Die Programmiersprache FORTH

1. über den Erfinder der Sprache

2. Allgemeines zu FORTH

- Besonderheiten
- Eckdaten
- Anwendungen

3. FORTH im Detail

- Das Wortkonzept
- Vokabulare
- Grundausrüstung eines Programmierers

Charles H. Moore

- geb 1938 in McKeesport, Pensilvania

- Studium der Physik am MIT mit einem

  National Merit – Stipendium

-1960 Doktor der Physik

  - Arbeiten an Datenreduktion für

    Gammastrahlensatelliten Explorer IX

  - danach Wechsel nach Stanford

    - zwei Jahre Studium der Mathematik

Charles H. Moore

- Arbeitsfeld: Astronomie und Astrophysik sowie Steuerungstechnik
- Nutzung von 'elektronischen Rechenmaschinen' sehr von Vorteil
- Autodidakt auf dem Gebiet der Informatik
- erste Kontakte mit Programmiersprachen ab 1958
    - erlernte LISP sowie FORTRAN2 von John McCarthy
    -> Entwicklung von Programmen zur Verarbeitung von Daten mondbeobachtender Satelliten am Smithsonian Astrophysical Observatory
    - nächster Schritt: Algol für Burroughs B5500

Computer··Science Center
University of Virginia

Burroughs B5500 Computer
Installed July 1964

Charles H. Moore

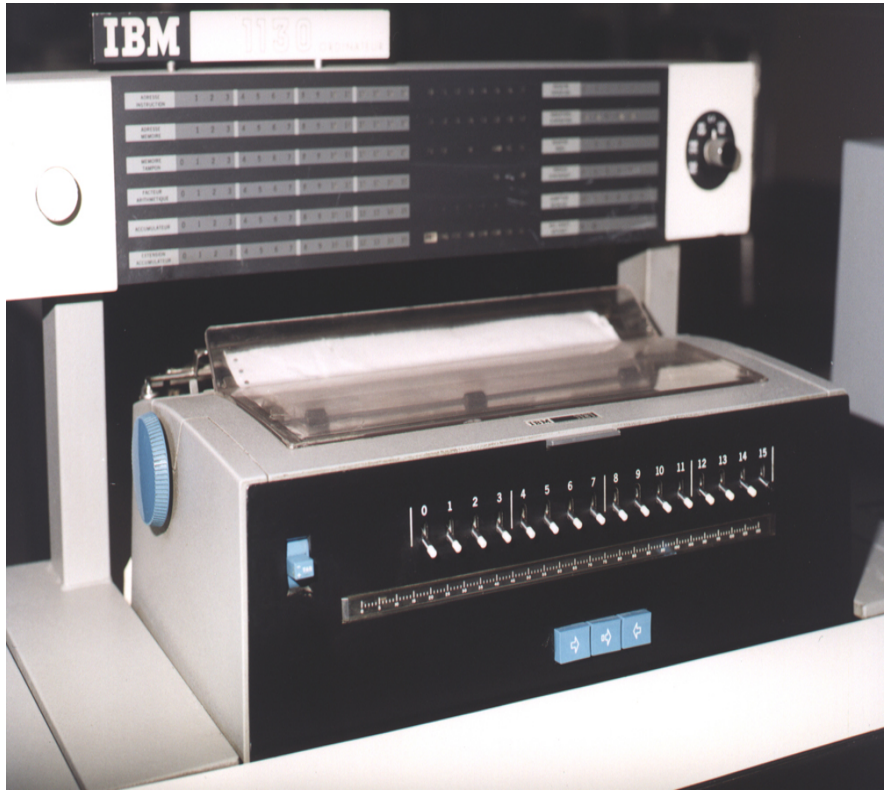
- erarbeitete damit Optimierungen zur Steuerung von Elektronenstrahlen am Stanford Linear Accelerator Center 1962
- mit den gesammelten Kenntnissen entwickelte er einen Fortran– Algol – Übersetzer für einen Timesharing service (1964)
- Echtzeit - Gaschromatograph auf seinem ersten Minicomputer (1965)
- Aneignung von Cobol für ein Netzwerksystem (1968)

- ab 1968: Entwicklung der Sprache FORTH

Charles H. Moore

- baute Stück für Stück seine persönliche Softwarebibliothek auf
- sein Arbeitsgerät war ein IBM 1130, mit dem ein erstes graphisches (buntes!) Terminal angeschlossen war (IBM 2250)
- die erste Anwendung seiner eigene Programmiersprache war eine Steuerungssoftware für ein 9m – Radioteleskop zur Beobachtung von Radiowellen im Weltall
- 1973 erste 'fertige' Version von FORTH
- Gründung der FORTH Inc. ebenfalls 1973
- in den nächsten 10 Jahren portierte er FORTH-Systeme auf viele Mini-, Micro- und Mainframecomputer

