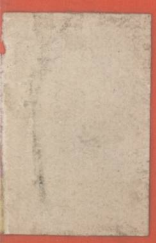
Horst Pelka

Mikrocomputer Programme

Werkzeuge und Hilfsmittel zu ihrer Erstellung



Horst Pelka

Mikrocomputer-Programme

Werkzeuge und Hilfsmittel zu ihrer Erstellung

mit 33 Abbildungen

Carl Hanser Verlag München Wien

Oberingenieur *Horst Pelka*
ist in einem Industrieunternehmen
auf dem Gebiet der Mikrocomputer tätig.

CIP-Kurztitelaufnahme der Deutschen Bibliothek

Pelka, Horst:
Mikrocomputer-Programme : Werkzeuge u. Hilfsmittel zu ihrer Erstellung / Horst Pelka.–
München ; Wien : Hanser, 1983
ISBN 3-446-13700-9.

Gesamtherstellung: Sellier Druck, Freising
Printed in Germany

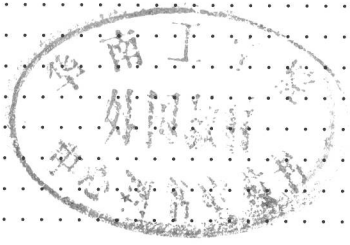
Vorwort

Heute gibt es kaum noch ein Gebiet der Elektronik, in dem nicht Mikrocomputer eingesetzt sind oder eingesetzt werden könnten. Die Hardware läßt sich in vielen Fällen als Standardware (Baugruppen-Systeme oder Einchip-µC) fertig kaufen. Der Entwicklungsingenieur sieht sich dann vor die Aufgabe gestellt, möglichst schnell und rationell die benötigte Software zu erstellen. Als Hilfen werden auf dem Markt eine breite Palette von Mikrocomputer-Entwicklungsplätzen angeboten. Wichtig ist jedoch, daß von vorn herein eine logische Planung und wenn möglich modulare Erstellung der Software vorgenommen wird.

Angeregt durch das unerwartet hohe Interesse an meiner Artikelserie „Werkzeuge und Hilfsmittel zur Erstellung von Mikrocomputer-Programmen" in der Zeitschrift Feinwerktechnik und Meßtechnik habe ich mich entschlossen, die Serie in diesem Buch zusammenzufassen. Es soll allen helfen, die sich in die Materie der Mikrocomputer-Software-Erstellung einarbeiten wollen. Bei dieser Gelegenheit möchte ich Herrn Professor Dr.-Ing. Joachim Heinzl, Lehrstuhl für Feingerätebau und Getriebelehre der Technischen Universität München für die Anregungen zu dieser Arbeit danken.

München, im April 1983
Horst Pelka

Inhaltsverzeichnis



1 Einleitung

Zur Kommunikation von Mensch und Computer sind viele Hilfsmittel bekannt. Leider ist es heute nicht möglich, daß der Mensch direkt mit einem Computer in Verbindung treten kann. Dazu müßte der Computer einerseits die menschliche Sprache verstehen können und andererseits das Ergebnis als Sprache wieder ausgeben. Wenn auch heute schon gewisse Redewendungen durch Sprachsynthese vom Computer ausgegeben werden, so ist man doch weit davon entfernt, daß sich Mensch und Maschine direkt verstehen können. Statt dessen muß der Mensch noch tief in die Sprache des Computers eindringen, um seine Wünsche wirkungsvoll an den Computer herantragen zu können.

Bild 1.1 zeigt die Skala der Verständigungsmittel zwischen Computer und Mensch. Wenn wir links beim Computer anfangen, so sind der Maschine nur direkte Zahlencodes verständlich. Diese Zahlencodes müssen z. B. bei einfachen Geräten, wie dem Experimental-Computer-Board ECB 85 bzw. dem Lern- und Übungsgerät Mikroset 8080 direkt eingegeben werden.

