

FORTH intern

Jeder Befehl besteht aus einem Kopf und einem Programmteil (Bild 1). Der Kopf enthält Daten und wird vom Compiler benötigt. Das was der Befehl tut steht als Assembler im Programmteil.

Man kann Befehle mit UNC name disassemblieren. Sich mit ^ name die Adresse CFA des Programmteils auf den Stack holen. Oder mit ^^ name die Adresse NFA des Kopfes auf den Stack legen. Der Kopf an der NFA-Adresse („Name Field Address“) besteht aus dem Header-Byte, den ASCII's für den Namen und der LFA-Adresse („Link Field Address“). Das Header-Byte enthält in den unteren 4 Bits die Länge des folgenden Namensfeldes. Und in den oberen Bits einige Steuerflags des Compilers. Das Namensfeld enthält den Namen als ASCII-Bytes. Die LFA die NFA des letzten zuvor definierten Namens und führt damit die Verkettung der Befehle durch.

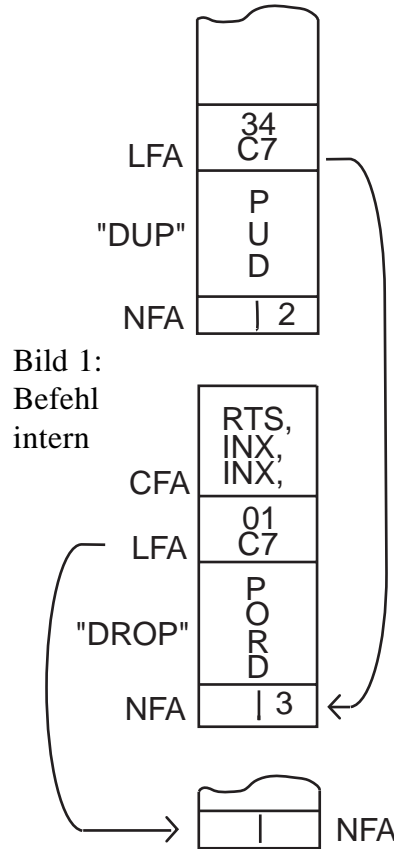


Bild 1: Befehl intern

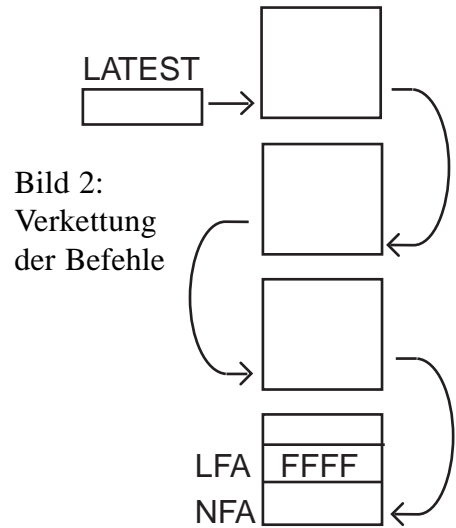


Bild 2: Verkettung der Befehle

Code

Historische Implementierungen erzeugten FORTH-Befehle („Words“) typisch als Adreßlisten die durch einen einfachen Interpreter abgearbeitet wurden („threaded code“). Das Verfahren ist unter dem Namen p-code auch für Pascal in den 70er Jahren favorisiert worden.

