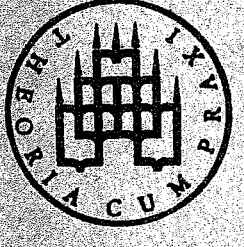
Applikation von FORTH



WILHELM-PIECK-UNIVERSITÄT
ROSTOCK

Interessengemeinschaft FORTH

Das moderne Programmierwerkzeug FORTH findet national eine staendig wachsende Beachtung. Um dem ansteigenden Informationsbeduerfnis Rechnung zu tragen, sowie den Kontakt zwischen Anwendern und System-implementierern zu verbessern, wurde durch die Kammer der Technik, Bezirksverband Suhl, Bezirksfachsektion Elektrotechnik/Elektronik eine

"Interessengemeinschaft FORTH"

ins Leben gerufen. Diese Interessengemeinschaft stellt sich unter anderem die Aufgaben,

- ueber FORTH-Grundlagenliteratur sowie Applikationsbeispiele zu informieren,
- Hinweise zu Eigenschaften und Bezugsmoeglichkeiten der in der DDR verfuegbaren FORTH-Systeme zu geben,
- den Erfahrungsaustausch zwischen FORTH-Anwendern zu foerdern
- sowie Erstanwender in der Einarbeitungsphase zu unterstuetzen.

Interessenten an Mitarbeit bzw. weiterer Information koennen ihre Wuensche, Vorstellungen und Angebote der Interessengemeinschaft FORTH, die unter der Anschrift

Kammer der Technik
Interessengemeinschaft FORTH
Postschliessfach 190
I L M E N A U
6 3 0 0

erreichbar ist, mitteilen.

A p p l i k a t i o n v o n F O R T H

Wilhelm-Pieck-Universitaet Rostock

Sektion Technische Elektronik

1987

Herausgeber: Der Rektor der Wilhelm-Pieck-Universitaet Rostock

Wissenschaftliche Leitung: Prof. Dr. sc. techn Karl Hormann

Herstellung der Druckvorlage: Dipl.-Ing. Egmont Woitzel
Ute Woitzel

Redaktionsschluss: 10. Februar 1987

Applikation von FORTH / Wilhelm-Pieck-Univ. Rostock, Sektion Techn. Elektronik. Wiss. Leitung: Karl Hormann. - Rostock, 1987. - 96 S.
ISBN 3-86009-022-4

Wilhelm-Pieck-Universitaet Rostock

Abteilung Wissenschaftspublizistik

Vogelsang 13/14, Rostock, DDR-2500, Telefon 369 577

Genehmigungs-Nr.: C 72/87

Druck: Ostsee-Druck Rostock, BT Ribnitz II-15-14 0,50

Applikation von FORTH

Inhalt

G e l e i t w o r t

Mit der vorliegenden Broschuere praesentiert der Wissenschaftsbereich Automatische Steuerungen an der Sektion Technische Elektronik der Wilhelm-Pieck-Universitaet Rostock einige Ergebnisse seiner Entwicklungen auf dem Gebiet der Programmiersprache FORTH. Das Wesen des Programmsystems comFORTH ueberzeugt durch seine breit angelegte Basissoftware und sein grosses Applikationsspektrum. So koennen neben Aufgaben aus dem Bereich der Prozessidentifikation und der optimalen und adaptiven Steuerung und Regelung auch Probleme der Prozessmesstechnik, der Stochastik und der unscharfen Systemtheorie behandelt werden.

Allgemein formuliert: Das comFORTH-Konzept besticht durch den hohen Grad an Modularitaet seiner Elemente, seine Flexibilitaet fuer den Anwender sowie seine Erweiterbarkeit. Obwohl es primaer in der 8-Bit-Rechentechnik erprobt wurde, ist eine Uebertragung der Ergebnisse auf die 16-Bit-Technik mit grosser Effizienz moeglich. Ich bin ueberzeugt, dass die vorgelegten Beitraege vielen Interessenten Denkanregungen bieten und den Weg zur umfassenden und zweckmaessigen Rechneranwendung ebnen.

Prof. Dr. sc. techn. Karl Hormann
Wissenschaftsbereichsleiter
WB Automatische Steuerungen

Applikation von FORTH

Egmont Woitzel; Ralf Neuthe

Basissystem comFORTH

Das Basissystem comFORTH ist ein erweitertes FORTH-Kernsystem nach dem figFORTH-Standard. In der vorliegenden Arbeit soll eine Vorstellung des Leistungsangebotes des Kernsystems erfolgen. Dazu wird die Handhabung der qualitativ neuen Moeglichkeiten des Basissystems anhand konkreter Beispiele demonstriert.

1. Das Programmierwerkzeug comFORTH

An der Sektion Technische Elektronik der Wilhelm-Pieck-Universitaet Rostock wird die Programmiersprache FORTH seit geraumer Zeit zur Programmierung von vorwiegend steuerungs- und regelungstechnischen Problemen eingesetzt. Die Auswahl der Sprache FORTH kann fuer diesen Anwendungsfall als guenstiger Kompromiss angesehen werden, da sie sowohl eine streng modulare Programmierung unterstuetzt als auch alle Abstraktionsniveaus zwischen Assemblersprache und beliebig hohen Fachsprachen zulaesst.

Um den vorwiegend anwendungsorientierten Programmierern eine genuegend komfortable Programmierumgebung zu schaffen, war es jedoch notwendig, ihnen ein Niveau zur Verfuegung zu stellen, das wesentlich ueber dem vom Standard angebotenen lag. Dies bezog sich einerseits auf die Leistungsfaehigkeit des Sprachkerns und andererseits auf die Verfuegbarkeit von Erweiterungsmoduln, die Standardfunktionen zur Programmentwicklung und zum Aufbau von Applikationen anbieten. Da die potentiellen Moeglichkeiten eines FORTH-Systems bereits durch die Leistungsfaehigkeit des Kerns festgelegt werden, wurde nach einer Analyse verfuegbarer Kernsysteme auf Grundlage der dabei gesammelten Erfahrungen das Basissystem comFORTH entwickelt.

Das Basissystem comFORTH stellt ein FORTH-Kernsystem dar, dessen Leistungsfaehigkeit gegenueber dem DDR de-facto-Standard figFORTH wesentlich gesteigert wurde. Zu den qualitativ neuen Eigenschaften des Kerns gehoeren unter anderem

- die Variabilitaet des Dialogsystems,
- die Erweiterbarkeit des Textinterpreters,
- die Moeglichkeit des temporaeren Ladens von Hilfsprogrammen,
- die logische Trennung von Anwender- und Dialogsystems sowie
- die flexible Ansteuerung der Peripheriegeraete.

Die Implementierung des Basissystems erfolgte unter minimaler Verletzung der Festlegungen des figFORTH-Standards, so dass das System comFORTH die Worte dieses Standards als Untermenge enthaelt. Ein weiteres wesentliches Konzept ist die strenge Anbindung des FORTH-Betriebssystems an die logische Schnittstelle des Betriebssystems CP/M, die notwendig ist, um insbesondere bei der Massenspeicherarbeit keine Konflikte mit anderen Systemkomponenten zu verursachen. Die fuer die Anwendung in der Prozessautomatisierung wichtige Voraussetzung der Unterbrechbarkeit aller Systembestandteile und Wiedereintrittsfaehigkeit der Routinen des Adressinterpreters wird von comFORTH erfuellt. Um den Anwenderprogrammierer von der Erweiterung des Compilers weitgehend zu entlasten, wurden bereits in das Basissystem

eine Anzahl von zusaetzlichen Direktiven aufgenommen, die die Strukturierung von Programmen und Daten verbessern.

Zur Arbeit mit dem Basissystem comFORTH wurde bereits eine grosse Anzahl von Erweiterungspaketen geschaffen, die die Programmierung und Testung von Anwendersystemen bereits auf einem hohen Niveau gestatten. Zu den verfuegbaren Programmen gehoeren:

Standardpakete:	- Modulkompressor zur Erzeugung von teilweise headerlosem Code
	- Assembler (U880, U881/82, U8000, i8086)
	- Locator fuer relativen Objektcode
	- Dateiarbeit fuer CP/M-Dateien
	- Stringverarbeitung auf eigenem Stapelspeicher
Arithmetikpakete:	- ueberlange Integerarithmetik
	- verschieden genaue Fliesspunktpakete
	- komplexe Fliesspunktarithmetik
	- Vektor- und Matrixalgebra
Entwicklungshilfen:	- Editoren verschiedenen Komforts
	- Debugwerkzeuge (diverse Anzeigen, Discompiler)
	- interaktiver Tracer
	- Crosscompiler fuer zugeschnittene Zielsysteme
Echtzeitsysteme:	- Multitaskkernsystem als Erweiterung
	- modifiziertes Basissystem comFORTH (rt) mit multitaskfaehigem Dialogsystem

Damit stellt das Programmsystem comFORTH dem System- und Anwendungsprogrammierer ein Umfeld zur Verfuegung, das eine effektive Loesung von Aufgaben eines breiten Spektrums ermoeglicht.

2. Die Anbindung an das CP/M-System

2.1. Grundlegende Konzepte

Die Anbindung eines FORTH-Systems an ein Betriebssystem bestimmt die Auslegung der folgenden Schnittstellen:

- den Aufruf des Systems aus der Betriebssystemumgebung,
- den Aufruf der Systemdienste von FORTH aus,
- die Massenspeicherverwaltung und
- die Hauptspeichereinteilung.

Das Betriebssystem CP/M, das wegen seiner grossen Verbreitung auf 8-Bit-Systemen als Umgebung fuer das zu implementierende Basissystem ausgewaehlt wurde, bietet fuer die Anbindung sehr einfach zu handhabende Schnittstellen an. Der Aufruf des Basissystems erfolgt als Kommandodatei vom CCP aus, wobei die Moeglichkeit der Parameteruebergabe genutzt wird.

Eine sehr wichtige Frage bei der Arbeit mit FORTH ist die Frage der Organisation des Massenspeichers. Da FORTH den Massenspeicher zunaechst rein physisch behandelt, d. h. dem Nutzer den gesamten verfuegbaren Massenspeicherraum direkt adressierbar zur Verfuegung stellt, muessen Organisationsformen gefunden werden, die eine organische Einordnung des FORTH-Massenspeicherraums in die Verwaltung von CP/M gestatten. Volle Kompatibilitaet der Massenspeicheranbindung wird erreicht, wenn dem FORTH-System als virtueller Speicherraum anstelle der Diskette bzw. Disketten nur eine Datei zur Verfuegung gestellt wird. Dies bedeutet, dass zum Aufruf der Systemdienste die BDOS-Schnittstelle verwendet wird. Im Interesse einer einheitlichen Schnittstelle wurde auch die Dialogperipherie an das BDOS angebunden.

Zur Vergabe des Speicherplatzes fuer Anwenderprogramme wird in CP/M nur ein extrem einfacher Mechanismus verwendet, da davon ausgegangen wird, dass sich in der Regel nur ein Applikationsprogramm im Speicher befinden muss. Alle Anwenderprogramme duerfen den Hauptspeicher von der Adresse 100H bis zu der Adresse benutzen, die auf der Speicherzelle 0006H verzeichnet ist. Durch comFORTH wird dies ausgenutzt, um allen verfuegbaren Speicherraum auch tatsaechlich fuer den maximalen Ausbau des Woerterbuchs nutzbar zu machen.

2.2. Das Dateikonzept von comFORTH

Wie bereits begruendet, stellt comFORTH dem Anwender virtuellen Speicherraum innerhalb einer Datei zur Verfuegung. Dieser virtuelle Speicher wird durch das Standard-Handler-Wort BLOCK voll kompatibel zu der unter figFORTH ueblichen physischen Anbindung verwaltet. Zur Speicherung von Quelltext wird die Datei in Screens eingeteilt, deren Zaehlung mit eins beginnt. Zusaetzlich dazu besteht jedoch die Moeglichkeit, waehrend der Arbeit unter FORTH die Arbeitsdatei zu wechseln.

Eine weitere Besonderheit von comFORTH ist die strikte Trennung von System und Anwenderprogramm in Bezug auf den Peripheriezugriff. Dies bedeutet, dass dem Dialogsystem (Betriebssystem) von FORTH eine eigene Datei zur Verfuegung steht, die unabhaengig von der gerade aktuellen Arbeitsdatei ist. Sie dient vorwiegend der Speicherung der Systemmeldungen, kann aber durch den Nutzer auch zur Aufnahme eigener Texte benutzt werden. Auch die aktive Systemdatei kann durch den Nutzer veraendert werden.

FORTH-Screen-Dateien werden zweckmaessigerweise mit dem Extend SCR gekennzeichnet, wogegen fuer die Systemdateien das Extend MSG verwendet werden sollte. Die vom System geforderten Extends koennen jedoch auch anders installiert werden.

Zur Eingabe von Dateinamen stellt comFORTH dem Nutzer einen Datentyp Filename zur Verfuegung, der durch den Textinterpreter vergleichbar den Integerzahlen unterstuetzt wird. Filenamen bestehen aus einem Laufwerksbezeichner d, einem Namen nnnnnnnn und einem Extend eee in der Form d:nnnnnnnn.eee, wobei im Gegensatz zur Arbeit unter dem CCP nur das Extend optional ist. Das momentan aktive Laufwerk wird durch das Symbol '@' markiert. Als Uebergabestelle fuer Dateinamen wird der Standard-Filecontroleblock von CP/M verwendet.

Auch die Umschaltung des aktiven Laufwerks ist ueber einen Datentyp des Textinterpreters moeglich. Die Syntax entspricht der CP/M-ueblichen. Durch die Einbindung der CP/M-typischen Erweiterungen als Datentypen des Textinterpreters wird einerseits die FORTH-uebliche umgekehrt polnische Notation auch fuer die Dateioperationen ermoeglicht und andererseits die Verkopplung des comFORTH-Kerns mit CP/M minimiert. Fuer die Anzeige des Inhaltsverzeichnisses des momentan aktive Laufwerks stellt comFORTH den Befehl DIR zur Verfuegung.

Eine kurze Beispielsitzung demonstriert die Anwendung der Befehle zur Dateiarbeit unter FORTH:

```
A:(cr)   ok
DIR(cr)
A:TRACY.SCR     A:ASM.SCR      A:COMFORTH.COM  A:DEBUG.SCR
A:ED.SCR        A:FCOPY.MSG    A:FCOPY.SCR     A:SCREDIT.COM
```

```
A:DEBUG.SCR    A:INSTALL.SCR   A:LOCATOR SCR   A:LOCATOR.MSG
A:FORTH.SCR    A:FORTH.MSG
 ok
B:(cr)  ok
DIR(cr)
B:DEMOS.SCR    B:GERMAN.MSG
 ok
```

Die momentan aktive Arbeitsdatei wird beim Listen eines Screen mit ausgegeben.

```
1 LIST(cr)
A:FORTH.SCR  Screen 1
 0
```

Mit Hilfe des Befehls USING kann sie umgeschaltet werden.

```
@:DEMOS USING(cr)   ok
1 LIST(cr)
B:DEMOS.SCR  Screen 1
 0 ( Bemerkungen                              Stand:  26/01/87  )
```

Das Extend SCR wird dabei durch USING automatisch angenommen. Als Laufwerk fuer die Datei wird das zur Zeit aktive Laufwerk B eingetragen. Ebenso ist die Systemdatei schaltbar. Ihr Inhalt ist z. B. fuer die ausgegebenen Fehlertexte relevant.

```
 .(cr) 46 .? parameterstack underflow
@:GERMAN MSG-USING(cr)   ok
 .(cr) 46 .? Stapelunterlauf
```

Die durch die Abarbeitung des Wortes . verursachte Ausgabe eines Integerwertes unter dem Stapelboden loest die Fehlermeldung mit der Nummer 1 aus. Die im Beispiel verwendete Datei GERMAN.MSG enthaelt deutsche Fehlertexte.

2.3. Der Start des comFORTH-Basissytems

Der Start des Basissystems erfolgt von CP/M aus durch den Aufruf der Kommandodatei COMFORTH.COM. Beim Aufruf koennen ihr die Namen der Anwender- bzw. Systemdatei uebermittelt werden, mit denen die Arbeit beginnen soll. Das Kommando besitzt folgendes Format:

COMFORTH [Anwenderdatei [Systemdatei]]

Die eckigen Klammern symbolisieren, dass die Systemdatei oder Systemdatei und Anwenderdatei im Aufruf entfallen duerfen, falls die Standarddateien verwendet werden sollen, fuer die das System installiert wurde. Dies sind im allgemeinen die Dateien FORTH.SCR und FORTH.MSG.

Bei einem Aufruf der Form:

B>a:comforth demos german

reagiert das System folgendermassen:

```
Z80  comFORTH 1.9
containing: standard commands
```

```
1 LIST(cr)
B:DEMOS.SCR  Screen 1
  0 ( Bemerkungen                                  Stand:  26/01/87  )
2 MESSAGE(cr) Woerterbuchueberlauf   ok
```

Auf einen Aufruf des Systems in der Form:

```
A>comforth
```

werden dagegen die Dateien FORTH.SCR und FORTH.MSG auf dem momentan aktiven Laufwerk A geoeffnet.

```
Z80  comFORTH 1.9
containing: standard commands
1 LIST(cr)
A:FORTH.SCR  Screen 1
  0
2 MESSAGE(cr) dictionary overflow (transient)   ok
```

Falls die Startdateien nicht gefunden werden, wird der Systemstart abgebrochen und ein Warmstart von CP/M ausgefuehrt.

3. Spracherweiterungen gegenueber figFORTH

3.1. Zur Notwendigkeit der Spracherweiterungen

FORTH ist eine semantisch offene Sprache. Dies bezieht sich auf Worte aller Klassen, also auch auf die des Compilers, d. h. der Worte, die zur Erzeugung von Worten sowie Programm- und Datenstrukturen eingesetzt werden. Aus der FORTH-Philosophie, die die Einfachheit als wichtigstes Prinzip enthaelt, folgt zwingend ein minimal ausgebauter Kern, der dann wiederum maximal in eine spezielle Richtung entwickelt werden kann. Dies fuehrt allerdings zu einem Widerspruch mit der Forderung der Anwender, bereits ueber einen gewissen Komfort verfuegen zu koennen, ohne vorher Systemprogrammierung im Sinne einer Compilererweiterung betrieben zu haben. Aus diesem Grund wurden eine Reihe von Spracherweiterungen in den Kern aufgenommen, die die vom Standard angebotenen Konstruktionen ergaenzen.

3.2. Neue Programmstrukturen

Die Programmstrukturen wurden um eine weitere Form der Selektion bereichert. In sehr vielen Faellen wird eine CASE- bzw. ORIF-aehnliche Struktur vermisst. Dies ist zum Beispiel bei der Programmierung von Menues der Fall. Unter comFORTH ist eine sehr verallgemeinerte Form verfuegbar, die SELECT-Struktur. Sie ist wie die IF-ELSE-ENDIF-Struktur ebenfalls nur innerhalb von :-Definitionen anwendbar.

Die SELECT-Struktur besteht aus einem Rahmen

```
SELECT   ...   ...   END-SELECT
```

in den drei verschiedene Austrittssequenzen eingefuegt werden koennen. Dies sind die Konstruktionen

```
CANCEL
IF_TRUE   ...          END-PATH
IF_EQUAL  ...          END-PATH
```

'CANCEL' wertet den uebergebenen Wert als Flag aus und verzweigt an das Ende der SELECT-Struktur, falls es logisch wahr ist. Ebenso arbeitet das IF_TRUE, nur dass vor dem Austritt die Passage abgearbeitet wird, die zwischen IF_TRUE und dem zugehoerigen END-PATH liegt. Das IF_EQUAL entspricht in seiner Funktion der Sequenz OVER = IF_TRUE DROP und wurde zur Verbesserung der Lesbarkeit eingefuehrt. Mit diesem Konstrukt koennen z. B. CASE-Konstruktionen realisiert werden.

Die Leistungsfaehigkeit der SELECT-Struktur wird durch die Einfuehrung einer zusaetzlichen Klausel, dem CARRY_ON weiter erhoeht. Durch sie kann der Rahmen in der Form

SELECT ... ... CARRY_ON ... END-SELECT

erweitert werden. Die Abbruchzweige werden hierbei nicht nach END-SELECT, sondern bereits nach CARRY_ON fortgesetzt, waehrend die normale Bearbeitung ohne Austritt durch eine der oben genannten Konstruktionen vor CARRY_ON abgebrochen und nach END-SELECT fortgesetzt wird. Dieser Steuerfluss laesst sich mit Hilfe von IF-ELSE-ENDIF-Konstruktionen nur bei Einfuehrung von Hilfsflags realisieren.

Beispiele fuer die Anwendung der SELECT-Struktur sind im Listing auf den Screens 13, 14 und 18 zu finden.

3.3. Moeglichkeiten zur Datenstrukturierung

Im FORTH-Sprachkern finden sich nur wenige Worte, die fuer die Behandlung von Daten vorgesehen sind. Der Nutzer erhaelt lediglich die Moeglichkeit, Datenfelder anzulegen. Fuer ihre Strukturierung ist er selbst verantwortlich. Dies gibt ihm zunaechst sehr viele Freiheiten. Allerdings wird er dabei auf bewaehrte Konzepte zurueckgreifen, die zumindestens als Ergaenzungspakete vorhanden sein muessen, wie z. B. Vektoren und Matrizen. Ebenso bewaehrt ist das Konzept der Records, das sich in praktisch allen modernen Hochsprachen finden laesst.

Durch Records koennen Datenfelder verschiedenen Typs zusammengefasst und als eine Einheit behandelt werden. Dabei koennen Records auch Datenfelder enthalten, die selbst wieder strukturiert sind, d. h. auch Realisierungen eines Records sind. Die Verwendung eines Record-Konstrukts bietet u. a. folgende Vorteile:

- Durch die symbolische Adressierung der Felder wird das Programm besser lesbar.
- Durch die Konzentration der Strukturbeschreibung ist das Programm sehr leicht aenderbar. Die Funktionen werden von der physischen Organisation der Daten unabhaengig.
- Da die Organisation hierarchisch erfolgen kann, ist es moeglich, die Daten in denselben Stufen zu strukturieren wie das zugehoerige Programmsystem.
- Da der Programmierer unabhaengig von der physischen Organisation programmieren kann, steigt seine Produktivitaet.

Aufgrund dieser offensichtlichen Vorteile und der haeufigen Anwendbarkeit wurde eine Record-Konstruktion mit in den comFORTH-Sprachkern aufgenommen. Ihre Anwendung soll im folgenden an einem groesseren Beispiel erlaeutert werden. Es handelt sich dabei um die Implementierung von Menues zur Ausgabe bzw. Veraenderung von Variableninhalten verschiedenen Typs.

Dabei wird ein Menue durch eine Liste von Eingabefeldern bestimmt. Ein Eingabefeld besteht aus einer Variablen, einer Eingabefunktion, einer Ausgabefunktion und einem kommentierenden Text. Dabei werden durch die Ein- und Ausgabefunktion und die Variable der Typ der Daten festgelegt. Die Eingabefelder des Menues koennen deshalb auch unterschiedliche Datentypen behandeln. Alle Felder des Menues werden in einer Liste zusammengefasst, so dass die Reihenfolge ihrer Bearbeitung festgelegt werden kann. Der Bediener soll sich mit Hilfe von Funktionstasten durch die Liste der Eingabefelder bewegen bzw. den Inhalt des aktuellen Eingabefeldes veraendern koennen.

Das Programm wird in drei Hierarchieebenen strukturiert, die sich auch in den Datenstrukturen wiederfinden. In der untersten Ebene werden die Funktionen zur Arbeit mit verketteten Listen und mit den Eingabefeldern definiert. Dem entsprechen die Records 'Zeiger' und 'Daten'. In der naechsten Hierarchieebene werden 'Zeiger' und 'Daten' als eine Einheit behandelt. Dem entspricht als Datenstruktur der Record 'Feld'. Als Funktion steht das Wort 'FELD' zur Definition und Initialisierung eines Eingabefeldes zur Verfuegung. In der obersten Hierarchieebene wird das gesamte Menue als eine Einheit behandelt. Sie wird durch die Funktion 'EINGEBEN' repraesentiert. Die korrelierende Datenstruktur ist die verkettete Liste der einzelnen Eingabefelder.

Im folgenden soll demonstriert werden, wie die Benutzung der Record-Konstruktion erfolgt. Dazu werden die entsprechenden Funktionen von der Beispieldatei ab Screen 4 benutzt. Die leichte Aenderbarkeit des Programms wird durch die Verwendung von zwei unterschiedlichen Varianten fuer die Implementierung der Liste demonstriert.

Auf Screen 9 der Beispieldatei befindet sich das Programm zur Implementierung einfach verketteten Listen. Als erste Definition befindet sich dort die Beschreibung des Datentyps 'Zeiger'. Dies erfolgt durch Anwendung der Record-Konstruktion. Sie besitzt folgende Syntax:

```
RECORD struc-typname  :=  feldname  OF typname
                      :=  feldname  OF typname
                          ...           ...        END-RECORD
```

'RECORD' definiert dabei ein Wort, das einen strukturierten Datentyp beschreibt, der aus beliebig vielen Feldern bestehen kann. Den Feldern selbst ist ebenfalls ein Typ zugeordnet, der entweder nicht strukturiert (atomar) ist oder ebenfalls strukuriert ist. Man kann deshalb zwischen struc-feldnamen und atom-feldnamen unterscheiden. Atomare Typen existieren bereits vordefiniert im Kern, koennen aber auch durch den Nutzer mit dem Wort 'ATOM' definiert werden. Der benutzte Typ 'ADDRESS' wurde beispielsweise im Kern wie folgt definiert:

```
2 ATOM ADDRESS
```

Die Integerzahl 2 gibt die Anzahl der fuer ein Feld benoetigten Bytes an. Damit kann man sich alle benoetigten atomaren Datentypen leicht selber definieren. Durch die strukturbeschreibenden Typworte, d. h. struc-typname oder atom-typname koennen Objekte erzeugt werden, die Repraesentanten der jeweiligen Typen sind. Dies geschieht einfach durch Anweisungen der Form:

```
atom-typname atom-objektname  bzw.  struc-typname struc-objektname
```

Man kann also mit dem Wort '{ZEIGER}' vom Screen 9 unseres Beispiels mehrere Zeigerobjekte definieren.

```
9 LOAD(cr)   ok
{ZEIGER} ZEIGER1(cr)   ok
{ZEIGER} ZEIGER2(cr)   ok
```

Auf die Datenfelder der Objekte kann durch Nennen der Namen der Objekte und der Felder zugegriffen werden. Bei hierarchisch organisierten Datentypen erfolgt der Zugriff auf ein Datenfeld in Reihenfolge der Typen solange, bis ein atomares Feld erreicht ist. Die allgemeine Syntax lautet:

```
struc-objektname [struc-feldname[...[struc-feldname]]] atom-feldname
```

Im Beispiel besitzt ein Objekt vom Typ '{ZEIGER}' nur ein einziges Feld, naemlich das Feld 'NACHFOLGER'. Seine Adresse erhaelt man durch die Anweisungen:

```
ZEIGER1 NACHFOLGER U. (cr) 36601   ok
```

Es ist auch moeglich, die Adresse eines strukturierten Feldes zu beschaffen. Dies erfolgt durch den Abbruch einer Feldliste mit '^':

```
struc-objektname [struc-feldname[...[struc-feldname]]] ^
```

Im weiteren sollen die Sequenzen

```
[struc-feldname[...[struc-feldname]]] atom-feldname        bzw.
[struc-feldname[...[struc-feldname]]] ^
```

mit 'feldliste' abgekuerzt werden. Mit Hilfe einer abgebrochenen Feldliste koennen z. B. die Adressen der Zeigerfelder unabhaengig von ihren Feldern ermittelt werden.

```
ZEIGER1 ^ U.(cr) 36601   ok
ZEIGER2 ^ U.(cr) 36619   ok
```

Diese Adresse ist fuer die Initialisierung mit dem Wort 'OZEIGER' zu uebergeben. Da alle Felder zu einem Ring verkettet werden sollen, muss bei einer einelementigen Liste das Zeigerfeld die eigene Adresse beinhalten. Die Funktionen zum Aufbau der Liste gehoeren bereits in diese Hierarchieebene, da sie von der konkreten Implementierung des Datentyps 'Zeiger' abhaengig sind.

```
OZEIGER1 ^ OZEIGER(cr)   ok
ZEIGER1 NACHFOLGER @ U.(cr) 36601   ok
ZEIGER2 ^ OZEIGER(cr)   ok
```

Mit dem Wort '>LISTE' kann ein Zeigerfeld als Nachfolger hinter einem anderen in die Liste eingeordnet werden.

```
ZEIGER1 ^ ZEIGER2 ^ >LISTE(cr)   ok
ZEIGER1 NACHFOLGER @ U.(cr) 36619   ok
ZEIGER2 NACHFOLGER @ U.(cr) 36601   ok
```

In den Nachfolger-Feldern der Zeiger befinden sich jetzt die Adressen der jeweiligen Nachfolgerobjekte. Wenn man mit einer so uebergebenen Basisadresse weiterarbeiten will, muss man die AT-Funktion benutzen. Sie verwendet folgende Syntax:

```
AT struc-typname feldliste
```

Da unsere Liste jetzt zwei Elemente besitzt, muss der Nachfolger des ersten Zeigers wiederum den ersten Zeiger als Nachfolger besitzen.

```
ZEIGER1 NACHFOLGER @ AT {ZEIGER} NACHFOLGER @ U.(cr) 36601  ok
```

Als zweite Moeglichkeit steht die Verwendung von zweifach verketteten Listen zur Verfuegung. Dazu muessen die Funktionen von Screen 10 geladen werden. Jetzt besitzen die Zeigerobjekte zwei Felder, naemlich fuer Vorgaenger und Nachfolger. Durch eine identische Syntax werden die Implementationsunterschiede gegenueber den hoeheren Hierarchieebenen verdeckt.

```
FORGET {ZEIGER}(cr)  ok
10 LOAD(cr)  ok
{ZEIGER} ZEIGER1(cr)  ok
ZEIGER1 ^ U.(cr) 36654  ok
ZEIGER1 NACHFOLGER U.(cr) 36654.  ok
ZEIGER1 VORGAENGER U.(cr) 36656  ok
```

Nach der Initialisierung zeigen sowohl das Vorgaenger als auch das Nachfolgerfeld auf die eigene Objektadresse.

```
ZEIGER1 ^ OZEIGER(cr)  ok
ZEIGER1 NACHFOLGER @ U.(cr) 36654  ok
ZEIGER1 VORGAENGER @ U.(cr) 36654  ok
```

Wie bereits oben vorgefuehrt, koennen mehrere Objekte definiert und in einer Liste zusammengefasst werden. Da die Liste nur zwei Elemente enthaelt, sind Vorgaenger und Nachfolger identisch.

```
{ZEIGER} ZEIGER2(cr)  ok
ZEIGER2 ^ U.(cr) 36674  ok
ZEIGER2 ^ OZEIGER(cr)  ok
ZEIGER1 ^ ZEIGER2 ^ >LISTE(cr)  ok
ZEIGER1 VORGAENGER @ U.(cr) 36674  ok
ZEIGER1 NACHFOLGER @ U.(cr) 36674  ok
```

Zur untersten Ebene gehoeren weiterhin die Manipulationen mit den einfachen Eingabefeldern, die sich ab Screen 6 auf der Beispieldatei befinden. Auch die Eingabefelder werden mit Hilfe der Record-Konstruktion beschrieben. Sie bestehen aus vier Feldern, die jeweils die Adresse der Eingaberoutine, Ausgaberoutine, Variablen bzw. des Textes enthalten. Durch die Abspeicherung der Adressen kann die Implementierung unabhaengig von den konkreten Datentypen erfolgen, die tatsaechlich benutzt werden.