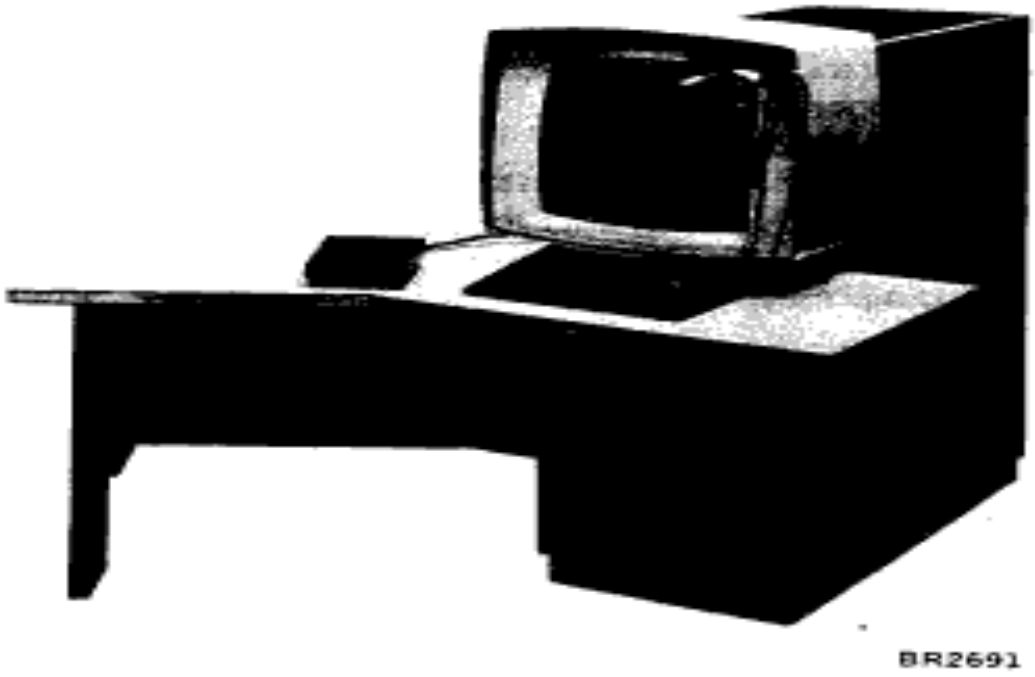
Charles H. Moore



IBM 1170

Charles H. Moore

Charles H. Moore



IBM 2250

Charles H. Moore

- suchte neue Herausforderung in der Chipentwicklung
- er fasste den Entschluss, die FORTH-Architektur in Hardware zu realisieren
- wurde auf dem Weg Gründungsmitglied von Novix Inc.
- Ergebnis seiner Bemühungen: NC4000 Chip
- Chip arbeitete in einem Gate-Array
- die Architektur des NC4000 wurde weiter verfeinert und später in Bausätzen vertrieben
- Derivat dieses Modells wurde kommerziell recht erfolgreich "RTX2000" (Harris Semiconductor 1988)
- dieser wurde in Weltraumanwendungen verwendet

Charles H. Moore

- Moore beschäftigte sich weiterhin mit der Entwicklung von Steuerchips
- Früchte seiner Arbeit u.a.
    - ShBoom (1985)
        - Low-cost-Prozessor (Herstellungskosten ca. 20$)
        - wird noch heute vertrieben
    - MuP21 (1988)
        - entwickelte eigenes Entwurfswerkzeug für dessen Modellierung
        - vereinte mehrere spezialisierte Funktionsgruppen in einem Chip
    - F21 (1993)
        - Weiterentwicklung des MuP21
        - integrierte Netzwerkschnittstelle

Charles H. Moore

- .21 (ab 1996)
  - ähnliche Struktur wie Vorgänger (modularer Aufbau)
  - getrimmt auf hohe Leistung für Inernetanwendungen
- c18 (2001)
  - sehr einfacher Kern, von dem mehrere auf einen Chip passten
  - vorgesehen für Hochleistung bei geringem Stromverbrauch (Satellitentechnik)

FORTH - Applikationen

- Steuerung von Radioteleskopen und Satelliten
- der Flughafen von Riad
- Steuerung verschiedener Experimente im Space Shuttle
- das Paketverfolgungssystem von Federal Express
- verschiedenste Robotersteuerungen
- die Steuerung des gemeinsamen Teilchenbeschleunigers der TU München und der Ludwig-Maximilian-Universität
- Vielzahl von Gerätesteuerungen auf Mikrocontrollerbasis
- die Firmware der der durch die Internationale Atomenergiebehörde eingesetzten Überwachungskameras
- seismische Aufzeichnungsgeräte
- Steuerung von Zementdrehöfen, Sägewerken
- Meßgeräte zur Belastungsprüfung in der Automobilindustrie
- OpenFirmware in SUN-Workstations und PowerMacs