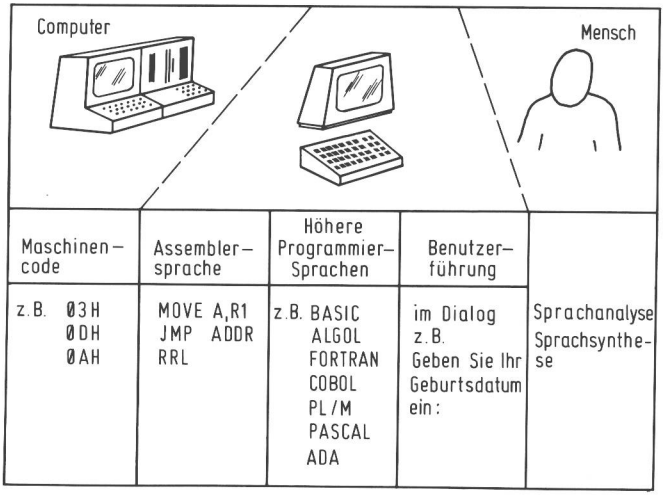


Bild 1.1: Skala der Verständigungsmittel zwischen Computer und Mensch

Wesentlich einfacher ist der memonische Code zu handhaben. Man nennt ihn auch Assemblersprache. Mittels eines Editierprogrammes wird ein Quellprogramm in der sogenannten Assemblersprache geschrieben. Es wird anschließend vom Assembler in den Maschinencode (Objektprogramm) und in ein sogenanntes „Listing" umgewandelt.

Noch bequemer hat es der Programmierer, welcher seine Programme in einer höheren Programmiersprache, wie BASIC, PL/M, PASCAL, FORTRAN, COBOL oder ADA schreibt.

Bei den bisher beschriebenen Hilfsmitteln ist eine spezielle Ausbildung des Programmierers notwendig. Diese Ausbildung umgeht man, indem die Programme des Computers soweit automatisiert werden, daß ein Direktdialog zwischen dem Benutzer und dem Computer stattfinden kann. Durch Ausgabe einer Aufforderung mit einem Doppelpunkt oder einem Fragezeichen veranlaßt der Computer den Benutzer irgendwelche Daten einzugeben. Man nennt diese Art daher Programme mit Benutzerführung. Hierbei ist für den Benutzer lediglich nur noch eine manuelle Fähigkeit zur Bedienung des Computers notwendig, nämlich die Bedienung der Tastatur, ähnlich wie bei einer Schreibmaschine.

Erst wenn der nächste Schritt gelingt, daß der Computer nämlich die menschliche Sprache versteht und die Ergebnisse in einer gewählten Sprache auch ausgibt, wäre eine zufriedenstellende Verständigungsmöglichkeit von Mensch zu Computer hergestellt.

Scherzweise würde das bedeuten, daß über ein computergesteuertes Telefon ein Fernsprechteilnehmer in Deutschland auf deutsch in das Telefon spricht und der Teilnehmer in England das Gespräch auf englisch übermittelt bekommt. Aber dies dürfte eigentlich noch weit in die Zukunft reichen.

Realistisch sind heute die drei Möglichkeiten, mit dem Computer in Assemblersprache, in einer höheren Programmiersprache und im Direktdialog per Tastatur zu verkehren.

2 Software

Unter Software versteht man alle zur Funktion eines Computers notwendigen Programme. Mittlerweile hat sich allgemein die Erkenntnis durchgesetzt, daß beim Einsatz von Mikrocomputern die Software der Hauptkostenfaktor bei der Realisierung von Systemen darstellt. Der Mikrocomputer wird in einer großen Stückzahl gefertigt und nach Liste verkauft. Die Intelligenz der Anlage steckt in der projektspezifischen Software. Das heißt, daß der Entwicklungsaufwand und das Entwicklungsrisiko in der Software steckt. Software kennt heute kaum mehr physikalische Grenzen. Es ist möglich, sehr komplexe und umfangreiche Programme und damit Anlagenfunktionen über das Programm zu implementieren.

Betrachtet man den Aufwand für die Erstellung eines Programmes in Abhängigkeit von dessen Umfang, so zeigt sich, daß der Aufwand überproportional mit dem Umfang steigt. Als Umfang bezeichnet man dabei die Anzahl der im Programm sichtbaren Anweisungen.