In Bezug auf die Benutzung der Record-Konstruktion bietet die Implementierung der Datenfelder keine Besonderheiten. Datenfelder koennen mit Hilfe des Beschreibungsworts '{DATEN}' definiert werden. Zur Ein- und Ausgabe der Variableninhalte muessen Eingabe- und Ausgabefunktion definiert werden, die als Parameter die Adresse der Variablen uebermittelt bekommen. Im Fall von Integervariablen kann als Ausgabefunktion das Fragezeichen verwendet werden. Zur Vereinfachung der Definition von Textfeldern steht das Wort ':TEXT' zur Verfuegung. Die Initialisierung erfolgt mit dem Wort 'ODATEN'. Zur Illustration der Arbeit der einzelnen Worte wird an dieser Stelle ein Testprotokoll eingefuegt.

```
6 LOAD(cr)  ok
{DATEN} DATENFELD1(cr)  ok
: INT-EIN GET_INUM SWAP ! ;(cr)  ok
```

```
0 VARIABLE X(cr)   ok
X INT-EIN(cr) 33(cr)   ok
X ?(cr) 33   ok
:TEXT TEXT1 PROBETEXT FUER VARIABLE X : "(cr)   ok
TEXT1 COUNT TYPE(cr) PROBETEXT FUER VARIABLE X :   ok
TEXT1 X DATENFELD1 ^ ODATEN INT-EIN ?(cr)   ok
DATENFELD1 ^ .TEXT(cr)
PROBETEXT FUER VARIABLE X :   ok
DATENFELD1 ^ ANZEIGEN(cr)
PROBETEXT FUER VARIABLE X : 33   ok
DATENFELD1 ^ AENDERN(cr)
PROBETEXT FUER VARIABLE X : 25(cr)   ok
```

Als naechster Schritt erfolgt die Implementierung der naechsten Hierarchieebene, die Zeigerfeld und Datenfeld als neuen Datentyp zusammenfasst und sie als Einheit behandelt. Die Struktur des Datentyps wird mit Hilfe von 'RECORD' auf Screen 11 definiert. Da die Implementierung der Zeiger vollstaendig in der unteren Ebene verdeckt wird, ist der Quelltext dieser Ebene fuer einfach und doppelt verkettete Listen identisch. Wenn ohne die Record-Konstruktion gearbeitet worden waere, muesste an sehr vielen Stellen eine Korrektur der Adressrechnungen erfolgen, die die Wahrscheinlichkeit fuer Fehler stark erhoeht. In der Praxis haette dies meist zum Schreiben eines voellig neuen Programms gefuehrt. Im folgenden sollen die auf dieser Ebene neu verwendeten Befehle zur Arbeit mit Records erlaeutert werden.

```
11 LOAD(cr)   ok
{FELD} EINGABE1(cr)   ok
EINGABE1 ^ U.(cr) 36785   ok
TEXT1 X EINGABE1 ^ OFELD INT-EIN ?(cr)   ok
```

Das auf Screen 12 definierte Wort 'FELD' dient zur Definition und Initialisierung eines Feldes. Es enthaelt eine weitere neue Klausel, die ALLOCATE-Konstruktion mit der Syntax:

```
ALLOCATE typname
```

Mit Hilfe der Allocate-Konstruktion ist es moeglich, ohne Erzeugung eines Woerterbucheintrags am Ende des Woerterbuchs Platz fuer ein Objekt zu reservieren. Die mit dem Wort 'FELD' definierten Worte werden bei ihrer Ausfuehrung die Adresse eines Objektes vom Typ 'Feld' hinterlassen, ohne dass eine Feldliste folgen muss. Falls ein Zugriff auf einzelne Felder erfolgen soll, muss er mit Hilfe der AT-Konstruktion eingeleitet werden.

```
0 VARIABLE Y(cr)   ok
:TEXT TEXT2 TEXT FUER VARIABLE Y : "(cr)   ok
TEXT2 Y FELD EINGABE2 INT-EIN ?(cr)   ok
EINGABE2 U.(cr) 36812   ok
```

Die Verkettung der Felder realisiert das Wort 'FOLGT'.

```
EINGABE1 ^ EINGABE2 FOLGT(cr)   ok
```

Da ein Feldobjekt aus strukturierten Feldern besteht, ist der Zugriff auf ein atomares Feld mehrstufig.

```
EINGABE1 ZEIGER NACHFOLGER @(cr)   ok
```

Diese Sequenz liefert die Adresse des Feldes 'ZEIGER' des Nachfolgerfelds, da die Verkettung auf Ebene der Zeiger realisiert wird. Um aus

der Adresse eines Feldes die Adresse einer uebergeordneten Struktur oder eines anderen ihrer Felder zu berechnen, ist eine weitere Konstruktion erforderlich, die INSIDE-Konstruktion. Sie besitzt folgende Syntax:

INSIDE struc-typname FROM feldliste TO feldliste

Aus der Adresse des Zeigerfeldes kann deshalb die Adresse des Feldes folgendermassen bestimmt werden:

```
INSIDE {FELD} FROM ZEIGER ^ TO ^ U.(cr) 36812   ok
```

Zur Benutzung der Funktionen zur Behandlung der Daten muss die Adresse des Datenfeldes an sie uebergeben werden.

```
EINGABE2 AT {FELD} DATEN ^ ANZEIGEN(cr)_
TEXT FUER VARIABLE Y : 0   ok
```

Die dritte Hierarchieebene fasst mehrere Eingabefelder in einer Liste zusammen und stellt dem Nutzer eine Spezialsprache zur Definition und Benutzung eines bzw. mehrerer Menues zur Verfuegung. Die eigentliche Menuearbeit existiert in zwei Varianten. Da bei Verwendung zweifach verketteter Listen auch eine Rueckwaertsbewegung innerhalb der Liste moeglich ist, kann der Funktionsumfang des Menues groesser sein. Dazu braucht lediglich ein weiterer IF_EQUAL-Zweig in das einfache Menue mit aufgenommen werden, ohne dass die uebrigen Passagen geaendert werden muessen. In Bezug auf die Record-Konstruktion enthaelt diese Ebene keine weiteren Konstruktionen mehr. Deshalb soll an dieser Stelle durch das Benutzen des Beispielmenues von Screen 15 ein Eindruck von der Arbeitsweise des Programms vermittelt werden.

```
FORGET {ZEIGER} (cr)   ok
5 LOAD(cr)   ok
15 LOAD(cr)   ok
EINGEBEN(cr) (N)
double    y: 0 (N)
character z: A(N)
integer   x: 0 (E)
integer   x: 111(cr)_
integer   x: 111 (N)
double    y: 0 (E)
double    y: 222,(cr)_
double    y: 222 (N)
character z: A(E)
character z: 'x'(cr)_
character z: x(L)
double    y: 222 (L)
integer   x: 111 (L)
character z: x(X)   ok
```

3.4. Dynamische Daten- und Programmstrukturen

Zur Unterstuetzung dynamischer Organisationsformen fuer Programme und Daten wurde in den comFORTH-Kern eine Konstruktion aufgenommen, die es gestattet, spezielle einfach verkettete Listen zu definieren. Dabei wird aehnlich wie bei der Record-Konstruktion zunaechst ein Listentyp definiert, der dazu benutzt werden kann, Repraesentanten dieses Typs zu definieren.

Ausgangsueberlegung bei der Implementierung war dabei, dass die Aktionen, wie das Durchsuchen einer Liste, ihre Verlaengerung oder Verkuerzung unabhaengig von den Inhalten der Liste erfolgt. Damit koennen grosse Programmanteile fuer alle Listentypen gemeinsam benutzt werden. Die fuer die Listentypen verschiedenen Aktionen bestehen lediglich im Anlegen eines neuen Datenfeldes, in seiner Ausgabe fuer den Dialog sowie in der durch das Listenwort selbst auszufuehrenden Aktion. Durch indirekte Ausfuehrung dieser Routinen ist es moeglich, eine einheitliche Syntax fuer alle Listen zu benutzen.

Die Definition eines Listentyps erfolgt durch das Wort 'LINLIST' unter Angabe der Worte zum Ausfuehren der Liste, zum Erweitern der Liste und zum Anzeigen eines Listenelements.

LINLIST listentypname ausfuehren erweitern anzeigen

Mit Hilfe des Worts 'listentypname' kann man Repraesentanten erzeugen.

listentypname listenname

Das Wort 'listenname' ist der Kopf einer einfach verketteten Liste. Bei dem Aufruf von 'listenname' wird das Wort 'ausfuehren' ausgefuehrt. Als Parameter erhaelt es die Adresse des Listenkopfes uebergeben. Mit den Worten 'EXTEND' und 'CUT' kann eine Liste vergleichbar einem Stapelspeicher erweitert bzw. verkuerzt werden. Das Wort '.LINLIST' kann dazu verwendet werden, den Inhalt einer Liste anzuzeigen. Die Syntax lautet:

```
EXTEND listenname
CUT listenname
.LINLIST listenname
```

Als ein spezieller Listentyp existiert im Kern bereits das Wort 'LORIF' (linked orif). Dieser Listentyp gestattet die Implementierung dynamisch veraenderbarer Programme. Die Elemente der Liste sind Verweise auf auszufuehrende FORTH-Worte, die im folgenden als Elementarroutinen bezeichnet werden sollen. Die Elementarroutinen einer Liste muessen eine standardisierte Parameteruebergabe besitzen. Dazu gehoert, dass sie als Ergebnis mindestens ein Flag liefern. Durch die LORIF-Liste werden nacheinander alle Elementarroutinen abgearbeitet, bis die erste als Ergebnis ein logisch wahres Flag liefert. Daraus ergibt sich logisch, dass die Ergebnisparameter im false-Fall als Eingangsparameter der danach abgearbeiteten Elementarroutinen dienen. Meist wird man festlegen, dass im false-Fall die Ergebnisparameter gleich den Eingangsparametern sein sollen. Die Flags der Elementarroutinen werden zur Steuerung der Listenaktion verwendet. Die Listenaktion selbst stellt dem aufrufenden Programm ebenfalls ein Flag zur Verfuegung, das angibt, ob eine Elementarroutine mit 'Erfolg' abgearbeitet wurde.

Im residenten Kern von comFORTH werden LORIF-Listen im Textinterpreter und im Fehlersystem verwendet. Damit wird erreicht, dass diese Bestandteile des Systems durch den Nutzer waehrend der Arbeit erweitert werden koennen. Im Abschnitt 5.3. wird die Erweiterung einer LORIF-Liste demonstriert.

4. Das Woerterbuchkonzept von comFORTH

Das Programmiersystem comFORTH besitzt zwei Woerterbuecher, die dem Nutzer zur Verfuegung stehen. Das Standardwoerterbuch ist in der Handhabung identisch mit dem Woerterbuch einfacher FORTH-Umgebungen.

Das transiente Woerterbuch dagegen ist ein fluechtiges Woerterbuch in comFORTH. Es befindet sich auf den hoechsten Adressen, die comFORTH belegt. Seine Groesse wird in der Boot-up Area festgelegt. Soll die Groesse des transienten Woerterbuchs geaendert werden, muss der Inhalt der entsprechenden Speicherzelle ueberschrieben und anschliessend ein Kaltstart zur Neukonfiguration des Memory-Map durchgefuehrt werden.

Zur Arbeit mit dem transienten Woerterbuch existieren 4 Kommandos: 'BEGIN-TRANS', 'END-TRANS', 'FORGET-TRANS' und 'EMPTY-TRANS'. Dabei dienen die ersten beiden Instruktionen zum Umschalten der Woerterbuecherbereiche, wogegen mit 'FORGET-TRANS' der letzte transient genutzte Bereich und mit 'EMPTY-TRANS' der gesamte transiente Bereich wieder freigegeben werden koennen.

Ein klassisches Anwendungsbeispiel ist die Verwendung eines transient geladenen FORTH-Assemblers. Er wird bekanntlich nur fuer die Assemblierung eines Programms benoetigt. Zur Laufzeit des Programmes ist er wieder entbehrlich und kann Platz fuer weitere Applikationen machen. Im folgenden Beispiel soll das nun demonstriert werden:

Als erstes muss man sich ueber die Groesse des zu ladenden Assemblers informieren. Dies ist in der Regel von Screen 1 der Quelldatei ablesbar. Der verwendete komfortable Assembler (Originalmnemonics) benoetigt 3884 Bytes im Woerterbuch. Da das System fuer die Arbeit mit dem transienten Dictionary sich dort auch einen Stack einrichtet, der aehnlich wie der Parameterstack gegen die Woerterbuchspitze laeuft, reserviert man etwas mehr Platz, damit es nicht zur Kollision kommt (wird durch eine Fehlermeldung angezeigt).

```
4500 112 +ORIGIN !(cr)   ok
FREEZE COLD(cr)
Z80  comFORTH 1.9
containing: standard commands
```

Nach dem Kaltstart meldet sich das System wieder. Jetzt liegt die neue Speicheraufteilung vor. Die Arbeit mit dem transienten Dictionary kann nun begonnen werden.

```
BEGIN-TRANS(cr)   ok
B:ASM80 USING 1 LOAD(cr)
wait ---  loading Z80 assembler
     ---  loading utility words
     ---  loading instruction mnemonics
     ---  loading load instruction
     ---  loading structuring words
     ---  Z80 assembler loaded   ok
```

Zur Demonstration einer Fehlerreaktionen wird noch eine Leerdefinition angefuegt.

```
: TEST ;(cr)   ok
VLIST(cr)
D19E TEST        C29B LABEL        C288 CODE        C272 ASSEMBLER
```

Die Kompilation ins transiente Woerterbuch wird nun beendet.

```
END-TRANS(cr)  ok
```

Transient definierte Worte sind weiterhin erreichbar und ausfuehrbar, aber nicht kompilierbar, da sie tilgbar bleiben muessen.

```
TEST(cr)  ok
: PROVOKATION TEST ;(cr) TEST? reference into transient area
```

Die mit dem Assembler zu uebersetzenden Primitivworte 'UNDER+', 'BOUNDS', 'NIP', '0,' und '0>' befinden sich auf Screen 2 und 3 der Demonstrationsdatei.

```
2 LOAD(cr)  ok
VLIST(cr)
660D 0>          65FF 0,          65F2 NIP          65DF BOUNDS
65C9 UNDER+      65B9 PROVOKATION D19E TEST         C29B LABEL
```

Nach der Assemblierung wird der Assembler nicht mehr gebraucht.

```
FORGET-TRANS(cr)  ok
VLIST(cr)
660D 0>          65FF 0,          65F2 NIP          65DF BOUNDS
65C9 UNDER+      65B9 PROVOKATION 6593 SESSION      627E (SESSION)
```

An dieser Stelle sind nur noch die ins Standardwoerterbuch kompilierten Definitionen verfuegbar.

```
0, D.(cr) 0  ok
ASSEMBLER(cr) ASSEMBLER? unknown word
```

Das transiente Dictionary kann auch wiederholt aktiviert werden. Es gelten im einzelnen folgende Regeln:

- 'BEGIN-TRANS' darf nicht waehrend des Transientmodes aufgerufen werden (keine Verschachtelung moeglich)
- 'END-TRANS' darf nur waehrend des Transientmodes aufgerufen werden
- 'FORGET-TRANS' und 'EMPTY-TRANS' sind waehrend des Transientmodes verboten

Jede Sequenz 'BEGIN-TRANS' ... 'END-TRANS' kann mit einmal 'FORGET-TRANS' nach den LIFO-Prinzip getilgt werden. Das folgende Beispiel soll das noch einmal verdeutlichen.

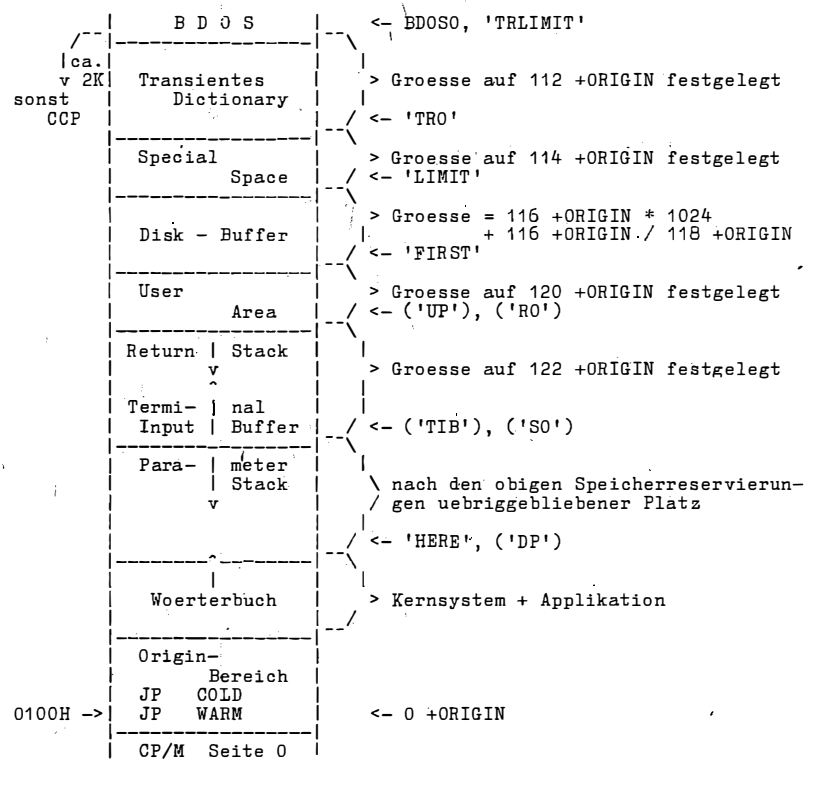
```
: S. ." Standard" ;(cr)  ok
: T. ." Transient" ;(cr)  ok
: S1 S. 1 . ;(cr)  ok
BEGIN-TRANS : T1 T. 1 . ; END-TRANS(cr)  ok
: S2 S. 2 . ;(cr)  ok
BEGIN-TRANS : T2 T. 2 . ; END-TRANS(cr)  ok
: Sx S. 'x' EMIT SPACE ;(cr)  ok
VLIST(cr)
6602 Sx          CC45 T2          65F3 S2          CC36 T1
65E4 S1          65CE T.
S1 S2 Sx(cr) Standard1 Standard2 Standardx  ok
T1 T2 (cr) Transient1 Transient2  ok
FORGET-TRANS(cr)  ok
VLIST(cr)
6602 Sx          65F3 S2          CC36 T1          65E4 S1
65CE T.
S1 S2 Sx(cr) Standard1 Standard2 Standardx  ok
```

```
T1 (cr) Transient1   ok
T2(cr) T2? unknown word
FORGET-TRANS(cr)   ok
VLIST(cr)
6602 Sx          65F3 S2          65E4 S1          65CE T.
S1 S2 Sx(cr) Standard1  Standard2  Standardx   ok
T1(cr) T1? unknown word
T2(cr) T2? unknown word
```

Zum Schluss sei noch erwaehnt, dass es Faelle gibt, in denen der transiente Bereich nicht freigegeben werden kann. Das ist immer dann so, wenn an ein transient kreiertes Vokabular weitere Definitionen im Standardwoerterbuch angefuegt wurden.

5. Die Speicheraufteilung innerhalb von comFORTH

In comFORTH wurde eine dynamische Speicheraufteilung organisiert, die sich an der Groesse der TPA orientiert. Diese Organisation laeuft waehrend des Kaltstarts ab. Als Bezugspunkt dient dabei die erste von BDOS belegte Adresse (mit BDOSO bezeichnet).



Im laufenden System sind der Sprung zum Warm- bzw. Kaltstart gegeneinander ausgetauscht, da bei einem eventuellem Wiederanlauf lediglich ein Warmstart ausgeloest werden soll. Das Kommando 'SAVE' macht diesen Tausch wieder rueckgaengig, um ein kaltstartfaehiges comFORTH im Speicher zu hinterlassen.

Damit ein spaeterer Warmstart auch wirklich gelingen kann, muss bei der Speicheraufteilung darauf geachtet werden, dass vom CCP (nach dem Verlassen von comFORTH) nur unwichtige Teile des Systems ueberschrieben werden koennen. Das waeren z.B. ein leeres transientes Dictionary, ein leerer Special-Space (soweit er noch betroffen wird) und die Disk-Buffer. Die User-Area darf um keinen Preis zerstoert worden sein.

Nachfolgend werden einige Richtlinien fuer die Groesse der einzelnen Bereiche des comFORTH-Systems angegeben, die bei einer Uminstallierung des Basissystems beachtet werden sollten.

Transientes Dictionary:

Die Groesse ist voellig frei waehlbar, d.h. wenn es nicht benoetigt wird, kann es entfallen (Groesse = 0).

Special Space:

Hier gilt das Gleiche wie fuer das transiente Dictionary. Der Special-Space wird z. B. vom Stringpaket verwendet.

Disk Buffer:

Es muss mindestens Platz fuer einen Screen reserviert werden. Als zweite Bedingung steht die Forderung nach mindestens zwei Disk-Bloekken, da sonst die Verwaltung nicht arbeiten kann. Ueblich ist die Anlage von zwei 1K grossen Bloecken. (Einige Programme wie z.B. der Screeneditior verlangen es sogar so.) Je mehr Bloecke vereinbart werden, um so seltener muss auf den Massenspeicher zugegriffen werden.

User Area:

Das comFORTH-Grundsystem benoetigt minimal 0E4H Byte fuer den Userbereich.

Returnstack und Terminal-Input-Buffer:

Als Mindestwert sei hier 80H angegeben. In einigen harten Faellen kann es aber noch zur Kollision kommen. Deshalb sei der Wert von 0A0H empfohlen.

Die Groessen des Transienten Woerterbuch, Special Space, User Area und Returnstack-TIB werden im ORIGIN-Bereich in der Masseinheit Byte gehalten. Die genauen Positionen innerhalb des Origin muessen der Systemdokumentation entnommen werden

6. Systemschnittstellen in comFORTH

6.1. Ein- Ausgabefunktionen

Die Dialogperipherie wird in comFORTH gegenueber figFORTH wesentlich komfortabler angebunden. Zwischen der eigentlichen CP/M-Schnittstelle und den FORTH-Schnittstellenworten befindet sich eine zusaetzliche Programmebene, die durch den Nutzer modifizierbar ist.

Die CP/M-Schnittstelle wird durch die Worte '(EMIT)', '(KEY)', '(?TERMINAL)' und '(PRINT)' realisiert, die den entsprechenden Aufruf des BDOS ausfuehren. Die Funktion '(?TERMINAL)' ruft dabei die Funktion Primitiv-Ein/Ausgabe auf, so dass der richtige Tastencode gelesen wird. Es muss allerdings dabei beachtet werden, dass dieses Zeichen dann auch bereits als ausgelesen gilt und nicht noch einmal gelesen werden kann.

Die FORTH-Schnittstelle besteht aus den Worten 'EMIT', 'KEY' und '?TERMINAL', die in bekannter Art und Weise verwendet werden koennen. Die Worte dieser Schnittstelle fuehren die Programme der Zwischenebene indirekt aus. Dazu stehen die Pointervariablen '<USER>', '<DLG>' und '<KEY>' zur Verfuegung. Aehnlich wie bei der Benutzung des Massenspeichers wird auch bei der Benutzung des Dialogausgabegeraets zwischen Dialog- und Anwendersystem unterschieden. Die Ausgaben des Anwenders werden an die Routine uebergeben, deren Codefeldadresse in '<USER>' eingetragen wurde, waehrend die des Systems ueber '<DLG>' ausgegeben werden.

Im Basissystem werden im Zwischenniveau im allgemeinen die Funktionen 'CONIN' und 'CONOUT' verwendet. Die 'CONIN'-Funktion wertet zusaetzlich zum Lesen der Tastatur Steuerzeichen aus. Dies sind:

```
^A Maskieren von Bit 7 bei Ausgabe ueber 'CONOUT' ein/ausschalten
^B Alternieren von Gross/Kleinbuchstaben          ein/ausschalten
^C Diskettensystem ruecksetzen
^P Parallen Druck bei Ausgabe ueber 'CONOUT'      ein/ausschalten
^[ Abbruch des bearbeiteten Programms zu 'QUIT'
```

Die Funktion 'CONOUT' stellt zusaetzlich zu den implizit bereits aufgefuehrten Funktionen Hardcopy und Maskierung in der Variablen 'OUT' die aktuelle Spaltennummer bereit.

Die Zwischenebene ist ohne Schwierigkeiten durch den Bediener veraenderbar. Zum Beispiel koennen alle Ausgaben des Anwenderprogramms durch folgende Sequenz auf den Drucker umgeleitet werden:

```
' (PRINT) CFA <USER> !(cr)  ok
```

Die Ausgaben des Dialogsystems, d. h. die Echos der Kommandoeingabe und das abschliessende 'ok' bzw. die Fehlermeldungen werden weiterhin auf dem vorher aktiven Ausgabegeraet erscheinen, waehrend alle anderen Ausgaben gedruckt werden.

Die Existenz dieser Schnittstelle bietet noch viele weitere Moeglichkeiten, wie beispielsweise die Ansteuerung eines fensterorientierten Ausgabesystems oder das Einlesen von Quelltexten, die mit einem herkoemmlichen Editor erstellt wurden, ueber die Tastaturschnittstelle.

6.2. Schnittstellen des Dialogsystems

Die Struktur des comFORTH-Dialogsystems entspricht der seines Vorbildsystems figFORTH. Es setzt sich aus drei Komponenten zusammen. Dies sind die Initialisierung, die Kommandobearbeitung und die Fehlerbehandlung. Diese Funktionen realisieren die Worte 'COLD', 'WARM', 'QUIT', 'ERROR' und 'MESSAGE'. Die Struktur des Dialogsystems zeigt Bild 1.

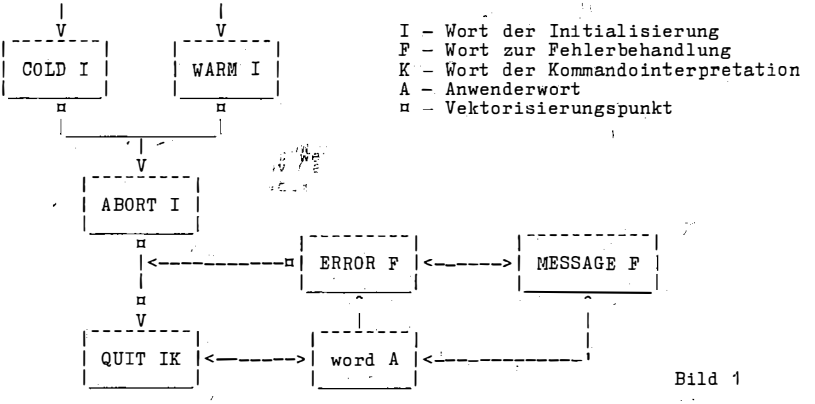


Bild 1

Der groessten Anteil der Initialisierung wird durch das Wort 'COLD' ausgefuehrt. Er beinhaltet unter anderem die Initialisierung des Dateisystems, die Einteilung des Memory-Maps, die Initialisierung des User-Bereichs und der residenten Vokabulare. Der Systemwarmstart 'WARM' initialisiert dagegen nur den Pufferspeicherbereich neu. Durch 'ABORT' und 'QUIT' werden weitere notwendige Initialisierungen durchgefuehrt, ohne die das System nicht lauffaehig ist. Dazu gehoert die Einstellung der interpretierenden Betriebsart und der Suchordnung des Textinterpreters sowie der Zahlenbasis. Durch 'QUIT' wird zusaetzlich der Returnstack reinitialisiert, um einen Ueberlauf bei laengerer Arbeit mit dem System zu vermeiden.

Das Fehlersystem von figFORTH enthaelt die Moeglichkeit entweder ueber das Wort 'ERROR' das ausloesende Wort abzubrechen oder ueber das Wort 'MESSAGE' lediglich eine Mitteilung an den Bediener auszugeben. Zur Ausgabe der Fehlerursache wird von 'ERROR' ebenfalls 'MESSAGE' benutzt.

Das eigentliche Bedienerinterface wird durch 'QUIT' repraesentiert, dass die Eingabe einer Kommandozeile fordert und diese dann an den Textinterpreter uebergibt. Der Textinterpreter ist die Schnittstelle zwischen dem Anwender- und dem Dialogsystem. Durch die Abgabe der Systemkontrolle an das vom Anwender aufgerufene Wort wird das Dialogsystem verlassen und in das Anwenderprogrammsystem uebergegangen. Dies wird unter anderem bei der Benutzung der Peripherie beruecksichtigt. Der Zustand des Systems kann aus der Variablen 'DLG' abgerufen werden, deren Inhalt durch den Nutzer allerdings nur mit grosser Vorsicht veraendert werden sollte. Das Anwendersystem gibt durch fehlerfreie Ausfuehrung des angegebenen Wortes bzw. den Aufruf eines Wortes des Dialogsystems die Steuerung wieder ab.

Bei der Implementierung eines anwenderspezifischen Dialogsystems unter figFORTH muss man feststellen, dass die Systemworte zur Initialisierung und Fehlerbehandlung in der Regel nicht verwendbar sind, da sie zum Aufruf des FORTH-Kommandointerpreters 'QUIT' fuehren. Dies ist besonders deshalb tragisch, weil bei einem Anwendersystem in der Regel:

- zusaetzliche Initialisierungen,
- zusaetzliche Fehlerbehandlungen und
- ein eigener Kommandointerpreter

notwendig sind. Besonders die Initialisierungen und Fehlerbehandlungen sind jedoch meist gegenueber denen, die fuer die Funktion des Basissystems notwendig sind, sehr kurz. Deshalb wurden in das Dialogsystem von comFORTH Vektorisierungspunkte aufgenommen, mit deren Hilfe das residente System um die zusaetzlichen Passagen erweitert werden kann. Die Vektorisierung wurde durch die Implementierung der zusaetzlichen Variablen 'USER-COLD', 'USER-WARM', 'USER-ABORT', 'USER-ERROR' und 'USER-QUIT' realisiert, die die Adressen der spezifischen Nutzerprogramme enthalten. Die Variablen korrespondieren mit den gleichnamigen Systemeintrittspunkten. Bis auf die Routine von 'USER-QUIT', die vor Ausfuehrung von 'QUIT' aufgerufen wird, werden alle anderen zusaetzlichen Routinen nach Ausfuehrung des korrespondierenden Wortes aufgerufen.

Die Benutzung dieser Moeglichkeiten soll daran demonstriert werden, wie einfach es ist, unter comFORTH alle Fehler zaehlen zu lassen, die man im Verlaufe einer Sitzung ausloest. Ein entsprechendes Programm ist auf Screen 20 der Beispieldatei zu finden. Die Variable '#ERRORS' enthaelt die aktuelle Anzahl an ausgefuehrten Systemfehlern. Das Wort '.ERRORS' inkrementiert deren Inhalt und gibt ihn als freundlichen Hinweis an den Bediener aus.

```
20 LOAD(cr)   ok
 .ERRORS(cr)
Bisher haben Sie schon 1 Fehler verursacht!!!   ok
```

Damit das Wort '.ERRORS' tatsaechlich nach jedem verursachten Fehler aufgerufen wird, ist es ueber die Pointervariable 'USER-ERROR' auszufuehren. Dazu ist seine Codefeldadresse dort einzutragen.

```
' .ERRORS CFA USER-ERROR !(cr)   ok
```

Damit reagiert das System jetzt auf einen vom Bediener verursachten Fehler wie folgt:

```
 .(cr) 46 .? parameterstack underflow
Bisher haben Sie schon 2 Fehler verursacht!!!
```

Die Pointervariablen werden beim Kaltstart des Systems aus Werten der Boot-up-Area voreingestellt, so dass das Nutzersystem bei entsprechender Installation bereits direkt nach dem Systemstart aktiv sein kann.

6.3. Erweiterbarkeit des Textinterpreters

Die erreichbare Qualitaet einer Mensch-Maschine-Schnittstelle wird bei der Verwendung eines FORTH-Systems im wesentlichen durch die Leistungsfaehigkeit des Interpreters bestimmt. Die bekannt gewordenen FORTH-Basissysteme (figFORTH, LMI-FORTH, polyFORTH, ZIP) verwenden

einen mehr oder weniger starren Interpreter, der im allgemeinen dadurch gekennzeichnet ist, dass ausser der Interpretation von Worten lediglich einfach- und/oder doppeltgenaue Integerzahlen verarbeitet werden koennen. Die Einbindung weiterer Datentypen, wie z. B. von Fliesspunktzahlen ist nur ueber Praefixworte moeglich, deren Parameteruebergabe eigentlich nicht FORTH-typisch ist.

Um die Implementierung einer leistungsfaehigen Mensch-Maschine-Schnittstelle zu ermoeglichen, wurde comFORTH mit einem erweiterbaren Textinterpreter ausgestattet, der die nachtraegliche Einfuehrung zusaetzlicher konvertierbarer Datentypen ermoeglicht. Die Erweiterbarkeit beruht auf der Verwendung einer LORIF-Konstruktion, deren Prinzip bereits unter Punkt 3.5. besprochen wurde.

Zum Verstaendnis des Konzepts der konvertierbaren Datentypen muss der Begriff 'Interpretation' als Aktion zur Selektion einer Zeichenkette aus dem Quelltextstrom, zu ihrer Klassifikation und zum nachfolgenden Aufruf eines typspezifischen Programms definiert werden. Zu einem konvertierbaren Datentyp gehoeren dann alle diejenigen selektierten Zeichenketten, deren Aufbau die entsprechenden Konventionen erfuellen. Fuer alle Zeichenketten eines Typs erfolgt der Aufruf des spezifischen Programms nach demselben Muster.

Nach dieser Definition sind auch FORTH-Worte ein eigener konvertierbarer Datentyp. Die Konvention fuer die Zugehoerigkeit einer Zeichenkette zu ihm besteht darin, dass diese bei einer Suche im Woerterbuch aufgefunden werden muss. Das aufzurufende Programm liegt in diesem Fall im Woerterbuch.

Im Fall der Integerzahlen besteht die Konvention darin, dass in der Zeichenkette nur gueltige Ziffernsymbole und gegebenenfalls ein negatives Vorzeichen enthalten sind. Die typspezifische Routine ist in diesem Fall fuer alle Repraesentanten identisch, naemlich die Konvertierung in eine Zahl und ihr Ablegen auf dem Stapel bzw. ihre Kompilation.

Durch das relativ einheitliche Grundmuster des Interpretationsprozesses ist es moeglich, den Rahmen der Interpretation von der Behandlung der einzelnen Typen abzuspalten und diese separat zu implementieren. Dazu wurde der comFORTH-Textinterpreter mit einer LORIF-Liste ausgestattet, die als Elementarroutinen die Tests auf jeweils einen konvertierbaren Datentyp enthaelt. Davon ist nur der Datentyp FORTH-Wort ausgenommen, da eine Zeichenkette stets als erstes auf Zugehoerigkeit zu diesem Typ getestet werden muss. Der Rahmen des Textinterpreter besteht deshalb lediglich aus der Selektion einer Zeichenkette aus dem Quelltextstrom, der Woerterbuchsuche und der nachfolgenden Behandlung als FORTH-Wort bzw. dem Aufruf der LORIF-Liste DATA?. Der Inhalt dieser Liste und damit die augenblickliche 'Intelligenz' des Systems kann sich der Bediener durch Anwendung der entsprechenden Listenbefehle anzeigen lassen.

```
 .LINLIST DATA?(cr)_
DRIVE                (CP/M-Laufwerksbezeichner   )
FILENAME             (CP/M-Dateinamen            )
DNUMBER              (doppeltgenaue Integerzahlen)
ASCII                (druckbare ASCII-Zeichen    )
CASCII               (ASCII-Steuerzeichen        )
INUMBER   ok         (einfachgenaue Integerzahlen)
```

Im folgenden Beispiel soll demonstriert werden, wie der Textinterpreter um einen anwenderspezifischen Datentyp erweitert werden kann. Das zugehoerige Programm ist den Screens 17 bis 19 zu entnehmen.

Der zu implementierende Datentyp 'Mark' soll es gestatten, Geldbetraege als doppeltgenaue Integerzahlen einzugeben, wobei das letzte Komma die Stellen der Pfennige einleitet. Da gegebenenfalls diese Stellen bei der Eingabe entfallen koennen sollen, muss eine von der Kommastelle abhaengige Skalierung erfolgen. Als Konvention fuer den Datentyp wird eine positve Integerzahl mit mindestens einem Komma und folgendem 'M' vereinbart. Zusaetzlich wird darauf geachtet, dass sich das letzte Komma an letzter, vor- oder vorvorletzter Stelle befindet.

