Zeile in Puffer übernehmen, neue Zeile beginnen
Control X	Zeichen über dem Cursor löschen, Rest nach links verschieben
Control S	Zeichen über dem Cursor einfügen, Rest nach rechts verschieben

Bild 2.8 Arbeitsweise des Editors

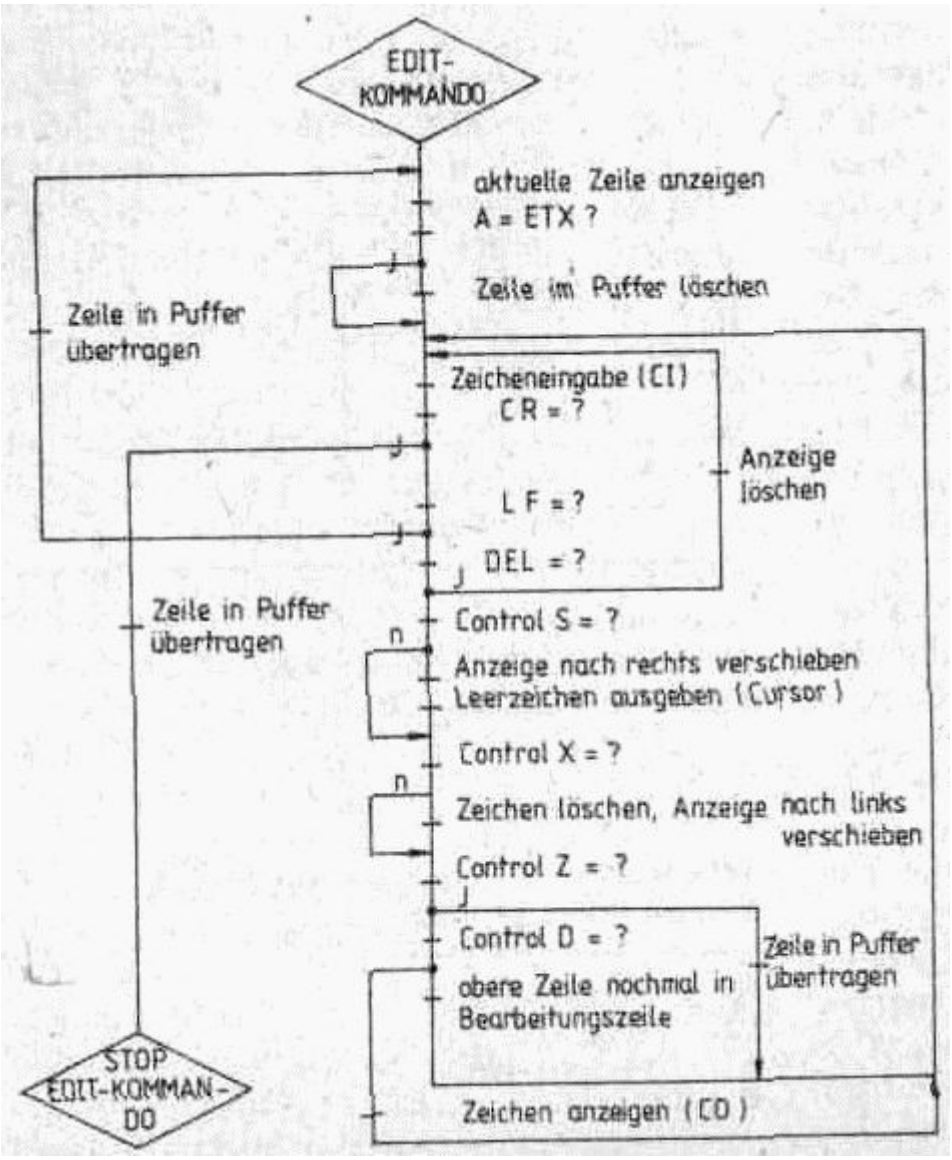


Tabelle 2.22. Editor-Fehlermeldungen

E1	Kommandofehler
E2	Textpuffer voll
E3	Parameterfehler

2.5.2. Assemblerbeschreibung

Gegenüber der manuellen Übersetzung eines Programms mit Tabellen der hexadezimalen Kodierung von Befehlen hat ein Assemblerprogramm zwei wesentliche Vorteile:

Die Übersetzung längerer Programme verläuft um mehrere Größenordnungen schneller und ist syntaktisch fehlerfrei. Das bedeutet aber nicht die Beseitigung logischer Fehler. Ist z. B. bei einem Sprung das Ziel im Quellprogramm falsch bezeichnet oder ein Flag falsch ausgewertet, dann erfolgt die Übersetzung zwar syntaktisch richtig als Sprung, aber logisch falsch beim Operanden. Das Programm kann nicht richtig arbeiten. Trotzdem lohnt sich der Aufwand für die Nutzung eines Assemblers. Sehr wertvoll ist auch die übersichtliche Darstellung der Programme, wie das Beispiel des Monitorprogramms im Anhang zeigt. Der Ausdruck ist in mehrere, unterschiedlich breite Spalten formatiert. Links befindet sich die Adresse, für die das Maschinenprogramm übersetzt wurde. Neben der üblichen hexadezimalen Angabe bieten einige Assembler sie noch in dezimaler Schreibweise. Damit kann der Programmierer besser Berechnungen ausführen oder für den Aufruf durch den BASIC-Interpreter ablesen, denn in dieser Programmiersprache muß man Adressen meist dezimal angeben.

Die nächste Spalte des Assemblerausdrucks enthält das Maschinenprogramm (Objektcode) in hexadezimaler Kodierung, wobei nicht mehr als ein Befehl pro Zeile steht. Entsprechend den unterschiedlichen Befehlslängen kann diese Spalte bis zu 4 zweistellige hexadezimale Zahlen enthalten.

Die immer besetzte Spalte mit der dezimalen Zeilennummer trennt den Maschinencode vom Quellprogramm. Nicht immer muß das Quellprogramm Befehle enthalten, auch Erläuterungen, die den Sinn von Programmteilen oder für den Eintritt in ein Programm erforderliche Parameter für eine spätere Bearbeitung festhalten, sind äußerst nützlich. Diese Kommentare können entweder hinter einem Befehl stehen oder schon unmittelbar in der Spalte nach der Zeilennummer beginnen. Sie sind durch ein vorangestelltes Semikolon gekennzeichnet. Unmittelbar auf die Zeilennummer folgt das Markenfeld. Wie bereits erwähnt, symbolisiert eine Marke den Wert einer Adresse. Meist kennzeichnet sie ein Sprungziel, manchmal auch den Beginn oder das Ende von Tabellen. Eine Marke besteht aus maximal 5 Zeichen, von denen das erste ein Buchstabe sein muß, und wird von einem nachfolgenden Doppelpunkt gekennzeichnet. Den konkreten Wert, den eine Marke symbolisiert, berechnet der Assembler erst bei der Übersetzung. Hier zeigt sich der Sinn symbolischer Adressen und Operanden.

Hätte man das Sprungziel gleich als Adresse angegeben, müßte bei Änderung möglicherweise an vielen Stellen ein neuer Wert stehen. So kann man etwas dazuschreiben oder wegstreichen, und der Assembler setzt an allen Stellen des Programms die neuen Sprungadressen ein. Nach dem Markenfeld folgt im Assemblerlisting die Mnemonik eines Befehls. Neben den auf den Prozessor bezogenen gibt es auch sogenannte Pseudobefehle, die nur die Arbeitsweise des Assemblers steuern und natürlich nicht in den Maschinencode zu übersetzen sind. Tabelle 2.23. zeigt die Pseudobefehle des Assemblers, im Monitorlisting kann man bei Unklarheiten den Gebrauch nachlesen.