Ein weiterer wesentlicher Aspekt, der vor allem verstärkt bei Mikrocomputern eintritt, ist die Frage nach Sicherheit und Zuverlässigkeit der Programme.

Mikroprozessoren bieten heute schon die Möglichkeit der Verarbeitung höherer Programmiersprachen. Gerade diese sind ein Hilfsmittel, um erhöhte Zuverlässigkeit der Software und sinkende Kosten bei der Wartung zu erreichen. Die Wahl der am besten geeigneten höheren Programmiersprache zur Implementierung eines Problems ist abhängig vom Problem selbst und von der Art der zugrunde liegenden Hardware. Denn nicht alle Programmiersprachen stehen für alle Mikrocomputertypen zur Verfügung. Bei Ein-Chip-Mikrocomputern war es z. B. bis kürzlich noch üblich, diese in Assemblersprache zu programmieren. Neuerdings gibt es auch bei vereinzelten Typen Compiler für höhere Programmiersprachen (z. B. PL/M).

Bild 2.1 (siehe folgende Seite) zeigt den Weg eines Problemes vom Anwender über die Software-Entwicklung und zurück als Problemlösung (Softwareprodukt) zum Anwender.

Egal, ob in höherer Programmiersprache oder in Assemblersprache muß eine gute Systematik bei der Programmentwicklung eingehalten werden. In einem Dialog zwischen Anwender und Entwickler soll die Aufgabe klar herausgearbeitet werden und die Aufgabenstellung in einem Pflichtenheft eindeutig spezifiziert werden. Auch sollten die Randbedingungen wie Prozessor, Speicher, periphere Bausteine, Schnittstellen usw. geklärt sein.

Bereits vor der Designphase kann herausgefunden werden, ob es bereits ähnliche Entwicklungen gab. Hier muß entschieden werden, was billiger ist: Eine bestehende Lösung zu adaptieren oder noch einmal alles neu zu entwickeln. Wird zur ersteren Lösung gegriffen, sollte festgestellt werden, welche Schwierigkeiten aufgetreten waren und ob die Aufwandsabschätzung richtig war. So kann man sich von vornherein viel Ärger ersparen.

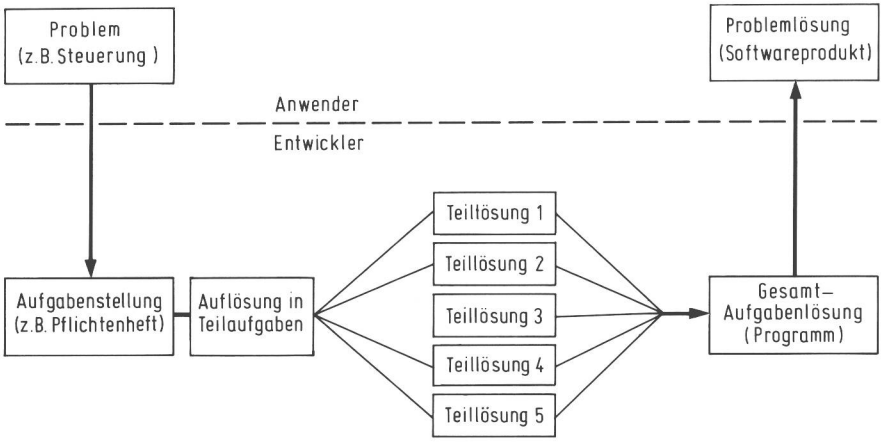


Bild 2.1: Vom Problem zum Software-Produkt

Auch wenn alles neu gemacht werden soll, ist die Kenntnis der für das ähnliche Problem verwendeten Hilfsmittel wie Entwicklungssystem, Programmiersprachen, etc. von großem Nutzen, um die bestehende Erfahrung mit einzubringen. Bei größeren Projekten ist eine wichtige Frage stets zu stellen: Ist das Projekt teilbar? Wenn ja, bedeutet es, daß die Software von vielen Entwicklern geschrieben werden kann und in einzelne Software-Module aufgeteilt werden muß. Bei jedem Zerlegungsschritt muß sichergestellt sein, daß die Lösung der Teilaufgaben auch die Lösung der Gesamtaufgabe beinhaltet und die festgelegte Folge der Teilhandlungen sinnvoll ist. Ebenfalls sollte bei jedem Zerlegungsschritt sichergestellt sein, daß er weitgehend autonome, lokal zu lösende parametrisierte Aufqaben enthält.