Als erste Funktion der Implementierung des Datentyps wird eine Ausgabeoperation definiert, die eine doppeltgenaue Integerzahl in einer der Konvention entsprechenden Form ausgibt.

```
17 LOAD(cr)   ok
100,00 MARK.(cr) 100,00M   ok
```

Zum Test der korrekten Funktion der Test- und Konvertierungsroutine 'MARK' werden die Bedingungen simuliert, die beim Aufruf durch den Textinterpreter auftreten werden. Dieser uebergibt der Testroutine die Adresse einer Zeichenkette im Standardformat, d. h. die Zeichen folgen einem Zaehlbyte. Durch die Testroutine wird ein Flag uebergeben, das den Wert 'wahr' besitzt, falls die Konvertierung erfolgreich war.

```
: MARKTEST BL WORD HERE MARK IF ." mark: " MARK.(cr)
ELSE ." ungueltig " DROP ENDIF ;(cr)   ok
MARKTEST    100,(cr) ungueltig   ok
MARKTEST   100,M(cr) mark: 100,00M   ok
MARKTEST 100,00M(cr) mark: 100,00M   ok
MARKTEST 10,000M(cr) ungueltig   ok
```

Nachdem der Test erfolgreich bestanden wurde, kann die Behandlungsroutine an den Textinterpreter angebunden werden. Dies erfolgt mittels der 'EXTEND'-Anweisung.

```
EXTEND DATA? MARK(cr)   ok
MARK
DRIVE
FILENAME
DNUMBER
ASCII
CASCII
INUMBER   ok
```

Da der Datentyp 'Mark' jetzt integraler Bestandteil des Textinterpreters ist, koennen Geldwerte jetzt ohne Praefix direkt durch den Interpreter verarbeitet werden.

```
100,M  MARK.(cr) 100,00M   ok
```

Falls der Datentyp nicht weiter benutzt werden soll, laesst er sich mit Hilfe der 'CUT'-Anweisung auch wieder aus der Liste entfernen.

```
CUT DATA?(cr)   ok
100,0M(cr) 100,0M? unknown word
```

Die Moeglichkeit der Erweiterung der Datentypliste wird von verschiedenen Ergaenzungsprogrammen des comFORTH-Systems benutzt, so z. B. von

der Fliesspunktarithmetik und dem ueberlangen Integerpaket. Durch die bessere Strukturierung des Textinterpreters ergibt sich weiterhin die Moeglichkeit, die Konvertierungs- und Testroutinen auch ausserhalb des Interpreters im Anwendersystem zu nutzen. Ein interessantes Beispiel ist die Implementierung von Request-Funktionen zur Realisierung einer Dateneingabe vom Bediener. Unter comFORTH wird die Implementierung einer solchen Funktion bereits weitgehend unterstuetzt. Es muss lediglich die Codefeldadresse der Konvertierungsroutine in die Variable 'CONVERT', das Begrenzerzeichen der Zeichenkette, im allgemeinen das Leerzeichen, in die Variable 'DELIM' und die maximale Laenge der einzugebenden Zeichenkette in die Variable 'FLD' eingetragen werden, bevor das Wort 'INPUT' aufgerufen wird. Durch das Wort 'INPUT' wird der Bediener aufgefordert, eine Zeichenkette einzugeben, die durch die angegebene Typroutine konvertierbar ist. Falls die eingegebene Kette die Konventionen nicht erfuellt, wird die Eingabeaufforderung wiederholt. Auf Screen 19 ist mit dem Wort 'GET_MARK' ein Beispiel dafuer zu finden.

```
GET_MARK MARK.(cr)_100,0M(cr)  100,00M   ok
```

Mit diesen Betrachtungen ueber die Erweiterbarkeit des Dialogsystem soll die Vorstellung des Basissystems comFORTH abgeschlossen werden.

Verfasser: Dipl.-Ing. Egmont Woitzel
stud.-ing. Ralf Neuthe
Wilhelm-Pieck-Universitaet Rostock
Sektion Technische Elektronik
Albert-Einstein-Str. 2
Rostock 6
DDR - 2500

Literatur

/1/ Brodie, L.:
Starting FORTH
Englewood Cliffs NJ: Prentice-Hall Inc. 1981

/2/ Brodie, L.:
Thinking FORTH
Englewood Cliffs NJ: Prentice-Hall Inc. 1984

/3/ Loeliger, R. G.:
Threaded interpretive languages
Peterborough: BYTE BOOKS 1981

/4/ Neuthe, R.:
Modulkonstruktion fuer comFORTH
Grosser Beleg an der WPU Rostock, STE/AS 1986

/5/ Woitzel, E.:
comFORTH - Ein flexibles Programmierwerkzeug zur Prozessautomatisierung
Diplomarbeit an der WPU Rostock, STE/AS 1985

/6/ Woitzel, E.:
comFORTH - Das Programmierwerkzeug FORTH unter SCPX
Berlin: edv-Aspekte 6 (1986) Heft 4

/7/ Zech, R.:
Die Programmiersprache FORTH
Muenchen: Franzis-Verlag 1984

Demonstrationsbeispiele fuer comFORTH

```
Screen  1
0 ( Bemerkungen                                  Stand:  26/01/87  )
1 ;S
2 ============================================================
3         Demonstrationsbeispiele fuer comFORTH
4 ============================================================
5
6 Bearbeiter: E. Woitzel
7             R. Neuthe
8 begonnen:   23/01/87
9
10 Anzahl Screens:
11
12
13
14
15
```

```
Screen  2
0 ( Primitivdefinitionen                              26/01/87  )
1
2 CODE UNDER+ ( w1 16b w2 ===> w3 16b ; w3=w1+w2 )
3             ( in high level: "ROT + SWAP"  3.9 mal langs.  )
4             EXX, DE POP, BC POP, HL POP, DE HL ADD,
5             HL PUSH, BC PUSH, EXX,       NEXT # JP, END-CODE
6
7 CODE BOUNDS ( w1 w2 ===> w3 w1 ; w3=w1+w2 )
8             ( in high level: "OVER + SWAP" 5.1 mal langs.  )
9             HL POP, DE POP, DE HL ADD, HL PUSH,
10            DE PUSH,                      NEXT # JP, END-CODE
11
12 CODE NIP     ( 16b1 16b2 ===> 16b2 )
13              ( in high level: "SWAP DROP" 4.5 mal langsamer )
14              HL POP, DE POP,             HPUSH # JP, END-CODE
15 -->
```

```
Screen  3
0 ( Primitivdefinitionen Fortsetzung                  26/01/87  )
1
2 CODE 0, ( ===> 0 0 )
3         ( in high level: "0 0"   5.1 mal langsamer          )
4         (          oder: "0,"    7.1 mal langsamer          )
5         0 # HL LD, HL PUSH,             HPUSH # JP, END-CODE
6
7 CODE 0> ( n ===> ? )
8         ( in high level: "0 >"   7.8 mal langsamer          )
9         DE POP, DE DEC, 1 # HL LD, D A LD, A OR,
10        M IF,   HL DEC,  ENDIF,         HPUSH # JP, END-CODE
11
12
13
14
15
```

Demonstrationsbeispiele fuer comFORTH

```
Screen  4
0 ( Einfache Daten-Ein/Ausgabe                          26/01/87  )
1
2  6 LOAD ( Datenbehandlung )
3  9 LOAD ( einfach verkettete Listen )
4 11 LOAD ( Definition von Eingabefeldern )
5 13 LOAD ( einfaches Menue )
6
7
8
9
10
11
12
13
14
15
```

```
Screen  5
0 ( Erweiterte Daten-Ein/Ausgabe                        26/01/87  )
1
2  6 LOAD ( Datenbehandlung )
3 10 LOAD ( doppelt verkettete Listen )
4 11 LOAD ( Definition von Eingabefeldern )
5 14 LOAD ( erweitertes Menue )
6
7
8
9
10
11
12
13
14
15
```

```
Screen  6
0 ( Unterstuetzung Zeichenketten                        26/01/87  )
1
2   :TEXT  ( ib: name text" ; erzeugt Textvariable )
3          0 VARIABLE -2 ALLOT '" WORD HERE C@ 1+ ALLOT
4 -->
5
6
7
8
9
10
11
12
13
14
15
```

Demonstrationsbeispiele fuer comFORTH

```
Screen  7
0 ( Struktur Eingabedatum und Initialisierung      26/01/87  )
1
2 RECORD {DATEN} :=  EINGABE OF ADDRESS
3                :=  AUSGABE OF ADDRESS
4                :=  DATEN   OF ADDRESS
5                :=  TEXT    OF ADDRESS
6                END-RECORD
7
8 : ODATEN ( 'text 'var 'daten ===> ; ib: input output )
9          ( initialisiert Datenvariable )
10         [COMPILE] ' CFA OVER  AT {DATEN} EINGABE !
11         [COMPILE] ' CFA OVER  AT {DATEN} AUSGABE !
12                   SWAP OVER  AT {DATEN} DATEN   !
13                              AT {DATEN} TEXT    ! ;
14 -->
15
```

```
Screen  8
0 ( Manipulationen mit Daten                       26/01/87  )
1
2 : .TEXT    ( 'daten ===> ; output: text )
3            CR AT {DATEN} TEXT @ COUNT TYPE  ;
4
5 : ANZEIGEN ( 'daten ===> ; output: text daten )
6            DUP .TEXT
7            DUP AT {DATEN} DATEN   @ SWAP
8                AT {DATEN} AUSGABE @ EXECUTE ;
9
10 : AENDERN  ( 'daten ===> ; output: text ; input: daten )
11           DUP .TEXT
12           DUP AT {DATEN} DATEN   @ SWAP
13               AT {DATEN} EINGABE @ EXECUTE ;
14
15
```

```
Screen  9
0 ( Einfach verkettete Liste                       26/01/87  )
1
2 RECORD {ZEIGER} :=ACHFOLGER OF ADDRESS
3                  END-RECORD
4
5 : OZEIGER  ( 'zeiger ===> ; Initialisierung als Ring )
6            DUP AT {ZEIGER} NACHFOLGER ! ;
7
8 : >LISTE   ( 'vorgaenger 'zeiger ===> ; einfuegen )
9            OVER AT {ZEIGER} NACHFOLGER @
10           OVER AT {ZEIGER} NACHFOLGER !
11           SWAP AT {ZEIGER} NACHFOLGER ! ;
12
13
14
15
```

Demonstrationsbeispiele fuer comFORTH

```
Screen 10
 0 ( Doppelt verkettete Liste                              26/01/87  )
 1
 2 RECORD {ZEIGER} := NACHFOLGER OF ADDRESS
 3                 := VORGAENGER OF ADDRESS
 4                  END-RECORD
 5
 6 : OZEIGER  ( 'zeiger ===> ; Initialisierung als Ring )
 7          DUP DUP AT {ZEIGER} NACHFOLGER !
 8              DUP AT {ZEIGER} VORGAENGER ! ;
 9
10 : >LISTE   ( 'vorgaenger 'zeiger ===> ; einfuegen )
11            OVER AT {ZEIGER} NACHFOLGER @
12            2DUP AT {ZEIGER} VORGAENGER !
13            OVER AT {ZEIGER} NACHFOLGER !
14            2DUP AT {ZEIGER} VORGAENGER !
15            SWAP AT {ZEIGER} NACHFOLGER ! ;

Screen 11
 0 ( Rahmen-Struktur und Initialisierung                   26/01/87  )
 1
 2 RECORD {FELD} := ZEIGER OF {ZEIGER}
 3                := DATEN  OF {DATEN}
 4                 END-RECORD
 5
 6 : OFELD ( 'txt 'var 'feld ===> ; ib: input output )
 7         DUP AT {FELD} ZEIGER ^ OZEIGER
 8             AT {FELD} DATEN  ^ ODATEN  ;
 9
10 -->
11
12
13
14
15

Screen 12
 0 ( Definition Eingabefeld                                26/01/87  )
 1
 2 : FELD  ( 'txt 'var ===> ; ib: name input output )
 3         <BUILDS  HERE ALLOCATE {FELD} OFELD
 4          DOES>  ( ===> 'feld ) ;
 5
 6 : FOLGT  ( 'feld1 'feld2 ===> ; verbindet felder )
 7          SWAP AT {FELD} ZEIGER ^
 8          SWAP AT {FELD} ZEIGER ^ >LISTE ;
 9
10 0 VARIABLE FELD^ ( zeigt auf aktuelles Eingabefeld )
11
12
13
14
15
```

Z80 comFORTH 1.9 12/01/87 WPU Rostock STE/AS

Demonstrationsbeispiele fuer comFORTH

```
Screen 13
0 ( Einfaches Menue                                        26/01/87 )
1
2 : EINGEBEN ( ===> ; eingeben in Felder der Liste )
3    BEGIN  KEY
4       SELECT
5        'N' IF_EQUAL ( naechster ) FELD^ @
6            AT {FELD} ZEIGER NACHFOLGER @
7            INSIDE {FELD} FROM ZEIGER ^ TO ^  FELD^ ! END-PATH
8        'E' IF_EQUAL ( eingeben  ) FELD^ @
9            AT {FELD} DATEN ^ AENDERN                  END-PATH
10       'X' IF_EQUAL ( beenden   ) ;S                  END-PATH
11      DROP CARRY_ON
12           FELD^ @ AT {FELD} DATEN ^ ANZEIGEN
13     END-SELECT
14   AGAIN  ;
15

Screen 14
0 ( Erweitertes Menue                                      26/01/87 )
1
2 : EINGEBEN ( ===> ; eingeben in Felder der Liste )
3    BEGIN  KEY  SELECT
4         'N'  IF_EQUAL ( naechster ) FELD^ @
5              AT {FELD} ZEIGER NACHFOLGER @
6              INSIDE {FELD} FROM ZEIGER ^ TO ^ FELD^ ! END-PATH
7         'L'  IF_EQUAL ( letzter   ) FELD^ @
8              AT {FELD} ZEIGER VORGAENGER @
9              INSIDE {FELD} FROM ZEIGER ^ TO ^ FELD^ ! END-PATH
10        'E'  IF_EQUAL ( eingeben  ) FELD^ @
11             AT {FELD} DATEN ^ AENDERN                END-PATH
12        'X'  IF_EQUAL ( beenden   ) ;S                 END-PATH
13        DROP CARRY_ON
14             FELD^ @ AT {FELD} DATEN ^ ANZEIGEN
15             END-SELECT   AGAIN  ;

Screen 15
0 ( Eingabemenuebeispiel                                   26/01/87 )
1
2 : I-EINGABE  ( addr ===> ) GET_INUM SWAP ! ;
3 : D-EINGABE  ( addr ===> ) GET_DNUM ROT 2! ;
4 : D-AUSGABE  ( addr ===> ) 2@ D. ;
5 : C-EINGABE  ( addr ===> ) GET_CHAR SWAP ! ;
6 : C-AUSGABE  ( addr ===> ) @ EMIT ;
7
8 0 VARIABLE X  0, VARIABLE Y , 'A' VARIABLE Z
9
10 :TEXT TX integer   x: " :TEXT TY double    y: "
11 :TEXT TZ character z: "
12
13 -->
14
15
```

Demonstrationsbeispiele fuer comFORTH

```
Screen 16
0 ( Eingabemenuebeispiel - Fortsetzung                    26/01/87 )
1
2 TX X  FELD FELD1  I-EINGABE ?
3 TY Y  FELD FELD2  D-EINGABE D-AUSGABE
4 TZ Z  FELD FELD3  C-EINGABE C-AUSGABE
5
6 FELD1 FELD2 FOLGT
7 FELD2 FELD3 FOLGT           FELD1 FELD^ !
8
9
10
11
12
13
14
15
```

```
Screen 17
0 ( Datentyp Mark - Ausgabe                               26/01/87 )
1
2 : D*M ( d1 n ===> d2 )
3       OVER OVER XOR >R   ABS >R   DABS R U* DROP
4       SWAP R> U*   ROT +   R> D+-   ;
5
6 : MARK. ( mark ===> )
7         BASE @ >R DECIMAL   SWAP OVER DABS
8         <# # # ',' HOLD #S SIGN  #> TYPE ." M "
9         R> BASE ! ;
10
11 -->
12
13
14
15
```

```
Screen 18
0 ( Datentyp Mark - Test                                  26/01/87 )
1
2 : MARK? ( addr ===> wd ?t | addr ?f )
3         SELECT  DUP C@ OVER + C@ 'M' -                    CANCEL
4                 BASE @ >R DECIMAL
5                 0, 3 PICK COUNT 1- (DNUMBER) R> BASE !
6                 0=                    IF_TRUE  2DROP  END-PATH
7                 DPL @ 2 >             IF_TRUE  2DROP  END-PATH
8                 ROT DROP 2 DPL @ -
9                 BEGIN  DUP  WHILE 1-
10                       >R 10 D*M R>  REPEAT    DROP 1
11                CARRY_ON                              0
12          END-SELECT ;
13 -->
14
15
```

Demonstrationsbeispiele fuer comFORTH

```
Screen  19
 0 ( Datentyp Mark - Interpretation und Eingabe     26/01/87 )
 1
 2 : MARK ( interpret: addr ===> wd ?t | addr ?f )
 3        ( compile:   addr ===>    ?t | addr ?f )
 4        MARK? DUP
 5        IF  DROP [COMPILE] DLITERAL  1 ENDIF ;
 6
 7 : GET_MARK ( interpret: ===> wd )
 8            ( compile:   ===> )
 9            [ ' MARK CFA ] LITERAL CONVERT !
10            16 FLD ! BL DELIM ! INPUT ;
11
12
13
14
15

Screen  20
 0 ( Spezial-Fehlerbehandlung                      26/01/87  )
 1
 2 0 VARIABLE #ERRORS ( Zaehlvariable fuer Nutzerfehler )
 3
 4 : .ERRORS ( ===> ; Ausschrift der Fehleranzahl )
 5           [ USER-ERROR @ , ]   1 #ERRORS +!
 6           CR ." Bisher haben Sie schon " #ERRORS ?
 7           ." Fehler verursacht!!! " ;
 8
 9
10
11
12
13
14
15

Screen  21
 0
 1
 2
 3
 4
 5
 6
 7
 8
 9
10
11
12
13
14
15
```

Applikation von FORTH

Ralf Neuthe

Das comFORTH Stringpaket

In dieser Applikation soll das Stringpaket naeher beschrieben werden. Es werden das Implementierungskonzept, die existierenden Stringbefehle und ein Anwendungsbeispiel behandelt.

1. Einleitung

Die Programmiersprache FORTH unterstuetzt von sich aus die Zeichenkettenverarbeitung nur sehr schlecht. Es existieren zwar ein Standardformat fuer Strings und einige Befehle, die sich dessen bedienen, aber Befehle, die direkt auf die Verarbeitung von Strings ausgerichtet sind, gibt es nicht. Das wird besonders bei verschiedenen Implementierungen von Editoren sichtbar.

Deshalb wurde fuer comFORTH im Rahmen eines kleinen Beleges /1/ ein nachladbarer Befehlssatz geschaffen, der eigens nur auf die Verarbeitung von Zeichenketten ausgerichtet ist.

2. Das Implementierungskonzept

Die Zeichenkettenverarbeitung findet, typisch fuer FORTH, auf einem separaten Stack statt. Er wird in comFORTH im Special-Space eingerichtet, und zwar so, dass er von den hoeheren zu den niederen Adressen waechst. Die Strings liegen dort hintereinander im Standardstringformat.

Fuer diesen Stringstack gibt es syntaktisch aequivalente Befehle wie fuer den Parameterstack. Die Stringbefehle enthalten zur Kennzeichnung ihrer Spezifikation ein '"'-Zeichen (sprich: String-Zeichen). Die Kennzeichnung von Strings aus dem Eingabestrom wird ebenfalls durch ein fuehrendes '"' mit einem folgendem Leerzeichen vorgenommen. Das Ende wird auch durch '"' signalisiert.

Strings koennen sowohl waehrend des Execute- als auch waehrend des Compile-Modes entgegengenommen werden. In letzterem werden sie als In-Line-Literale compiliert und gelangen erst bei Ausfuehrung der Worte auf den Stringstack.

Im Einzelnen enthaelt das Stringpaket folgende Befehlsgruppen:

- Ein-/Ausgabefunktionen
- speicherbezogene Uebergabefunktionen
- adressbezogene Stringstackfunktionen
- Manipulationen mit Strings auf dem Stringstack
- Manipulationen von Strings
- Formatierungsfunktionen
- Konvertierungsfunktionen
    a) Datentyp in String
    b) String in Datentyp
- Definitionswoerter

3. Beschreibung der Funktionsgruppen

3.1. Ein-/Ausgabefunktionen

Zur Stringeingabe stehen die Funktionen '"' und 'GET_"' zur Verfuegung. Der Befehl '"' ist ein Immediate-Wort und die Benutzung erfolgt in der Form '" str"'. Der Befehl 'GET_"' erwartet vom Bediener die Eingabe eines Strings. Bei der Eingabe ist zu beachten, dass der String das Zeichen '"' nicht enthalten darf (ansonsten sind alle Zeichen zulaessig). Bei beiden Eingabefunktionen wird der String zum Stringstack uebergeben, wobei der Stringstackpointer entsprechend der Laenge des Strings weitergesetzt wird.

Zur Stringausgabe stehen die Funktionen '".' und '".S' zur Verfuegung. Mit '".' erfolgt eine Ausgabe des Strings mit Entfernung des Strings vom Top-of-"Stack, und mit '".S' erfolgt eine zerstoerungsfreie Anzeige des Stringstackinhaltes.

3.2. Speicherbezogene Uebergabefunktionen

Zur Stringuebergabe zwischen den verschiedenen Speicherbereichen stehen die Funktionen '"@' und '"!' sowie die Funktionen '("@)' und '("!)' zur Verfuegung. Die Stringbefehle '"@' und '"!' bewirken eine Stringuebergabe mit count-Byte und die Befehle '("@)' und '("!)' bewirken eine Stringuebergabe ohne count-Byte.

3.3. Adressbezogene Stringstackfunktionen

Zu den adressbezogenen Stringstackfunktionen gehoeren die Befehle '"P!', '"P@' und '"DEPTH'. Mit Aufruf von '"P!' erfolgt eine Neuinitialisierung des Stringstacks. Mit Aufruf von '"P@' erfolgt die Uebergabe der Adresse des Stringstackpointers auf dem TOS. Durch den Vergleich dieser Adresse mit der Adresse, die man durch den Aufruf von '"LIMIT' erhaelt, kann man z.B. die noch freie Stringstackapazitaet ermitteln. Mit '"DEPTH' wird die aktuelle Anzahl der Strings, die sich auf dem Stringstack befinden, auf dem TOS uebergeben.

3.4. Manipulationen mit Strings auf dem Stringstack

Zu dieser Funktionengruppe gehoeren die Stringbefehle '"DROP', '"DUP', '"SWAP', '"OVER', '"ROT', '"PICK', '"ROLL'. Die aufgefuehrten Stringbefehle entsprechen in ihrer Funktionsweise denen des Datenstacks.

Befindet sich beim Aufruf von '"PICK' bzw. '"ROLL' eine Zahl <1 auf dem TOS, dann erfolgt eine Fehlermeldung (Fehlerausschrift: impossible number).

3.5. Manipulation von Strings

Zu dieser Funktionengruppe gehoeren die Stringbefehle '"DEL' (DELete), '"/DEL', '"INS' (INSert), '"SPLIT', '"REP' (REPlace), '("LOC)' (LOCation), '"LOC', '"+', '"-', '"=', '"LEN' (LENgth).

Bei der Verwendung einiger dieser Stringbefehle ist die Zaehlweise der Stringpositionen innerhalb der Strings zu beachten.

Bild 1

Die Wirkungsweise der Stringbefehle wird im folgenden an jeweils einem Beispiel erlaeutert.

*****	Stringstack davor	:	Eingabe	:	*****	Stringstack danach
1	STRING	:	1 3 "DEL	:	1	SNG

Die Sequenz '1 3 "DEL' loescht aus dem String STRING ab der ersten Position (Bild 3) drei Zeichen heraus.

1	STRING	:	1 3 "/DEL	:	1	TRI

Die Sequenz '1 3 "/DEL' laesst von dem String STRING ab der ersten Position drei Stringcharakter stehen. Der Rest wird geloescht.

1	TRI	:		:		
2	SNG	:	1 "INS	:	1	STRING

Die Sequenz '1 "INS' bewirkt das Einfuegen des Strings TRI in den String SNG ab der ersten Position.

1	STRING	:	3 "SPLIT	:	1	ING
		:		:	2	STR

Die Sequenz '3 "SPLIT' bewirkt das Auftrennen des Ausgangsstrings in die beiden Teilstrings STR und ING und legt diese beiden Teilstrings in vertauschter Reihenfolge auf dem Stringstack ab.

1	TRA	:		:		
2	TRI	:		:		
3	STRING	:	"REP	:	1	STRANG

Befinden sich die drei Strings STRING TRI TRA auf dem Stringstack, dann bewirkt die Abarbeitung von '"REP' die Entfernung von TRI aus STRING und das Einfuegen von TRA ab der Anfangsposition von TRI. Der entfernte und der eingefuegte String koennen unterschiedliche Laenge besitzen.

Ist der zu entfernende String kein Teilstring des Ausgangsstrings erfolgt eine Fehlermeldung (Fehlerausschrift: isn't possible).

1	TRI	:		:		
2	STRING	:	("LOC)	:	1	TRI
		:		:	2	STRING

Der Befehl '("LOC)' bewirkt die Uebergabe der Startposition von TRI innerhalb von STRING auf dem TOS. In diesem Beispiel wird der Wert 1 uebergeben. Ist der zu lokalisierende String kein Teilstring des zu durchsuchenden Strings, erfolgt die Uebergabe des Wertes -1 auf dem

TOS.

```
-------------------------------------------------------------------
  1    TRI          :                          :
  2    STRING       :   "LOC                   :
```

Der Befehl '"LOC' hat die gleiche Funktion wie '("LOC)'. Im Unterschied zu '("LOC)' greift '"LOC' zerstoerend auf die beiden obersten Strings des Stringstacks zu.

```
-------------------------------------------------------------------
  1    ING          :                          :
  2    STR          :   "+                     :   1    STRING
```

Mit dem Befehl '"+' werden die beiden Strings ING und STR in vertauschter Reihenfolge zusammengesetzt.

```
-------------------------------------------------------------------
  1    TRI          :                          :
  2    STRING       :   "-                     :   1    SNG
```

Mit dem Befehl '"-' wird TRI aus STRING entfernt. Ist der zu entfernende String kein Teilstring des Ausgangsstrings, erfolgt eine Fehlermeldung (Fehlerausschrift: isn't possible).

```
-------------------------------------------------------------------
  1    STRING       :                          :
  2    STRING       :   "=                     :
```

Mit dem Befehl '"=' werden die beiden zuoberst auf dem Stringstack liegenden Strings verglichen und es wird ein Flag erzeugt, das in diesem Beispiel 'wahr' ist.

```
-------------------------------------------------------------------
  1   STRING        :   "LEN                   :   1   STRING
```

Mit dem Befehl '"LEN' wird die Anzahl der Stringcharakter des aktuellen Top-of-"Stack auf dem TOS uebergeben. In diesem Beispiel wird der Wert 6 uebergeben.

Fuer die aufgefuehrten Befehle '"INS', '"REP' und '"+' ist die Besonderheit zu beachten, dass es zum Ueberschreiten der maximal moeglichen Stringlaenge von 255 Stringcharaktern kommen kann. Entsteht bei der Benutzung einer dieser drei Befehle ein String mit einer Laenge von mehr als 255 Bytes, so wird dieser ohne Rueckmeldung nach 255 Stringcharaktern abgeschnitten.

3.6. Formatierungsfunktionen

Mit Hilfe der Stringbefehle '"LEFT', '"RIGHT' und '"MIDDLE' ist eine Formatierung des Strings, der sich auf dem Top-of-"Stack befindet, moeglich.

Die Wirkungsweise der Stringbefehle wird im folgenden an jeweils einem Beispiel erlaeutert.

```
*****  Stringstack  :        Eingabe        :  *****  Stringstack
          davor     :                       :           danach
====================:======================:====================
  1    STRING       :   3 "LEFT             :    1    STR
```

Die Sequenz '3 "LEFT' bewirkt die Uebergabe der drei linksseitigen Stringcharakter STR zum neuen Top-of-"Stack.

```
------------------:----------------------:------------------
  1    STRING      :  3 "RIGHT            :   1    ING
```

Die Sequenz '3 "RIGHT' bewirkt die Uebergabe der drei rechtsseitigen Stringcharakter ING zum neuen Top-of-"Stack.

```
------------------:----------------------:------------------
  1    STRING      :  3 "MIDDLE           :   1    RIN
```

Die Sequenz '3 "MIDDLE' bewirkt die Uebergabe der drei mittleren Stringcharakter RIN. An diesem Beispiel ist ersichtlich, dass die Formatierung bevorzugt rechtsseitig erfolgt, wenn die exakte Uebergabe der angegebenen Anzahl mittlerer Stringcharakter nicht moeglich ist.

Ist bei den angegebenen Formatierungsbefehlen die Anzahl der zu uebergebenen Stringcharakter groesser als die Stringlaenge, dann wird der String mit Leerzeichen ergaenzt.

3.7. Konvertierungsfunktionen

3.7.1. Konvertierung Datentyp in String

Zu dieser Funktionengruppe gehoeren die Befehle 'I>"', 'D>"', 'U>"', 'UD>"', 'FILE>"', und 'DR>"'(DRive). Mit diesen Stringbefehlen ist die Konvertierung von einfach-genauen Zahlen, doppelt-genauen Zahlen, einfach-genauen Absolutzahlen und doppelt-genauen Absolutzahlen in einen Ziffernstring moeglich, welcher zum Stringstack uebergeben wird.

Der Befehl 'FILE>"' konvertiert den momentanen Filenamen im STD-FCB in einen String und uebergibt ihn auf dem Stringstack. Der Befehl 'DR>"' konvertiert die momentane Laufwerknummer in einen String und uebergibt ihn auf dem Stringstack.

3.7.2. Konvertierung String in Datentyp

Zu dieser Funktionengruppe gehoeren die Befehle '">I', '">D', '">FILE' und '">DR'. Mit diesen Befehlen erfolgt die Konvertierung eines Strings auf dem Stringstack in eine einfach- bzw. doppelt-genaue Zahl auf den TOS. Wird bei der Konvertierung ein ungueltiges Ziffernsymbol angetroffen, dann erfolgt eine Fehlermeldung (Fehlerausschrift: isn't convertible).

Der Befehl ">FILE konvertiert einen String in einen Filenamen, der in den STD-FCB eingetragen wird. Entspricht der String nicht dem Format des STD-FCB, dann erfolgt eine Fehlermeldung (Fehlerausschrift: isn't convertible).

Der Befehl '">DR' konvertiert den String vom Top-of-"Stack in eine Laufwerknummer. Weiterhin erfolgt Aktuellsetzen der konvertierten Laufwerknummer. Ist der String nicht in eine Laufwerknummer konvertierbar, dann erfolgt eine Fehlermeldung (Fehlerausschrift: isn't convertible).

3.8. Definitionswoerter

Zu dieser Funktionengruppe gehoeren die beiden Woerter '"CONSTANT' und '"VARIABLE'.

Der Gebrauch von '"CONSTANT' entspricht sinngemaess der Benutzung von 'CONSTANT'. Die Sequenz '"CONSTANT name' bewirkt die Definition einer Stringkonstante, die den String vom Top-of-"Stack erhaelt (wo er dabei zerstoert wird). Bei Aufruf von 'name' wird der String wieder auf dem Stringstack uebergeben.

Der Gebrauch von '"VARIABLE' entspricht ebenfalls sinngemaess der Benutzung von 'VARIABLE'. Dieses Wort wird benutzt in der Form '"VARIABLE name', d.h. es wird ein Wort aufgebaut, welches mit 'name' aufgerufen werden kann. Nach der Definition haelt diese Variable einen String der Laenge 0. Die Sequenz 'name "!' bewirkt die Uebergabe des Strings vom Stringstack in die Stringvariable, und die Sequenz 'name "@' bewirkt die Uebergabe des Strings von der Stringvariablen zum Stringstack. Eine Stringvariable kann Strings bis Laenge = 255 Charakter (maximale Laenge in FORTH) halten.

4. Anwendungsbeispiel

4.1. Einschaetzung

Zur Verdeutlichung der Arbeitsweise des String-Paketes wurde als kleines Beispiel ein Zeileneditor programmiert. (Das Listing folgt im Anschluss an diesen Artikel.)

Bei dieser Arbeit wurde der Befehlssatz des Stringpaketes als eine gute Auswahl empfunden. Fuer die einzelnen Aufgaben fand sich immer ein entsprechender Stringbefehl. Dadurch konnte das Editorprogramm sehr kurz gestaltet werden.

Ein Problem gab es allerdings bei der Einfuegefunktion, da sie eine unregelmaessige Parameteruebergabe besitzt. Dieses wird vom Autor als eine Schwaeche angesehen, die bei einer spaeteren Ueberarbeitung unbedingt beseitigt werden muss. Eine weitere Schwaeche ist die zu haeufige Fehlergenerierung. In vielen Faellen waere ein Misserfolg (z. B. bei der Konvertierung) auch durch ein Flag anzeigbar. Das hat den Vorteil, dass in solchen Situationen die Fehlerreaktion vom uebergeordneten Programm bestimmt werden kann.

4.2. Das Beispielprogramm

Es wurden nur die Kernbefehle, die sich direkt mit der Handhabung von einzelnen Strings befassen, erstellt. Das mag zwar billig erscheinen, aber gerade sie machen in den Editoren den Loewenanteil aus. Im folgenden Text soll die Arbeitsweise des Editors kurz beschrieben werden.

Der zu editierende String (Screenzeile) befindet sich in einer String-Variablen ('"PUFFER'). Die Stelle des Kursors in diesem String wird in der Variablen 'KURSOR' gehalten. Bei der Anzeigefunktion ('.PUFFER') wird dieser Kursor ("¤") an der entsprechenden Stelle nur fuer die Ausgabe eingeblendet. Er befindet sich aber selber nicht direkt in dem zu editierenden String.

Die Aenderung der Kursorposition wird einfach durch Aenderung des Wertes der Variablen 'KURSOR' vorgenommen. Anschliessend wird der Wert mit dem Wort 'KORREKTUR' gegebenenfalls korrigiert (Einstellung auf

die jeweilige Aussenposition). Nach jeder Kursorverschiebung wird fuer eine bessere Uebersicht ueber die Aktionen der String mit der neuen Kursorposition ausgegeben.

Bei den Einfuege- bzw. Loeschfunktionen wird der String auf dem Stringstack an der Kursorposition geteilt. Dann werden speziell die entsprechenden Manipulationen vorgenommen. Anschliessend wird der String wieder zusammengefuegt und in den Puffer zurueckgeschrieben. Auch hiernach erfolgt wieder eine Ausgabe des neuen Pufferinhaltes.

Mit einem entsprechendem Rahmenprogramm koennte man somit auf einfache Weise Screens editieren.