Tabelle 2.23. Pseudobefehle des Assemblers

Pseudobefehl	Bedeutung
EQU	Wertzuweisung einmalig
DEF	Wertzuweisung mehrmalig
DB	Wertzuweisung für Bytes
DA	Wertzuweisung für Adressen

Pseudobefehl	**Bedeutung**
BER	Reservierung eines Speicherbereiche!
ORG	Wertzuweisung für die Adresse des folgenden Programms
*	aktueller Adressenwert
PN	Programmname, Anfang eines Programms
END	Ende eines Programms

Die auf den Befehl häufig folgenden Operanden sind trotz der symbolischen Bezeichnung oft konkrete Adressen oder andere Zahlen. Sie können sowohl in hexadezimaler Kodierung als auch dezimal dargestellt werden. Im Gegensatz zu den Marken und anderen symbolischen Werten muß ihr erstes Zeichen eine Ziffer sein. Bei hexadezimalen Zahlen, die mit einem Buchstaben beginnen, stellt man die Ziffer 0 davor.

2.5.3. Bedienung des Assemblers

Das vom Assembler zu bearbeitende Quellprogramm kann sowohl im RAM stehen als auch von einem externen Gerät über den Inputkanal in den Computer gelangen. Ist kein externes Gerät zugewiesen (IT), kann das Programm nur mit Hilfe des Editors in den RAM gelangt sein, und der Assembler erwartet es auf den damit festgelegten Textpuffer-Adressen. Da der Assembler etwa 8k RAM benötigt, können dann nur kurze Programme bei der in Bild 3.1 gezeigten Aufteilung des Speichers bearbeitet werden (maximal ca. 1 k Objektcode). Die Übersetzung längerer Programme ist zwar möglich, jedoch müssen dann 2 externe Geräte am Computer arbeiten: Der Inputkanal muß das Quellprogramm liefern, Out- und Listkanal müssen den Objektcode und die Liste aufnehmen und speichern bzw. drucken. Ist als Listgerät der Bildschirm zugewiesen (LT), erscheint die Liste mit je 16 Zeilen. Nach Eingabe eines Leerzeichens folgen die nächsten 16. Der Maschinencode gelangt nach der Übersetzung auf die im Quellprogramm durch ORG-Befehl festgelegte Adresse, wenn kein externes Gerät dem Outputkanal zugewiesen ist. Dabei muß man darauf achten, daß nicht der Textpuffer oder der Arbeitsspeicher des Assemblers (1000H bis 2FFFH) überschrieben werden. Um sicherzugehen und auch für diese Adreßbereiche Programme erstellen zu können, sollte die Ausgabe des Objektcodes bei längeren Programmen immer auf Kassetten erfolgen.

Die Übersetzung des Quellprogramms erfolgt in mindestens 2 Schritten. Zunächst muß der Assembler alle Befehle auf ihre Länge prüfen und damit für symbolische Adressen konkrete Werte berechnen. Erst in einem zweiten Durchlauf können diese Werte dann als Operanden bei den Befehlen eingesetzt werden. Nach dem Aufruf des Assemblers muß deshalb zunächst Pass 1 ablaufen. Erst danach kann mit Pass O das Objektprogramm oder mit Pass P die Liste entstehen. Bei Eingabe von Kassette oder einem anderen externen Gerät muß man das Quellprogramm also mindestens zweimal «abspielen». Selbstverständlich entstehen bei der Programmierung mit Assembler trotz aller Sorgfalt oft Fehler.

Tabelle 2.24. erläutert die Bedeutung der Fehlermeldungen des Assemblers.

Tabelle 2.24. Assembler-Fehlermeldungen

20	Fehlerhafter Programmname
21	Markenfehler (zu lang, Syntax)
22	Marke doppelt

24	Fehlerhalte Mnemonik
25	Operandenfehler
27	Zeichenkette zu lang
28	Symboltabelle voll
29	Relativsprung zu weit