Wichtig ist, daß von vornherein sichergestellt ist, daß möglichst wenig logische Fehler unterlaufen, da diese erst relativ spät zu finden und schwer zu entdecken sind, weil übliche Compiler und Assembler nur syntaktische Fehler erkennen.

Diese Forderungen führen automatisch zur strukturierten Programmierung.

3 Strukturierte Programmierung

In den letzten Jahren hat die strukturierte Programmierung erheblich an Bedeutung gewonnen. Sie wurde zuerst von *Nassi/Shneidermann* [1] zur Diskussion gestellt. Durch den Einsatz von PASCAL, das zunehmend an Beliebtheit gewinnt, wird die strukturierte Programmierung für alle Arten von Anwendungen eingesetzt.

Die strukturierte Programmierung geht davon aus, tatsächlich nötige Sprünge nur in einem Modul zuzulassen, der problemlos überschaut werden kann. Dieser Modul muß einzeln testbar sein, ohne daß das Gesamtprogramm verhanden ist. Kennt man von jedem Modul genau das Verhalten gegenüber seinen Eingangsdaten, dann kann man die Moduln ohne Schwierigkeit in linearer Reihenfolge miteinander verknüpfen.

Die strukturierte Programmierung erlaubt folgende Programmbausteine:

Anweisung / Verzweigung / Schleife

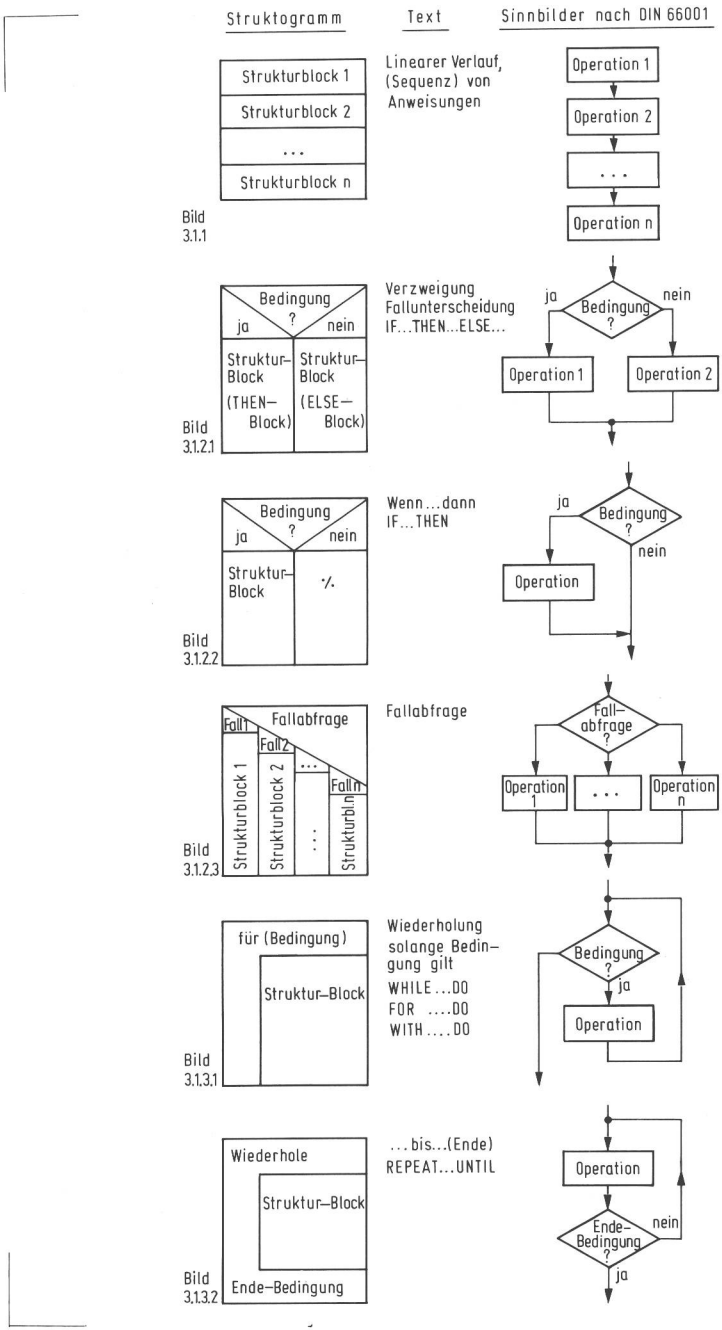
Die Vorteile der strukturierten Programmierung sind nicht von der Hand zu weisen. Im Verlauf der Programmierarbeit entsteht ein sehr übersichtliches Gebilde miteinander je einmal verknüpfbarer Anweisungsblöcke, die einzeln prüfbar sind. Dabei wird insbesondere bei größeren Programmen – verglichen mit klassischen Methoden – Zeit eingespart. Die Fehlerrate sinkt, die Pflege der Software wird vereinfacht. Insbesondere mit PASCAL sind Datenstrukturen möglich, die ebenso den kommerziellen wie den technisch wissenschaftlichen Problemstellungen angepaßt werden können.