4.3. Vorfuehrung

Um die Arbeitsweise zeigen zu koennen, wird dieser Editor auf das schon vorhandene Kernsystem mit String-Paket geladen.

```
B:EDITOR USING(cr)   ok
1 LOAD(cr)   ok
```

Dann muss der Stringpuffer geladen werden. (Diese Arbeit wuerde sonst das Rahmenprogramm uebernehmen.)

```
" Das Stringpaket wurde schon vorher geladen." "PUFFER "!(cr)   ok
 .PUFFER(cr)
¤Das Stringpaket wurde schon vorher geladen.  ok
```

Der Stringpuffer ist geladen. Das "¤"-Zeichen stellt die aktuelle Kursorposition dar.

Die folgenden Anweisungen dokumentieren sich selbst.

```
4 RECHTS(cr)
Das ¤Stringpaket wurde schon vorher geladen.  ok
11 LOESCHEN(cr)
Das ¤ wurde schon vorher geladen.  ok
" Beispiel" EINFUEGEN(cr)
Das Beispiel¤ wurde schon vorher geladen.  ok
" w" SUCHEN(cr)
Das Beispiel w¤urde schon vorher geladen.  ok
1 LOESCHEN(cr)
Das Beispiel w¤rde schon vorher geladen.  ok
" i" EINFUEGEN(cr)
Das Beispiel wi¤rde schon vorher geladen.  ok
2 RECHTS(cr)
Das Beispiel wird¤e schon vorher geladen.  ok
7 LOESCHEN(cr)
Das Beispiel wird¤ vorher geladen.  ok
" Bei" SUCHEN(cr)
Das Bei¤spiel wird vorher geladen.  ok
3 LINKS(cr)
Das ¤Beispiel wird vorher geladen.  ok
8 LOESCHEN(cr)
Das ¤ wird vorher geladen.  ok
" Stringpaket" EINFUEGEN(cr)
Das Stringpaket¤ wird vorher geladen.  ok
```

Dieses Beispiel zeigt, wie durch eine geeignete Wortwahl ein Programm sich selbst dokumentieren kann. Auf die Dauer kann aber die Laenge der Befehle stoeren, da sie haeufig benutzt werden. In der Regel werden deshalb kurze Namen (oft nur ein Buchstabe) verwendet. Durch haeufige Anwendung praegen sich solche Kuerzel auch ein.

Als zweite Moeglichkeit fuer eine Arbeitserleichterung bestuende die Verschluesselung der Befehle ueber die Funktionstasten fuer die Kursorsteuerung usw. Solche Aufgaben muesste das Rahmenprogramm uebernehmen.

Literatur

/1/ Farchmin, Udo
"Stringpaket fuer comFORTH"
Kleiner Beleg an der WPU Rostock 1986

Verfasser: stud. ing. Ralf Neuthe
Wilhelm-Pieck-Universitaet Rostock
Sektion Technische Elektronik
Albert-Einstein-Strasse 2
Rostock 6
DDR - 2500

Kleiner Zeileneditor als Beispiel

```
Screen  1
0 ( Neue Insert-Funktion mit erweitertem Bereich   26/01/87  )
1
2 : "EINFUEGEN  ( pos ===> | str1 str2 ---> str3             )
3   SELECT
4   DUP 1 <      IF_TRUE  DROP "SWAP "+  END-PATH
5   DUP "SWAP "LEN "SWAP < 0=
6               IF_TRUE  DROP       "+  END-PATH
7               "INS                    END-SELECT ;
8 -->
9
10
11
12
13
14
15
```

```
Screen  2
0 ( Kursor "Puffer Korrektur Ausgabe Rechts Links  26/01/87  )
1
2  0 VARIABLE KURSOR     "VARIABLE "PUFFER
3
4 : KORREKTUR  ( ===> ; haelt den Kursor im Bereich fest      )
5   KURSOR @  "PUFFER "@ "LEN "DROP MIN  0 MAX  KURSOR ! ;
6
7 : .PUFFER  ( ===> | ---> ; gibt den Pufferinhalt aus        )
8   "PUFFER "@    " ¤"  KURSOR @  "EINFUEGEN  CR ". ;
9
10 : RECHTS  ( n ===> ; bewegt den Kursor um n nach rechts     )
11   KURSOR +!  KORREKTUR  .PUFFER ;
12
13 : LINKS    ( n ===> ; bewegt den Kursor um n nach links     )
14   MINUS RECHTS ;
15 -->
```

```
Screen  3
0 ( Einfuegen Loeschen und Suchen                  27/01/87  )
1
2 : EINFUEGEN  ( ===> | str ---> ; fuegt str am Kursor ein    )
3   KURSOR @  "LEN KURSOR +!
4   "PUFFER "@   "SWAP  "EINFUEGEN  "PUFFER "!  .PUFFER ;
5
6 : LOESCHEN   ( n ===> | ---> ; loescht n Zeichen            )
7   "PUFFER "@  KURSOR @ SWAP "DEL  "PUFFER "!  .PUFFER ;
8
9 : SUCHEN     ( ===> | str ---> ; sucht str, setzt Kursor    )
10   "LEN "PUFFER "@ "SWAP "LOC  DUP 0<
11   IF    2DROP
12   ELSE  + KURSOR !  ENDIF .PUFFER ;
13
14
15
```

Z80 comFORTH 1.9 29/09/86 WPU Rostock STE/AS

Applikation von FORTH

Hans-Peter Dünow

Grafikanwendungen in comFORTH

Im Beitrag wird gezeigt, wie bestimmte comFORTH-Funktionen zur Erzeugung und Verwaltung von Displaylisten und zur grafischen Ausgabe genutzt werden koennen. Die Schaffung problemspezifischer Arbeitshilfen wird am Beispiel eines Grafikeditors zur Eingabe von Hintergrundgrafiken demonstriert.

1. Einleitung

Die Computergrafik spielt bei Betrachtungen ueber Rechnerapplikationen eine besondere Rolle, ist sie doch die z. Z. effektivste Methode bei der Gestaltung der Schnittstelle zwischen Computer und Bediener. Das Hauptproblem fuer den Anwender liegt gegenwaertig weniger bei der hardwaremaessigen Realisierung, als viel mehr bei der Erzeugung und rechentechnischen Verarbeitung der Bilddaten. Fuer jeden Anwender von Computergrafik stellt sich die Frage:

Wie und in welcher Form schaffe ich meine stark problemabhaengigen Daten auf den Bildschirm und in welcher Weise kann ich Manipulationen vornehmen, um eine moeglichst effektive Auswertung dieser Daten zu erreichen?

comFORTH bietet dafuer hervorragend geeignete Funktionen an. Einerseits werden durch die Maschinennaehe schnelle Manipulationen des Bildwiederholspeichers bzw. der Display- oder Datenlisten moeglich, was sich guenstig auf den bei der Computergrafik gewichtigen Zeitfaktor auswirkt. Ausserdem ist die Einbindung von Betriebssystemfunktionen sehr leicht moeglich. Andererseits ermoeglichen die Hochsprachelemente eine hohe Effektivitaet bei der Programmierung. comFORTH ist ein Programmierwerkzeug, mit dessen Hilfe man sich auf einfache Art und Weise auf bestimmte Anwendungsfaelle zugeschnittene Hilfsmittel fuer die Realisierung von Grafikfunktionen schaffen kann.

2. Hardwarehintergrund

Zunaechst soll kurz auf die verwendete Hardware eingegangen werden. Es wird eine Rastervollgrafik vorausgesetzt. Die verwendete Videosteuerung besitzt eine Aufloesung von 512*256 Bildpunkten. Durch einen auf der Steuerung installierten Prozessor werden bereits Grafikgrundroutinen realisiert, so dass die in FORTH programmierten Grundmodule sehr einfach gehalten werden koennen. Von den vielfaeltigen Funktionen, die von der Grafiksteuerung unterstuetzt werden, seien die im folgenden verwendeten kurz erlaeutert:

Das Datenformat hat immer die folgende Form:

```
code    Verschluesselte Grafikfunktion
a1      spezifische Informationen ( Koordinaten,Kennungen, ...)
a2
 ...
an
00      Endekennung
```

Funktionen:	code:	Spezifikation:	
Linie	82(H)	x1,y1,x2,y2	(Anfangspunkt,Endpunkt)
Pointer	94(H)	x1,y1	(Bildschirmposition)
Zeichen	84(H)	01, c,	(Anzahl,Zeichen)

Ein ASCII-Zeichen wird auf die eingestellte Pointerposition geschrieben. Weitere Funktionen sind z. B. Fuellen, Segmentloeschen, Interpolation, verschiedene Zeichengroessen und anderes mehr. Die Kopplung zwischen Rechner und Grafikst ueber ein PIO-Port realisiert, ueber welches Daten gesendet und empfangen werden koennen. Die Byteuebergabe wird im Beispiel durch ein Wort PP! ausgefuehrt. Die Grafikkarte arbeitet in den zwei Betriebsarten Grafik- und Normalmode, die jeweils durch das Wort UM eingeschaltet werden koennen.

3. Grundroutinen

Bei der Programmierung der anwendungsspezifischen Teile kann man sich zunaechst einen "Bausatz" moeglichst universell nutzbarer Grundmodule schaffen. Die hardware- und betriebssystemabhaengigen "Bausteine" sollten in einem geschlossenen Block untergebracht werden. Der Hauptteil des Programms wird dadurch von der Hardware unabhaengig. Wird neben einer Bildschirmgrafik z. B. ein Plotter als Ausgabegeraet verwendet, so muss nur ein Block von Grundbefehlen ausgetauscht werden.

Beispiel fuer Grundfunktion:

```
VEC ( y1,x1,y2,x2 --> ,Vektor ) 82 ( hex ) PP! H/L! H/L! H/L! H/L!
    0 PP! ;
```

Ist die Grafiksteuerung in den Grafikmode geschaltet und gibt man z. B. ein :

```
0  0  100 100 VEC(cr)  ok ,
```

so wird von der Grafik ein Vektor mit dem entsprechenden Anfangs- (0,0) und Endpunkt (100,100) auf den Bildschirm "gezeichnet". Der Nullpunkt liegt immer in der linken unteren Ecke des Bildschirms. Ausser VEC werden noch POS (y,x —>) und ZE1 (c -->) als Funktionen zur Ausgabe auf den Bildschirm und die Routinen -RE-, -LI-, -HO-, -RU- zur Erfassung und Abspeicherung der aktuellen Kursorposition aus einem Satz von Grundmodulen benutzt.

4. Beispiel: Grafikeditor

Der optische Effekt, den man mit den programmierten Grafikroutinen erreicht, laesst sich meist erst nach Testung einschaetzen, und es machen sich oft nachtraeglich Aenderungen erforderlich (Masstab, Aufteilung des Bildes, Groessenverhaeltnisse), d. h. die Erstellung von Computergrafik ist auch ein iterativer Prozess. Der Programmierer muss also nachtraeglich Veraenderungen an den Programmen vornehmen. Fuer eine solche Arbeitsweise bietet sich FORTH an. Als Beispiel fuer die Eingabe, Darstellung und Verwaltung von Displaylisten soll im folgenden ein kleiner Grafikeditor entworfen werden, mit dem man ueber die Tastatur gesteuert umfangreiche Hintergrundgrafiken fuer beliebige Anwendungsfaelle erzeugen kann.

4.1. Problemstellung

Wir wollen folgenden Fall konstruieren:

Fuer eine bestimmte Aufgabe sind viele verschiedene Hintergrundgrafiken zu erstellen, die in Form von Listen fest im Programm stehen und zu beliebigen Zeitpunkten darstellbar sein sollen. Der Begriff Hintergrundgrafik soll hier fuer das an einem Bild nicht zu veraendernde "Gerippe" stehen. Das koennen z. B. Signalflussplaene mit Beschriftung sein oder unterschiedliche Diagramme, die im Prozess nicht veraendert werden muessen. Auf diesen "Hintergrund" koennen dann, "wie auf einem Vordruck", die beweglichen oder veraenderlichen Funktionen gezeichnet werden (Funktionsverlaeufe, Zahlen). Die Hintergrundgrafiken sollen beliebig mischbar sein und muessen demzufolge auf dem Bildschirm verschiebbar sein . Der Programmierer hat jetzt zwei Alternativen: Entweder er setzt sich hin und erzeugt seine "Formulare" "zu Fuss", d. h. er bestimmt die Koordinaten mit Bleistift und Rasterpapier oder er programmiert sich ein Hilfsmittel, z. B. einen Grafikeditor.

4.2. Datenformate

Bevor die notwendigen Editorfunktionen bestimmt werden, muss man sich ueber das Datenformat im klaren sein. Der Editor soll als minimale Leistung die Eingabe von Linien und Text unterstuetzen. Ausserdem sollen Punkte fuer variablen Text festgehalten werden. Die Algorithmen fuer die Verwaltung der Displayliste werden besonders einfach, wenn das Datenformat von gleichmaessiger Struktur ist. Ausserdem wird Speicherplatz eingespart, da notwendige Kennungen zur Unterscheidung der Funktionen entfallen koennen. Aus diesen Gruenden sollen fuer Linien- und Textdaten unterschiedliche Listen gefuehrt werden.

Liste fuer Linienstrukturen:

Linienfolgen werden auf dem Papier erzeugt, indem der Stift an bestimmten Punkten abgesenkt und an anderen angehoben wird. Diese Technik wird hier einfach uebernommen und es ergibt sich daraus das Datenformat. Zunaechst ist eine Kennung erforderlich, die anzeigt, ob der "Stift" beim Anfahren einer Position abgesenkt oder angehoben wird. Zur Festlegung der Position ist ein Wert fuer die x-Richtung und ein Wert fuer die y-Richtung erforderlich. Am Anfang der Liste wird die Anzahl der Positionen eingetragen. Damit kann die Struktur der Liste angegeben werden:

Anzahl	(2 Byte)
Kennung 1	(1 Byte)
x1	(2 Byte)
y1	(2 Byte)
Kennung 2	

Liste fuer Strings:

Da die Laenge eines Strings stark variieren kann, ist die Struktur dieser Liste nicht so gleichmaessig. Es wurde folgendes Format festgelegt:

```
Anzahl der Strings                    (2 Byte)
Anzahl der Zeichen im String 1        (2 Byte)
x1      Startpunkt                    (2 Byte)
y1                                    (2 Byte)
Zeichen 1                             (1 Byte)
...
Anzahl der Zeichen im String 2          u.s.w.
```

Liste fuer variablen Text:

Fuer variablen Text werden nur Positionen eingegeben, die jeweils den Startwert eines variablen Textes angeben. Damit genuegt folgendes Format:

```
Startadresse
x1              (2 Byte)
y1              (2 Byte)
x2
y2          u.s.w.
```

Fuer alle drei Listen wird ein Grafikpuffer benoetigt, in dem die Daten bis zur Weiterverarbeitung abgelegt werden. Dieser Puffer beginnt ab PAD und ist dadurch verschieblich. Die Startadressen fuer die Listen werden in den Variablen (START), (STARTX) und (STARTV) festgehalten. Alle Funktionen zur Verwaltung der Listen beziehen sich auf diese Variablen. Da die editierten Grafikelemente auf dem Bildschirm verschiebbar sein sollen, sind alle Listeneintraege relativ zu einem Bezugspunkt zu sehen. Dieser Bezugspunkt kann waehrend des Editierens veraendert werden.

4.3. Editorfunktionen

Die Eingabe der Grafik soll ausschliesslich ueber die Tastatur erfolgen. Als Minimum sind folgende Funktionen notwendig:

- Position festhalten, Stift absenken/anheben (Leertaste,B)
- Bezugspunkt eingeben (N)
- Text eingeben (T)
- Position fuer variablen Text festhalten (V)
- Abbruch (C)

Eine Erweiterung des Editors bezueglich der Funktionen ist jederzeit moeglich.

4.4. Realisierung

Es wird davon ausgegangen, dass die Grundfunktionen bereits programmiert sind. Das Programm fuer die Erstellung von Hintergrundgrafiken kann in zwei Teile zerlegt werden: Einmal ist es der Teil, der ausschliesslich zur Eingabe der Position erforderlich ist und zum anderen der zur Ausgabe noetige Teil. Die Eingabe wird ueber die Tastatur vorgenommen. Mit den Kursortasten kann ein Grafikkursor auf dem Bildschirm bewegt werden. Die aktuelle Position steht in den Variablen KURX und KURY, die durch die Grundfunktionen -LI-, -RE-, -RU-, -HO- beeinflusst werden.

Linien:

Linienzuege werden durch Festhalten der Anfangs- und Endposition eines Vektors per Taste eingegeben. Die Positionen werden in der Liste abgespeichert und gleichzeitig wird der Vektor auf dem Bildschirm sichtbar, was durch die Worte PLO+>L bei abgesenktem und durch NPLO+>L bei angehobenem "Stift" ausgefuehrt wird. Die genannten Funktionen sind jeweils im Listing zu finden.

Text:

Die Texteingabe erfolgt folgendermassen: Nach Betaetigen der Taste T kann beliebiger Text eingegeben werden, der ab Grafikkursorposition sichtbar wird. Abgeschlossen wird die Eingabe mit (CR). Zwei wichtige comFORTH-Funktionen vereinfachen die Programmierung dieser Routine. Das ist die Funktion EXPECT, die eine Anzahl Zeichen von der Tastatur erwartet und ab einer bestimmten Adresse (z. B. aktuelle Listenadresse) ablegt. Die zweite Funktion wird in den Worten GKANAL und NKANAL benutzt und ist durch die Uservariable <USER> gekennzeichnet. In diese Variable wird einfach die CFA einer Anwenderroutine gelegt und alle vom Rechner ausgegebenen Zeichen laufen ab sofort ueber diese Routine. Durch diese einfache Aktion ist ploetzlich ein voellig anderer Treiber wirksam. Im Beispiel wird die CFA des Wortes ZE1 in <USER> eingetragen.

```
100 100 POS(cr)   ok
GKANAL(cr)   ok
300 .(cr)   ok
```

Diese Eingabe hat zur Folge, dass an der Position 100,100 die Zahl 300 erscheint. Dem Anwender bleiben also saemtliche Anstrengungen zur Zahlenkonvertierung erspart. Durch die Ausnutzung dieser Uservariablen wird eine sehr effektive Programmierung moeglich.

Variabler Text:

Positionen fuer Variablen Text koennen durch Betaetigen der Taste 'V' eingegeben werden. Im Anwenderprogramm hat man dann die Moeglichkeit, beliebige Zeichenketten auf die festgehaltene Position auszugeben, wozu wieder die Worte GKANAL und NKANAL genutzt werden koennen. Die Abfrage der Tastatur waehrend des Editierens wird mit KEY und einer folgenden SELECT-Struktur realisiert. Die Programme sind im Listing zu finden und leicht nachzuvollziehen. Da die Editorfunktionen zur Eingabe und Bearbeitung der Grafikelemente nur waehrend des Eingabevorgangs benoetigt werden, muesste man anstreben, diese Programme nur zu diesem Zweck einzuladen und nach getaner Arbeit zu "vergessen". Dazu bietet sich die Moeglichkeit von comFORTH an, Programme transient zu laden. Der Editor kann also sehr komfortabel sein und benoetigt trotzdem keinen Speicherplatz im Anwenderprogramm!

Wenn eine Hintergrundgrafik erstellt ist, muss diese noch in das Anwenderprogramm eingebunden werden und in eine moeglichst handhabbare Form gebracht werden. Guenstig waere es, die einzelnen Bilder und Texte einfach durch Namen aufzurufen. Das provoziert geradezu die Benutzung der <BUILDS-DOES>-Struktur. Durch den <BUILDS-Teil wird das Objekt unter einem bestimmten Namen im Woerterbuch eingetragen. Nach Aufruf des Namens erfolgt die Darstellung auf dem Bildschirm. Vorher kann man noch den Masstab und den Bezugspunkt in die Variablen (MASS), (XNP) und (YNP) eintragen. Dabei werden Linienstrukturen, fester und

variabler Text getrennt, da man so die Moeglichkeit hat, bestimmte Grafiken mit unterschiedlichen Beschriftungen zu versehen. Diese Funktionen werden z. B. in der Form

```
OBJEKT name1
TEXT   name2
VTEXT  name3
```

benutzt, wobei die Namen in den folgenden Programmteilen beliebig aufgerufen werden koennen und die Darstellung der Bilder auf dem Bildschirm zur Folge haben. Zum variablen Text ist noch zu bemerken, dass alle Positionen in einer Liste abgespeichert werden. Bei Aufruf des Namens fuer eine Liste mit variablen Text muss noch eine Nummer n angegeben werden, die anzeigt, an welcher Stelle in der Liste die n'te eingegebene Position steht.

Beispiel:

```
4 GKANAL 2 name3 . NKANAL(cr)  ok
```

Diese Eingabe hat zur Folge, dass an der zweiten eingegebenen Position die Ziffer 4 erscheint.

Der Editor wird mit den Worten NEW (neues Bild) und GEDIT (Bearbeiten der aktuellen Grafik nach Abbruch) aufgerufen.

5. Zusammenfassung

Es wurde gezeigt, wie mit comFORTH sehr effektiv auf bestimmte Anwendungsfaelle zugeschnittene Hilfsmittel programmiert werden koennen. Die speziell zur Realisierung der oben konstruierten Aufgabe notwendigen Funktionen beanspruchen im Anwenderprogramm lediglich 755 Byte, da wesentliche, speicherintensive Teile transient geladen werden koennen. Das Beispiel erhebt keinen Anspruch auf Vollstaendigkeit, allerdings kann man durch Manipulationen mit dem Bezugspunkt und dem Masstab sehr interessante Effekte erzielen. Weitere Editorfunktionen koennten z.B. Loeschfunktionen fuer Korrekturen, Koordinatentransformationen (Drehungen, Spiegelungen) und Module fuer die Diskettenarbeit sein.

Verfasser: Dipl.-Ing. Hans-Peter Dünow
Wilhelm-Pieck-Universitaet Rostock
Sektion Technische Elektronik
Albert-Einstein-Strasse 2
Rostock 6
DDR - 2500

Grafikeditor

```
Screen  1
( Basisfunktionen fuer Hintergrundgrafik        Due    30.01.87 )


0 VARIABLE (START) 0 VARIABLE (STARTX)  ( Listenanfaenge )
2000 CONSTANT #LINES  ( max. Anzahl von Linien )
0 VARIABLE (KENN) ( Kennung fuer akt. Farbe )
0 VARIABLE (XNP) 0 VARIABLE (YNP) ( Koordinaten fuer Bezugsp. )
0 VARIABLE (ZEIG) ( Zeiger fuer Darstellung v. Linien )
1 VARIABLE (MASS) ( Masstab )
0 VARIABLE (Ya) 0 VARIABLE (Xa) ( Linienstart )
0 VARIABLE ('START) ( Speicher fuer Grafikpuffer )
0 VARIABLE (USER> ( Speicher fuer <USER> )
0 VARIABLE (NTXT) ( naechster Texteintrag )
0 VARIABLE (STARTV)

                                                                -->

Screen  2
( Basisfunktionen fuer Hintergrundgrafik        Due  30.01.87 )

: 5* ( n --> n*5 ) DUP DUP + DUP + + ;
: PENS ( --> ,Stift absenken) (FARBE) @ (KENN) ! ;
: PENR ( --> ,Stift anheben ) 0 (KENN) ! ;
: YnXn ( --> ,neue Koord. ) (XNP) @ (ZEIG) @ 1+ @ (MASS) @ * +
       (YNP) @ (ZEIG) @ 3 + @ (MASS) @ * + 5 (ZEIG) +!
       (Ya) ! (Xa) ! ;
: PLOT ( --> ,Darstellg. einer Linie )
       (ZEIG) @ C@ IF (Ya) @ (Xa) @ YnXn (Ya) @ (Xa) @ VEC
       ELSE YnXn ENDIF ;
: GPLOT ( --> ,Gesamtdarstellung )
       (YNP) @ (Ya) ! (XNP) @ (Xa) ! (START) @ DUP 2+ (ZEIG) !
       @ 0 DO PLOT LOOP ;

                                                                -->

Screen  3
( Basisfunktionen fuer Hintergrundgrafik        Due  30.01.87 )

: GKANAL ( --> ,Grafik -> <USER> ) <USER> @ (USER> ! ' ZE1
         CFA <USER> ! ;
: NKANAL ( --> ,Normalbildschirm - <USER> ) (USER> @ <USER> ! ;
: AUSGTXT ( --> ,Textausgabe )
         GKANAL (STARTX) @ DUP 2+ SWAP @ 0 DO DUP 2+ DUP 2+
         SWAP OVER @ SWAP @ (XNP) @ + SWAP (YNP) @ + SWAP
         POS 1+ SWAP @ 0 DO 1+ DUP C@ EMIT LOOP 1+ LOOP DROP
         NKANAL ;
: STRINGS ( --> ,Stringaufbau im DIR )
         (NTXT) @ (STARTX) @ DO I C@ C, LOOP ;
: TEXT ( --> ,Text -> Name ) <BUILDS STRINGS DOES> (STARTX) !
         AUSGTXT ;

                                                                -->
```

Grafikeditor

```
Screen  4
( Basisfunktionen fuer Hintergrundgrafik          Due  30.01.87 )

: OBJCRE ( --> ,create object ) (START) @ DUP DUP @ 5* + 2+
        SWAP DO I C@ C, LOOP ;
: OBJEKT ( --> ,Objekt + Name )  <BUILDS OBJCRE DOES>
         (START) ! GPLOT ;
: VTEXT  ( n --> ,Eingabe variabler Strings )
         <BUILDS (STARTV) @ DUP 2+ SWAP @ 0 DO DUP 2@  , ,
         2+ 2+ LOOP DROP DOES> SWAP 1 - DUP + DUP + + 2@
         POS ;
: .VAR ( --> ,Print var. Text ) GKANAL . NKANAL ;
: LOAD_GEDIT 5 LOAD ;



Screen  5
( Grafikeditorfunktionen                          Due 30.01.87 )



BEGIN-TRANS

0 VARIABLE (LEAVE) ( Austrittsbedingung )
: >LIST ( y,x --> ,Listeneintrag )
        (XNP) @ - SWAP (YNP) @ - SWAP (START) @ DUP @ 5* 2+ +
        DUP (KENN) @ SWAP C! 1+ DUP >R ! R> 2+ ! 1 (START) @
        +! ;
                                                          -->


Screen  6
( Grafikeditorfunktionen                          Due 30.01.87  )

: PLO+>L ( --> ,plotten und eintragen )
         KURY @ KURX @ (KENN) @ IF 2DUP (Ya) @ (Xa) @ VEC ENDIF
         2DUP (Xa) ! (Ya) ! >LIST PENS ;
: NPLO+>L ( --> ,nicht plotten,eintragen )
         KURY @ KURX @ 2DUP 2DUP (Ya) @ (Xa) @ VEC (Xa) ! (Ya)
         ! >LIST PENR ;
: INITXT ( Initialisierung Texteingabe )
         ('START) @ #LINES + DUP (STARTX) ! 2+ (NTXT) ! (STARTX)
         @ 3000 ERASE  ;
: INIVTXT ( Initialisierung var. Text )
         ('START) @ #LINES DUP + + (STARTV) !  ;

                                                          -->
```

Grafikeditor

```
Screen  7
( Grafikeditorfunktionen                                Due  30.01.87

 : EINTXT ( --> ,Texteingabe )
        KURY @ KURX @ POS GKANAL (NTXT) @ 2+ DUP KURX @ (XNP)
        @ - SWAP ! 2+ DUP KURY @ (YNP) @ - SWAP ! 1+ DUP
        1+ 80 EXPECT BEGIN 1+ DUP C@ 1 (NTXT) @ +! 0= UNTIL
        (NTXT) DUP @ -1 SWAP +! 1 (STARTX) @ +! NKANAL ;
: POSVTXT ( --> ,Position var.Text)
        KURY @ KURX @ (XNP) @ - SWAP (YNP) @ - SWAP (STARTV)
        @ DUP @ DUP + DUP + 2+ + 2! 1 (STARTV) @  +! ;




                                                                -->

Screen  8
( Grafikeditorfunktionen                        Due  30.01.87
HEX
: GEDIT  ( --> ,Editorstart ohne Initialisierung )
         ('START) @ (START) ! 0 (LEAVE) ! 0 (KENN) !
         BEGIN KEY DKUR
    SELECT
      04 IF_EQUAL -RE- END-PATH
      08 IF_EQUAL -LI- END-PATH
      05 IF_EQUAL -HO- END-PATH
      18 IF_EQUAL -RU- END-PATH
      20 IF_EQUAL PLO+>L END-PATH
     'B' IF_EQUAL NPLO+>L END-PATH
     'C' IF_EQUAL 1 (LEAVE) ! (START) @ ('START) ! END-PATH
     'N' IF_EQUAL KURX @ (XNP) ! KURY @ (YNP) ! END-PATH
     'T' IF_EQUAL EINTXT END-PATH
     'V' IF_EQUAL POSVTXT END-PATH                              -->

Screen  9
( Grafikeditorfunktionen                        Due  30.01.87 )

           'S' IF_EQUAL KEY 30 - (S> ! END-PATH

   DROP END-SELECT SKUR (LEAVE) @ UNTIL ;  DECIMAL


: NEW ( --> ,neues Bild ) PAD DUP ('START) ! INITXT 0 SWAP !
       INIVTXT       GEDIT ;

  END-TRANS
```

Applikation von FORTH

Wolfgang Drewelow

Identifikationsalgorithmen in FORTH

In diesem Beitrag werden zwei Anwendungen von FORTH auf dem Gebiet der Identifikation dynamischer Systeme beschrieben. Das Korrelationsverfahren wird als eine Methode behandelt, bei welcher eine weitgehende Anlehnung an eine maschinennahe Programmierung sinnvoll ist, waehrend fuer die Realisierung der expliziten Methode der kleinsten Fehlerquadrate ein komplexer, hochsprachiger Befehlsvorrat bevorzugt werden sollte.

1. Einleitung

Die Konzipierung von Steuer- und Reglerstrategien erfordert Modelle der zu steuernden und zu regelnden Prozesse. Eine Vielzahl von experimentellen Verfahren zur Modellgewinnung sind aus Veroeffentlichungen bekannt (z.B. /1/,/2/). Hierbei wird das Eingangs- und das Ausgangssignal des zu untersuchenden Systems messtechnisch erfasst und dann analysiert. Diesen als Identifikationsverfahren bezeichneten Methoden ist vielfach ein mittlerer bis grosser numerischer Aufwand gemeinsam, der auf eine geeignete Weise zunaechst programmtechnisch erledigt werden muss.

Die Zeiten der Maschinenprogrammierung von Mikrorechnern, in der bezueglich der Softwaretechnik scheinbar alle Fehler der Anfangszeit der Rechentechnik wiederholt wurden, scheinen ueberwunden. Auch ein "eingefleischter" Maschinenprogrammierer oder Assemblerspezialist wird heute anerkennen, dass bezueglich des Aufwands zur Erstellung und Wartung und bezueglich der Zuverlaessigkeit seine alten Methoden gegenueber Hochsprachen keine vertretbare Alternative mehr darstellen. Allerdings erfordern gerade Prozessanpassungen, wie sie z.B. bei den Aufgaben der Messplatzautomatisierung, der digitalen Regelung und Ueberwachung und eben auch bei der Identifikation erforderlich sind, die organische Einbindung von Maschinenanteilen. Hinzu kommt, dass prozessgebundene Algorithmen nach einem festgelegten Zeitregime arbeiten sollen, so dass Forderungen nach der Echtzeit- oder Multitaskfaehigkeit des Programmierergebnisses gestellt werden muessen. Viele der gegenwaertig verfuegbaren Hochsprachen bieten diesbezueglich keine oder nur unbefriedigende Moeglichkeiten an. Die Spezifik prozessorientierter Software liegt offensichtlich etwas abseits von der jetzt schon zwangsweise sehr breit gefaecherten Entwicklungsrichtung der Informatik.

Der entscheidende Vorteil von FORTH aus der Sicht des Prozessautomatisierungsingenieurs ist gerade das breite und dabei dennoch homogene Spektrum, welches dieses Mittel zur Programmerstellung anbietet: Sowohl ausserordentlich prozessnahe (maschinennahe) Instruktionen als auch sehr komplexe, nutzerdefinierte Hochworte sind die gleichberechtigten Elemente seines "Wortschatzes". Zudem stellt FORTH als Dialogsprache dem Nutzer seine gesamten Systemanteile zur Verfuegung (die Nutzung verlangt allerdings auch die Kenntnis der inneren FORTH-Struktur), so dass die Entwicklung von Dialogkonzepten fuer prozess- und problemorientierte Fachsprachen weitgehend vereinfacht wird.

Im vorliegenden Beitrag soll insbesondere die verfuegbare Spannweite des FORTH-Konzepts anhand von zwei Beispielen aus dem Problembereich der Prozessidentifikation verdeutlicht werden. Das Beispiel im 2. Abschnitt befasst sich mit dem Korrelationsverfahren und verdeutlicht den prozessornahen Bereich der FORTH-Programmierung einschliesslich der Wandleransteuerung, einer einfachen Echtzeitanbindung und einer laufzeitguenstigen Testsignalgenerierung. Im 3. Abschnitt wird als Beispiel die Programmierung der Methode der kleinsten Fehlerquadrate behandelt. Hier erfolgt die Formulierung der Algorithmen weitgehend auf der Grundlage eines vordefinierten Wortvorrats fuer komplexe Operationen der Matrixalgebra. Der Autor hofft, dass mit diesen Beispielen eine Vorstellung ueber die sinnvolle Anwendbarkeit von FORTH im Bereich der Identifikation vermittelt werden kann.

2. Korrelationsverfahren

2.1. Einige Vorbemerkungen zur Theorie

Als parameterfreies Modell fuer einen abgetasteten, linearisierbaren Prozess diene die Gewichtsfolge g(k), die ueber die Faltungssumme (1) das Verhalten der Ausgangsgroesse y(k) zur Eingangsgroesse u(k) beschreibt.

$$y(k)=\sum_{i=0}^{\infty} g(i)\ u(k-i) \qquad (1)$$

Eine zentrale Methode, die Gewichtsfolge auch an gestoerten Prozessen zu bestimmen, stellt die Korrelationsmethode dar. Sie basiert auf dem zu (1) analogen Zusammenhang zwischen der Gewichtsfolge sowie der Autokorrelationsfunktion Ruu(k) des Eingangssignals und der Kreuzkorrelationsfunktion Ruy(k) zwischen Ausgangs- und Eingangsgroesse (2).

$$Ruy(k)=\sum_{i=0}^{\infty} g(i)\ Ruu(k-i) \qquad (2)$$

$$Ruu(k)=1/N\sum_{i=0}^{N} u(i)\ u(i+k) \qquad (3)$$

$$Ruy(k)=1/N\sum_{i=0}^{N} u(i)\ y(i+k) \qquad (4)$$

N...Messpunktezahl

Praktisch werden haeufig als Testsignale periodische Binaerfolgen maximaler Laenge benutzt, ein Beispiel fuer ein solches Signal ist in Bild 1 zu sehen.