Die Programmiersprache FORTH

FORTH – eine Programmiersprache der Vierten Generation

Generationen von Programmiersprachen

1. Generation:

Maschinencode an sich — 1011 0011 0000 0011
1000 0000 1100 0011
0000 0100

2. Generation: — mov bl,3
Assembler und Dialekte — add bl,4

3. Generation: — summe = 3 + 4
Abstraktion vom Assembler
Einführung von Schleifenkonstrukten, Variablen
jede höhere Sprache

Die Programmiersprache FORTH

4. Generation:

alle Eigenschaften der 3. Generation,
erweitert um Konstrukte für spezielle Probleme
SQL, Scriptsprachen

5. Generation:

theoretische Beschreibung
'deklarative PL'
in der Sprache wird die Aufgabe beschrieben, die der Computer dann automatisch löst/lösen soll
Prolog (Begriff auch hauptsächlich in der KI verwandt)

Die Programmiersprache FORTH

Besondere Eigenschaften

- eine der jüngeren Programmiersprachen
- sehr vielseitige Sprache mit breitem Anwendungsspektrum
- nach den Vorlieben seines Erfinders schnell und kompakt, vorgesehen für Echtzeitanwendungen
- betriebssystemtauglich
- große Unterschiede zu Sprachen wie Pascal, Algol oder Basic
- Stackkonzept verhindert Entsprechung bestimmter Datenstrukturen
- lauffähig auf Maschinen unterschiedlichster Größe
- sehr geeignet für Meßdatenerfassung, Prozess- und Maschinensteuerung, automatische Testsysteme

Die Programmiersprache FORTH

Besondere Eigenschaften:

- Anliegen des Erfinders war Kombination der Geschwindigkeit von Asm-Code mit den Strukturierungsmöglichkeiten von Hochsprachen

(komplexe Programme in reinem Assembler zu schreiben ist hart, fragen Sie einen Coder Ihrer Wahl ;)

Herangehen Moores:

- Datenablagemöglichkeiten im asm sind RAM, Register, Stack
- will man komplexere Programme ausführen (Multitasking, rekursive Programme) muss man Daten sichern
- einzige praktikable Moglichkeit ist der Stack

-> besondere Sorgfalt bei der Arbeit mit dem Stack, durch den Mischmasch von Daten und Adressen

Die Programmiersprache FORTH

seine Idee für FORTH:

man betreibe zwei separate Stacks

- 'Datenstack' dient zur Ablage von Daten
- 'Returnstack' trägt Daten von Kontrollstrukturen, Rücksprungadressen

dadurch Vereinfachung für Programmierer, da Stackjonglieren entfällt!

Die Programmiersprache FORTH

weitere Eigenschaften:

- FORTH ist typenlos (alles ist int bzw 8/16/32-Bit binär)
- bedingt durch Stackarchitektur: Verwendung der UPN oder postfix-Notation

  prefix: +34

  infix: 3+4

  postfix: 34+
- FORTH ist eine strukturierte Sprache (kein GOTO,...)
- FORTH ist erweiterbar
- Programme haben typischerweise einen Zeitoverhead von 50% - 200 % ggü optimierten Maschinenprogrammen
- Wortkonzept der Sprache lenkt einen automatisch in Richtung eines modularen Programms (Top-Down Entwurf und Bottom-Up Implementierung)