3.1 Struktogramme

Nachstehend werden die möglichen Programmbausteine besprochen. Zur besseren Verständlichkeit sind rechts neben den Struktogrammen die bisher nach DIN 66001 üblichen Sinnbilder gezeichnet.

3.1.1 Anweisungen

Die einfachste Art eines Struktogrammes besteht aus einer Anzahl aufeinanderfolgender Strukturblöcke, die wiederum aus einer beliebigen Anzahl zu bearbeitender *Anweisungen* zusammengesetzt sind. Das Programm durchläuft linear alle Strukturblöcke in der vorgegebenen Richtung von oben nach unten. Ein weiteres Zusammenführen mehrerer Software-Bausteine ergibt neue Strukturblöcke (Bild 3.1.1).


Struktogramm
Text
Sinnbilder nach DIN 66001
Strukturblock 1
Strukturblock 2
...
Strukturblock n
Linearer Verlauf, (Sequenz) von Anweisungen
Operation 1
Operation 2
...
Operation n
Bild 3.1.1
Bedingung ?
ja
nein
Struktur-Block (THEN-Block)
Struktur-Block (ELSE-Block)
Verzweigung Fallunterscheidung IF...THEN...ELSE...
Operation 1
Operation 2
Bild 3.1.2.1
Wenn...dann IF...THEN
Operation
Bild 3.1.2.2
Fallabfrage
Fall1
Fall2
Falln
Strukturblock 1
Strukturblock 2
Strukturbl.n
Fall-abfrage ?
Operation 1
Operation n
Bild 3.1.2.3
für (Bedingung)
Struktur-Block
Wiederholung solange Bedingung gilt
WHILE...DO
FOR ....DO
WITH....DO
Operation
Bild 3.1.3.1
Wiederhole
...bis...(Ende)
REPEAT...UNTIL
Ende-Bedingung
Bild 3.1.3.2

3.1.2 Verzweigungen

Bedingte Sprungbefehle leiten den Arbeitsablauf abhängig von einer in einem Steuerblock angegebenen Bedingung weiter (Bild 3.1.2.1). Ein Steuerblock hat zwei Ausgänge: Einen für die erfüllte Bedingung und einen für die nicht erfüllte. Beiden Ausgängen können entweder unterschiedliche Strukturblöcke wie in Bild 3.1.2.1 folgen oder es folgt nur ein Strukturblock nach erfüllter Bedingung (Bild 3.1.2.2).