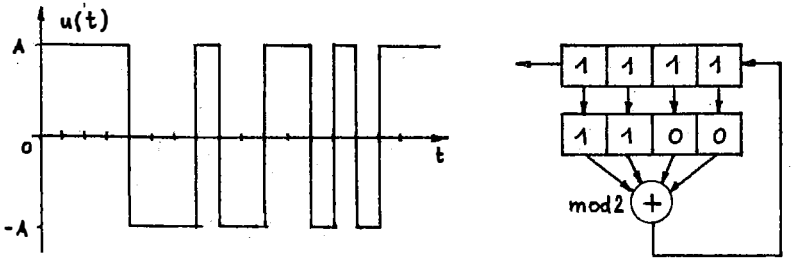


Bild 1: Beispiel eines pseudostochastischen Binaersignals und Moeglichkeit fuer dessen Erzeugung

Durch die spezielle Konstruktion der Testsignale ueber rueckgekoppelte Schieberegister - die Schieberegisterzustaende werden am Ausgang mit der jeweiligen Ausgangsbewertung multipliziert, diese Produkte modulo2 addiert und auf den Eingang zurueckgekoppelt (Bild 1) - nimmt die Autokorrelationsfunktion Ruu(k) eine solche Form an, dass zwischen der Kreuzkorrelationsfunktion Ruy(k) und der Gewichtsfunktion g(k) ein angenaehert proportionaler Zusammenhang besteht, d.h.

$$Ruy(k) \approx K\, g(k).$$

Voraussetzung fuer die Gueltigkeit dieser Naeherung ist zum einen, dass vor der eigentlichen Messung zunaechst eine vollstaendige Testsignalfolge auf das System gegeben wird, um dieses in den eingeschwungenen Zustand zu versetzen, zum anderen muss die Dauer einer Testsignalperiode groesser als die Abklingzeit des zu modellierenden Systems sein.

Unter Einhaltung der genannten Bedingungen ist das Verfahren also bestechend einfach: Auf das zu untersuchende System werden mehrere Perioden (einschliesslich der Einschwingperiode) einer pseudostochastischen Testfolge gegeben, das zugehoerige Ausgangssignal zu den Zeitpunkten des Signalwechsels erfassst und entsprechend (4) mit den Werten des verschobenen Eingangssignals korreliert. Diese Kreuzkorrelationsfunktion repraesentiert dann bereits die Gewichtsfunktion g(k) (angenaehert proportional). Eine ausfuehrlichere Beschreibung dieses Verfahrens ist z.B. in /1/ enthalten.

2.2. Realisierung des Korrelationsverfahrens in FORTH

Von dem Rechnerprogramm sind im wesentlichen die folgenden Aktivitaeten zu realisieren:

1. Erzeugung des Testsignals und Ausgabe des Testsignals ueber den Digital-Analog-Wandler im Rhythmus der vorgebbaren Abtastzeit
2. Erfassung des Ausgangssignals (nach erfolgtem Ablauf der Einstellfolge) ueber den Analog-Digital-Wandler im Abtastrhytmus
3. Ermittlung der zurueckliegenden (verschobenen) Werte des Eingangssignals
4. Multiplikation des zuletzt gemessenen Ausgangssignals mit den verschobenen Werten des Eingangssignals
5. Aufsummation dieser Produkte fuer die unterschiedlichen Verschiebungen zwischen Eingangs- und Ausgangswerten

Die Algorithmen zur Testsignalbehandlung machen einen Grossteil des numerischen Aufwands aus. Da das Testsignal nur zwei Zustaende annehmen kann, bietet sich fuer eine rechenzeit- und speicherplatzguenstige Realisierung eine bitweise Zustandsverwaltung an. In dem nachfolgend angefuehrten Demonstrationsprogramm ist der Eingangssignalverlauf ueber die letzten 16 Schritte in einem Wort (Variable SCHIEBEREGISTER) registriert, wobei der letzte Wert durch das Bit 0 repraesentiert wird. Gleichzeitig werden die letzten Bits dieses Wortes als Zustaende des erzeugenden Schieberegisters benutzt, wobei die Bewertungsfaktoren in analoger Reihenfolge in der Variablen BEWERTUNGEN abgespeichert sind. Eine effektive Bearbeitung dieser Datenstrukturen erfordert geeignete Maschinenpassagen, die als CODE-Definitionen auf den Screens 2 und 3 angegeben sind. Die Routine SR<- summiert modulo 2 die Schieberegisterinhalte der Stufen auf, die durch eine Eins in der Variablen BEWERTUNGEN an der entsprechenden Stelle markiert sind. Diese Summe ist der neue Folgenwert, der nach einer Linksverschiebung der Variablen SCHIEBEREGISTER auf das freiwerdende Bit 0 eingetragen wird. Die Routine u(k-n) erwartet auf dem Datenstapel den Index der zu ermittelnden (vergangenen) Eingangsgroesse, bestimmt den Folgenwert aus der Bitposition (=Index) und legt diesen als Ergebnis auf den Stapel.

Eine weitere programmtechnische Vereinfachung ist dadurch moeglich, dass die in jedem Rekursionsschritt durchzufuehrenden Multiplikationen des Ausgangssignals mit den verschobenen Eingangssignalwerten wegen der nur moeglichen zwei Zustaende durch einfache Vorzeichenwechsel des Ausgangssignals ersetzt werden koennen. Diese Moeglichkeit wird bei der Formulierung der Rekursion MESSWERTVERARBEITUNG zur Verarbeitung eines Messwertes (Screen 5) beruecksichtigt. Die durch die Routine u(k-n) ermittelten vergangenen Testsignalzustaende werden dabei als Flags benutzt.

Die bislang beschriebenen Routinen haetten selbstverstaendlich als Hochwortdefinitionen auch ueber anderen, nicht bitorientierten Datenstrukturen arbeiten koennen - sicher auf Kosten der Rechenzeit, eventuell aber bei groesserer Durchsichtigkeit dieser Routinen selbst. Allerdings wird man bei richtiger Programmierweise in FORTH die Schnittstellen von Programmoduln nicht so waehlen, dass diese von gewaehlten Datenstrukturen abhaengig sind. Damit sind Routinen ohne weiteres gegeneinander austauschbar, wenn nur diese Schnittstellen exakt eingehalten werden. Uebersichtlichkeit von FORTH-Programmen wird zum wesentlichen Teil durch geeignete Wahl der Parameteruebergabebedingungen erreicht.

Programme fuer die Analog-Digital-, die Digital-Analog-Wandlung sowie zur Initialisierung von Zeitsteuerbedingungen bzw. zu deren Abtestung werden ueblicherweise in Maschinensprache formuliert. Entsprechende Codepassagen sind peripherieabhaengig und waeren auf Screen 4 einzusetzen. Gegenwaertig sind die Wandlerroutinen ADU und DAU so abgefasst, dass ein Durchgang vom Testsignal auf den Messeingang erfolgt, wodurch die Korrelationsanalyse zur Ermittlung der Autokorrelationsfunktion fuehrt. Die Programme START und WARTE, die im Normalfall die Initialisierung der Hardware fuer eine externe Taktung entsprechend der Variablen ABTASTZEIT bzw. die Abtestung eines Taktzustandsflags vornehmen sollen, sind hier als NOP angegeben, so dass das Programm in dieser Form ohne eine Zeitausrichtung mit der maximalen Geschwindigkeit ablaeuft. Als einfacher Dialograhmen dient das Programm KORRELATION. Hier werden Werte fuer die Schieberegisterbewertungen in binaerer Form, die Abtastzeit, die Messperiodenzahl und die Amplitude des Testsignals eingegeben. Die Periodenlaenge L wird aus der Schieberegisterlaenge N (implizit in der Eingabe der Bewertungsfaktoren enthalten: Anzahl der signifikanten Binaerstellen) nach der Beziehung L=2**N-1 errechnet und angezeigt. Das Rahmenprogramm nutzt die Routine PLOT, die den Inhalt des Vektors Ruy - die ermittelten Kreuzkorrelationswerte Ruu(0) bis Ruu(15) im Double-Integer-Format - ausgibt.

```
A:KORRBSP.SCR  Screen 1
 0 ( Demonstrationsbeispiel - Korrelation 1 )
 1
 2 ( Zusatzarithmetik )
 3 CODE 4* ( n ===> 4n )
 4     HL POP,  HL HL ADD,  HL HL ADD,  HPUSH # JP, END-CODE
 5 : D+! ( d adr ===> ) DUP >R 2@ D+ R> 2! ;
 6
 7 ( Vektordefinition fuer Double-Integer )
 8 : DVECTOR ( n ===> ) <BUILDS 4* ALLOT DOES> ;
 9 : DVELEMENT ( dvectoradr n ===> elementadr )  4* + ;
10
11 ( Datenvereinbarungen )
12 32767 VARIABLE SCHIEBEREGISTER    20 VARIABLE BEWERTUNGEN
13     1 VARIABLE PERIODEN           31 VARIABLE FOLGENLAENGE
14     0 VARIABLE AMPLITUDE
15    16 DVECTOR  Ruy ( Vektor der Korrelationswerte )      -->
```

```
A:KORRBSP.SCR  Screen 2
  0 ( Demonstrationsbeispiel - Korrelation 2 )
  1 ( Verschiebung des Schieberegisters nach links und Erzeugung )
  2 ( des neuen Wertes )
  3
  4 CODE SR<- ( ===> )
  5     DE PUSH,                SCHIEBEREGISTER #) HL LD,  HL PUSH,
  6     BEWERTUNGEN #) DE LD,   L A LD,                      E AND,
  7     A L LD,                 H A LD,                      D AND,
  8     A H LD,              16 # DE LD,
  9      BEGIN,                 HL HL ADD,
 10           CY IF,               D INC,          ENDIF,
 11           E DEC,                                      Z UNTIL,
 12     D A LD,                 1 # AND,                   HL POP,
 13     HL HL ADD,                L OR,                    A L LD,
 14     HL SCHIEBEREGISTER #) LD,                        DE POP,
 15     NEXT # JP,              END-CODE -->

A:KORRBSP.SCR  Screen 3
  0 ( Demonstrationsbeispiel - Korrelation 3 )
  1
  2 ( Holen des Wertes des Eingangssignals u[k-n] zum Messzeit- )
  3 ( punkt k )
  4
  5 CODE u(k-n) ( n ===> folgenwert )
  6     DE POP,          E INC,          SCHIEBEREGISTER #) HL LD,
  7     BEGIN,
  8        H RR,         L RR,           E DEC,
  9     Z UNTIL,
 10     0 # HL LD,
 11     CY IF,           HL INC,      ENDIF,
 12     HPUSH # JP,      END-CODE           -->
 13
 14
 15

A:KORRBSP.SCR  Screen 4
  0 ( Demonstrationsbeispiel - Korrelation 4 )
  1
  2 ( Echtzeitsteuerung )
  3 CODE START ( ===> )
  4     ( Betriebssystemabhaengige Codepassage: Vereinbarung der )
  5     ( Interruptanbindung, Start des CTC ) NEXT JP, END-CODE
  6 CODE WARTE ( ===> )
  7     ( Betriebssystemabhaengige Codepassage: z.B. Abwarten, )
  8     ( bis Synchronisationsflag durch entsprechende Inter- )
  9     ( ruptserviceroutine gesetzt ist ) NEXT JP, END-CODE
 10
 11 ( Wandlersteuerung, peripheriespezifisch )
 12 : ADU    ( ===> y )    0 u(k-n) IF 1 ELSE -1 ENDIF ;
 13 : DAU    ( u ===> )    DROP ;
 14     ( In diesem Fall Simulation eines Durchgangs von der )
 15     ( Testsignalerzeugung auf den Messwerteingang ) -->
```

```
A:KORRBSP.SCR  Screen 5
  0 ( Demonstrationsbeispiel - Korrelation 5 )
  1
  2 : MESSWERTVERARBEITUNG ( y ===> ) S->D
  3      16 0 DO 2DUP I u(k-n) 0= IF DMINUS ENDIF
  4                          Ruy I DVELEMENT D+! LOOP 2DROP ;
  5 : MESSABLAUF ( ===> )
  6      16 0 DO 0, Ruy I DVELEMENT 2! LOOP
  7      START
  8      FOLGENLAENGE @ 0 DO
  9       SR<- AMPLITUDE @ 0 u(k-n) 0= IF MINUS ENDIF WARTE DAU LOOP
 10      PERIODEN @ 0 DO
 11        FOLGENLAENGE @ 0 DO
 12            SR<- AMPLITUDE @ 0 u(k-n) 0= IF MINUS ENDIF WARTE DAU
 13               ADU MESSWERTVERARBEITUNG
 14            LOOP
 15        LOOP ;      -->

A:KORRBSP.SCR  Screen 6
  0 ( Demonstrationsbeispiel - Korrelation 6 )
  1
  2 : PLOT ( ===> ) 16 0 DO CR Ruy I DVELEMENT 2@ D. LOOP ;
  3
  4 : KORRELATION ( ===> )
  5     CR ." Bewertungsfaktoren des erzeugenden Schieberegisters :"
  6        BINARY GET_INUM DECIMAL
  7        1 OVER BEGIN 2 / SWAP 2 * SWAP -DUP 0= UNTIL 1- DUP
  8     CR ." Folgenlaenge: " . FOLGENLAENGE ! BEWERTUNGEN !
  9     CR ." Messperioden: " GET_INUM PERIODEN !
 10     CR ." Abtastzeit: " GET_INUM
 11          ( muss Eingang in die Initialisierung des CTC bzw. )
 12          ( in die Interruptserviceroutine finden ) DROP
 13     CR ." Testsignalamplitude:" GET_INUM AMPLITUDE !
 14        MESSABLAUF CR ." Kreuzkorrelationsfunktion:" PLOT ;
 15
```

Das angegebene Programm benutzt bis auf die GET_INUM-Funktion (residenter Bestandteil von comFORTH) und die Assemblerdefinitionen ausschliesslich Worte des FIG-FORTH-Kerns.

Fuer eine professionelle Anwendung der Korrelationsanalyse sind selbstverstaendlich noch Erweiterungen dieses Programms vorzusehen - das angefuehrte Beispiel dient vorrangig der Demonstration der Moeglichkeiten maschinennaher Programmierung in FORTH. Sinnvoll ist so eine Erweiterung der Schieberegister-Datenstruktur, die dann Verschiebungswerte groesser 16 zulaesst. Die Ausgabe der Gewichtsfolge in pseudografischer und grafischer Form muesste ebenfalls in entsprechenden Erweiterungen enthalten sein. Ebenso wird man eine Umsetzung des parameterfreien in ein parametrisches Modell fuer die Weiterverarbeitung berucksichtigen.

Nachfolgend sind Ergebnisse eines Identifikationslaufes mit einem solchen komplexeren FORTH-Programm dargestellt.

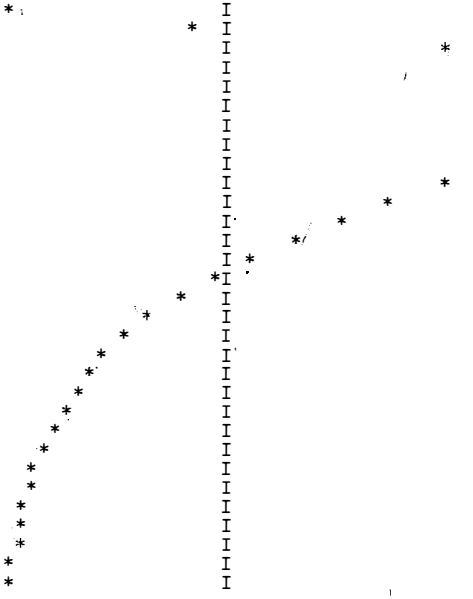
KORRELATION

Kommentar: ungestoerter Fall
Abtastzeit (in 10 ms-Schritten): 60
Wertigkeit: 2
Amplitude: .1
Arbeitspunkt: 0.
Schieberegisterlaenge: 5
Periodenlaenge: 31
Bewertung 0: 0
Bewertung 1: 0
Bewertung 2: 1
Bewertung 3: 0
Bewertung 4: 0
Folgenbeginn: 10
Perioden: 3
Korrelationspunkte: 31

Auswertung beendet
Verstaerkung bekannt (Y/N) N
Bezug auf Anfangswert (Y/N) N
Bezug auf Endwerte (Y/N) N

Summe der Gewichtsfolgenwerte: .03116

g(k) PLOT



0	-.29959E-01
1	-.47267E-01
2	.03830E-01
3	.49489E-01
4	.56984E-01
5	.57429E-01
6	.53513E-01
7	.47149E-01
8	.39641E-01
9	.31733E-01
10	.23983E-01
11	.16659E-01
12	.10009E-01
13	.39892E-02
14	-.12746E-02
15	-.59028E-02
16	-.99080E-02
17	-.13290E-01
18	-.16233E-01
19	-.18725E-01
20	-.20804E-01
21	-.22559E-01
22	-.24027E-01
23	-.25254E-01
24	-.26291E-01
25	-.27130E-01
26	-.27836E-01
27	-.28414E-01
28	-.28910E-01
29	-.20330E-01
30	-.29673E-01

```
PARAMETERSCHAETZUNG
Ordnung: 2
Totzeit: 0
Schrittzahl: 15

Parametervektor:
-1.53630  .61713  .04129  .01960
```

3. Methode der kleinsten Fehlerquadrate

3.1. Ansatz

Die von GAUSS begruendete Methode der kleinsten Fehlerquadrate (MKQ) ist das grundlegende Verfahren zur Bestimmung von Schaetzwerten fuer die Koeffizienten parametrischer Modelle. Auch bei der Parameterschaetzung dynamischer Systeme spielt dieses Verfahren eine zentrale Rolle. Wird dem Prozess ein Modell in Form der Differenzengleichung

$$y(k)+a_1y(k-1)+\ldots+a_ny(k-n) = b_1u(k-1-d)+\ldots+b_nu(k-n-d)$$

n...Modellordnung
d...Modelltotzeit

zugrunde gelegt, so ergibt sich fuer die Schaetzung der Parameter aus gemessenen Ein- und Ausgangssignalwerten u(0-d) bis u(N-1-d) bzw. y(0) bis y(N) die Gleichung

$$\hat{\underline{\Theta}} = (\underline{M}'\underline{M})\underline{M}'\underline{y} \qquad (5)$$

mit

$$\underline{M} = \begin{bmatrix} -y(n-1) & \ldots & -y(0) & u(n-1-d) & \ldots & u(0-d) \\ \vdots & & \vdots & \vdots & & \vdots \\ -y(N-1) & \ldots & -y(N-n) & u(N-1-d) & \ldots & u(N-n-d) \end{bmatrix} \qquad (6)$$

$$\underline{y}' = [y(n), y(n+1), \ldots y(N)] \qquad (7)$$

$$\hat{\underline{\Theta}}' = [a_1, \ldots a_n, b_1, \ldots b_n]$$

Wird die Gleichung (5) direkt zur Schaetzung der Parameter a1 bis an und b1 bis bn benutzt (es gibt auch andere numerische, insbesondere on-line Varianten), so ist es nicht guenstig, die (2n, N-n+1)-Matrix $\underline{M}'\underline{M}$ und den N-2n+1-Vektor $\underline{M}'\underline{y}$ zu verwalten, sondern statt dessen gleich im Sinne eines verringerten Speicherbedarfs mit der (2n,2n)-Matrix $\underline{M}'\underline{M}$ und dem 2n-Vektor $\underline{M}'\underline{y}$ zu arbeiten. Es ergibt sich dann der folgende rekursive Algorithmus: Mit der Messwertvektordefinition
$\underline{m}(k)' = [-y(k-1), \ldots, -y(k-n), u(k-1-d), \ldots, u(k-n-d)]$
gelten fuer den Zeitpunkt k+1 folgende Gleichungen fuer das Matrizenprodukt $\underline{M}'\underline{M}$ bzw. das Produkt $\underline{M}'\underline{y}$:

$$\underline{M}'\underline{M}(k+1) = \underline{M}'\underline{M}(k) + \underline{m}(k+1)\underline{m}(k+1)'$$

$$\underline{M}'\underline{y}(k+1) = \underline{M}'\underline{y}(k) + \underline{m}(k+1)y(k+1)$$

Diese Rekursionen sind der Ausgangspunkt fuer den im folgenden Abschnitt formulierten Algorithmus.

3.2. Formulierung der MKQ in FORTH

Waehrend im Abschnitt 2.2 gezeigt wurde, wie maschinennahe, dafuer numerisch aber relativ anspruchslose Algorithmen in FORTH zu behandeln sind, soll im Gegensatz dazu jetzt gezeigt werden, dass auch eine umfangreiche Numerik bei einem entsprechend dem Zielanwendungsspektrum ausgebauten Befehlssatz einfach programmierbar ist.

Da Parameterschaetzverfahren im allgemeinen komplexe Operationen der Matrixalgebra erfordern, wurde der bestehende FORTH-Kern um zwei Pakete erweitert. Das erste Paket liefert einen Fliesspunktarithmetik-Befehlssatz, der sich in der Bezeichnung an die im Kern residente Integerarithmetik anlehnt und zur Unterscheidung ein vorangestelltes F enthaelt, z.B. F@, FOVER, F*, F+!, F!, FDROP, FMINUS. Auf diesem Befehlssatz baut als zweites Paket ein Matrixmodul auf. Es enthaelt die Definitionsworte fuer die Datentypen VECTOR und MATRIX. Darueber hinaus sind eine Reihe von Operationen ueber diesen Datentypen definiert. Nachfolgend sind nur die Worte, die Eingang in das Anwenderprogramm MKQ finden, kurz erklaert.

VECTOR (n ===>)
Dieses Wort wird in der Form n VECTOR name angewandt. Daraufhin wird die Vektorstruktur fuer n Elemente angelegt. Die Nennung von name legt die Objektadresse auf den Datenstapel (identisch mit der Adresse des ersten Elements)

MATRIX (n1 n2 ===>)
Analog zu VECTOR wird bei Ausfuehrung von n1 n2 MATRIX name die Struktur der Matrix name fuer n1 Zeilen und n2 Spalten angelegt. Bei Anwendung von name wird die Matrixadresse auf den Stapel gelegt.

!VDIM (n vaddr ===>)
Die Dimension der bereits definierten Vektors mit der Objektadresse vadr wird neu auf n gesetzt. Der urspruenglich vorgesehene Speicherraum wird nicht veraendert.

!MDIM (n1 n2 maddr ===>)
Die Dimension einer bereits definierten Matrix wird neu gesetzt, ohne dass der urspruenglich vorgesehene Speicherraum geaendert wird.

@VDIM (vaddr ===> n)
legt die aktuelle Anzahl n von Komponenten des Vectors mit der Objektadresse vaddr auf den Stapel.

VELEMENT (vaddr k ===> addr)
legt die Adresse des k-ten Elements eines Vektors auf den Datenstack. Die Adressierung beginnt beim nullten Element.

MELEMENT (maddr n1 n2 ===> addr)
addressiert das Element der Matrix in der Zeile n1 und der Spalte n2. Die Zaehlung beginnt jeweils bei Null.

VRROT (f0_neu vaddr ===> fn_alt)
Rechtsrotation eines Vektors der Dimension n.

VSET (f vaddr ===>)
Alle Elemente eines Vektors werden gleich f gesetzt.

MSET (f maddr ===>)
Die Elemente der Matrix werden mit dem konstanten Wert f geladen.

Ax=b (maddr_A vaddr_x vaddr_b ===>)
Fuer eine gegebene quadratische (n,n)-Matrix A und den gegebenen n-Vektor x wird die Loesung x des linearen Gleichungssystems Ax=b ermittelt.

Das Programm zur Parameterschaetzung nach der MKQ arbeitet nach dem oben angegebenen rekusiven Prinzip. In Screen 1 sind die erforderlichen Datenstrukturen vereinbart, wobei weitgehend analoge Bezeichnungen zum vorhergehenden Text gewaehlt wurden. Der Vektor uhi dient der Zwischenspeicherung von Eingangssignalwerten bei a priori bekannten Modelltotzeiten (quasi-Simulation der Totzeit im Parameterschaetzalgorithmus). Die Variable #REK enthaelt die Rekursionsschrittzahl der MKQ. Auf Screen 2 wird die Datenrekursion fuer die MKQ, d.h. die Rekursion fuer M'y (Zeilen 4 und 5) sowie fuer M'M (Zeilen 6 bis 11) angegeben. Bei der Rekursion von M'M wird die Symmetrieeigenschaft von m(k)m(k)' ausgenutzt. In den Zeilen 12 bis 15 wird die neu hinzukommende Dateninformation in den Messvektor m(k) und in den Totzeitvektor uhi eingearbeitet und damit die naechste Rekursion vorbereitet. In der Rekursion wurde beruecksichtigt, dass eine Bildung der Matrix M'M bzw. des Vektors M'y erst sinnvoll ist, wenn der Messvektor m(k) ausschliesslich signifikante Messwerte enthaelt (Zeile 2).

Screen 3 enthaelt in der Routine MKQINI die Initialisierung der Dimensionen und des Inhalts der zum Algorithmus gehoerigen Datenstrukturen entsprechend der Vorgaben fuer Modellordnung und -totzeit. Das Rahmenprogramm MKQ steuert die wiederholte Abarbeitung der Rekursionen und die Loesung des linearen Gleichungssystems zur Bestimmung des gesuchten Parametervektor (Adresse wird auf dem Stapel uebergeben) nach Beendigung der Rekursionsfolge. Nicht definiert auf den dargestellten Screens sind die Eingaberoutinen fuer die Ein- und Ausgangsdaten get_u und get_y. Diese sind problemabhaengig - sie sollen die Daten als Fliesspunktzahlen auf dem Datenstack hinterlegen.

```
A:MKQ.SCR  Screen 1
  0 ( EXPLIZITE MKQ - Datenvereinbarungen )
  1
  2 8 VECTOR m(k)
  3 8 VECTOR M'y
  4 5 VECTOR uhi
  5
  6 8 8 MATRIX M'M
  7
  8 0 VARIABLE d          0 VARIABLE n
  9 0 VARIABLE #REK
 10
 11 -->
 12
 13
 14
 15
```

```
A:MKQ.SCR  Screen 2
 0 ( EXPLIZITE MKQ - Einarbeitung eines Messwertepaares )
 1 : MKQREK ( u y —> )
 2      1 #REK +! #REK @ d @ n @ + 1- >
 3    ( Neuberechnung von M'M und M'y, erst ab Schritt n+d+1 )
 4      IF m(k) @VDIM 0 DO
 5           m(k) I VELEMENT F@ FOVER F* M'y I VELEMENT F+! LOOP
 6      m(k) @VDIM 0 DO
 7           m(k) @VDIM I DO
 8             m(k) I VELEMENT F@ m(k) J VELEMENT F@ F* I J =
 9               IF M'M I DUP MELEMENT F+!
10               ELSE FDUP M'M I J MELEMENT F+!
11                    M'M J I MELEMENT F+! ENDIF
12        LOOP LOOP ENDIF
13    ( Einsortieren der neuen Messwerte in M'y und uhi )
14    FMINUS m(k) VRROT FDROP d @ IF uhi VRROT ENDIF
15    m(k) DUP @VDIM 2/ VELEMENT F! ;  —>


A:MKQ.SCR  Screen 3
 0 ( EXPLIZITE MKQ - Initialisierung, Rahmenprogramm )
 1
 2 : MKQINI ( Ordnung, Totzeit —> )
 3     DUP d ! uhi !VDIM
 4     DUP n ! 2* DUP m(k) !VDIM
 5       DUP M'y !VDIM DUP M'M !MDIM 0 #REK !
 6     0. uhi VSET 0. m(k) VSET 0. M'y VSET 0. M'M MSET ;
 7
 8 : MKQ ( vecadr von TETA, Totzeit, Schrittzahl --> )
 9     >R OVER @VDIM 2/ SWAP MKQINI R>
10        0 DO get_u get_y MKQREK LOOP
11     M'M SWAP M'y Ax=b ;
12
13
14
15
```

Literatur

/1/ Isermann, R.
Prozessidentifikation
Springer Verlag Berlin 1974

/2/ Unbehauen, H.; Goehring, B.; Bauer, B.
Parameterschaetzverfahren zur Systemidentifikation
Oldenbourg-Verlag, Muenchen 1974

Verfasser: Dr.-Ing. Wolfgang Drewelow
Wilhelm-Pieck-Universitaet Rostock
Sektion Technische Elektronik
Albert-Einstein-Strasse 2
Rostock 6
DDR - 2500

Applikation von FORTH

Hartmut Pfüller

FORTH-Software fuer Seegangs-Messeinrichtung (ein Bericht)

Der Beitrag gibt eine Uebersicht ueber den Verlauf einer der ersten FORTH-Anwendungen an der WPU Rostock.

1. Einfuehrung

In der Seeschiffahrt spielen meteorologische Verhaeltnisse eine erhebliche Rolle. Deshalb werden z. B. Staerke und Richtung des Windes staendig mit Hilfe der allgemein bekannten Apparaturen erfasst. Aehnlich einfache Einrichtungen fuèr die Messung der Staerke des Seegangs als Komplex von Wellenhoehe und Wellendauer gibt es bisher nicht. Wegen der Wichtigkeit der Seegangsstaerke wird diese aber regelmaessig ins Logbuch eingetragen, wozu der entsprechende Wert jeweils geschaetzt werden muss. Von Fall zu Fall ist die Schiffbaubranche allerdings daran interessiert, die Bedingungen fuer Versuchsfahrten weitgehend zu objektivieren, d. h. Wellenhoehe und Wellendauer moeglichst genau zu messen. In diesem Sinne ergab sich fuer die Sektion Technische Elektronik der WPU Rostock die Moeglichkeit, fuer eine vom Schiffbau auf Mikroprozessorbasis konzipierte Seegangs-Messeinrichtung die Software-Entwicklung zu uebernehmen.

2. Technische Rahmenvorgaben

2.1. Messwerterfassung

Vorgesehen war das folgende Messverfahren: Eine Boje folgt freischwimmend der Wellenbewegung und erfasst mittels eines eingebauten Sensors die senkrechte Komponente der Beschleunigung. Der so registrierte Messwert wird ueber einen spannungsgesteuerten Oszillator in eine beschleunigungsproportionale Tonfrequenz umgewandelt und ueber einen in der Boje installierten Funksender zum Schiff uebertragen. An Bord wird in einem Funkempfaenger die beschleunigungsproportionale Tonfrequenz zurueckgewonnen und zur Verarbeitung an ein mikroprozessorgesteuertes Auswertegeraet weitergegeben.

2.2. Das Auswertegeraet

Als Herzstueck des Auswertegeraetes war die K1520-ZRE K2521 vorgesehen. Das Rechenprogramm war in den verfuegbaren 3 kByte EPROM unterzubringen. Fuer transiente Daten enthaelt die ZRE 1 kByte RAM. Das Geraet hatte die folgenden Auswertefunktionen zu erfuellen:

- Online-Berechnung der aktuellen senkrechten Bojenbewegung durch zweimalige Integration der aufgenommenen Beschleunigungswerte.
- Online-Ausgabe der berechneten Bojenbewegung ueber D/A-Umsetzer an einen y-t-Schreiber.
- Aufbereitung der erfassten Wellenbewegung nach verschiedenen Vorschriften zur Seegangsklassifikation.
- Darstellung der Klassifikationsergebnisse, der mittleren Wellendauer

und -hoehe sowie bei Bedarf der hoechsten Welle unter den letzten einhundert mittels zu multiplexender 7-Segment-Anzeigen.

- Bedienung von 7 einzelnen LEDs zur Betriebszustandsanzeige (z.T. blinkend, z.B. fuer die Meldung "Akku in Boje leer").

- Entgegennahme von Knopfdruck-Auswahlkommandos fuer verschiedene Anzeigemodi der 7-Segment-Anzeigen.

Um den Hardware-Aufwand moeglichst gering zu halten, wurde versucht, die geforderten Funktionen weitgehend softwaremaessig zu unterstuetzen.

Als Gefaess aus dem K1520-System kam ein Einschubrahmen fuer fuenf Steckplaetze zum Einsatz, in dem aber nur drei Plaetze belegt werden mussten:

- ZRE K2521
- D/A-Umsetzer zur Schreiberansteuerung
- Dekoder und Treibertransistoren fuer 7-Segment-Anzeigen.

3. Voraussetzungen fuer die Loesung der Aufgabe

Die rechentechnische Basis fuer die Bearbeitung der Aufgabe war ein K1520-Entwicklungsplatz mit EPROM-residentem Editor, Assembler und Debugger. Als Massenspeicher-Peripherie konnten Robotron-Digitalkassettengeraete K5200 benutzt werden.

In vorangegangenen Arbeiten waren auf dem genannten Entwicklungsplatz erste Versionen eines FORTH-Interpreters nach dem Modell von LOELIGER /1/ implementiert worden. Auf Basis dieses Interpreters konnte wenig spaeter von HALWASS /2/ ein FORTH-Targetcompiler fertiggestellt werden. Dieser Target-Compiler bot fuer die Bearbeitung der Aufgabe ausgezeichnete Voraussetzungen und kann grob etwa wie folgt charakterisiert werden:

- Vollstaendige Beibehaltung der interaktiven Benutzungsweise, so wie von herkoemmlichen FORTH-Systemen bekannt.

- Fest eingebauter Trace-Modus (also Einzelschrittabarbeitung von FORTH-Secondaries) zur Unterstuetzung bei Fehlersuche.

- Organische Zulassung des Datentyps "Fliesskommazahl" und Einbeziehung der entsprechenden Vereinbarungen (von Konstanten und Variablen) und Operationen fuer Fliesskommazahlen.

- Nach Ausloesen des Target-Uebersetzungsvorganges werden fuer den RAM-Anteil und den ROM-Anteil separate Zielcode-Segmente ausgegeben.

- Diese Code-Segmente liegen in verschieblichem HEX-Code vor und koennen jeweils fuer beliebige Zieladressen gelinkt werden.

- In den Zielcode werden nur die Ruempfe (Bodies) der benutzten FORTH-Worte uebernommen; die Koepfe (Header) werden ausnahmslos weggelassen.

- Die Code-Segmente enthalten ausschliesslich diejenigen FORTH-Befehle, auf die im zu uebersetzenden Programm tatsaechlich Bezug genommen wird. Auf diese Weise liegt als Uebersetzungsprodukt die wirklich minimal erforderliche Menge an Code vor.