Die Programmiersprache FORTH

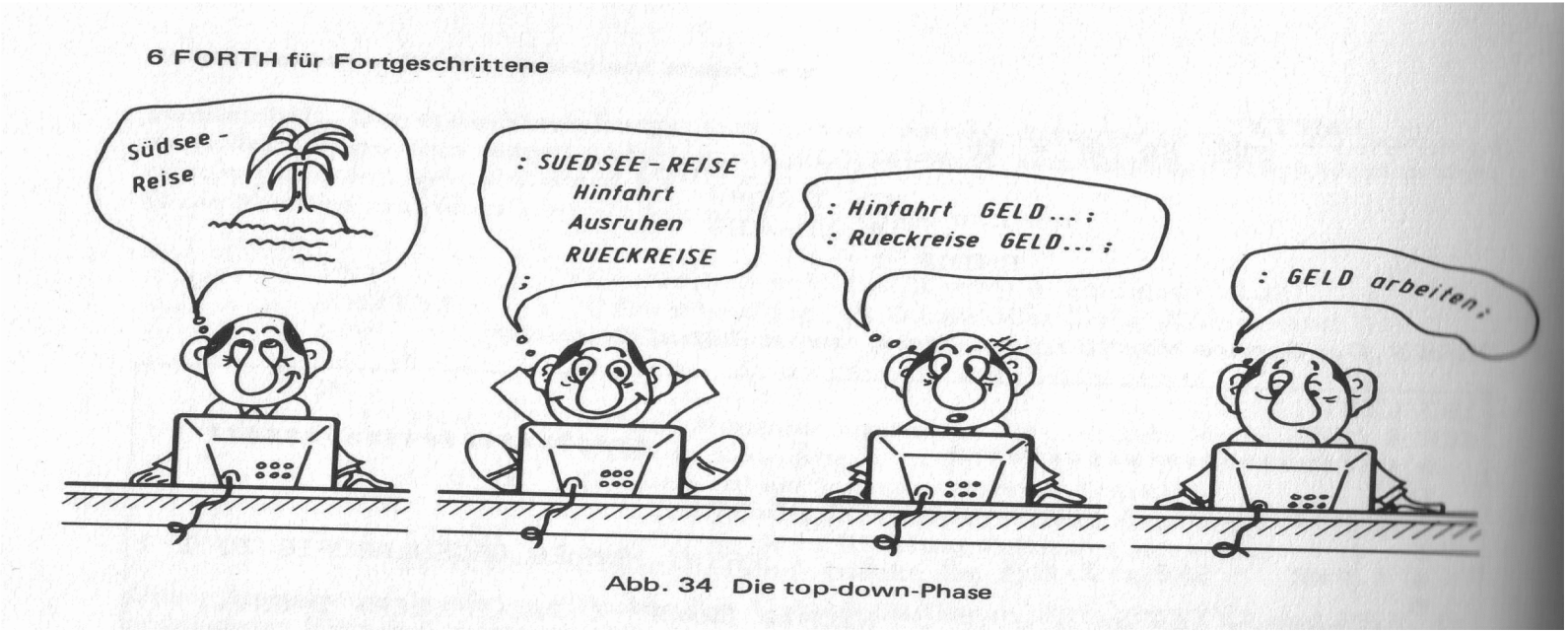


Abb. 34 Die top-down-Phase

Die Programmiersprache FORTH

weitere Eigenschaften:

- OS- Eigenschaften

-zu den Grundmodulen gehören Terminal I/O Handler und DOS-Bestandteile

- alle diese Werkzeuge sind direkt aus FORTH nutzbar

-> sehr umfangreiche Kontrolle über das System (z.B Real- u. Protected-Mode)

Die Programmiersprache FORTH

weitere Eigenschaften:

es existieren (immer noch) eine Vielzahl von Dialekten

- URTH
- STOIC
- CONVERS
- MMSFORTH

manche entwickelt für bestimmte Rechner (CONVERS für Digital-Maschinen) andere für spezielle CPU (ZIP für Z80)

FORTH - Details

Das Wortkonzept

- jedes Haupt- oder Unterprogramm, jede Subroutine ist ein Wort
- Wort String aus ASCII-Zeichen der Länge 1 – 31
- Wort setzt sich zusammen aus anderen Worten oder Maschinen- bzw Asm- Code
- erzeugt der User ein neues Wort, wird im 'Dictionary' ein neuer Bereich angelegt
  - Name in ASCII-Bytes
  - Codebereich mit Wortadressen bzw Maschinen-/Asm-Code

FORTH - Details

Erzeugung eines neuen Wortes:

- Zuhilfenahme einiger Schlüsselworte

CREATE:

- legt neuen Wortbereich im Dictionary an
- Wort noch nicht nutzbar

CODE und END-CODE:

- klammern einzufügenden Asm-Code
- so die Möglichkeit, tief ins System einzugreifen -> Steuerung

SMUDGE:

- setzt das S-Bit und macht das Wort somit gültig

FORTH - Details

- Spezielle Arten von Worten

  (interner Datentyp ist immer 16Bit-Integer!)

xxxx	CONSTANT	name
xxxx	VARIABLE	name
xxxx	USER	name

xxxx.. 16 Bit- Integer

FORTH - Details

CONSTANT:

- liefert bei Aufruf 16 Bit Integer auf den Stack

VARIABLE:

- liefert bei Aufruf eine Speicheradresse auf den Stack, in dem eine 16 Bit Integer abgelegt werden kann

USER:

- liefert eines Adressoffset x bezogen auf y, so daß a=x+y
- in gängigen FORTH-Systemen werden USER-Variablen als Systemvariablen benutzt (ca. zwei Duzent)

z.B:

DP.. dictionary pointer

BASE.. Zahlenbasis

S0.. Parameterstack (oder einfach Stack)

R0.. Returnstack

FORTH - Details

- USER-Variablen ermöglichen
  Multi-User-Betrieb
    oder
  Multitasking
  (einfacher Weg: UP (user area pointer) auf Adressbereich des
    nächsten Users setzen)

FORTH - Details

Die Vokabulare

- der Programmierer hat jederzeit die Option, ein neues Vokabular/Wörterbuch/Dictionary anzulegen

--> Möglichkeit, Pakete oder Namespaces zu erzeugen

> vocabulary wörterbuchname

- legt neues WB an
- Wechsel zwischen WB:

> wörterbuchname (Basisdictionary: FORTH)

Die Erweiterbarkeit der Sprache steckt in den Vokabularen

Der Entwickler kann sich passend für die entsprechenden Aufgaben 'Wergzeugkästen' zusammenstellen und Teile in neuen Worten wiederverwenden!