Eine weitere Art von Verzweigung ist die Fallabfrage (Bild 3.1.2.3). Während bei den früher üblichen Sinnbildern (rechts) mit einfachen Mitteln nur drei Fälle unterschieden werden konnten – sonst mußte man die Rauten kaskadieren – kann beim Struktogramm (links) eine beliebige Anzahl von Fällen unterschieden werden. Jedem Fall kann entweder ein Strukturblock ohne ein Leerfeld (keine Anweisungen) nachgeordnet sein.

3.1.3 Schleifen

Es gibt generell zwei Arten von Schleifen:

1. Die Zählschleife oder sukzessive Schleife, bei der die Anzahl der Wiederholungen von vorneherein festgelegt ist.
2. Die datenabhängige oder induktive Schleife, bei der das Verlassen der Schleife entweder von den Eingangsdaten oder den Verarbeitungsdaten abhängt.

Andererseits kann zwischen der Abfrage vor dem Strukturblock (Bild 3.1.3.1) oder nach dem Strukturblock (Bild 3.1.3.2) unterschieden werden. In den einzelnen Programmiersprachen sind dies die Schleifen nach den Befehlen WHILE ... DO, FOR ... DO, WITH ... DO bzw. REPEAT ... UNTIL ...

Bild 3.1.1: Struktogramm mit linearem Verlauf mehrerer sequentieller Anweisungen

Bild 3.1.2.1: Struktogramm für Verzweigung mit zwei nachfolgenden Strukturblöcken

Bild 3.1.2.2: Struktogramm für Verzweigung mit Strukturblock nach Bedingungserfüllung

Bild 3.1.2.3: Struktogramm für eine Fallabfrage

Bild 3.1.3.1: Struktogramm für Schleife mit Eingangsabfrage

Bild 3.1.3.2: Struktogramm für Schleife mit Abfrage der Ende-Bedingung

3.2 Struktogrammgenerator

Um die Erstellung von Struktogrammen zu automatisieren, wurden Hilfsmittel erstellt, die es erlauben, aus einem vorhandenen Programm oder einem erstellten Text mit dem Drucker ein Struktogramm zu zeichnen. Nachstehend wird der von der Siemens AG vertriebene Struktogrammgenerator STRUCT kurz erläutert. Das Programm ist auf einer Diskette und läßt sich durch Aufruf „STRUCT" in das *S*iemens *M*ikrocomputer *E*ntwicklungssystem SME laden. Es arbeitet mit Benutzerführung.

Nach Aufruf „STRUCT" meldet es sich mit:

[] STRUCT / NASSI-SHNEIDERMANN GENERATOR
[] VERSION 2.Ø
[] ENTER MODE (DIAGRAMM / EDIT / OFF)
[] (D/E/O)?

Zur Erstellung eines Diagrammes ist die Taste D, für Korrekturen die Taste E und zum Aussteigen aus dem Programm die Taste O zu drücken.

Beim Drücken der Taste D erscheint:

[] INPUT FROM CONSOLE OR FILE (C/F)?

Will man ein neues Struktogramm erstellen, ist die Console (C) zu wählen. Es erscheint die Frage:

[] HOLDING FILE =

Hier wird ein Name mit max. sechs Buchstaben vor einem Punkt und max. drei Buchstaben hinter dem Punkt eingegeben, z. B.

UEBUNG. ØØ1

und die Return-Taste gedrückt. Es erscheint:

[] MAX DIAGRAM WIDTH

Der Wert darf 2 < Breite < 133 sein.

Dann wird noch nach dem Versatz des vorderen Bildrandes gefragt:

[] STARTING COLUMN =

Hier geben wir beispielsweise 1Ø ein und drücken die Return-Taste. Es erscheint auf dem Bildschirm ein Feld und darunter:

!LEVEL = Ø! *BEGIN, *CASE, *FOR, IF, *REPEAT, *WHILE

Die erste Zeile darf also mit einem der sechs Worte beginnen. Wollen wir ein Struktogramm nach Bild 3.1.1 erstellen, so ist der Text nach Bild 3.2.1 a einzugeben und die Eingabe mit ESC abzuschließen.

Ist der Text bereits auf einer Diskette gespeichert, muß nach der Frage:

[] INPUT FROM CONSOLE OR FILE (C/F)?