Als Nachteile waeren aus heutiger Sicht lediglich zu nennen, dass der Umfang des uebersetzbaren Programms durch die Groesse des resident verfuegbaren RAM-Bereichs begrenzt wird und dass die Bezugnahmen auf das benutzte Betriebssystem die Portabilitaet des Target-Compilers erschweren.

4. Software-Konzept

4.1. Physikalisch-mathematische Aspekte der Wellenhoehenberechnung

Aus der Physik ist bekannt, dass durch zweimalige Integration der Beschleunigung eines Koerpers der waehrend der Integrationszeit zurueckgelegte Weg berechnet werden kann. Unklar ist allerdings zunaechst, wie vorzugehen ist, wenn die absolute Position errechnet werden soll und wie im vorliegenden Fall keine Informationen ueber die Anfangswerte der zu ermittelnden Groessen beschafft werden koennen.

Bei naeherer Betrachtung wird dann klar, dass sich der konkrete physikalische Sachverhalt (= staendiges Pendeln der Boje um das Nullniveau des Wasserspiegels) hier dadurch auszeichnet, dass die Mittelwerte der Vertikalkomponenten von Beschleunigung, Geschwindigkeit und Hoehe der Boje jeweils Null sein muessen. Die Berechnung des augenblicklichen absoluten Hoehenwertes der Boje wird hier moeglich, indem aus dem zufaelligen Anfangspunkt der Rechnung resultierende Gleichanteile von Geschwindigkeit und Hoehe erfasst und vor der Weiterintegration jeweils weggefiltert werden.

Da wegen der im Gesamtsystem enthaltenen messtechnischen und numerischen Abweichungen vom linear-kontinuierlichen Idealfall die Moeglichkeit von Drifterscheinungen der erwaehnten Gleichanteile besteht, ist es erforderlich, die Information ueber den Gleichanteil in einer Art Lernfunktion mit Vergessensrate laufend zu aktualisieren /3/.

Wenn eine gleichartige Behandlung auch mit dem Mittelwert der Beschleunigung vorgenommen wird, erreicht man durch Softwaremittel auf sehr bequeme Weise einen zuverlaessigen automatischen Nullpunktabgleich fuer die Beschleunigungs-Tonfrequenz-Kennlinie des spannungsgesteuerten Oszillators in der Boje.

4.2. Funktionelle Zerlegung der benoetigten Software-Anteile

Aus dem Anforderungsspektrum im Punkt 2.2 lassen sich drei weitgehend eigenstaendige Funktionen ableiten:

a) Multiplex-Vorgang der 7-Segment-Anzeigen
b) Online-Erfassung der Beschleunigungs-Messwerte und Berechnung der Vertikalbewegung der Boje sowie Ausgabe dieser Werte zum Analogschreiber
c) Fuehren der Statistik ueber alle Werte von Hoehe und Dauer der letzten einhundert Wellen sowie Anzeige der ueber Knopfdruck angewaehlten Statistikwerte.

Dazu jeweils einige kurze Bemerkungen:

Zu a: Hier handelt es sich um einen reinen Hilfsvorgang fuer die Anzeige-Hardware. Dafuer wurde ein CTC-Kanal der ZRE so programmiert, dass mit einer Frequenz von 300 Hz Interrupts erzeugt werden. Die zugehoerige Service-Routine schaltet nur die Anzeigestelle zyklisch weiter und ist in herkoemmlicher Assemblertechnik programmiert.

Zu b: Zur Erfassung der eingehenden Tonfrequenz als Mess-Signal fuer die vertikale Bojenbeschleunigung wurde ein zweiter CTC-Kanal als Frequenzzaehler eingesetzt, dessen Inhalt dann in einem festen Zeitrahmen ausgelesen werden musste. Aus der Gesamtheit der physikalischen, rechentechnischen und softwaremaessigen Randbedingungen ergab sich fuer diesen Zeitrahmen schliesslich eine optimale Taktdauer von 50 ms. Die volle Kapazitaet eines CTC-Zaehlregisters mit 8 bit Breite entsprechend n=255 erlaubt damit eine maximale Messtonfrequenz von 255*20 Hz = 5,1 kHz. Dass diese Frequenz nicht ueberschritten werden kann, war durch Auslegung der Messtrecke in der Boje mit Rueoksicht auf die maximal zu erwartende Beschleunigung zu sichern.

Fuer die Erzeugung des erwaehnten 50-ms-Taktes wurde ein dritter CTC-Kanal eingesetzt. Mittels der zugehoerigen Interrupts wurde eine Service-Routine gestartet, die in FORTH geschrieben war und die aufgefuehrten Online-Aufgaben vollstaendig zu erledigen hatte.

Zu c: Die Abarbeitung aller nicht zeitkritischen Funktionen wie Statistik und Anzeigebedienung wurde in einem Programmkomplex zusammengefasst, der als Hintergrundroutine fungierte und ebenfalls in FORTH programmiert war.

4.3. Ausprogrammierung in FORTH

Obwohl es im hier vorgegebenen Rahmen nicht moeglich ist, auf viele Einzelheiten des FORTH-Programms einzugehen, wird es im Anhang komplett abgedruckt, und zwar einmal um der Vollstaendigkeit des Gesamteindrucks willen und zum anderen, weil es heute schon fast wieder aus historischer Sicht interessant ist, den damaligen Stand im Verstaendnis der FORTH-Programmierung zu rekapitulieren.

Screen 1: Auffaellig ist allenfalls, dass im crossFORTH neben den ueblichen 2-Byte-Konstanten und Variablen auch die Vereinbarungen CVARIABLE fuer 1-Byte-Variable und FVARIABLE fuer 4-Byte-Fliesskommavariablen benutzbar sind.

Screen 2 und 3 enthalten versohiedene Sortierungen und Tabellenfuehrungen fuer die laufende Aktualisierung der gewuenschten Statistik-Kenngroessen im Hintergrund-Programm. (Das Wort I> lieferte im LOELIGER-Modell den Laufindex der DO ... LOOP - Schleife.)

Im Screen 4 fasst das Programm HINTERGRUND-AUSWERTUNG bereits die vier Hauptfunktionen der Statistikfuehrung buendig zusammen, indem bereitgestellt werden:

- Die Seegangsstaerken fuer alle vorgesehenen Klassifikationen,
- die Maximalhoehe unter den letzten hundert Wellen,
- der Mittelwert der 8, 10 bzw. 33 (klassifikationsbedingt) hoechsten Wellen unter den letzten hundert,
- der Mittelwert der Wellendauer derjenigen Wellen, die zur Berechnung des Hoehenmittels herangezogen wurden.

In den naechsten Screens schliesst sich die Realisierung verschiedener Anzeigevarianten an. Die Wortwahl ist offenbar nicht immer guenstig, die Kommentierung ist straeflich vernachlaessigt. (Nicht einmal die Stack-Effekte werden angezeigt!)

Dann wird im Screen 7 mit der Wahl der gelungenen Bezeichnungen AUSFILTERN und AUFINTEGRIEREN die Hoehenermittlung vorbereitet. Das

Wort AUSFILTERN enthaelt dabei die gleitende Mittelwertbildung mit Vergessensfaktor (/3/, S. 32 f.); leider auf wenig schoene Weise notiert.

Interessant ist - fuer Nichtkenner von FORTH - wie unbekuemmert am Ende von Screen 7 im fortlaufenden Quelltext die freie Mischbarkeit von Hochsprache und Maschinensprache benutzt wird. Das Wort D/A-UMSETZUNG wuerde in herkoemmlicher Assemblernotation so lauten:

```
E1        POP       HL
7D        LD        A,L
D3 85     OUT       (85H),A
```

(Ueber das Tor 85H der ZRE-PIO wurde der D/A-Umsetzer bedient.)

Screen 8 schliesslich enthaelt in fuer damalige Verhaeltnisse gelungener Notation im Wort WEGERMITTLUNG die komplette Durchfuehrung der Integrations- und Mittelungsoperationen nach Abschnitt 4.1. In der letzten Zeile der Definition wird der ermittelte Hoehenwert mit einem messstreckenspezifischen Umrechnungsfaktor multipliziert und anschliessend in eine Ganzzahl zurueckkonvertiert (FRECON).

An dieser Stelle ist eine Bemerkung zu der stereotyp grassierenden Behauptung angebracht, dass FORTH-Programme "schwer lesbar" seien. Wie der Leser sich selbst ueberzeugen kann, duerfte es kaum eine andere Programmiersprache geben, mit der die Funktion WEGERMITTLUNG auf aehnlich komfortable - schon fast natuerlichsprachliche Weise - ausgedrueckt werden kann. Es waere an der Zeit, auch in der Argumentation der Tatsache Rechnung zu tragen, dass - weitgehend unabhaengig von der Programmiersprache - die Lesbarkeit eines Programmes in erster Ordnung von zwei wesentlichen Punkten abhaengt, naemlich:

a) von der Vertrautheit des Lesers mit der Programmiersprache und
b) vom Geschick und vom guten Willen des Programmierers.

Denn, um das Gesagte an einer Ueberspitzung in der Umkehrung deutlich zu machen: Durch entsprechend irrefuehrende Wortwahl und Programmierung kann jedes Programm in jeder Programmiersprache weitgehend unlesbar gemacht werden.

Weiter enthaelt Screen 8 mit der FORTH-Interruptserviceroutine (50-MS-TAKT-ROUTINE) die Zusammenfassung aller wesentlichen Online-Funktionen.

Waehrend der Anfang von Screen 9 noch einmal Beispiele fuer die Benutzung von Maschinencode enthaelt, zeigt das endgueltige Programm WELLENREITER in einer Endlosschleife die HINTERGRUND-AUSWERTUNG und im ELSE-Zweig der IF-Anweisung die bei allen Unterbrechungen aktivierte 50-MS-TAKT-ROUTINE mit der am Anfang desselben Screens programmierten Rueckkehr aus dem Interrupt (RUECK).

5. Erprobung

5.1. Laborerprobung

Das uebersetzte und gelinkte Programm belegte wie vorgegeben weniger als 3 kByte Speicher und wurde so auf die ZRE-EPROMs gebracht. In der ersten Laborerprobung lieferte das Programm sehr haeufige Abstuerze, fuer die schliesslich zwei Fehlerursachen lokalisiert wurden:

a) Beim Eintritt in die FORTH-Interruptserviceroutine wurde das IY-Register nicht initialisiert, das im LOELIGER-Modell zwar durchgaengig als Ruecksprungziel (NEXT) in den Adressinterpreter gilt, in einzelnen Arithmetikprogrammen aber doch kurzzeitig anderweitig verwendet wird.

b) In den DO-Schleifen-Handlern *CDO und *DO wird bei LOELIGER der Returnstack nicht wiedereintrittsfaehig verwaltet (/1/, S. 114 f.). Dieser Defekt ist durch Umstellen der beteiligten Anweisungen behebbar.

Nach Beseitigung dieser Fehler lieferte das Programm fuer im Labor simulierte Beschleunigungswerte augenscheinlich sinnvolle Ergebnisse. Insbesondere konnte am y-t-Schreiber deutlich verfolgt werden, wie das Programm nach jeweils kurzer Einschaltzeit zuverlaessig "lernte", die die Integration stoerenden Mittelwerte wegzufiltern.

5.2. Erste See-Erprobung

Die erste See-Erprobung fand ohne die Bearbeiter seitens der WPU statt und lieferte voellig sinnlose Ergebnisse. Ueber die Ursachen konnte nur nachtraeglich spekuliert werden. Klar war lediglich, dass es dem Programm auch bei sehr langer Laufzeit offensichtlich nicht gelang, die auszufilternden Mittelwerte zu finden oder zu halten.

5.3. Vorbereitung der zweiten See-Erprobung

Fuer die zweite See-Erprobung des Geraetes wurde die Anwesenheit eines WPU-Vertreters vorgesehen. Weiter sollte wenigstens fuer die Erprobungsphase der Nachteil abgestellt werden, dass der Programmablauf im abgeschlossenen Geraet nicht verfolgt werden konnte. Zu diesem Zweck wurde kurzfristig eine Moeglichkeit gefunden, das eigentliche Einzweck-Auswertegeraet in einen FORTH-Dialogrechner umzubauen. Dazu wurden die beiden freien Steckplaetze benutzt, um zusaetzlich zwei K1520-Karten unterzubringen:

- Video-Karte
- 6 kROM/ 2 kRAM,

wobei auf die EPROMS der ROM/RAM-Karte eine "normale" interaktive FORTH-Version gebracht wurde, die ihrerseits einen Bildschirm ueber die Video-Karte und eine serielle Tastatur ueber die ZRE-PIO bediente. Im Cross-Kompilat auf den ZRE-EPROMs wurde die Adressreferenz zur HINTERGRUND-AUSWERTUNG (in Screen 8) durch einen Zeiger zum Textinterpreter des Dialog-FORTH ersetzt. Damit ergab sich der Effekt, dass das Dialog-FORTH staendig als Hintergrund arbeitete, in der FORTH-Interruptserviceroutine (50-MS-TAKT-ROUTINE) aber die Online-Messauswertung durchgefuehrt wurde. Damit war es moeglich, vom Dialog-FORTH aus die interne Arbeitsweise des Cross-Kompilats bis in alle Einzelheiten waehrend des Echtzeitbetriebes zu inspizieren.

5.4. Zweite See-Erprobung

Bei der Arbeit mit dem um die Dialogmoeglichkeiten erweiterten Geraet erwies sich die erreichte Flexibilitaet als weit groesser, als vorher abgesehen worden war. Die Eigenschaften von FORTH erlaubten nicht nur das passive Inspizieren der Online-Arbeitsweise des Cross-Kompilats, sondern es war auch moeglich, beliebige Funktionen daraus wieder in

das Dialog-FORTH einzubinden (ueber vektorierte Ausfuehrung der zugehoerigen Codefeldadresse).

Unter Zuhilfenahme dieser Dialogmoeglichkeiten konnten im See-Einsatz relativ schnell zwei Ursachen fuer das Fehlverhalten gefunden werden:

a) Durch unguenstige Auslegung der Messstrecke in der Boje wurde zuweilen die zulaessige Tonfrequenz (nach Punkt 4.2.b) ueberschritten, so dass immer wieder voellig falsch uebermittelte Beschleunigungswerte die Stabilisierung der Mittelwerte verhinderten.

b) Nach Abstellen dieses Fehlers zeigte sich, dass der entsprechend den Laborbedingungen im Interesse hoher Messgenauigkeit sehr schwach eingestellte Vergessensfaktor fuer die Mittelwertbildung zu starke Empfindlichkeit gegen Stoerungen verursachte. Um unter realen Bedingungen ein ausreichend robustes Verhalten zu sichern, wurden fuer den Vergessensfaktor guenstigere Werte als Kompromiss zwischen Stabilitaet und Genauigkeit ausprobiert.

Von da an arbeiteten Geraet und Software durchweg stoerungsfrei. Die Ergebnisse fuehrten zu einer Veroeffentlichung in der Zeitschrift Seewirtschaft /4/.

Literatur

/1/ R. G. Loeliger:
Threaded Interpretive Languages.
BYTE BOOKS Peterborough 1981

/2/ M. Halwass:
crossFORTH.
WPU Rostock, Sektion Technische Elektronik 1983
(unveroeffentlicht)

/3/ W. Schoenborn, K. Fritzsch, G. Stanke:
Lernverfahren fuer technische Systeme.
Akademie-Verlag Berlin 1983

/4/ A. Kiekbusch:
Wellenhoehen- und Seegangsmesseinrichtung.
Seewirtschaft 18(1986) H. 2, S. 80-81

Verfasser: Dr.-Ing. Hartmut Pfüller
Wilhelm-Pieck-Universitaet Rostock
Sektion Technische Elektronik
Albert-Einstein-Strasse 2
Rostock 6
DDR - 2500

Anhang: Programm-Listings

```
( BLOCK 1     WELLENREITER     WPU/PWW      22.6.83 )   HEX
 C09  CONSTANT ROHBESCHLEUNIGUNG      0 CVARIABLE STARTVERTEILER
 C01  CONSTANT DAUER                  0 CVARIABLE ZEIGER
 CDC  CONSTANT SORTLISTE              0  VARIABLE TIEF
 C14  CONSTANT HOEHENABLAGE           0  VARIABLE GANZAHL
 C0E  CONSTANT <SEEGG>              100  VARIABLE GBASIS
 C05  CONSTANT <SEEGANGSDISPLAY>      0  VARIABLE ZEIT
 C78  CONSTANT ZEITABLAGE             0  VARIABLE ALTZEIT
 C07  CONSTANT TASTERADRESSE          0  VARIABLE ALTHOEHE
 C06  CONSTANT <LED'S>                0  VARIABLE HOCH
 C0A  CONSTANT <MITTELHOEHE>      187.5 FVARIABLE MITTELBESCHL.
 C0C  CONSTANT <MAXIMALHOEHE>        0. FVARIABLE GESCHWINDIGKEIT
 C03  CONSTANT <HOEHE>               0. FVARIABLE MITTELGESCHWDKT
  D2  CONSTANT I                     0. FVARIABLE HOEHE
  DC  CONSTANT II                    0. FVARIABLE MITTELHOEHE
                                     0. FVARIABLE FANZAHL


( BLOCK 2     WELLENREITER      WPU/PWW     14.6.83 ) DECIMAL

: UMSPEICHERN   100 0
  DO HOEHENABLAGE I> + C@ SORTLISTE I> + C!
  LOOP ;
: SORTIERUNG   33 0
  DO SORTLISTE I> + SORTLISTE 100 + OVER
     DO DUP C@ 255 AND I> C@ 255 AND <
        IF DUP C@ I> C@ SWAP I> C! OVER C!
        THEN
     LOOP DROP
  LOOP ;
: CMITTEL   SWAP OVER OVER + SWAP 0 RROT
    DO I> C@ 255 AND +
    LOOP
  SWAP / ;


( BLOCK 3     WELLENREITER     WPU/PWW     21.6.83 )

: TABELLE   10 0
    DO OVER I> + C@ 255 AND > NOT
      IF DROP I> LEAVE
      THEN
    LOOP
  <SEEGG> ;
: SEEGANG   UMSPEICHERN SORTIERUNG
  SORTLISTE  8 CMITTEL I TABELLE     C!
                      II TABELLE 3 + C! DROP
  SORTLISTE 10 CMITTEL I TABELLE  1+ C!
                      II TABELLE 4 + C! DROP
  SORTLISTE 33 CMITTEL I TABELLE 2 + C!
                      II TABELLE 5 + C! DROP ;
: DISPLAYKONVERTIERUNG   10 /MOD SWAP CJOIN ;
```

```
( BLOCK 4     WELLENREITER     WPU/PWW     21.6.83 )

: MAXIMALHOEHE   SORTLISTE C@ 255 AND
  10 51 */ DISPLAYKONVERTIERUNG <MAXIMALHOEHE> ! ;
: HOEHENMITTELUNG   HOEHENABLAGE 100 CMITTEL
  10 51 */ DISPLAYCONVERTIERUNG <MITTELHOEHE> ! ;
: DAUERMITTEL   ZEITABLAGE 100 CMITTEL 99 MIN
  DISPLAYKONVERTIERUNG DAUER ! ;
: HINTERGRUND-AUSWERTUNG
  SEEGANG MAXIMALHOEHE HOEHENMITTELUNG DAUERMITTEL ;
: MASKENUEBERTRAGUNG   DUP <R AND DUP 0=
    IF R> DROP DROP
    ELSE R> 255 XOR <LED'S> C@ AND IOR <LED'S> C!
    THEN ;
: AUSWAHL-8/10/33   DUP 56 MASKENUEBERTRAGUNG ;
: SEEGANG/I-II   DUP 192 MASKENUEBERTRAGUNG ;


( BLOCK 5     WELLENREITER     WPU/PWW     21.6.83 )   HEX

: WERTANZEIGE   <LED'S> C@ 4 AND
    IF <MAXIMALHOEHE>
    ELSE <MITTELHOEHE>
    THEN
  @ <HOEHE> ! <LED'S> C@ 40 AND
    IF 3
    ELSE 0
    THEN
  4 3 0
    DO 2* DUP <LED'S> C@ AND
      IF DROP I> LEAVE
      THEN
    LOOP
  + <SEEGG> + <SEEGANGSDISPLAY> C! ;


( BLOCK 6     WELLENREITER     WPU/PWW     21.6.83 )

: MAXIMUM/J-N   DUP 4 AND
    IF <LED'S> C@ 4 IOR
    ELSE <LED'S> C@ FB AND
    THEN
  <LED'S> C! ;
: BEDIENUNG/EIN-AUSGABE   TASTERADRESSE @ 2* 2*
  MAXIMUM/J-N AUSWAHL-8/10/33 SEEGANG/I-II WERTANZEIGE DROP
: ABLAGE   ZEIGER C@ HOEHENABLAGE + C! ZEIT @ DUP ALTZEIT @
  - 2/ FF MIN ZEITABLAGE ZEIGER C@ + C! ALTZEIT !
  HOCH OSET TIEF OSET GBASIS @ 800 < IF 1 GBASIS +! THEN ;
: ZYKLUSSCHALTUNG   ZEIGER C@ 1+ DUP 64 =
    IF DROP 0
    THEN
  ZEIGER C! ;
```

```
( BLOCK 7     WELLENREITER      WPU/PWW       22.6.83 )

: PROTOKOLLIERUNG    DUP 0=
  IF DROP
  ELSE ALTHOEHE @ 0 >
     IF DUP 0<
        IF ZYKLUSSCHALTUNG HOCH @ TIEF @ - FF MIN ABLAGE THEN
     THEN DUP DUP HOCH @ MAX HOCH ! TIEF @ MIN TIEF ! ALTHOEHE !
  THEN ;
: AUSFILTERN    DUP <R F@ F- FDUP FANZAHL F@ F/ R>
  DUP <R F@ F+ R> F! ;
: AUFINTEGRIEREN    DUP <R F@ F+ FDUP R> F! ;
CREATE BLINKTAKT    82DB , 8E6 , 26 , B56F , NEXT
: BLINKT    <LED'S> C@ SWAP BLINKTAKT
    IF FF XOR AND ELSE IOR THEN <LED'S> C! ;
CREATE D/A-UMSETZUNG    7DE1 , 85D3 , NEXT


( BLOCK 8     WELLENREITER      WPU/PWW       21.6.83 )

: ZAEHLEN    GANZAHL @ GBASIS @ <
    IF 1 GANZAHL +! GANZAHL @ FCON FANZAHL F! THEN ;
: WEGERMITTLUNG    ZAEHLEN          MITTELBESCHL.    AUSFILTERN
 GESCHWINDIGKEIT AUFINTEGRIEREN MITTELGESCHWDKT AUSFILTERN
 HOEHE           AUFINTEGRIEREN MITTELHOEHE     AUSFILTERN
 0.01228797 F* FRECON ;
: ANALOGAUSGABE    DUP -80 MAX 7F MIN 80 + D/A-UMSETZUNG ;
: 50-MS-TAKT-ROUTINE    1 ZEIT +!
  ROHBESCHLEUNIGUNG C@ FF AND DUP 6E <
    IF DROP 1 BLINKT
    ELSE <LED'S> C@ FE AND <LED'S> C!
         FCON WEGERMITTLUNG ANALOGAUSGABE PROTOKOLLIERUNG
    THEN
  BEDIENUNG/EIN-AUSGABE ;


( BLOCK 9     WELLENREITER      WPU/PWW       21.6.83 )

CREATE INTERRUPTZULASSUNG FB C, NEXT
CREATE RUECK    C3 C, C1 , NEXT
: WELLENREITER    STARTVERTEILER C@ NOT
    IF FF STARTVERTEILER C!
      SORTLISTE 1+ TASTERADRESSE
        DO I> COSET LOOP
      48 <LED'S> C! DAUER OSET INTERRUPTZULASSUNG
        BEGIN 2 BLINKT ZEIGER C@ 63 =
        END <LED'S> C@ FD AND <LED'S> C!
        BEGIN HINTERGRUND-AUSWERTUNG 0
        END
    ELSE
      50-MS-TAKT-ROUTINE RUECK
    THEN ;
```

Applikation von FORTH

Rainer Korpal

Prozessteuerung "Brauerei"

1. Prozessbeschreibung

Bei der Herstellung von Bier hat die moeglichst exakte Einhaltung vorgegebener Temperatur-Zeit-Verlaeufe entscheidenden Einfluss auf die Qualitaet. Seit einigen Jahren erfolgt die Herstellung der Standardbiere in der Rostocker Brauerei nach dem Reaktorverfahren. Fuer den Reaktorkeller II, in dem 22 Reaktoren mit jeweils einem maximalen Inhalt von 2500 hl zusammengefasst sind, wird seit 1984 die Temperaturregelung fuer die Reaktoren mittels Mikrorechner realisiert. Die alkoholische Gaerung ist ein exothermer Prozess, so dass in den Prozesstufen Gaerung und Reifung eine Waermeabfuhr erforderlich ist. Dies erfolgt ueber eine gesteuerte Zu- bzw. Abschaltung der Reaktoren an den Kuehlkreislauf.

Neben der Temperaturregelung der Reaktoren uebernimmt der Mikrorechner noch folgende Funktionen:

- Uebersichtsdarstellung des Reaktorkellers mit Anzeige der augenblicklich in den einzelnen Reaktoren ablaufenden Prozesstufen.
- Wahlweise Anzeige des Zustandes eines einzelnen Reaktors
- Journalfunktion fuer die Prozesstufen Gaerung, Reifung und Reinigung (Temperaturverlaeufe und prozesspezifische Daten)
- Steuerung der Temperatur-Zeit-Verlaeufe bei der Gaerung

Alle Darstellungen koennen zusaetzlich ueber einen Typenraddrucker als Hardcopy ausgegeben werden. Die prozesspezifischen Angaben mit Ausnahme der Isttemperaturen in den einzelnen Reaktoren und der Endzeit der Gaerung werden im Dialog eingegeben und koennen zum Teil nachtraeglich modifiziert werden.

Der Mikrorechner wurde aus OEM-Karten des Kombinat VEB robotron zusammengestellt, die Prozessanschlusskarten aus dem reichhaltigen Nachnutzungsangebot ausgewaehlt. Der Dialog erfolgt ueber Bildschirm (MON 1) und Tastatur (K 7634). Zur Abspeicherung der Prozessdaten wird ein gestuetzter CMOS-RAM eingesetzt, um nach Netzausfall einen Wiederanlauf zu ermoeglichen. Ein Massenspeicher wurde nicht vorgesehen. Der Einsatz der ZRE K 2521 ergibt eine unguenstige Speicherstruktur, die bei der Programmgestaltung beruecksichtigt werden muss.

2. Programmstruktur

Das Programm besteht aus einem Hintergrundprogramm und einem Vordergrundprogramm, das ueber Interrupt von der CTC im Sekundenabstand aktiviert wird. Das Hintergrundprogramm ist als Endlosschleife, in der staendig die Tastatur abgefragt und alle 30 s die Anzeige aktualisiert wird, aufgebaut. Auf den Einsatz eines Echtzeitbetriebssytems wird verzichtet, um den Aufwand gering zu halten. Dies ist zulaessig, da keine asynchronen Prozessignale behandelt werden muessen. Es gibt nur zeitliche Ereignisse, die im Vordergrundprogramm ueber getrennte Zaehlzellen behandelt werden. Durch den Prozess werden geringe Anforderungen an die Echtzeitfaehigkeit des Programms gestellt. Beispielsweise betraegt die Abtastzeit bei der Temperaturregelung 15

min. Alle zeitlichen Ereignisse werden mit Minutengénauigkeit erfasst. Das gesamte Programm wurde in Teamarbeit von 2 Kollegen programmiert. Einer bearbeitete das Vordergrund- und der andere das Hintergrundprogramm. Voraussetzung fuer die Teamarbeit ist die Zerlegung der Aufgabe in moeglichst lose gekoppelte Teilkomplexe. Je geringer die Kopplung ist, um so kleiner ist der Koordinierungsaufwand zwischen den Bearbeitern, um so einfacher lassen sich die Schnittstellen zwischen den Teilprogrammen definieren und um so geringer ist der Programm-Overhead, der durch ungenuegende Abstimmung zwischen den Programmierern entsteht. In diesem Fall war es relativ einfach. Das Hintergrundprogramm realisiert alle Dialogfunktionen und das Vordergrundprogramm alle zeitlich bezogenen Aufgaben. Als Schnittstelle zwischen beiden Programmen dient eine exakt definierte Datenbasis ueber alle Reaktoren. Fuer jeden Reaktor existiert ein Datensatz konstanter Laenge. Das erste Element in diesem Datensatz kennzeichnet den Prozesschritt, in dem der Reaktor sich gerade befindet (Technologie = Befuellen, Gaerung, Reifung, Schlauchen, Filtrieren, Reinigung, Leerzustand), und das zweite Datenelement enthaelt die momentan gemessene Isttemperatur des Reaktors. Die weitere Struktur des Datensatzes ist aus Effektivitaetsgruenden abhaengig von der momentanen Technologie, aber einheitlich fuer alle Reaktoren. Nach Festlegung der einzelnen Datenstrukturen konnten beide Programme unabhaengig voneinander bearbeitet werden.

Der Programm-Overhead, der bei Teamarbeit durch das mehrfache Programmieren gleicher Programmpassagen fast unvermeidlich ist, wird durch die Bereitstellung vieler Grundworte in FORTH erfreulich klein gehalten.

Da zum Zeitpunkt der Programmerstellung noch kein Assembler unter FORTH verfuegbar war, wurden alle Maschinenkodepassagen (physische E/A-Treiber, Registerrettung und -rueckrettung bei Interrupt, Initialisierung, Kaltstart u. ae.) in Assemblersprache geschrieben und getrennt uebersetzt. Aehnlich wurden auch die Zeichenketten der darzustellenden Texte vereinbart.

3. Programmbeschreibung

Nachfolgend werden einige ausgewaehlte Screens von den 45 Screens, aus denen das Programm insgesamt besteht, naeher erlaeutert. Ziel ist es, die Vorgehensweise bei der Bottom-up-Programmierung mittels FORTH zu veranschaulichen und die Wohlstrukturiertheit und Verstaendlichkeit des entstehenden Programms zu zeigen. Ausserdem wird die Maechtigkeit der Ausdrucksmittel von FORTH sichtbar. Da es keine Beschraenkungen bei der Wahl der Namen der zu definierenden Worte gibt, kann der Programmierer selbst im grossen Umfang die Lesbarkeit seiner Programme beeinflussen.

Screen 1

Durch die Definition des Wortes TABELLE wird ein neues Definitionswort geschaffen und bei der Definition der Liste KALENDER, die die um 1 erhoehte Anzahl der Tage des jeweiligen Monats enthaelt, gleich benutzt. Diese wird bei der Eingabe des Datums zum Test auf Korrektheit und beim Uhrprogramm zum Weiterschalten auf den naechsten Monat benoetigt. In Zeile 7 wird das Definitionswort CONSTANT in === umbenannt. Ziel ist eine Verringerung des Schreibaufwandes. Zur Laufzeit des Programmes hat diese Umbenennung keinen Einfluss auf die Ausfuehrungszeit. == ist ein Definitionswort fuer Bytekonstanten.

Durch die Einfuehrung dieses Datentyps wird pro Definition 1 Byte gegenueber der Wortkonstanten eingespart. Zeile 8 enthaelt die Definitionen von Konstanten. CRAM ist die Anfangsadresse des CMOS-RAMs. PFL ist die Laenge des Datensatzes eines Reaktors. Die in den restlichen Zeilen vereinbarten Konstanten dienen als Zeiger auf Variable bzw. Vektoren. Die in FORTH uebliche Variablendeklaration mittels Def.-Wort VARIABLE konnte nicht benutzt werden, da die dann belegten Speicherplaetze Bestandteil des Woerterbuchs werden, dieses aber bei dieser Anwendung spaeter im EPROM abgelegt wird. Bei dem benutzten Verfahren ist der Programmierer fuer die Konsistenz der Variablen selbst verantwortlich. Nachherein scheint die Schaffung neuer Def.-Worte, die dies automatisch sichern, sinnvoller. PUFFER ist der Vektor, der die Prozessdaten der Reaktoren enthaelt.

Screen 2

In Zeile 2 werden weitere Konstanten vereinbart. ROM ist die Anfangsadresse des EPROMs, der die Ausgabetexte enthaelt, NRAM ist die Anfangsadresse eines RAM-Bereiches fuer Variable, die bei Netzausfall nicht gesichert werden muessen, und .BILD ist die Anfangsadresse des Bildwiederholspeichers. Der Bildschirm wird in zwei logische Bereiche (Anzeigebereich und Kommandobereich) eingeteilt. Zur Verwaltung der Kursorposition werden daher zwei Variable (KURSOR fuer den Anzeigebereich und AKURSOR fuer den Kommandobereich) benoetigt. In den Zeilen 6 - 15 werden Zeiger auf einzelne Textketten vereinbart. Durch die Verwaltung der Textketten ueber Zeiger ergibt sich die Moeglichkeit, auch nur einen bestimmtem Teil einer Textkette auszugeben.

Screen 3

Die hier vereinbarten Zeiger widerspiegeln die Struktur des Datensatzes eines Reaktors in Abhaengigkeit von der entprechenden Technologie. Die Laenge des Datensatzes betraegt 164 Byte (PFL), so dass Bytekonstanten ausreichend sind.

Screen 4

Die Eingabe bzw Aenderung von Parameter fuer einen Reaktor kann nicht direkt in der Datenbasis erfolgen, da die einzelnen Parameter nicht unabhaengig von einander sind und ein unvollstaendiger Datensatz zum Absturz des Vordergrundprogramms fuehren kann. Deshalb wird bei Ausfuehrung des Kommandos "Parametereingabe" der Datensatz des jeweiligen Reaktors in einen Dialogpuffer umgeladen. Dort erfolgt dann die Neueingabe bzw. Aenderung und erst nach korrektem Abschluss wird der geaenderte Datensatz in einer unteilbaren Aktion in die Datenbasis eingetragen. Die definierten Zeiger ermoeglichen den Zugriff auf einzelne Datenelemente entsprechend der Technologie.

