Klaus Kohl-Schöpe
Prof.-Hamp-Str. 5
87745 Eppishausen
kks@designin.de

mcFORTH

Ein FORTH für viele Microcontroller

Jahrestagung 2016 der FORTH-Gesellschaft e.V.

Einleitung

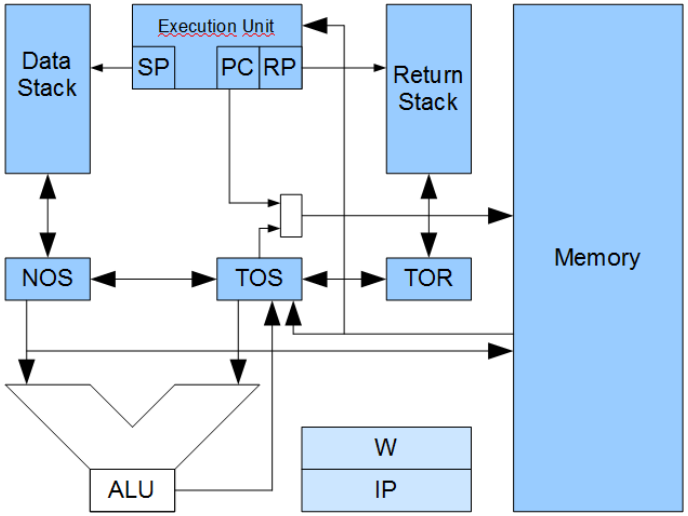
Seit über 35 Jahren befasse ich mich mit Microcontroller und konnte dieses Hobby als Field Application Engineer (FAE) zu meinen Beruf machen. Bei einem der größten Bauteile-Distributoren arbeite ich mit Bausteine fast aller Hersteller wie Atmel, Cypress, Infineon, NXP (Freescale), Renesas, ST, TI und viele mehr.

Mit der FORTH-Gesellschaft e.V. besteht der Kontakt auch schon über 25 Jahren, was zur Entwicklung eigener 16-Bit-FORTH-Versionen für Zilog Super8 und Z80, Harris RTX-2000 und x86-Versionen (PC, V20) führte. Dieses KKFORTH habe ich sowohl beruflich als auch privat bei vielen Projekten genutzt.

Es hat aber 20 Jahre gedauert, um jetzt mit einem neuen FORTH – dem mcFORTH - endlich die Themen wie 32-Bit, Direkt/Indirekt Nesting oder getrennte Speicher und Befehle für RAM und Flash erneut anzugehen. Wichtig war dabei die Nähe zum aktuellen FORTH-Standard und vor allem die leichte Portierbarkeit. Als erste Versionen wurde ein virtueller Prozessor (VP32) unter Windows und der ARM Cortex-M0 auf den Infineon XMC1xxx-Kits (XMC2Go, Bootkits) realisiert.