Bei etwas höherem Speicherverbrauch, aber deutlich höherer Geschwindigkeit verwendet nanoFORTH „JSR-threading“, Befehle werden also normalerweise als Unterprogrammlisten kompiliert. Z.B. würde

```
: X SWAP DROP ;
```

äquivalent kompiliert durch

```
:CODE X
^ SWAP JSR,
^ DROP JSR,
RTS,
CODE;
```

Für Zahlen („literals“) und Sprungbefehle muß man diese Unterprogramme durch echten Assembler „inline“ ergänzen. So sorgt der Compiler z.B. dafür, daß zur Laufzeit des Programms die Zahl 1234h auf den Stack gelegt wird:

Die Befehlsliste(„Dictionary“) dient zu Verwaltung der Befehle. Sie ist als eine Verkettung der Befehlsköpfe durch die LFA-Zeiger realisiert (Bild 2). Die Variable LATEST enthält die NFA-Adresse des neuesten Befehls. Dessen LFA enthält die NFA des nächsten Befehls. Und das setzt sich fort bis zum letzten Befehl dessen LFA den illegalen Wert FFFF enthält. Hier ist die Liste zuende.

Liste

Da Befehle die neu definiert werden sich immer nur auf Befehle beziehen können die bereits definiert wurden, ergibt sich in der Befehlsliste bezüglich der Funktion eine natürliche Anordnung wie in Bild 3 dargestellt. Die „Primitives“ bilden den Teil

der Befehle die die virtuelle 16 Bit Stackmaschine auf der 8 Bit CPU nachbilden. Dieser Nucleus ist in Assembler definiert. Alles was darüber liegt ist fast durchwegs in FORTH geschrieben. Ausnahme sind Befehlssteile die auf Geschwindigkeit optimiert werden müssen. Außer dem Parser ist das z.B. die Suchroutine für die Befehlsliste. Da das Suchen durch die Liste zeitraubend ist, erreicht man eine Verbesserung der Compilergeschwindigkeit wenn Teile der Liste ausgeblendet werden können. Das ist für den Assembler gangbar, er ist nur innerhalb :CODE CODE; und [CODE ... CODE] zugeschaltet (Bild 4).

```

...
X.   DEC,
H% 34 #. LDA,
    0,X STA,
X.   DEC,
H% 12 #. LDA,
    0,X STA,
...

```

Der Befehl UNTIL mit kurzem Rücksprung wird z.B. so kompiliert:

```

^ (BR) JSR,
  hhhh  BEQ,

```

Der Befehl (BR) nimmt eine Zahl vom FORTH-Stack und setzt das Zero-Flag in der CPU entsprechend.

Die Wahl wie man diese Funktionen im Compiler auslegt ist hauptsächlich ein Kompromiß zwischen Speicherverbrauch und Geschwindigkeit des erzeugten Codes. Ausreichende Compiliertgeschwindigkeit ist wegen der interaktiven Betriebsart erforderlich. Fehler-sicherheit ergibt sich automatisch wenn man das System simpel und transparent hält. Bedeutet aber auch, daß der Compiler auf Optimierungen in der Codeerzeugung verzichtet.

Bild 3: Anordnung der Funktionsgruppen.

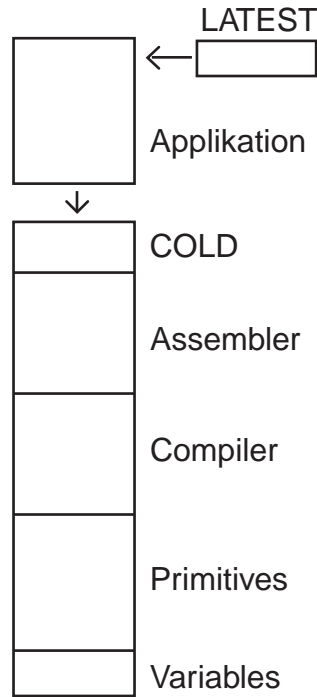
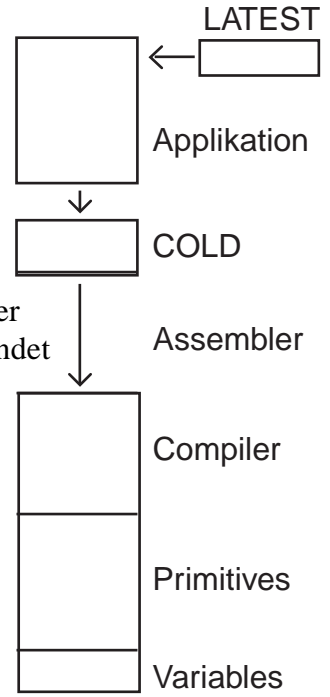