Screen 5

Zur Darstellung der Daten eines Reaktors werden verschiedene Zahlenformate benoetigt. .RO liefert eine rechtsbuendige Integerzahl. Das Wort ZAHL realisiert die vorbereitenden Aktionen fuer die Konvertierung, waehrend ZAHLAUS die entstandene Zeichenkette dann an der in KURSOR stehenden Bildschirmposition zur Anzeige bringt. Darin unterscheidet sich .RO von dem sonst ueblichen Wort .R . Die Einbeziehung von >R in ZAHL bzw. R> in ZAHLAUS ist nicht moeglich, da

>R und R> in einer Definition auftreten sollten. .R1 liefert eine rechtsbuendige Zahlendarstellung mit einer Stelle nach dem Komma und darzustellenden Wert mit OFFFFH und erhoeht bei Uebereinstimmung nur die Kursorposition um die Spaltenbreite, ohne den Wert darzustellen, waehrend es sich sonst wie .R1 verhaelt. Damit hat man die Moeglichkeit, in einem Vektor den Wert OFFFFH als Dummy-Wert einzutragen und erspart sich so unterschiedliche Darstellungsroutinen. Die Darstellung von Minuten, Stunden, Tag und Monat soll ohne Unterdrueckung der Vornullen erfolgen. .2. realisiert dieses Darstellungsformat. In Zeile 12 werden zwei Maschinenkodepassagen erzeugt, die Bit 7 eines Speicherplatzes setzen (SETK) bzw. loeschen (CLEARK). In den Worten SKURSOR und CKURSOR werden sie benutzt, um auf dem Bildschirm die naechste Eingabeposition zu markieren bzw. diese Marke wieder zu loeschen. BILDLOE loescht den Bildschirm durch Ueberschreiben mit Leerzeichen (BLANKS). STC wandelt die relativ bezueglich der Anfangsadresse des Bildschirms angegebene Position im Anzeigebereich in eine absolute Adresse des Bildwiederholspeichers um.

Screen 6

Mit den hier definierten Worten wird der Zustand eines einzelnen Reaktors zur Anzeige gebracht. Es erfolgt eine Trennung in allgemeine Daten, die unabhaengig von der Technologie sind und in technologieabhaengige Daten. Die Worte 0BILD, 1BILD, 2BILD, 3BILD, 4BILD, 5BILD und 6BILD realisieren die Darstellung der technologieabhaengigen Daten. Sie bestehen aus der Anzeige des Namens der Technologie (Zeiger auf entsprechende Textkette und Ausgabewort BILD*) und dem von der Technologie abhaengigen Darstellungsformat, auf das hier nicht weiter eingegangen werden soll. Einige Technologien haben das gleiche Darstellungsformat (1TECH) und die Technologie "Leer" enthaelt keine weiteren Anzeigen. CASE: ist ein Def.-Wort zur Definition von Fall-Entscheidungen in einer sehr einfachen Form. Ueber die Fall-Entscheidung TECHNOLOGIE erfolgt die Auswahl des technologieabhaengigen Darstellungsformates. BILD realisiert die vollstaendige Anzeige des Zustandes eines Reaktors. Zuerst werden allgemeine Texte ausgegeben (relative Position, wo der Text erscheinen soll; Zeiger auf den Text; Ausgabewort). Dann werden die als Parameter uebergebene Reaktornummer (intern 0..21) dargestellt (F STC 1+ 3 .R0), der dazugehoerige Datensatz ermittelt (PFL * PUFFER +) und die Isttemperatur dargestellt sowie die technologieabhaengigen Daten angezeigt.

Screen 7

DRUCKERINIT ist der Aufruf einer Maschinenkodepassage zur Initialisierung der Druckerschnittstelle und DRUCKER realisiert eine Einzelzeichen-Ausgabe auf dem Drucker. HARDCOPY gibt den Bildschirminhalt auf dem Drucker aus. Die Variable NUMBILD enthaelt die Nummer des darzustellenden Reaktors bzw. den Wert OFFH, wenn die Uebersichtsdarstellung gewuenscht wird. Falls der Zustand eines bestimmten Reaktors gedruckt werden soll, wird dieser zuerst zusammen mit Datum und Uhrzeit auf dem Bildschirm dargestellt. Dann wird zeilenweise Zeichen fuer Zeichen der Bildwiederholspeicher gelesen und gedruckt. Die 16. Zeile (Kommandozeile) wird nicht mit ausgegeben. Es wird ein Drucker vom Typ 1152 eingesetzt. Bei diesem Typ sind der Buchstabe "O" und die Ziffer Null nicht zu unterscheiden, so dass eine Null durchstrichen dargestellt werden muss. Deshalb wird jedes zu druckende Zeichen geprueft, ob es eine Null (ASCII = 30H) ist. Wenn ja, wird ein Schraegstrich (ASCII = 2FH) gedruckt, dann der Druckkopf um eine Position zurueckgesetzt (ASCII = 08) und die Null gedruckt. Nach jeder

Zeile werden CR (ASCII = ODH) und LF (ASCII = OAH) ausgegeben.

Screen 8

Dieser und die nachfolgenden Screens zeigen, wie die Eingabe ueber die Tastatur organisiert wird. Es existiert ein Puffer TEXTPUFFER, in dem alle Zeichen von der Tastatur nacheinander eingetragen werden, bis die Eingabe mittels Taste ENTER fuer gueltig erklaert wird. Der Puffer wird durch einen Zeiger, der sich auf dem ersten Speicherplatz des Puffers befindet, verwaltet. Das aufrufende Wort muss vor Eintritt in den logischen Tastaturtreiber die Adresse des Auswerteprogramms in die Variable KOMMANDO? und die maximale Laenge der einzugebenden Zeichenkette in die Variable LAENGE eintragen. Das Wort GETCHAR prueft, ob die Eingabe fuer abgeschlossen erklaert wird (ENTER - Kodierung 9DH) und bringt in dem Fall das Wort, dessen Adresse in der Variable KOMMANDO? steht, zur Ausfuehrung. Weiter wird geprueft, ob BS (Kodierung 07) eingegeben wird und wenn ja, ob dies ausgefuehrt werden kann. Zur Ausfuehrung wird der Zeiger von TEXTPUFFER um 1 verringert, der Kursor um eine Position zurueckgesetzt und die neue Eingabeposition geloescht. Wenn weder ENTER noch BS eingegeben wird, wird geprueft, ob die Zeichenkette ihre maximale Laenge (Variable LAENGE < Zeiger) erreicht hat. Wenn ja, wird das Zeichen ignoriert. Anderenfalls wird das Zeichen im Puffer eingetragen, auf dem Bildschirm dargestellt und die Eingabe des naechsten Zeichens vorbereitet.

Screen 9

TASTATUR ist der physische Tastaturtreiber, der bei betaetigter Taste einmal den Zeichenkode und sonst eine Null liefert. *TAST ist der logische Tastaturtreiber, der erst verlassen wird, wenn das Auswerteprogramm die Fertigmeldung (Variable FERTIG) gesetzt hat.

Screen 10

Am Beispiel der Eingabe der Uhrzeit wird die Vorgehensweise bei der Parametereingabe gezeigt. UHRZEIT ist das Rahmenprogramm. VORBE bereitet die Eingabe vor. Die Eingabezeile und der Textpuffer werden geloescht und der Kursor auf die erste Eingabeposition gesetzt. Dann wird der Anforderungstext (3CO STC 29TX BILD*) ausgegeben und der logische Tastaturtreiber aufgerufen. Nach Abschluss der Eingabe wird geprueft, ob die Eingabe korrekt ist (FERTIG C@ 2 =). Wenn ja, werden die Werte fuer Minuten und Stunden abgespeichert, sonst wird die Eingabe wiederholt. UHRSET ist das Auswerteprogramm fuer die Eingabe der Uhrzeit. Die Variable FERTIG wird auf 1 gesetzt, damit anschliessend der logische Tastaturtreiber verlassen werden kann. Dann wird geprueft, ob zwei Werte, getrennt durch Leerzeichen, eingegeben sind, diese separiert und in Binaerzahlen konvertiert. 1TXZEIGER und 2TXZEIGER enthalten nach Abarbeitung von SUCHE Anfang und Ende der Zeichenkette eines Parameters. HOLE transportiert die Zeichenkette auf den Stapel und KONV fuehrt die Konvertierung in eine Binaerzahl aus. Abschliessend werden die eingegebenen Werte auf Korrektheit geprueft und gegebenenfalls die Variable FERTIG um 1 erhoeht.

Screen 11

Dieser und der nachfolgende Screen enthalten alle Worte zur Erzeugung der Uebersichtsdarstellung. Bei den Technologien "Leer" und "Reini-

gung" wird nur die Isttemperatur angezeigt. Fuer "Filtrieren", "Schlauchen" und "Befuellen" werden zusaetzlich die Sorte und die Fuellmenge und ausserdem bei "Gaerung" und "Reifung" das Anfangsdatum angezeigt. UEIST stellt die Isttemperatur und UEMENGE die Fuellmenge dar. UESORTE bringt die Sortenbezeichnung, die als Textkette im Datensatz des jeweiligen Reaktors abgespeichert ist, zur Anzeige. Fuer die Uebersichtsdarstellung wird die Textkette auf max. 8 Zeichen beschraenkt. UEDAT ist das Ausgabeformat fuer ein Datum ohne Jahreszahl. In den Zeilen 12 bis 15 werden die Worte zur Darstellung der einzelnen Technologien definiert, wobei die Technologie durch ihren Anfangsbuchstaben (ASCII-Kodierung) dargestellt wird.

Screen 12

UETECH ist eine Fall-Entscheidung zur Darstellung der Werte entsprechend der jeweiligen Technologie. Die Uebersichtsdarstellung erfolgt in zwei Spalten, einmal Reaktor 1 bis 11 und zum anderen Reaktor 12 bis 22. UEBER realisiert die Darstellung einer Spalte und erfordert die Parameter: relative Bildschirmposition, Datensatzadresse, letzter Reaktor+1, erster Reaktor. Datensatzadresse und Bildschirmposition werden hochgezaehlt und als Ausgabeparameter zurueckuebermittelt. UEBERSICHT realisiert dann die ganze Uebersichtsdarstellung. Nach Ausgabe der Ueberschrift, die sich aus zwei Textketten zusammensetzt (31TX und 2TX) und fuer jede Spalte erfolgt, wird die erste Spalte (Parameter: CO PUFFER C 1) und dann die zweite Spalte angezeigt. Fuer die Darstellung der zweiten Spalte wird die Datensatzadresse, die vom ersten Aufruf von UEBER stammt, benutzt, waehrend die Bildschirmposition neu angegeben wird (SWAP DROP EO SWAP).

Screen 13

HINTERGRUND ist das Hintergrundprogramm. Nach Loeschen des Bildschirms wird geprueft, ob es sich um einen Neustart oder um einen Wiederstart nach Netzausfall handelt. Die Variable ?RESTART muss ein bestimmtes Bitmuster (BLACKOUT), das sich mit grosser Wahrscheinlichkeit nicht zufaellig beim Einschalten im CMOS-RAM einstellt, enthalten, damit die Wiederstartroutine, die fuer das Hintergrundprogramm nur in einer entsprechenden Meldung besteht, ausgefuehrt wird. Nach Quittierung dieser Meldung bzw. bei Neustart sofort beginnt die Abarbeitung der Endlosschleife, in der staendig die Tastatur abgefragt wird und alle halbe Minute, gesteuert ueber die Variable MERKZELLE, die Anzeige einschliesslich der Uhrzeit aktualisiert wird.

Screen 14

Ab diesem Screen beginnt das Vordergrundprogramm. Es beginnt genau wie das Hintergrundprogramm mit der Definition von Zugriffsoperationen auf den Datensatz eines Reaktors. Dieser Screen ist ein typisches Beispiel dafuer, wie unterschiedlich verschiedene Programmierer eine aehnliche Aufgabe loesen, und dass an sich an dieser Stelle eine staerkere Absprache zwischen den Programmierern wuenschenswert gewesen waere. DATENELEMENT ist die Zugriffsoperation, die waehrend der Compile-Zeit dem Datennamen einen Wert zuordnet und zur Laufzeit daraus und dem Inhalt der Variablen BASIS die absolute Speicheradresse ermittelt. BASIS enthaelt die Anfangsadresse des Datensatzes des gerade bearbeiteten Reaktors, deren Wert durch BASIS-SETZEN eingetragen wird.

Screen 15

Zur Kuehlung verfuegt jeder Reaktor ueber Kuehltaschen, die durch Oeffnen des Motorventils in den Kuehlkreislauf einbezogen bzw. bei dessen Schliessen davon getrennt werden. Darueber erfolgt die Regelung der Temperatur im Reaktor. Ausserdem kann die Kuehlsole durch Betaetigen des Magnetventils aus den Kuehltaschen abgelassen werden. Nur waehrend der Gaerung und Reifung sollen die Kuehltaschen mit Sole gefuellt sein. Der physische Treiber zur Ansteuerung der Schuetzensteuerung der Motor- und Magnetventile ist eine Maschinenkodepassage, der als Parameter die Reaktornummer, die Schaltfunktion und der Ventiltyp uebergeben werden. Das Motorventil verfuegt ueber einen Schuetz zum Oeffnen und einen zum Schliessen. EIN-KUEHL oeffnet das Motorventil und AUS-KUEHL schliesst es. KUEHLHALT und KUEHLABLAUF erfordern keine Erlaeuterung. ABHALT prueft jeden Reaktor, ob er sich in der Technologie Gaerung (Kodierung = 04) bzw. Technologie Reifung (Kodierung 05) befindet, und steuert das Magnetventil entsprechend an.

Screen 16

Die AD-Wandler-Routine ist eine Maschinenkodepassage. Durch nicht abgleichbare Leitungswiderstaende und Uebergangswiderstaende im Relais-Multiplexer sind die Messwerte verfaelscht und erfordern eine kanalabhaengige Korrektur. Die Korrekturwerte sind in der Tabelle OFFSET zusammengefasst. KORREKTUR holt den entsprechenden Wert aus der Tabelle und subtrahiert ihn vom Messwert. ISTTEMPERATUR uebergibt die Adresse des Speicherplatzes, auf dem der Messwert abgelegt wird.

Screen 17

Bei jeder Aktivierung des Vordergrundprogramms wird ein Reaktor bearbeitet. In dem die 22 Reaktoren noch um 8 fiktive Reaktoren, bei denen keinerlei Aktionen ausgefuehrt werden, erweitert werden, dauert ein vollstaendiger Durchlauf 30 s und wird zur Zeitzaehlung benutzt. Bei Bearbeitung des ersten Reaktors jedes Durchlaufs wird die interne Zeit, auf der das gesamte Kalenderprogramm aufbaut, erhoeht und die Variable MERKZELLE gesetzt, womit die Aktualisierung der Anzeige durch das Hintergrundprogramm freigegeben wird. So lange reale Reaktoren (Reaktornummer < 23) bearbeitet werden, schliessen sich die Aktionen: Ermittlung Datensatzadresse (BASIS-SETZEN), Steuerung Magnetventil (ABHALT), Ermittlung der Reaktortemperatur (MESSWERTERFASSUNG) und technologische Funktion (REINIGUNG, REIFUNG bzw GAERUNG) an. Die technologische Funktion die Journalfunktion, die Temperaturregelung und die Sollwertvorgabe. Abschliessend wird die folgende Reaktornummer ermittelt und der Relais-Multiplexer auf den entsprechenden Kanal geschaltet. INTSRV ist der Rahmen des Vordergrundprogramms. POPRETI enthaelt die Rueckrettung der CPU-Register, die Interruptfreigabe und den Befehl RETI. Der Anfang der Interruptroutine, der die Registerrettung beinhaltet, ist eine hier nicht dargestellte Maschinenkodepassage, die nach dem Laden des Instruktionsregisters der FORTH-Maschine (Registerpaar BC der CPU) mit der Adresse von INTSRV zum Inneren Interpreter der FORTH-Maschine verzweigt.

Screen 18

Das Wort ANFANG repraesentiert das Gesamtprogramm "Prozessteuerung Brauerei". Zuerst wird geprueft, ob der Aufruf von ANFANG ein Neustart oder ein Wiederstart nach Netzausfall ist. Bei Wiederstart enthaelt

die Variable ?RESTART das Bitmuster BLACKOUT und ueber RELREENTRY wird der Schaltzustand der Relais vor dem Netzausfall wiederhergestellt. Bei Neustart wird die Datenbasis und alle Variablen geloescht, ein Anfangsdatum eingetragen und alle Relaisausgaenge rueckgesetzt. Nach Initialisierung der Hardware (PIO und CTC) wird der Multiplexer auf den ersten Kanal geschaltet und das Hintergrundprogramm gestartet.

4. Programmausfuehrung

Das Programm wurde auf einem Buerocomputer entwickelt. Nach dem die logische Korrektheit des Programms nachgewiesen war, erfolgte die Umsetzung auf den Prozessmikrorechner. Der Nachweis der Korrekheit auf dem Entwicklungssystem kann nicht vollstaendig erfolgen, da sich nicht alle E/A-Operationen simulieren lassen. Bei der Umsetzung muss die vom Buerocomputer abweichende Speicherstruktur (Aufteilung in RAM und ROM) beruecksichtigt werden. Als erste Speicheradresse des FORTH-Systems wurde die Adresse 2000H festgelegt. Das Programm muss ROM-faehig sein. Es wurde ein Standard-FIG-FORTH mit der Startadresse 2000H auf dem Buerocomputer generiert, dann die einzelnen Screens geladen, die Systemvariablen entsprechend den Erfordernissen des Prozessmikrorechners (Zahlenbasis dezimal, Woerterbuchzeiger auf RAM-Bereich usw.), andere Treiber fuer Tastatur und Bildschirm eingebunden und ein Speicherabzug auf Diskette vorgenommen. Vorher wurden noch die Eintrittspunkte (Adressen von INTSRV, ANFANG, NEXT und OUTER) bestimmt. Die Startroutine (ab Adresse 0 auf dem ZRE-EPROM) wertet auch die Variable ?RESTART (Adresse 9000H) aus und laedt Registerpaar BC entweder mit der Adresse ANFANG bei Wiederstart oder mit der Adresse des Aeusseren Interpreters (OUTER) von FORTH, bevor sie die Adresse NEXT (Innerer Interpreter) anspringt. Damit ist die Moeglichkeit gegeben, FORTH selbst als Testmonitor zu benutzen, in dem RESET ausgeloest und nach Ausfuehrung der gewuenschten Operationen (z. B. Zugriff auf die Datenbasis, Test einzelner Worte, E/A-Operationen) die Variable ?RESTART mit BLACKOUT geladen und ANFANG ausgefuehrt wird (wird als Wiederstart interpretiert). Im Monitormode koennen alle FORTH-Worte benutzt werden. Dies ist moeglich, da das Woerterbuch beim Speicherabzug vollstaendig erhalten geblieben ist und somit auf alle implementierten Worte Zugriff besteht. Erkauft wird dies mit einem erheblichen Speicheraufwand (> 16 K Byte). Das wiegt aber nicht die Vorteile beim Einfahren des Programms und bei der Programmwartung auf. Beispielsweise ist die Uebersichtsdarstellung des Reaktorkellers erst nachtraeglich implementiert worden. Die einzelnen Worte wurden im Monitormode erprobt, bevor das neue Programm erzeugt wurde, so dass es sofort fehlerfrei lief. Es ist moeglich, die Adresse einzelner Worte zu ermitteln und durch Umprogrammieren der EPROMs Kleinigkeiten direkt zu korrigieren.

Dank seiner Flexibilitaet ist FORTH fuer Prozessteuerungen, die eine staendige Softwarewartung erfordern, hervorragend geeignet, sofern die Echtzeitbedingungen eingehalten werden koennen.

Verfasser: Dipl.-Ing. Rainer Korpal
Wilhelm-Pieck-Universitaet Rostock
Abteilung Wissenschaftlicher Geraetebau
Albert-Einstein-Str. 2
Rostock 6
DDR - 2500

Prozessteuerung Brauerei

```
Screen  1
0 ( BRAUEREI          VEREINBARUNGEN                )    HEX
1
2 : TABELLE <BUILDS DOES> ;
3               DECIMAL
4 TABELLE KALENDER       32 C, 29 C, 32 C, 31 C, 32 C, 31 C,
5                        32 C, 32 C, 31 C, 32 C, 31 C, 32 C,
6                 HEX
7 : === CONSTANT ;          : == <BUILDS C, DOES> C@ ;
8 9000         === CRAM                    A4 === PFL
9 CRAM         === ?RESTART        CRAM  3 +     ===  3OSEKZ
10 CRAM  4 +   === 3MINZ           CRAM  7 +     ===  TAG
11 CRAM  8 +   === MONAT           CRAM  9 +     ===  JAHR
12 CRAM  B +   === MINUTEN         CRAM  C +     ===  STUNDEN
13 CRAM  D +   === #REAKTOR        CRAM  E +     ===  BASIS
14 CRAM 10 +   === BELEGUNG        CRAM 1A +     ===  PUFFER
15 CRAM 1A + 16 PFL * + === NUMBILD

Screen  2
0 ( BRAUEREI          VEREINBARUNGEN                )    HEX
1
2 400  === ROM          C00 === NRAM          F000 === .BILD
3 NRAM === KURSOR NRAM 2 + === AKURSOR  NRAM E0 + === TEXTPUFFER
4
5  ( ZEIGER AUF STANDARDTEXTE                        )
6 ROM  46 + ===  1TX   ROM  52 + ===  2TX   ROM  59 + ===  3TX
7 ROM  61 + ===  4TX   ROM  66 + ===  5TX   ROM  70 + ===  6TX
8 ROM  7E + ===  7TX   ROM  8A + ===  8TX   ROM  8F + ===  9TX
9 ROM  97 + === 10TX   ROM  A6 + === 11TX   ROM  B5 + === 12TX
10 ROM C4 + === 13TX   ROM  E0 + === 14TX   ROM  F3 + === 15TX
11 ROM 106 + === 16TX  ROM 10B + === 17TX   ROM 115 + === 18TX
12 ROM 120 + === 19TX  ROM 12B + === 20TX   ROM 133 + === 21TX
13 ROM 13B + === 22TX  ROM 145 + === 23TX   ROM 153 + === 24TX
14 ROM 15F + === 25TX  ROM 17D + === 26TX   ROM 187 + === 27TX
15 ROM 3A2 + === 28TX

Screen  3
0 ( BRAUEREI          VEREINBARUNGEN                )    HEX
1
2 (         ZEIGER ZU DATENPLAETZEN                          )
3          3 == REIST      ( ISTTEMPERATURFELD REINIGUNG      )
4         38 == GIST       ( ISTTEMPERATURFELD GAERUNG        )
5         24 == RIST       ( ISTTEMPERATURFELD REIFUNG        )
6         1A == HE1        ( HEFE GEDRUECKT 1                 )
7         1E == HE2        ( HEFE GEDRUECKT 2                 )
8         22 == ST         ( SOLLTEMPERATUR REIFUNG           )
9         1C == STZA       ( SOLLWERTE GAERUNG                )
10        16 == BR         ( BEGINN REIFUNG                   )
11        10 == BG         ( BEGINN GAERUNG                   )
12        16 == EG         ( ENDE GAERUNG                     )
13        2A == EXT        ( EXTRAKTANGABEN                   )
14
15
```

Prozessteuerung Brauerei

```
Screen  4
 0 ( BRAUEREI          VEREINBARUNGEN                )    HEX
 1
 2 ( ZEIGER ZUM DIALOGBEREICH                                        )
 3 NRAM 10 + === DIAREAKTOR        (          DIALOGBEREICH         )
 4 DIAREAKTOR  3 + === *REIST      ( ISTTEMPERATURFELD REINIGUNG    )
 5 DIAREAKTOR 24 + === *RIST       ( ISTTEMPERATURFELD REIFUNG      )
 6 DIAREAKTOR 38 + === *GIST       ( ISTTEMPERATURFELD GAERUNG      )
 7 DIAREAKTOR 22 + === *ST         ( SOLLTEMPERATUR REIFUNG         )
 8 DIAREAKTOR 1C + === *STZA       ( SOLLWERTE GAERUNG              )
 9 DIAREAKTOR 1A + === *HE1        ( HEFE GEDRUECKT 1               )
10 DIAREAKTOR 1E + === *HE2        ( HEFE GEDRUECKT 2               )
11 DIAREAKTOR 16 + === *BR         ( BEGINN REIFUNG                 )
12 DIAREAKTOR 10 + === *BG         ( BEGINN GAERUNG                 )
13 DIAREAKTOR 16 + === *EG         ( ENDE GAERUNG                   )
14 DIAREAKTOR 2A + === *EXT        ( EXTRAKTANGABEN                 )
15

Screen  5
 0 ( BRAUEREI          ZAHLENFORMATE                 )    HEX
 1
 2 : .R0 >R ZAHL <# #S SIGN #> R> ZAHLAUS ;
 3 : .R1 >R ZAHL <# # 2C HOLD #S #> R> ZAHLAUS ;
 4 : .R2 >R ZAHL <# # # 2C HOLD #S #> R> ZAHLAUS ;
 5 : ..R1 OVER -1 =
 6        IF      KURSOR +! DROP
 7        ELSE    .R1
 8        ENDIF   ;
 9 : .2. ZAHL <# # # SIGN #> 0 ZAHLAUS ;
10
11 ( KURSORFUNKTIONEN                                    )
12 CREATE CLEARK CBE1 , BE C, NEXT CREATE SETK CBE1 , FE C, NEXT
13 : CKURSOR AKURSOR @ CLEARK ;    : SKURSOR AKURSOR @ SETK ;
14 : STC  .BILD + KURSOR ! ;
15 : BILDLOE .BILD 400 BLANKS ;

Screen  6
 0 ( BRAUEREI          ANZEIGE REAKTORZUSTAND        )    HEX
 1
 2 : 0BILD 16TX BILD* ;          : 1BILD 17TX BILD* 1TECH ;
 3 : 2BILD 18TX BILD* 1TECH ;    : 3BILD 19TX BILD* 1TECH ;
 4 : 4BILD 20TX BILD* 2TECH ;    : 5BILD 21TX BILD* 3TECH :
 5 : 6BILD 22TX BILD* 4TECH ;
 6
 7 : CASE: <BUILDS ] DOES> SWAP 2* + @ EXECUTE ;
 8 CASE: TECHNOLOGIE 0BILD 1BILD 2BILD 3BILD 4BILD 5BILD 6BILD [
 9
10 : BILD 0 STC 23TX BILD* 40 STC 24TX BILD* 59 STC 6TX BILD*
11       DUP F STC 1+ 3 .R0 PFL * PUFFER + DUP 1+ @ 68 STC 5 .R1
12       4D STC DUP C@ TECHNOLOGIE DROP ;
13
14
15
```

Z80 comFORTH 1.9 12/01/87 WPU Rostock STE/AS

Prozessteuerung Brauerei

```
Screen  7
 0 ( BRAUEREI          HARDCOPY                          )    HEX
 1
 2 CREATE DRUCKERINIT C3 C,  F  , NEXT
 3 CREATE DRUCKER     C3 C, 2E  , NEXT
 4
 5 : HARDCOPY NUMBILD C@ DUP FF =
 6         IF   DROP NUMBILD COSET
 7         ELSE BILDLOE 2D STC TAG .DAT DROP .
 8              39 STC MINUTEN .ZEIT BILD
 9         ENDIF DRUCKERINIT A DRUCKER .BILD F 0
10         DO    40 0
11               DO     DUP C@ DUP 30 =
12                      IF   2F DRUCKER 8 DRUCKER
13                      ENDIF DRUCKER 1+
14               LOOP   D DRUCKER A DRUCKER
15          LOOP   DROP A DRUCKER A DRUCKER     ;

Screen  8
 0 ( BRAUEREI          LOG.TASTATURTREIBER 1             )    HEX
 1 NRAM 4 + === KOMMANDO?  NRAM 6 + === LAENGE NRAM C + === FERTIG
 2 : GETCHR CKURSOR DUP 9D =
 3          IF      DROP KOMMANDO? @ CFA EXECUTE
 4          ELSE    DUP 7 =
 5                  IF     DROP TEXTPUFFER C@
 6                         IF    -1 TEXTPUFFER C+! -1 AKURSOR +!
 7                               20 AKURSOR @ C!
 8                         ENDIF
 9                   ELSE  TEXTPUFFER C@ 1+ DUP DUP LAENGE C@ >
10                         IF   DROP DROP DROP
11                         ELSE 2OVER AKURSOR @ C! 1 AKURSOR +!
12                              TEXTPUFFER C! TEXTPUFFER + C!
13                         ENDIF
14                   ENDIF
15           ENDIF    SKURSOR   ;

Screen  9
 0 ( BRAUEREI          LOG.TASTATURTREIBER 2             )    HEX
 1
 2 CREATE TASTATUR C3 C, ROM , NEXT
 3
 4 : *TAST BEGIN    TASTATUR DUP
 5                  IF     GETCHR  FERTIG C@
 6                  ENDIF
 7         END     ;
 8
 9
10
11
12 : VORBE .BILD 3C0 + 40 BLANKS 10 LAENGE C! TEXTPUFFER COSET
13         .BILD 3DA + AKURSOR ! SKURSOR    ;
14
15
```

Prozessteuerung Brauerei

```
Screen 10
0 ( BRAUEREI          EINGABE UHRZEIT                )    DECIMAL
1
2 : UHRSET  FERTIG C1SET TEXTPUFFER C@
3         IF   TEXTPUFFER 1+ 1TXZEIGER ! SUCHE ENDE?  0=
4              IF  HOLE KONV 2TXZEIGER @ 1+ 1TXZEIGER ! SUCHE
5                  HOLE KONV DUP 60 <
6                  IF    OVER 24 <
7                        IF    1 FERTIG C+!
8                        ELSE  DROP DROP
9                        ENDIF DROP DROP
10                 ENDIF
11             ENDIF
12          ENDIF  ;                                      HEX
13 : UHRZEIT BEGIN  FERTIG COSET VORBE 3C0 STC 29TX BILD*
14                  ' UHRSET KOMMANDO? ! *TAST FERTIG C@ 2 =
15          END     MINUTEN C! STUNDEN C! CKURSOR   ;

Screen 11
0 ( BRAUEREI          UEBERSICHTSBILD 1              )    HEX
1
2 : UEIST   OVER C + STC DUP 1+ @ 4 .R1 ;
3 : UEMENGE OVER 11 + STC DUP 3 + @ 5 .R0 ;
4 : UESORTE OVER 17 + STC DUP 5 + DUP C@ 8 MIN 0
5           DO      1+ DUP C@ !KURSOR
6           LOOP    DROP  ;
7 : UEDAT   2 PICK 5 + STC OVER + DUP C@ .2. 2E !KURSOR 1+
8           C@ .2. 2E !KURSOR ;
9 : TECHLR  2 PICK 3 + STC !KURSOR UEIST ;
10 : TECHFSB TECHLR UEMENGE UESORTE   ;
11
12 : UEFILT 46 TECHFSB ;          : UELEER  4C TECHLR ;
13 : UESCHL 53 TECHFSB ;          : UECLEAR 43 TECHLR ;
14 : UEBEFL 42 TECHFSB ;
15 : UEGAER 47 TECHFSB BG UEDAT ; : UEREIF  52 TECHFSB BR UEDAT ;

Screen 12
0 ( BRAUEREI          UEBERSICHTSBILD 2              )    HEX
1
2 CASE: UETECH  UELEER UEBEFL UEFILT UESCHL UEGAER UEREIF
3               UECLEAR   [
4
5 : UEBER DO    OVER STC I 2 .R0 DUP C@ UETECH PFL + SWAP
6               40 + SWAP
7         LOOP   ;
8
9 : UEBERSICHT  80 STC 31TX BILD* 2TX BILD* A0 STC 31TX BILD*
10              2TX BILD* C0 PUFFER C 1 UEBER SWAP DROP E0
11              SWAP 17 C UEBER DROP DROP ;
12
13
14
15
```

Prozessteuerung Brauerei

```
Screen 13
0 ( BRAUEREI          HINTERGRUND                    )    HEX
1
2     1FE === BLACKOUT
3 : HINTERGRUND    BILDLOE ?RESTART @ BLACKOUT =
4          IF      220 STC 27TX BILD*
5                  BEGIN    TASTATUR      END
6          ENDIF VORBE ' KOMMANDO KOMMANDO? ! NUMBILD COSET
7          3C0 STC 26TX BILD* MERKZELLE C1SET
8          BEGIN  TASTATUR -DUP
9                 IF      GETCHR
10                ENDIF   MERKZELLE C@
11                IF      BILDLOE1 2D STC TAG .DAT DROP 39 STC
12                        MINUTEN .ZEIT UEBERSICHT MERKZELLE COSET
13                ENDIF   0
14           END  ;
15

Screen 14
0 (-BRAUEREI          VEREINBARUNGEN REGLER          )    HEX
1
2 : BASIS-SETZEN #REAKTOR C@ 1- PFL * PUFFER + BASIS ! ;
3
4 : DATENELEMENT <BUILDS C, DOES> C@ BASIS @ + ;
5
6  1 DATENELEMENT ISTTEMPERATUR   3 DATENELEMENT RZEIGER
7 22 DATENELEMENT RFSOLLTEMP     24 DATENELEMENT RFZEIGER
8 16 DATENELEMENT GAERENDZEIT    1C DATENELEMENT GSTUFEN
9 1D DATENELEMENT GPOINT         38 DATENELEMENT GZEIGER
10 98 DATENELEMENT 30GSEKZ       99 DATENELEMENT GSTUNZ
11 9A DATENELEMENT ZUSTAND
12
13
14
15

Screen 15
0 ( BRAUEREI          KUEHLUNG                       )    HEX
1
2 CREATE RELAIS C3 C, ROM CC + , NEXT
3      0 == RES                          1 == SET
4      0 == AUF-MOTORVENTIL              1 == ZU-MOTORVENTIL
5      2 == MAGNETVENTIL
6 : EIN-KUEHL DUP RES ZU-MOTORVENTIL  RELAIS
7                 SET AUF-MOTORVENTIL RELAIS ;
8 : AUS-KUEHL DUP RES AUF-MOTORVENTIL RELAIS
9                 SET ZU-MOTORVENTIL  RELAIS ;
10 : KUEHLABLAUF SET MAGNETVENTIL RELAIS ;
11 : KUEHLHALT   RES MAGNETVENTIL RELAIS ;
12 : ABHALT #REAKTOR C@ BASIS @ C@ DUP 4 = SWAP 5 = OR
13          IF      KUEHLHALT
14          ELSE    DUP KUEHLABLAUF AUS-KUEHL
15          ENDIF   ;
```

Prozessteuerung Brauerei

```
Screen 16
0 ( BRAUEREI          TEMPERATURERFASSUNG              )    HEX
1
2 CREATE ADU C3 C, ROM 13E + , NEXT
3
4 TABELLE OFFSET
5         23 C, 25 C, 22 C, 22 C, 2E C, 25 C, 29 C, 28 C,
6         2B C, 2C C, 2E C, 35 C, 2B C, 2C C, 30 C, 35 C,
7         23 C, 21 C, 27 C, 26 C, 35 C,  0 C,  0 C,  0 C,
8          0 C,  0 C,  0 C,  0 C,  0 C,  0 C,  0 C,  0 C,
9
10 : KORREKTUR #REAKTOR C@ 1- OFFSET + C@ - ;
11
12 : MESSWERTERFASSUNG  ADU KORREKTUR ISTTEMPERATUR ! ;
13
14
15


Screen 17
0 ( BRAUEREI          INTERRUPTROUTINE                 )    DECIMAL
1
2 : VORDERGRUND  #REAKTOR C@ DUP 1 =
3         IF     INC-ZEIT MERKZELLE C1SET
4         ENDIF 23 <
5         IF     BASIS-SETZEN ABHALT MESSWERTERFASSUNG REINIGUNG
6                REIFUNG GAERUNG
7         ENDIF #REAKTOR C@ 1+ 31 MOD DUP #REAKTOR C! 0=
8         IF     #REAKTOR C1SET
9         ENDIF #REAKTOR C@ MULTIPLEXER ;
10
11 : ISR     VORDERGRUND POPRETI ;
12
13
14
15


Screen 18
0 ( BRAUEREI                                           )    DECIMAL
1
2 : ANFANG ?RESTART @ BLACKOUT =
3         IF     RELREENTRY
4         ELSE   CRAM 4024 0 FILL #REAKTOR C1SET RELINI
5                10 TAG C! 9 MONAT C! 1984 JAHR !
6         ENDIF
7         1 MULTIPLEXER CTC-INITIALISIERUNG HINTERGRUND ;
8
9
10
11
12
13
14
15
```

Applikation von FORTH

Helmut Darmuntzel

Rechnergestuetzte Stundenplangestaltung

Der Leser soll mit einer mittleren comFORTH - Applikation auf dem Gebiet der nicht-numerischen Datenverarbeitung bekannt gemacht werden. Die Problemstellung und das Loesungskonzept werden vorgestellt und die Eignung der Programmiersprache FORTH in der Version comFORTH 1.x bestaetigt.