FORTH - Details

Programmieren in FORTH

- FORTH-Programmierung ist stackorientiert
- im Gegensatz zu anderen (höheren) Sprachen ist die Arbeit mit dem Stack essentiell
- alle Notationen in UPN
- Programmierer ist dafür zuständig, dass der Operator die entsprechenden Daten auf dem Stack vorfindet (und auch in der richtigen Reihenfolge!)
- Ergebnis wird ebenfalls auf dem Stack abgelegt
- kein Typechecking o.ä. (alles Int)
- Interpretation der 16 Bit-Werte grundsätzlich im Zweierkomplement

FORTH - Details

Programmieren in FORTH

___- Notation/Beschreibung von Worten:

(+: n1, n2 -> summe)

Wort erwartet legt ab

- wichtige Worte zur Darstellung/Wiedergabe von Werten:

x	EMIT	x.. ASCII-Zeichen
	CR	carriage return
	SPACE	ein Leerzeichen
x	SPACES	x Leerzeichen

- recht spartanischer Bausatz, aber den Möglichkeiten (Terminals) angemessen

FORTH - Details

Programmieren in FORTH

___- Arithmetikbefehle:

PLUS +

MINUS -

TIMES *

DIV /

DECIMAL

HEX

*/M Division mit Rest (*/M p,q ->rest,erg)

DPLUS, DMINUS,... für 32Bit-Werte

CPLUS, CMINUS,... für 8Bit-Werte

Bsp:

>5000 <-- 16Bit

>5000. <-- 32Bit

FORTH - Details

Programmieren in FORTH

___- Gleitkommabefehle:

Gleitkommaoperationen nutzen eigenen FP-Stack

FLOAT

FPLUS f+

FMINUS f-

FTIMES f*

FDIV f/

Bsp:

>25e-1 FDUP f* f. <enter>

>6.25 ok.

FORTH - Details

Programmieren in FORTH

___- Logikbefehle:

recht überschaubar ;)

drei an der Zahl

AND

OR

XOR

kein einfaches NOT (Kombination der drei anderen)

- Vergleichsoperatoren:

= (n1,n2 -> f)

< (n1,n2 -> f)

> (n1,n2 -> f)

FORTH - Details

Programmieren in FORTH

___- adressbezogene Befehle:

!	n	addr	speichert n unter Adresse addr
@		addr	liest 2 Byte ab Adresse addr und schreibt sie auf den TOS(**T**op**O**f**S**tack)
analog			
C!			
C@			für 8 Bit

--> diese beiden Befehle sind Grundlage für die Kommuikation mit dem Rest des Systems

FORTH - Details

Programmieren in FORTH

___- weitere adressbezogene Befehle:

TOGGLE	addr	b	xor zwischen Adressinhalt von addr und Byte b
CMOVE	von nach	b	kopiert u Byte von 'von' nach 'nach'
BLANKS	addr	u	schreibt u Byte Leerzeichen ab Adresse addr
?			Interpretiere TOS als Adresse und lies Wert

FORTH - Details

Programmieren in FORTH

___- Befehle zur Stackmanipulation:

DROP	(n1 ->)
DUP	(n1 -> n1,n1)
SWAP	(n1,n2 -> n2,n1)
OVER	(n1,n2 -> n1,n2,n1)
ROT	(n1,n2,n3 -> n2,n3,n1)

- Stackmunipulationsbefehle sind notwendiges Übel der Sprache
- machen die Gruppe der mit am häufigsten verwendeten Befehle aus

FORTH - Details

Programmieren in FORTH

___- Befehle zur Manipulation des Returnstacks:

- normale Nutzung zur Haltung von Schleifendaten, Rücksprungadressen u.ä
- Vorsicht ist geboten, damit nicht aus Versehen Rücksprungadressen verloren gehen

>R	(n ->)	TOS -> TOR
<R	(n ->)	TOR -> TOS
R	(-> n)	TOR -> TOS (TOR wird erhalten)
I	(-> n)	kopiert den aktuellen Loop Index nach TOS

FORTH - Details

Programmieren in FORTH

___- Elementare Stringbefehle:

." Das ist ein String"

- ' ." ' und abschliessendes " umfassen String
- darf kein "" beinhalten
- String wird intern wie Befehlswort behandelt und kann auch so verwendet werden

EXPECT	(addr, n ->)	legt String von n chars ab Adresse addr ab
TYPE	(addr, n ->)	gibt n chars ab Adresse addr aus
COUNT	(addr1 -> addr2,n)	gibt von addr ab die Anzahl der chars im String und dessen Startaddresse zurück
KEY	(-> n)	einzelnes ASCII-Zeichen auf TOS

FORTH - Details

Programmieren in FORTH

___- Strukturierung von Programmen:

Verzweigung:

IF -ELSE -ENFIF

IF (TOS ungleich 0)
{ Code im IF-Zweig}
ELSE (TOS = 0)
{ Code im ELSE-Zweig}
ENDIF

FORTH - Details

Programmieren in FORTH

___- Strukturierung von Programmen:

Schleifen:

DO LOOP

Endwert
Startwert
DO
{ Code}
LOOP

bzw

DO +LOOP

Endwert
Startwert
DO
{ CODE }
Inkrement
+LOOP

Bemerkung: DO-Loops sind das langsamste Schleifenkonstrukt, das FORTH zu bieten hat (aber immer noch 5x bis 10x schneller als vergleichbares in BASIC (Verzicht auf Typechecking)

FORTH - Details

Programmieren in FORTH

___- Strukturierung von Programmen:

Schleifen:

Vorzeitiges Verlassen von Schleifen mit

LEAVE

sofortiges Verlassen:

R> R> DROP DROP ;S (;S.. Fortsetzung an Rückkehraddresse der Schleife)

undefinierte Anzahl an Durchläufen:

Nutze DO +LOOP und wähle Inkrement 0

FORTH - Details

Programmieren in FORTH

___- Strukturierung von Programmen:

weitere Schleifen:

BEGIN UNTIL

Verarbeitung: -Lesen von BEGIN sichert Startaddresse auf TOS

-Lesen von UNTIL prüft, ob TOS ungleich 0

- falls ja -> neuer Durchlauf

BEGIN WHILE REPEAT

BEGIN

{ Manipulation des Abbruchflags }

WHILE

{ Code }

REPEAT Prüfung, ob Abbruchflag ungleich 0 ->BEGIN

FORTH - Details

Programmieren in FORTH

___- Strukturierung von Programmen:

weitere Schleifen:

Endlosschleife

```
BEGIN
    { Code }
AGAIN
```

- keine Prüfung von Parametern -> schnellste Schleife
- Verlassen mit ' ;S ' (Fortsetzen amRückkehrpunkt)

FORTH - Details

Programmieren in FORTH

___-die häufigsten Befehle:

Tafel 4 Die häufigsten FORTH-Befehle

Operanden :

n , n1 ...	16-Bit-Zweierkomplementzahlen
d , d1 ...	32-Bit-Zweierkomplementzahlen
u , u1 ...	vorzeichenlose 16-Bit-Zahlen
ud	vorzeichenlose 32-Bit-Zahlen
addr	Adresse
b	8-Bit-Byte
c	7-Bit-ASCII
f	Boolesches Flag (16 Bit, # ϕ = wahr)

Stackbewegungen :
Ein- und Ausgaben immer von links nach rechts ;

der *Top-Of-Stack* (TOS) ist stets rechts außen dargestellt

Second = Zahl unter dem TOS
Third = Zahl unter dem Second

Zahlensysteme

DECIMAL	(→)	deklariert Dezimal-System
HEX	(→)	deklariert Hexadezimales Zahlensystem
BASE	(→ addr)	System-Variable, welche die Zahlenbasis hält

Stack – Manipulationen

DUP	(n → n n)	kopiert (dupliziert) den TOS
–DUP	(n → n ?)	dupliziert nur dann, wenn ungleich Null
DROP	(n →)	beseitigt den (aktuellen) TOS
SWAP	(n1 n2 → n2 n1)	vertauscht die beiden oberen Zahlen des Stacks
OVER	(n1 n2 → n1 n2 n1)	kopiert den Second zum (neuen !) TOS
ROT	(n1 n2 n3 → n2 n3 n1)	rotiert den Third zum TOS
>R	(n →)	bringt den TOS zum Return-Stack (Zwischenspeicherung, Gebrauch mit Vorsicht !)
R>	(→ n)	holt Wert vom Return-Stack zum TOS zurück
R	(→ n)	kopiert den Return-Stack-Top zum TOS

Arithmetik

+	(n1 n2 → Summe)	Addition von 16-Bit-Zahlen (16-Bit-Summe)
D+	(d1 d2 → Summe)	Addition von 32-Bit-Zahlen (32-Bit-Summe)
–	(n1 n2 → Diff.)	Differenz n1 – n2
*	(n1 n2 → Produkt)	16-Bit-Produkt zweier 16-Bit-Zahlen
/	(n1 n2 → Quotient)	16-Bit-Division mit 16-Bit-Ergebnis

FORTH - Details

Tafel 4 Die häufigsten FORTH-Befehle (Fortsetzung)