Bild 4: Assembler ausgeblendet



Ein Beispiel dafür sind Befehle wie H% oder ". . . ." die sich interpretierend und (innerhalb von Befehlen) compilierend unterschiedlich verhalten („state-smart“). Diese automatische Umschaltung erscheint zwar praktisch. Man kann dann eventuell aber durch Blick ins Listing

nicht immer sicher vorhersagen was der Compiler tun wird, da man den Zustand des steuernden Flags ja nicht kennt. Aus diesem Grunde so weit wie möglich in nanoFORTH vermieden worden.

Kompatibilität

Da sich andere Systeme an verschiedensten Spezifikationen wie fig-FORTH, FORTH-83 oder ANS-Forth anlehnen ist eine exakte Beschreibung der Unterschiede kaum möglich. Deshalb können in Tabelle 1 nur die Problempunkte angesprochen werden, wenn man nanoFORTH Source portieren will. Schematische Lösungen sind nicht möglich.

Umschreiben ist meist nur für Systemen die auch ein 16 Bit Stackmodell verwenden sinnvoll. Sowie bei Programmen die nur wenig Assembler enthalten.

Portierung von nanoFORTH für Bit Befehlen geschrieben und gelesen werden sollten. 6502 auf 68HC08 ist natürlich einfacher. Das FORTH ist praktisch identisch, der Assembler sehr ähnlich. Es ergeben sich hier jedoch andere Probleme.

Der 68HC08 ist big-endian , 6502 little-endian. Das kann auch innerhalb FORTH können bei unsauberem Speicherzugriff zur Problemen führen. Deshalb soll man aus Gründen der Portabilität in Speicher den man mit ! schreibt nicht mit C@ herumstochern. Das gilt sinngemäß auch bei 32 Bit Speicheroperationen die nur mit den dafür gedachten 32

Problem der MemoryMap ist, daß der 6502 Einplatinencomputer extern 32k ByteRAM hat. Der 68HC08 wird für einige Applikationen nicht genügend RAM haben.

Der 6502 hatte einen Metacompiler. Befehle wie META-ON , META1 können für den 68HC08 entfallen.

Tabelle 1: Portierung von nanoFORTH Code

Code	Funktion nanoFORTH	Funktion andere FORTHS
Der kritischste Punkt sind Befehle die identische Namen haben, sich aber unterschiedlich verhalten:		
4 1 DO ... LOOP	Zählt: 1, 2, 3, 4	Zählt: 1, 2, 3
+LOOP	Schleifenendwert: punktgenau treffen	Endwert: überschreiten
DO ..IF LEAVE THEN xyz LOOP	Beendet nach LEAVE Schleife bei Erreichen von LOOP	LEAVE springt sofort hinter LOOP
name	maximal 16 ASCII Zeichen, durchgehend Großschreibung	typisch bis 32 Zeichen, Groß/Kleinschreibung
U*	(UN1 UN2 - UD1)	
U/MOD	(UD1 UN2 - UN3 UN4)	
U/	(UD1 UN2 - UN4) UN3 = REM	
ERASE	FLASH löschen	ca. 00 FILL
Kleineres Problem sind Befehle die nur in nanoFORTH existieren und modifizierte Namen und Funktion haben:		
3 ZVARIABLE name	Variable liegt in ZeroPage belegt 3 Bytes	ca. VARIABLE name 1 ALLOT unklar ob ZeroPage.
LAND, LOR, LNOT	Verknüpfung von Flags	Befehle AND, OR, NOT verwendbar soweit das Flag sauber 0000 oder FFFF (besser: -1) ist.
TABLE	Kopf im Programmspeicher	Vgl. CREATE
1<SHIFT 4<SHIFT ... 1SHIFT> 4 SHIFT> ...	Shiftbefehle	1 <SHIFT oder 1 LSHIFT 1 SHIFT> oder 1 RSHIFT
B! B0! B@	Bitbefehle	mit AND, OR basteln
HOPP	3. Wert vorholen	ca. 2 PICK
IF. ELSE.	langer Sprung langer Sprung	IF ELSE
+C!	+! für Bytes	