1. Das konventionelle Vorgehen

Das Erstellen von Stundenplaenen ist eine je nach Groesse der Lehreinrichtung weniger oder mehr zeitraubende Aufgabe, die vielerorts zu Beginn des Schul- oder Lehrjahres bzw. Semesters und danach in Abhaengigkeit vom Lehrerausfall geloest werden muss. Prinzipiell geht es bei allen Plaenen darum, eine bestimmte Lehrvorgabe (Meier gibt Russisch in 6a viermal in der Woche) so auf die einzelnen Stunden in der Woche zu verteilen, dass einerseits Widerspruchsfreiheit erreicht wird und andererseits Einschraenkungen erfuellt werden. Speziell an POS sind Beispiele solcher Einschraenkungen:

1. Bestimmte Lehrer sind an bestimmten Tagen ganz oder stundenweise verhindert.
2. Unterstufenklassen duerfen am Tagesende keinen Unterricht erhalten.
3. Fuer bestimmte Kombinationen gibt es aufgrund des Schulfernsehens nur wenige, in der Woche verstreute Moeglichkeiten.

Die herkoemmliche Methode zur Stundenplanerstellung besteht nun darin, dass sich einige Lehrer vor einer grossen Wandtafel versammeln und nacheinander die geforderten Stunden in diesem Plan vermittels farbiger Reiter markieren, ihn "stecken". Ist der Plan fertig, wird auf Einhaltung aller Beschraenkungen und Widerspruchsfreiheit geprueft und solange korrigiert, bis beides gewaehrleistet ist. Dies erfordert Logik, Erfahrung, Disziplin und Ruhe. Da es sich aber um ein algorithmisch abarbeitbares Problem zu handeln scheint, ergibt sich zwangslaeufig die Frage nach einer rechentechnischen Behandlung.

2. Das Stundenplanproblem

Die exakte Aufgabenstellung lautet, Schulstundenplaene nach gegebenen Ressourcen, Restriktionen und Anforderungen mittels einer EDVA zu konstruieren. Vom konventionellen Verfahren ohne Rechnerunterstuetzung ist bekannt, dass

1. von den Bearbeitern an den verschiedenen Schulen Art und Menge der Restriktionen sehr unterschiedlich bewertet werden und
2. die Anforderungen an das Ergebnis nur verbal und unscharf formuliert werden koennen.

Zusaetzlich wirkt sich erschwerend aus, dass

3. Existenzkriterien und exakte Loesungsalgorithmen fehlen.

Dadurch bedingt, verwenden alle auf der Welt von 1961 bis heute entwickelten Rechnerprogramme zur Erstellung von Stundenplaenen in unterschiedlichem Umfang heuristische Methoden zur Loesung. Dies sind aus der praktischen Arbeit auf den Rechner uebertragene Methoden ohne Loesungsgarantie. In Verbindung mit der Programm-Abarbeitung auf zentralen Gross-EDVA erfordert dies einen unvertretbar hohen Aufwand fuer Datenvorbereitung und Loesungskorrektur bei den Anwendern. Oft mussten diese dem Rechenzentrum die Daten in abgelochter Form uebergeben und erhielten nach mehreren Laeufen einen Stundenplan, den sie mit relativ wenig Aufwand in einen praktisch verwendbaren umwandeln konnten (siehe /1/).

Durch Verfuegbarkeit der Mikrorechentechnik ist inzwischen eine dezentrale Verarbeitung moeglich geworden, wodurch folgende Vorteile wirksam werden:

1. Schulspezifische Restriktionen und Loesungsanforderungen sind im Dialogbetrieb besser und schneller formulierbar.
2. Erfahrung und Intuition des bisher konventionellen Bearbeiters gehen positiv in den erstellten Plan ein.
3. Der Rechner stellt sich dem Bearbeiter nicht mehr als Black-Box dar, sondern wird fuer ihn zum Werkzeug, wodurch er mehr Vertrauen zur erarbeiteten Loesung gewinnt.

Als Nachteile sind die beim Mikrorechner um Groessenordnungen geringeren Leistungsmerkmale Rechenzeit und Speicherplatz zu nennen. Wesentliche Aufgaben, die der Rechner uebernehmen kann, sind:

- Speicherung der Daten und Sicherung zur Wiederverwendung
- Suche und Anzeige in vielfaeltiger Form
- Verwaltung der elementaren Logik
- Unterbreitung plausibler Vorschlaege
- Durchfuehrung von Abbruchtests
- Verfielfaeltigung der einzelnen Plaene durch Druck

Daraus und aus der fuer die Stundenplanerstellung in der DDR verfuegbare Mikrorechentechnik mit ihrer Standardkonfiguration (8-Bit-CPU, 64 KByte Hauptspeicher, 2 * Floppy-Disk) ergeben sich folgende Anforderungen an das zu erstellende Programm:

Es muss die geforderten Aufgaben in einer fuer den Bediener akzeptablen Zeit loesen (Reaktionszeit des Rechners im Sekundenbereich) und sich an der konventionellen Arbeitsweise des Bearbeiters orientieren, wobei ihm als EDV-Laien eine sichere und einfache Bedienerfuehrung zur Verfuegung gestellt werden muss.

3. Loesungskonzept

Aus den vielen Ueberlegungen sollen beispielhaft vier Komplexe vorgestellt werden. Als Realisierung der Forderung nach einfacher und sicherer Bedienerfuehrung wurde die Menuetechnik gewaehlt. Zur Demonstration der Ueberlegenheit der Arbeit mit Rechner gegehueber der konventionellen Arbeit am Wandsteckbrett wird die elementare Logik erlaeutert. Dem Anspruch nach schneller Abarbeitung und Datensicherheit kann entsprochen werden, wenn alle zur Erarbeitung eines Schulstundenplanes notwendigen Daten im RAM gehalten werden und der Massenspeicher lediglich als Archiv dient. Als Beispiel fuer die heuristische Vorgehensweise wird das Grundprinzip des Rechnervorschlags erlaeutert (siehe auch /2/).

Insgesamt ergibt sich eine prinzipielle Programmstruktur nach Bild 1. Der Funktionsumfang des gesamten Programms ist aus dem Menuebild fuer das Hauptmenue (Screen 5) ersichtlich.

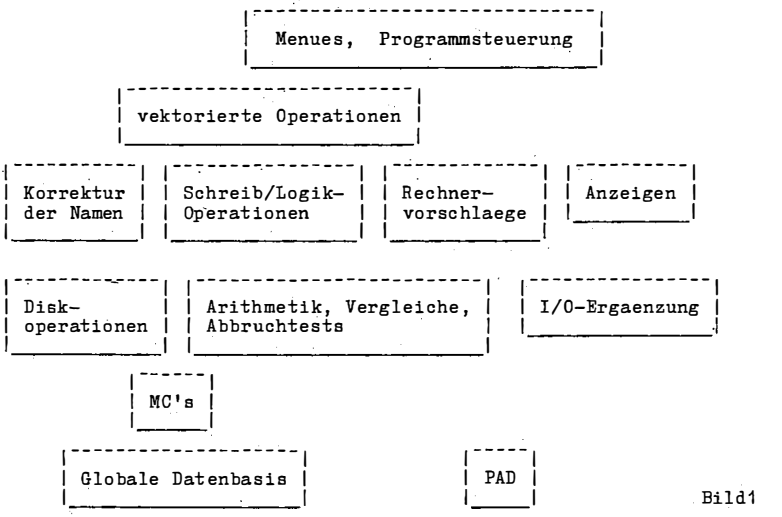


Bild1

3.1. Menuearbeit

Der Rechner uebernimmt dabei die aktive Rolle im Dialog, wobei dem Bearbeiter in jedem Fall gezeigt wird, was er in der jeweiligen Handlungsphase tun kann bzw. muss. Seine Aktionen werden unterschieden zwischen Kommandos und Dateneingaben. Kommandos sind festgelegte Buchstaben oder Zeichen, die aus dem Menuebild ersichtlich sind und zur Ausfuehrung einer Funktion fuehren. Einzugebende Daten koennen Wochentage, Stunden und die Namen der beteiligten Lehrer, Faecher, Klassen und Fachraeume sein. Als Kompromiss zwischen Eingabeschnelligkeit, Speicherplatzeinsparung und genuegender Verstaendlichkeit wurde vereinbart, genannte Namen durch drei signifikante Grossbuchstaben, Zahlen oder Zeichen abzukuerzen.

Die Menuebilder werden auf Massenspeicher in einer gesonderten Datei abgelegt und bei Bedarf von dort auf den Bildschirm gebracht. Danach wird der Cursor zur Eingabe positioniert und ein Kommando oder eine Eingabe erwartet. Bei der Vielfalt der zu realisierenden Funktionen muessen die Menues geschachtelt werden, bevor die eigentliche Aktion ausgefuehrt wird.

3.2. Logik

Die Anwendung der elementaren Logik bedeutet hier die Verwaltung der Einzigkeitsbedingungen durch den Rechner. Das bedeutet:

1. Ein Lehrer kann zu einer bestimmten Zeit nur eine Klasse unterrichten.
2. Eine Klasse kann zur einer bestimmten Zeit nur von einem Lehrer unterrichtet werden.
3. In einem als solchen gekennzeichneten Fachraum koennen zu einer Zeit nur ein Lehrer und eine Klasse beschaeftigt sein.

In Verbindung mit der Moeglichkeit, Stunden fuer einzelne Lehrer, Faecher, Klassen, Raeume oder aber fuer Kombinationen von Lehrer, Fach, Klasse und evtl. Fachraum sperren zu koennen (Restriktionen), wird der aktuelle Zustand fuer jede Kombination zu jeder Zeit durch vier Zustaende repraesentiert:

M ..... Moeglich (kann in den Plan aufgenommen werden, Anfangszustand bei der Planerstellung)
G ..... Gesetzt (in den Plan aufgenommen)
U-S ..... Statisch unmoeglich (von Hand gesperrt)
U-D ..... Dynamisch unmoeglich (durch Setzen einer anderen Stunde und Anwendung der Einzigkeitsbedingung gesperrt)

Somit werden Fehlermitteilungen ermoeglicht, falls der Bediener versucht, eine Kombination zu einer bestimmten Zeit zu setzen, obwohl er vorher den Lehrer oder die Klasse von Hand gesperrt hat oder der Lehrer oder die Klasse oder der Raum durch Anwendung der Einzigkeitsbedingungen von anderen gesetzten Stunden dynamisch unmoeglich wurden. Zur Anwendung der Logik dienen die FORTH-Worte in Screen 1 und 2.

3.3. Speichertechnik

Namen werden geordnet nach Lehrern, Faechern usw. in jeweils einer Variablen ohne Count-Byte abgelegt, da die Laenge der Namen ja bereits standardmaessig festgelegt wurde. Intern wird statt mit der Zeichenkette mit einer Nummer gearbeitet, die sich aus dem Platz der Zeichenkette in der Variablen ergibt. Da die Anzahl der Namen in allen Variablen stets kleiner als 256 ist, genuegt ein 1-Byte-Integer als Nummer. Die Zuordnung der Kombinationen (also wer in welchem Fach in welcher Klasse evtl. in welchem Fachraum unterrichtet) erfolgt in einer weiteren Variablen, wobei fuer jede Kombination 4 * 1 Byte benoetigt werden. Da die Anzahl dieser Kombinationen schon fuer zweizuegige Oberschulen groesser als 256 ist, wird intern mit einem 2-Byte-Integer als Kombinationsnummer gearbeitet, der sich aus dem Platz der Kombination in der Variablen ergibt. Die Repraesentation der eigentlichen Datenbasis muss nun so erfolgen, dass fuer jede Kombination zu jeder Zeit in der Woche die vier genannten Zustaende,

die bekanntlich in zwei Bits verschluesselt werden koennen, abgelegt werden. Bei praktisch sinnvollen und handlich gerundeten maximal 680 Kombinationen und maximal 60 Wochenstunden ergeben sich als Groesse des eigentlichen Datenspeichers rund 10 KByte. Zusaetzlich zu den Variablen fuer Namen und Kombinationen und dem Datenfeld hat sich die Einrichtung einer Parametermatrix, in der fuer jede Kombination die Anzahl der woechentlich zu gebenden Stunden, die Anzahl der bereits in den Plan gesetzten und die Anzahl der noch moeglichen Stunden vorliegen und bei jeder Datenveraenderung aktualisiert werden, als zweckmaessig erwiesen.

Da bei jeder Leseoperation oder bei jeder Datenveraenderung

- die Zeit
- der Zustand
- die Nummer der Kombination
- die Nummer des Lehrers und/oder des Fachs und/oder der Klasse und/oder evtl. des Fachraums

anzugeben sind, wurde eine globale Variable fuer die Zeit (MZEIT), eine fuer den Zustand (MSTATUS) und eine fuer die Nummern der Lehrer usw. (MK, 4 Byte) eingefuehrt. Falls Operationen nur fuer eine Klasse oder nur fuer einen Fachraum o. ae. ausgefuehrt werden sollen, wurde die Moeglichkeit geschaffen, in MK einzelne Plaetze zu maskieren.

3.4. Rechnervorschlaege

Statt mittels kombinatorischer Ueberlegungen den gesamten Stundenplan auf einmal zusammenzustellen, wird in Anlehnung an die konventionelle Arbeit eine Kombination nach der anderen ausgewaehlt und zu einer sinnvoll erscheinenden Zeit in den Plan gesetzt, bis alle zu vergebenden Stunden realisiert sind. Als vernuenftige Ueberlegung zur Findung einer nun zu setzenden Kombination ("Wer") wird fuer jede untersucht, wieviele Stunden noch zu verteilen sind und wieviele Moeglichkeiten es dafuer noch gibt. Diejenige Kombination mit dem groessten Quotienten wird zur Setzung ausgewaehlt ("die am dringendsten gesetzt werden muss"). Zur Ermittlung einer guenstigen Zeit wird die ermittelte Kombination probeweise zu allen moeglichen Zeiten in den Plan gesetzt und die Anzahl der sich daraus ergebenden dynamischen Unmoeglichkeiten bei anderen Kombinationen gezaehlt. Die Zeit mit der geringsten Anzahl resultierender Unmoeglichkeiten wird ausgewaehlt ("Es wird dort gesetzt, wo es auf den Gesamtfonds der Moeglichkeiten den am wenigsten schaedlichen Einfluss hat."). Die beiden Worte zur Ermittlung der Unmoeglichkeiten fuer eine Kombination und eine Zeit sind in Screen 4 dargestellt.

4. Programmierungstechnik und Systeminstallation

4.1. Programmierung

Als erstes wurden die globale Datenbasis und die Worte zu ihrer Behandlung realisiert. Daraufhin wurde der Gesamtkomplex "Vereinbarung der Namen und Kombinationen" erstellt, um jederzeit schnell und effektiv Pruefdaten benutzen zu koennen. Als naechstes wurden die Anzeigefunktionen aufgebaut, wodurch in der weiteren Testphase aufwendige Speichervergleiche auf Bit-Niveau entfielen. Mit den nächfolgenden Manipulationsmodulen konnte das Logik-Konzept getestet und die Grundlage fuer komplexe Bearbeitungsformen ("Kombination aus Rechnervorschlag und Setzen von Hand") und fuer die Vollautomatik gelegt werden.

4.2. Einfahren

Da die Bildschirmausgaben sehr oft durch direkte Cursorpositionierung erzielt werden, musste ein Fuelle von Fehlern und Maengeln (die Zeichen ueberschrieben sich gegenseitig auf dem Bildschirm) beseitigt werden. Ebenfalls wurde ein ernsthafter Entwurfsfehler festgestellt, da in der ersten Programmversion noch nicht zwischen statischer und dynamischen Unmoeglichkeit unterschieden wurde. Nach dessen Beseitigung und Bearbeitung eines ersten praxisnahen Beispiels zeigte sich, dass die Reaktionszeiten im Minutenbereich lagen, wodurch sich die Notwendigkeit ergab, die FORTH-Worte zum Lesen, Schreiben, Parametrieren und Kombination suchen durch Maschinenroutinen zu ersetzen. Zum gegenwaertigen Zeitpunkt wird durch die Anwender die Forderung erhoben, die Vereinbarung der Kombinationen und der zugehoerigen Wochenstunden durch einen Full-Screen-Editor per Cursorpositionierung statt in vier Stunden in einer Stunde erledigen zu koennen.

4.3. Systeminstallation

Um Kompatibilitaet zwischen 1 K - und 2 K - Bildschirmen zu erreichen, wurde die gesamte Ausgabe auf das Format 16 * 64 Zeichen ausgerichtet. Damit nicht jedesmal jedes Menuebild von Disk gelesen werden muss, wurde die Anzahl der Disk-Buffer maximiert (12 unter Betriebssystem DAC, 7 unter SCP).

Weiterhin wurde zur Vermeidung von Rechenaufwand die Groesse der Felder auf Vielfache von 1 KByte gerundet und als Disk-Buffer-Groesse ebenfalls 1 KByte festgelegt.

Das automatische Laden der Namen und Daten von Disk beim Programmstart konnte durch Verwendung einer entsprechenden Routine in USER-COLD und die Fehlerbehandlung durch Fehlerausschrift (MESSAGE) und Ruecksprung ins Hauptmenue ueber USER-QUIT realisiert werden. Durch Umgehung von ABORT beim Programmstart wurde die Ausschrift "Z80 comFORTH ..." verhindert. Die realisierten Systemfunktionen sind aus Screen 6 ersichtlich.

5. Einschaetzung von comFORTH

Neben den prinzipiellen Vorteilen der Programmiersprache FORTH wie

- Moeglichkeit der natuerlichsprachigen Wahl der Modulnamen,
- vektorierte Ausfuehrung von Funktionen und
- Unterstuetzung des Konzeptes der schrittweisen Verfeinerung

wurden folgende Vorteile der Version comFORTH gegenueber aelteren fig-Versionen wirksam:

1. Bereitstellung umfangreicher und komfortabler Programmpakete fuer Erstellung und Test.
2. Die Moeglichkeit, SCREDIT-, DEBUG- und TRACY-Paket unter das zu entwickelnde Anwenderprogramm zu laden, beschleunigt die gesamte Programmerstellung um Groessenordnungen.
3. Durch die physische Trennung von Quellprogramm(teilen), Daten und Bildern werden Sicherheit und Uebersichtlichkeit wesentlich erhoeht.
4. Verbesserte Flexibilitaet des Gesamtsystems durch komfortables

Memory-Mapping und Eingriffsmoeglichkeit in Systemfunktionen.

Neben den prinzipiellen FORTH-Nachteilen der umgekehrt polnischen Notation (der hier durch die Menuetechnik nicht wirksam wurde) und der Eingrenzung der Laenge der gewaehlten Modulnamen durch die Beschraenkung, ein Modul guenstig auf einem Quelltextscreen (1K!) unterbringen zu muessen (was durch comFORTH wegen der fig-Kompatibilitaet nicht beseitigt werden konnte), traten Schwierigkeiten bei der Nutzung aller Eingabefunktionen auf, obwohl dem Anwender durch comFORTH schon prinzipiell fuer jeden Datentyp eine spezielle solche Funktion zur Verfuegung gestellt wird.

Bei allen Eingaben fehlt aber die Option, ob der Bediener etwas eingeben muss oder sich durch z.B. Druck auf "Enter" der Eingabe entziehen kann. Auch wird bei allen Eingabefunktionen ein Fragezeichen ausgegeben. Dies wird nicht nur als ueberfluessig erachtet (der Cursor zeigt ohnehin an, wo etwas eingegeben werden soll), sondern Ausgaben, die zu Kontrollzwecken getaetigt werden, werden dadurch ueberschrieben. Zum dritten sollten Ja/Nein-Abfragen, wie im deutschsprachigen Raum ueblich, durch "J" und "N" verschluesselt werden. Eine Auswahl eigener Eingabefunktionen enthaelt Screen 3.

Als Ausweg wird zur Diskussion vorgeschlagen, in einer Version comFORTH 2.x nur noch die Funktionen KEY, EXPECT, TYPE und EMIT im Sprachkern zu belassen und in einem I/O-Paket komfortable Funktionen einzeln oder insgesamt wahlweise ladbar bereitzustellen.

Literatur

/1/ Lesch,M.; Lieske,H.-J.:
"Zu einigen Problemen bei der Ermittlung von Schulstundenplaenen mit Hilfe der EDV" in:
"Leistungsermittlung und Diagnostik im Unterricht"
Teil III, S. 63-78
Berlin: Akademie der Paedagogischen Wissenschaften, Zentralstelle fuer paedagogische Information und Dokumentation 1979

/2/ Uhlemann, K. H.; Schöllkopf, K. H.; Knauer, B. A.:
"Untersuchungen zum Stundenplanproblem"
Elektronische Datenverarbeitung 11 (1969) Heft 3, S. 119-131
Braunschweig: F. Vieweg & Sohn

Verfasser: Dipl.-Ing. Helmut Darmüntzel
Wilhelm-Pieck-Universitaet Rostock
Sektion Technische Elektronik
Albert-Einstein-Strasse 2
Rostock 6
DDR - 2500

Programmlisting

```
  Screen   1
 0 ( Triviale Negation -  SCHREIBEN  VERAENDERN-AUF )
 1
 2  : SCHREIBEN      ( n --> ; Zusammenfassung d. Maschinenroutinen)
 3              DUP APM-M OVER {L} {P} {S} ;
 4
 5  : VERAENDERN-AUF ( alter_zustand  neuer_zustand -->)
 6
 7              DUP MSTATUS C!
 8              1 BEGIN  #LFKLFR         ( --> az nz n 1 oder --> 0)
 9                WHILE  DUP {L}                ( az nz n zust. -->)
10                       4 PICK =
11                       IF DUP SCHREIBEN  ENDIF 1+
12                REPEAT 2DROP ;
13  -->
14
15


  Screen   2
 0 ( Triviale Negation -  NEG-ALLE)
 1
 2  : NEG-ALLE  ( n -->)
 3      DUP {L} DUP GESETZT =   45 ?ERR             ( n zustand -->)
 4      DUP U-S = SWAP U-D = OR 40 ?ERR                      ( n -->)
 5      DUP ?DIFF 0=            39 ?ERR                      ( n -->)
 6      DUP K-AUS                                ( n fr kl fa le -->)
 7      MLEER MK C! MOEGLICH U-D VERAENDERN-AUF   ( n fr kl fa -->)
 8      DROP MLEER MK 2+ C! MOEGLICH U-D VERAENDERN-AUF ( n fr -->)
 9      DUP  KEIN-FR =
10           IF    DROP
11           ELSE  MLEER MK 3+ C! MOEGLICH U-D VERAENDERN-AUF
12           ENDIF                                        ( n -->)
13      GESETZT MSTATUS C! SCHREIBEN ; ( als gesetzt gekennzeichn.)
14
15   -->


  Screen   3
 0 ( Eingabe - ZEIN  ZEIN!  EIN  EIN!)
 1
 2  : ZEIN  ( x y --> f ; Eingabe einer Zahl ohne Zwang)
 3       XY PAD 5 ERASE PAD 1+ 3 EXPECT
 4       PAD 1+ BL ENCLOSE DROP SWAP DROP
 5       2DUP SWAP 1- C! + BL SWAP C!
 6       PAD INUMBER IF 1  ELSE DROP 0 ENDIF ;
 7  : ZEIN! ( x y --> ; Eingabe einer Zahl mit Zwang)
 8       BEGIN 2DUP ZEIN
 9       UNTIL ROT ROT 2DROP ;
10  : EIN   ( x y --> n ; Anzahl der eingegebenen Zeichen)
11       XY PAD 5 ERASE PAD 4 EXPECT PAD C@ ;
12  : EIN!  ( x y --> ; Eingabe einer Zeichenkette mit Zwang )
13       BEGIN 2DUP EIN
14       UNTIL 2DROP ;
15  -->
```

```
Screen   4
 0  ( Optimierung  -  NEG-TEST-1   NEGATEST)
 1
 2  : NEG-TEST-1 ( n --> ; #u in #U)
 3    1  BEGIN #LFKLFR
 4       WHILE DUP {L} MOEGLICH =  IF 1 #U +! ENDIF 1+    ( n+1 -->)
 5       REPEAT ;
 6
 7  : NEGATEST ( n --> #u)
 8     0 #U ! AK ( L.)  DUP C@ MLEER MK C! NEG-TEST-1
 9               ( Kl.) DUP 2+ C@ MLEER MK 2+ C! NEG-TEST-1
10               ( Fr.) 3+ C@ DUP KEIN-FR =
11                         IF DROP
12                         ELSE MLEER MK 3+ C! NEG-TEST-1  ENDIF
13                      #U @ ;
14   -->
15
```

```
Screen   5
 0                  Stundenplanprogramm
 1               Version 1.2  vom 22.01.1987
 2              ================================
 3
 4   Hilfen zur Arbeit mit dem Programm      ( H )
 5   Programmiersprache  FORTH               ( F )
 6   Erfassung aller Namen                   ( E )     Hauptmenue
 7   Weiterarbeit mit den Daten von Disk     ( W )     ---------
 8   Sichern der aktuellen Daten             ( S )
 9   Anzeige                                 ( A )   Bitte waehlen:
10   Manipulation                            ( M )
11   Rechnervorschlag + Setzen von Hand      ( K )          <---
12   Global optimierter Rechnervorschlag     ( G )
13   Vollautomatik                           ( V )
14   Druck fertiger Plaene                   ( D )
15   Beendigung der Arbeit mit dem Programm  ( B )
```

```
Screen   6
 0 ( Systemarbeit -  MYABORT  MYQUIT  MY(EMIT)  BILDSCHIRM  MYCOLD)
 1
 2  : MYABORT  ( Ersatz fuer das System-ABORT ohne RP! und .VER)
 3       [COMPILE] [ SP! 0 DLG ! 0 BLK !
 4       [COMPILE] FORTH DEFINITIONS DECIMAL ;
 5  : MYQUIT      MYABORT RP! HAUPTMENUE ;
 6
 7  : MY(EMIT)    XY-MODUS @ IF ELSE -DUP
 8                          IF ELSE BL ENDIF  (Leerz. statt 0)
 9                   ENDIF (EMIT) ;
10  : BILDSCHIRM  ' MY(EMIT) CFA <USER> ! ;
11
12  : MYCOLD   MYABORT RP! LADE-NAMEN LADE-DATEN BILDSCHIRM QUIT ;
13
14       ' MYQUIT CFA 58 +ORIGIN !  ( Init USER-QUIT)
15       ' MYCOLD CFA 50 +ORIGIN !  ( Init USER-COLD) BILDSCHIRM
```

Das Programmsystem comFORTH

Das Programmsystem comFORTH ist eine figFORTH-kompatible Implementierung der Sprache FORTH unter dem Betriebssystem CP/M, zu der eine ausgebaute Programmbibliothek existiert. Zu den wichtigsten Eigenschaften von comFORTH gehoeren die Flexibilitaet des Dialogsystems, die Erweiterbarkeit des Textinterpreters sowie die organische Einbindung des FORTH-Massenspeicherkonzepts in eine dateiorientierte Betriebssystemumgebung. Das Programmsystem comFORTH ist auf allen Z80-Rechnersystemen unter CP/M-kompatiblen Betriebssystemen lauffaehig und wird durch die WPU Rostock zur Nachnutzung angeboten. Ein entsprechendes Informationsblatt kann bei der

Wilhelm-Pieck-Universitaet Rostock
Sektion Technische Elektronik
Wissenschaftsbereich Automatische Steuerungen
Albert-Einstein-Strasse 2
R o s t o c k 6
DDR - 2500

angefordert werden. Die nachfolgende Aufstellung gibt einen kurzen Ueberblick ueber den Leistungsumfang von comFORTH.

Basispaket comFORTH

Das Basispaket besteht aus einem residenten Kernsystem, das als Kommandodatei geliefert wird, und einigen Standarderweiterungsmoduln in Quellform. Konzepte und Befehle des Basissystems werden in den mitgelieferten Dokumentationen ausfuehrlich beschrieben.

bestehend aus: residentem comFORTH-Kern
Standard Line-Editor
Kopierprogramm fuer Screentransfer zwischen Dateien
Debugpaket mit Dumpfunktionen und Discompiler
U880-FORTH-Assembler
Installationsprogramm fuer residente Kerne

Ergaenzungskomponenten

Die Ergaenzungskomponenten zum comFORTH-Basissystem werden in der Regel in Quellform geliefert. Zu allen Komponenten wird eine Anwenderdokumentation bereitgestellt.

Entwicklungswerkzeuge:

Full-Screen-Editoren
Locator fuer relativen Maschinencode
Interaktives Debugsystem

Standarderweiterungen:

Stringverarbeitung
Dateiarbeit fuer CP/M

Arithmetikpakete:

4 - Byte - Fliesspunktarithmetik
Doppeltgenaue Fliesspunktarithmetik
Komplexe Fliesspunktarithmetik
Vektor- und Matrixalgebra
Mehrfachgenaue Integerarithmetik

Cross-Entwicklungskomponenten

Als Erweiterungen zum comFORTH-System wird ein Crosscompiler angeboten, der in der Lage ist, FORTH-Programme in Zielcode fuer spezielle Applikationsrechner zu erzeugen. Gegenwaertig koennen durch den Crosscompiler Zielkompilate fuer die Prozessoren Z80 und Z8 erzeugt werden. Bibliotheken fuer die Prozessoren Z8000 und i8086 befinden sich in Vorbereitung.

Zur Implementierung von Echtzeitsystemen wird ein Multitaskkernsystem zur Verfuegung gestellt, mit dem die Behandlung von Echtzeitforderungen auf dem Sprachniveau von FORTH moeglich ist. Alle Systemrufe stehen dem Anwender als FORTH-Worte zur Verfuegung.

Ein Testsystem, das den Einzelschritt-Test von Multitaskprogrammen auf FORTH-Niveau gestattet, befindet sich in Vorbereitung.

Einfuehrende Literatur zu FORTH

Brodie, L.: Programmieren in FORTH
Muenchen; Wien: Carl-Hanser-Verlag 1984

Brodie, L.: Denken in FORTH
Muenchen; Wien: Carl-Hanser-Verlag 1986

Floegel, E.: FORTH-Handbuch
Muenchen: Hofacker-Verlag 1983

Floegel, E.: FORTH-Anwendungen
Muenchen: Hofacker-Verlag 1984

Loeliger, R. G.: Threaded Interpretive Languages
Peterborough: BYTE BOOKS 1981

Zech, R.: Die Programmiersprache FORTH
Muenchen: Franzis-Verlag 1984

Schriftenreihen der Wilhelm-Pieck-Universität Rostock

- Archiv der Freunde der Naturgeschichte in Mecklenburg ISSN 0518-3189
- Rostocker Agrarwissenschaftliche Beiträge ISSN 0138-3299
- Rostocker Betriebswirtschaftliche Manuskripte ISSN 0232-3066
- Rostocker Mathematisches Kolloquium ISSN 0138-3248
- Rostocker Philosophische Manuskripte ISSN 0557-3599
- Rostocker Physikalische Manuskripte ISSN 0138-3140
- Rostocker Wissenschaftshistorische Manuskripte ISSN 0138-3191
- Lateinamerika/Semesterbericht der Sektion Lateinamerikawissenschaften ISSN 0458-7944
- Erziehungswissenschaftliche Beiträge ISSN 0138-2373
- Fremdarbeiterpolitik des Imperialismus ISSN 0138-3396
- Beiträge zur Geschichte der Wilhelm-Pieck-Universität Rostock ISSN 0232-539X
- Beiträge zur Geschichte der FDJ ISSN 0233-0830
- Probleme der Agrargeschichte des Feudalismus und des Kapitalismus ISSN 0233-0636
- Rostocker Beiträge zur Hoch- und Fachschulpädagogik ISSN 0233-0539
- Rostocker Informatik-Berichte ISSN 0233-0784
- Studien zur Geschichte der deutsch-polnischen Beziehungen ISSN 0233-0687
- Rostocker Forschungen zur Sprach- und Literaturwissenschaft ISSN 0233-0644
- Rostocker Universitätsreden

Bezugsmöglichkeiten

- Bestellungen aus der DDR über die Wilhelm-Pieck-Universität Rostock, Abt. Wissenschaftspublizistik, Vogelsang 13/14, Rostock, DDR - 2500
- Bestellungen aus dem Ausland über die Firma Buchexport, Volkseigener Außenhandelsbetrieb der DDR, Leninstr. 16, Leipzig, DDR - 7010

Ferner sind die Hefte im Rahmen des Schriftentausches über die Wilhelm-Pieck-Universität Rostock, Universitätsbibliothek, Tauschstelle, Universitätsplatz 5, Rostock, DDR - 2500, zu beziehen.