MOD	(n1 n2 → Rest)	Modulo-Division (→ übergibt Teiler-Rest)
/MOD	(n1 n2 → Rest Quot.)	Division mit Rest und Quotient als Resultat)
*/MOD	(n1 n2 n3 → Rest Quot.)	Multiplikation und anschließende Division mit 32-Bit-genauem Zwischenergebnis (n1 * n2 / n)
*/	(n1 n2 n3 → Quotient)	wie */MOD, jedoch lediglich Quotient
M/MOD	(ud1 u2 → u3 ud4)	Division einer vorzeichenlosen 32-Bit-Zahl mit Übergabe des 16-Bit-Restes und des 32-Bit-Quot.
MIN	(n1 n2 → Minimum)	übergibt die kleinere von zwei Zahlen
MAX	(n1 n2 → Maximum)	übergibt die größere von zwei Zahlen
ABS	(n → u)	bildet Absolutwert einer 16-Bit-Zahl
DABS	(d → ud)	bildet Absolutwert einer 32-Bit-Zahl
MINUS	(n → −n)	wechselt das Vorzeichen einer 16-Bit-Zahl
DMINUS	(d → −d)	wechselt das Vorzeichen einer 32-Bit-Zahl
1+	(n → n+1)	incrementiere den TOS mit 1
2+	(n → n+2)	incrementiere den TOS mit 2
Logische Befehle		
AND	(n1 n2 → log. UND)	bitweise logische UND-Verknüpfung
OR	(n1 n2 → log. ODER)	bitweise logische ODER-Verknüpfung
XOR	(n1 n2 → log. EXOR)	bitweise Exclusiv-ODER-Verknüpfung
Vergleichs-Operatoren		
<	(n1 n2 → f)	Flag = 1 , falls n1 kleiner n2
>	(n1 n2 → f)	Flag = 1 , falls n1 größer n2
=	(n1 n2 → f)	Flag = 1 , falls n1 gleich n2
0<	(n → f)	Flag = 1 , falls der TOS negativ ist
0=	(n → f)	Flag = 1 , falls der TOS gleich Null ist (negiert Wahrheitswert von Flags)
Speicherbezogene Befehle		
@	(addr → n)	ersetzt Zellen-Adresse durch ihren Inhalt
C@	(addr → b)	wie @ , jedoch wird auf ein Byte zugegriffen
!	(n addr →)	speichere Second in die Adresse auf dem TOS
C!	(b addr →)	wie ! , jedoch wird ein Byte abgespeichert
+!	(n addr →)	addiere Second zum Inhalt der Adresse auf dem TOS
CMOVE	(from to u →)	verschiebe u Bytes im Adreßraum

FORTH - Details

FILL	(addr u b →)		fülle u Bytes im Speicher ab addr mit b
ERASE	(addr u →)		fülle u Bytes im Speicher ab ddr mit Null
BLANKS	(addr u →)		fülle u Bytes im Speicher ab addr mit $ 20
TOGGLE	(addr b →)		EXOR das Byte in Adresse addr mit Maske b
SP@	(→ addr)		übergibt aktuelle Pos. des Stack-Pointers
Strukturierte Worte			
DO ... LOOP	(n1 n2 →)		Schleife, Index läuft von n2 bis n1−1 mit Incr.=1
DO ... +LOOP	(n1 n2 →)		wie DO...LOOP, jedoch ist das Index-Increment hier (statt 1) nun beliebig (wird als zusätzlicher Parameter an +LOOP übergeben)
I	(→ Index)		Loop-Index → TOS
LEAVE	( )		erzwingt Abbruch der Schleife bei nächster Gelegenheit (Erreichen von LOOP oder +LOOP)
IF ...(wahr)... ENDIF	(f →)		führt Befehle aus, falls das Flag = 1 ist
IF ...(wahr)... ELSE ... (falsch) .. ENDIF	(f →)		dto., jedoch wird bei f=ϕ der FALSE-Teil ausgeführt
BEGIN ... UNTIL	(→ f →)		Schleife mit Abbruch, falls Flag für UNTIL = 1
BEGIN...WHILE...REPEAT	(→ f →)		wie BEGIN...UNTIL, jedoch Abbruch-Test am Anfang des Schleifen-Kernes; REPEAT schließt die Schleife
BEGIN...AGAIN			bedingungslos nach BEGIN Endlos-Schleife
Terminal Eingabe / Ausgabe			
.	(n →)		druckt die Zahl auf dem TOS aus (zerstörend !)
.R	(n Feldweite →)		druckt die Zahl (rechts adjustiert in Feld)
D.	(d →)	( )	druckt doppelt genaue Zahl
D.R	(d Feldweite →)		druckt 32-Bit-Zahl rechts adjustiert in Feld
CR	( )		Ausgabe eines Carriage-Return / Line-Feed
SPACE	( )		Ausgabe eines Space-Character
SPACES	(n →)		Ausgabe von n Space-Characters
."	( )		druckt einen nachfolgenden Text aus, welcher mit " beendet wird
TYPE	(addr u →)		druckt u Zeichen, startend ab Adresse addr
COUNT	(addr → addr+1 u)		wandelt length-Byte-String in die TYPE-Form
?	(addr →)		druckt den Inhalt der Adresse
?TERMINAL	(→ f)		übergibt den Tastatur-Status (1 = „betätigt")
KEY	(→ c)		wartet auf Tastatur-Eingabe und legt den Char. auf den Stack (ASCII)