Arithmetik

Nur ein Teil der in Tabelle 2 aufgeführten Befehle ist im Controller enthalten. Der Rest muß bei Bedarf kompiliert werden. Auch in einem 16 Bit FORTH treten insbesondere wegen Multiplikation und Division 32 Bit Daten auf. Aber auch

die Addition von zwei 16 Bit Zahlen kann ein 17 Bit Ergebnis haben. Da man neben vorzeichenlosen Zahlen auch Zahlen benötigt die im 2er-Komplement Vorzeichen beinhalten hat man letztlich 4 Formate.

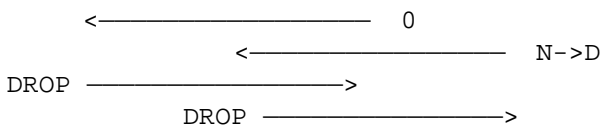
Format wandeln

Ein 32 Bit Datenwort liegt auf dem Stack als zwei 16 Bit Werte, wobei die obere Hälfte oben liegt. Einziger zur Formatwandlung benötigter Sonderbefehl ist N->D der entsprechend Bit 15 ein weiteres Datenwort mit dem Wert FFFF oder 0000 auf den Stack legt.

Tabelle 2: Arithmetische Befehle

32 Bit ohne Vorzeichen	32 Bit mit Vorzeichen	16 Bit ohne Vorzeichen	16 Bit mit Vorzeichen

Format wandeln



16 Bit auf 32 Bit erweitern

32 Bit auf 16 Bit kürzen

Arithmetik

D+	D+	+	+	Addition
D-	D-	-	-	Subtraktion
		U*	*	Multiplikation
		1<SHIFT	1<SHIFT	LSL „Logic Shift Left“ = ASL
		4<SHIFT	4<SHIFT	
		8<SHIFT	8<SHIFT	
D2*	D2*	2*	2*	Division
U/ U/MOD	/ /MOD			
U2/		1SHIFT>		LSR „Logic Shift Right“
		4SHIFT>		
		8SHIFT>		
	D2/		2/	ASR „Arithmetic Shift Right“
	DNEGATE		NEGATE	Vorzeichen wechseln
	DABS		ABS	Absolutwert bilden

Vergleichsbefehle

	U<	<	kleiner
UD>	U>	>	grösser
		0<	

Eingabe

D: D:

Ausgabe

D. SD. ND. SND.

Tabelle 3: erweiterte dezimale Ausgabebefehle

CD. \ (C1 -) C1 = 0 - 63, Prints 2 Chars, rightadjusted, with leading SPACES
 D. \ (UD1 -) Prints 10 Chars, rightadjusted, with leading SPACES
 SD. \ (D1 -) Prints Sign, 10 Chars rightadjusted, with leading SPACES

Arithmetik

Addition und Subtraktion haben den Vorteil, daß identische Befehle für Zahlen mit und ohne Vorzeichen verwendbar sind.

Die Umwandlung von positiv nach negativ oder umgekehrt geschieht mit NEGATE bzw DNEGATE. Man beachte bei der Verwendung, daß es für die negativste Zahl, also -32768, im positiven Bereich keine Entsprechung gibt. Dieser Wert muß also vermieden werden. Das Problem betrifft auch die Befehle ABS bzw. DABS.

Erweiterte Ausgabe & Eingabe

Für arithmetischen Anwendungen wird dezimale Ausgabe meist bevorzugt (Tabelle 3).

Der simple Befehl CD. kann kein volles Byte darstellen, weil er nur zwei Stellen ausgibt. Die beiden letzten Befehle beruhen auf(DD.) einen Grundbefehl, den man auf 2 - 5 Stellen einstellen kann.

Eingabe von Dezimalzahlen mit Vorzeichen oder 32 Bit Dezimalzahlen ist im Grundsystem nicht direkt möglich. Der Befehl D: liest eine 32-Bit Dezimalzahl mit Vorzeichen ein und legt sie auf den Stack.

```
---- ---- ---- | D: -2
---- FFFE FFFF |
```

Positives Vorzeichen und Vornullen sind optional. Verminderung auf 16 Bit kann durch einen folgenden DROP

erfolgen. Der Befehl ist nicht „state-smart“, deshalb nicht innerhalb Befehlen verwenden. Sondern mit DCONSTANT indirekt lösen.

Tabelle 2: Arithmetik
Erweiterung für 32 Bit

D+ \ (D1 D2 - D3) D1 + D2 = D3
 D- \ (D1 D2 - D3) D1 - D2 = D3
 D@ \ (Addr1 - D1) Speicher lesen
 D! \ (D1 Addr1 -) Speicher schreiben nur RAM
 D, \ (D1 -) nur RAM

Erweiterung für Vorzeichen

* \ (N1 N2 - D1) N1 * N2 = D3
 / \ (D1 N1 - N2) D1 / N1 = N2
 /MOD \ (D1 N1 - N3 N2) D1 / N1 = N2 N3 = Rest
 */ \ (N1 N2 N3- N4) (N1 * N2) / N3 = N4

Sonderfunktionen

NEGATE \ (N1 - N2) Vorzeichen wechseln
 DNEGATE \ (D1 - D2)
 ABS \ (N1 - UN2) Absolutwert bilden
 DABS \ (D1 - UD2)
 N->D \ (N1 - D1) 16 auf 32 Bit wandeln
 2* \ (N1 - N2) N1 * 2 = N2 Alias von 1<SHIFT
 D2* \ (D1 - D2) D1 * 2 = D2
 2/ \ (N1 - N2) N1 / 2 = N2
 D2/ \ (D1 - D2) D1 / 2 = D2
 > \ (N1 N2 - F1) Vergleichsbefehl
 < \ (N1 N2 - F1)
 0< \ (N1 - F1) entspricht: 8000h AND

Nutzungsvertrag

Für Eigengebrauch ist es dem Lizenznehmer beliebig erlaubt Controller mit der Firmware zu vervielfältigen oder Sicherheitskopien zu erstellen.

Wenn Controller Dritten zugänglich gemacht werden sind geeignete Schutzmaßnahmen gegen Auslesen der Firmware zu treffen (vgl. Seite 1.8) und Maßnahmen zu deren Umgehung im Applikationsprogramm zu unterlassen.

Da jeder Lizenznehmer ein individuelles, „steganographisch“ gekennzeichnetes Exemplar der Firmware erhält, ist die Quelle eventueller Raubkopien eindeutig identifizierbar.

Impressum

Rafael Deliano
 Steinbergstr. 37
 82110 Germering
 Tel 089/8418317
 mail@embeddedFORTH.de

Terminal- programme

Für nanoFORTH genügen die simpleren Versionen. Untersucht wurden nur stichprobenartig die benötigten Einstellungen 9600 Baud 8N1, XON/XOFF, TTY, Upload & Download Text. Option VT52 ist zusätzlich für Applikationen manchmal wünschenswert. Als Testrechner dienten ein alter XT-Laptop mit DOS 3.3 oder ein etwas neuerer PC mit Windows 98.

Viele DOS-Versionen laufen auch noch unter Windows, aber das ist eigentlich keine sinnvolle Kombination.