FORTH - Details

Tafel 4 Die häufigsten FORTH-Befehle (Fortsetzung)

EXPECT	(addr n →)	erwartet n Character (oder bis CR) und bringt sie nach Addr ...
EMIT	(c →)	gibt Character c aus
WORD	(c →)	liest ein Wort (bis zum Delimiter c) im gültigen Eingabe-Buffer
Eingabe – Ausgabe – Formatierung		
NUMBER	(addr → d)	wandelt einen String in addr um in 32-Bit-Zahl
<#	( )	eröffnet Zahlenwandlung für Ausgabe (String)
#	(d → d)	wandelt nächste Stelle der Zahl und fügt dem Ausgabe-String eine Ziffer hinzu (32-Bit-Zahlen !)
#S	(d → 0 0)	wandelt alle signifikanten Stellen um in String
SIGN	(n d → d)	fügt das Vorzeichen von n in den Ziffernstring ein
#>	(d → addr u)	beendet Umwandlung in Ziffern-String (String hat passende Form für TYPE)
HOLD	(c →)	Einfügung eines ASCII Characters in den String
Massenspeicher (Disk)		
LIST	(screen →)	Ausdrucken eines Screen von Disk
LOAD	(screen →)	Laden eines Screen (Compilation oder Interpret.)
BLOCK	(block → addr)	liest Disk-Block nach Adresse addr
B/BUF	(→ n)	Systemkonstante (Blockgröße in Bytes)
BLK	(→ addr)	Systemvariable (aktuelle Block-Nummer)
SCR	(→ addr)	Systemvariable (hält aktuelle Screen-Nummer)
UPDATE	( )	markiert zuletzt benutzten Buffer als ‚updated'
FLUSH	( )	schreibt alle ‚updated' Buffer auf die Disk
EMPTY-BUFFERS	( )	markiert alle Buffer als ‚leer'
Definitions-Worte		
: xyz	( )	Beginn einer Colon-Definition mit Namen xyz
;	( )	Abschluß der Colon-Definition
VARIABLE xxx	(n →)	erzeugt eine Variable xxx , die mit n initialisiert ist (xxx übergibt Adresse bei Aufruf)
CONSTANT yyy	(n →)	erzeugt eine Konstante yyy mit Wert n (bei Aufruf von yyy wird dieser Wert übergeben)
CODE zzz	( )	eröffnet die Definition eines Primitive mit dem Namen zzz (Assembler- bzw. Maschinen-Code)
;CODE	( )	Abschluß einer Colon-Definition, wenn es sich um die Definition eines Definitionswortes handelte, wobei die run-time-Executive in Assembler definiert werden soll (Code hinter ;CODE)

FORTH - Details

<BUILDS .. DOES>	does : (→addr)	wird zur Definition neuer Definitions-Worte benutzt, wobei jedoch im Gegensatz zu ;CODE die run-time-Executive in high-level definiert wird
Vokabulare		
CONTEXT	(→addr)	übergibt die Adresse eines Pointers zum Context-Vokabular (das zuerst abgesucht wird)
CURRENT	(→addr)	übergibt die Adresse eines Pointers zum Current-Vokabular (das z.Z. erweitert wird)
FORTH	( )	Name des Haupt-Vokabulars (setzt CONTEXT)
EDITOR, ASSEMBLER etc.	( )	weitere Vokabular-Namen (setzen CONTEXT)
DEFINITIONS	( )	macht das Current-Vokabular zum Context-Vokabular
VOCABULARY xyz	( )	deklariert ein neues Vokabular mit dem Namen xyz
VLIST	( )	druckt die Namen aller Worte im Context-Vokabular
System-Worte und Diverses		
(	( )	eröffnet Kommentar, der mit) abgeschlossen wird; hinter (muß ein Space kommen
FORGET abc	( )	vergißt alle neuen Definitionen ab (inclusive) abc
ABORT	( )	erzwingt Fehler-Abbruch einer Operation
' xxx	(→addr)	findet die Adresse (PFA) des Wortes xxx im Dictionary (in Definitionen : compiliert die Adresse)
HERE	(→addr)	übergibt die Adresse des nächsten freien Platzes im Dictionary
PAD	(→addr)	übergibt die Startadresse eines Zwischenspeichers, meist 68 Bytes oberhalb von HERE
IN	(→addr)	System-Variable, hält Input-Buffer-Offset für WORD
ALLOT	(n →)	hinterläßt eine ungenutzte Lücke (n Bytes) im Dictionary
,	(n →)	compiliert eine Zahl in das Dictionary (HERE)

Vielen Dank.

Quellen:

- Zech, “Die Programmiersprache FORTH”
- Forth Interest Group

http://www.forth.org

- Deutsche FORTH-Gesellschaft e.V.

http://www.forth-ev.de

- BigFORTH - Homepage

http://www.jwdt.com/~paysan/bigforth.html