Bezüglich der Bedienung sollte man sich keiner Illusionen hingeben: man ist auf MS-DOS mit Funktionstasten produktiver als mit Windows und der Maus. Allerdings haben mit Features überladene Programme meist komplizierte Konfigurationsmöglichkeiten, die durch die pull-down Menüs a la MS-DOS kaum übersichtlich darstellbar sind. Steuerung über Escape-Sequenzen ist die unterste Stufe der Ergonomie und bei einfachen Programmen auch noch anzutreffen.

Das gilt sinngemäß auch für den hier nicht angesprochenen Editor. Für zügiges Arbeiten ist ein schneller Wechsel zwischen beiden Programmen nötig. Bei kleinen, schnell startenden MS-DOS Programmen kann man problemlos wechseln. Besonders wenn man das wie bei Shamrock explizit vorgesehen hat. Bei Windows ist ein ins Terminalprogramm integrierter Editor oft die bessere Wahl.

Bei der Auswahl der Programme wurden nur solche berücksichtigt die im www zu finden sind. Früher bekannte Programme wie Telix, Telemate, Procomm sind nicht getestet worden, weil die im www noch vorhandenen Versionen nicht zeitgemäß sind bzw. nicht den aktuellen, kostenpflichtigen Versionen entsprechen. Wer Geld ausgeben will, soll das bei deutschen Anbietern tun. Dort erhält er dann auch meist problemlos effektiven Support.

Schnittstelle

Praktisch alle Programme senden für die Löschtaste den Code 08h. In nicht jedem Fall wird dabei das überschriebene Zeichen auf dem Bildschirm auf SPACE 20h gelöscht

wie das in nanoFORTH intern geschieht. Mit der Diskrepanz kann man aber leben.

Bezüglich des Zeilenendezeichens ist für nanoFORTH eigentlich vorgesehen, daß ein Programm das das Steuerzeichen LF für Zeilenvorschub braucht, dieses auch als CRLF selbst sendet. Dem ist aber nicht so, manche Programme senden nur CR, brauchen aber CRLF als Echo. Wurde in nanoFORTH per Flag-Byte LF' im EEPROM gepatcht das Senden von LF nach empfangenem CR erzwingt. Dazu muß man dieses Byte von 00 auf FF umprogrammieren: FF LF' EC! .

Deutlich problematischer ist es, wenn das Programm ein empfangenes Steuerzeichen XOFF 13h als Grafikzeichen am Bildschirm ausgibt. Das schließt den Betrieb mit nanoFORTH weitgehend aus, da hier am Zeilenanfang reichlich isolierte XOFF ohne vorangegangenes XON auftreten.

PCTERM 2.0
MS-DOS 1989 6,5kByte
www.shamrock.de

Leider auch nichtmehr als Freeware verfügbar. Das von mir bevorzugte Programm bietet genau den Funktionsumfang den man für Controller benötigt.

Unicom 2.08
MS-DOS 1989 25kByte
www.shamrock.de

Freeware. Hat gegenüber PCTERM mehr Features, aber schlechtere Bedienbarkeit. Kann mit Farbbildschirm und Maus besser sein als mit monochromen Laptop ohne Maus. Von Interesse besonders, daß man zwei Ports farblich unterschiedlich auf gleichem Bildschirm darstellen kann.

Mit passendem Adapterstecker und zwei COM-Ports kann man also bidirektionalen Datenverkehr auf einer V24 bequem überwachen.

CONEX 7.5
MS-DOS 1987 64kByte
Erhard Hilbig

Freeware. Hat sehr viele nützliche technische Features bezüglich, Ports, Filetransfer, Terminal-Emulationen. Oberfläche aber seehr spartanisch, auch wenn das knappe Handbuch hilfreich ist. Sendet nur CR, braucht aber LF für neue Zeile. Echtes Problem: zeigt Grafikzeichen für XOFF an. Für nanoFORTH nicht geeignet, aber technisch interessant.

Rterm32 0.199
Win95/98/NT 2004 220k
www.iep.de

Simpel, wirksam. Es gäbe sogar gute Dokumentation, aber die Einstellung ist wirklich selbsterklärend. Sendet nur CR, braucht aber CR LF für neue Zeile.

Terminal für Win3.11
(Win98) 1997 144k

Von Future Soft Inc für Microsoft entwickelt. Läuft auch noch als Variante auf Windows 98. Hat ziemlich genau den benötigten Funktionsumfang, viel mehr aber auch nicht. Keine Probleme festgestellt.

HyperTerminal PE 6.3
Win95/98/ME/2000/NT 2001
www.hillgraeve.com

Mit Windows95/98 wurde ein frühes Hyperterminal von Hilgraeve Inc. ausgeliefert, das nicht die Freude des Anwenders war. Hier wurde die Version PE verwendet die man bei Hilgraeve direkt erhält, für Privat-anwender kostenlos. Tut mehr als die Beilage zu Win3.11. Aber nicht zuviel, denn Hilgraeve will auch sein kostenpflichtiges Terminalprogramm verkaufen.
Keine Probleme festgestellt.

Teraterm Pro 2.3
Win95,98 1998

Freeware von Takashi Teranishi
Hat kein VT52 sondern nur VT100 aufwärts.
Keine Probleme festgestellt.

ZOC 4.14
Win9x/ME/NT/2000/XP 2004
Markus Schmidt
www.emtec.com

Kostenpflichtig, aber unbeschränktes Programm kann für 30 Tage getestet werden kann. Ziemlich viele Features die für Controller nicht benötigt werden. Aber natürlich sehr praktisch sein können speziell der eingebaute Editor.
Keine Probleme festgestellt.

S3Term V1.0
Win?
Carel Hauptfleisch
www.MPfree.zone

Freeware. Das SuperSimpelSerial Terminal hat zwar nicht für nanoFORTH erforderlichen Merkmale aber für Fehlersuche nützlich: es macht zwei Fenster auf. Mit passendem Adapterstecker und zwei COM-Ports kann man also bidirektionalen Datenverkehr auf einer V24 bequem darstellen.

Index Handbuch

GP32

- 1.1 Inbetriebnahme
- 1.2 Einführung in FORTH
Speicher RAM , I/O
- 1.3 Speicher FLASH , EEPROM
- 1.4 Schreibschutz mit Jumper
EEPROM SYS-EE
Prüfsummen
- 1.5 Reset
- 1.6 Bustakt ändern
- 1.7 Programmiergerät
- 1.8 Sicherungen, Bradboards

FORTH

- 2.1 Kommentare, Stack, Zahlenformat
- 2.2 Zahlenbasis, Stackbefehle, Arithmetik
- 2.3 Boolesche-, Schiebebefehle, Speicherzugriff
- 2.4 Bitbefehle, Konstanten, Variablen
- 2.5 Flags, Terminal-I/O
- 2.6 Befehle definieren
- 2.7 Sprungbefehle
- 2.8 Source compilieren, Marker, Fehlersuche
- 2.9 Parser
- 2.10 Zeit- & Speicherbedarf
- 2.11 Fehlermeldung
- 2.12 Index FORTH-Befehle

Assembler

- 3.1 Schreibweise Adressierung
- 3.2 Einbindung in FORTH, CCR-Flags
- 3.3 X-Register, CPU-Stack, Labels
- 3.4 Sprungbefehle, Unterprogramme
- 3.5 Daten bewegen
- 3.6 Arithmetik
- 3.7 Boolesche Befehle
- 3.8 Schiebebefehle Interrupts
- 3.9 Sonderbefehle, Liste der Opcodes
- 3.10 Liste der Opcodes
- 3.12 Tabellen Zahlenwandlung

Anhang

- 4.1 FORTH intern
- 4.2 Kompatibilität
- 4.4 Arithmetik
- 4.5 Nutzungsvertrag
Impressum
- 4.6 Terminalprogramme
- 4.7 Index Handbuch