AUFBAU EINES FORTH-SYSTEM

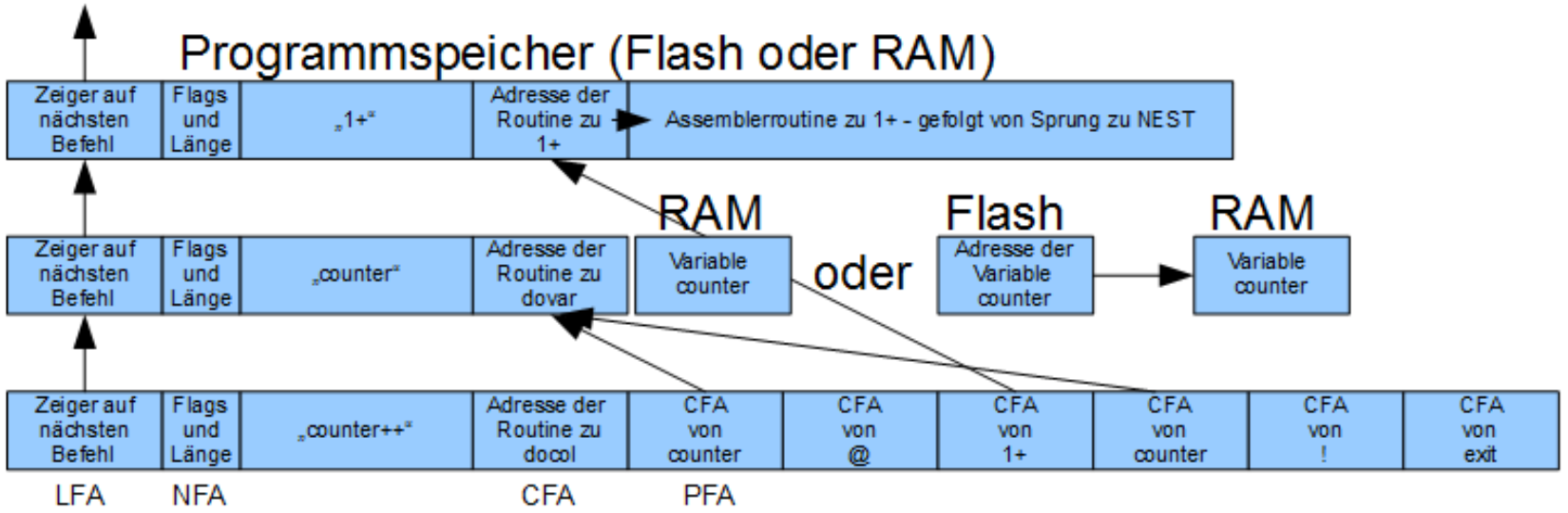
Prozessor und Speicher



- Prozessor mit
  - Programmzeiger (PC)
  - Recheneinheit (ALU)
  - für FORTH geeigneten Befehlssatz
- Datenstack (SP – evtl. TOS/NOS)
- Returnstack (RP – evtl. TOR)
- Programmspeicher
- RAM-Speicher für Variablen/Strings
- Evtl. zusätzliche Register (z.B. IP und W)

Aufbau von FORTH (1 - ITC)

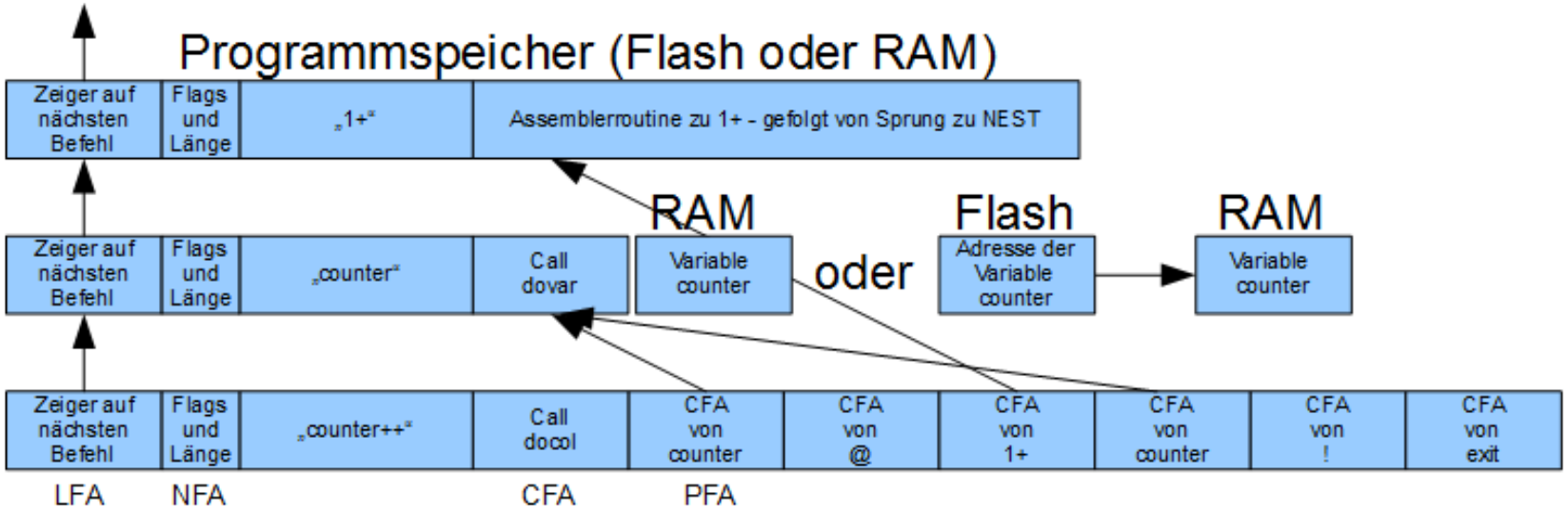
- Beispielprogramm:
  Variable counter
  : counter++ counter @ 1+ counter ! ;
- Aufbau bei **Indirect Threaded Code** (ITC):



Hinweis: Benötigt ein IP und W-Register und FORTH-Routinen wie NEST: IP++ => W, JMP (W++)

Aufbau von FORTH (2 - DCT)

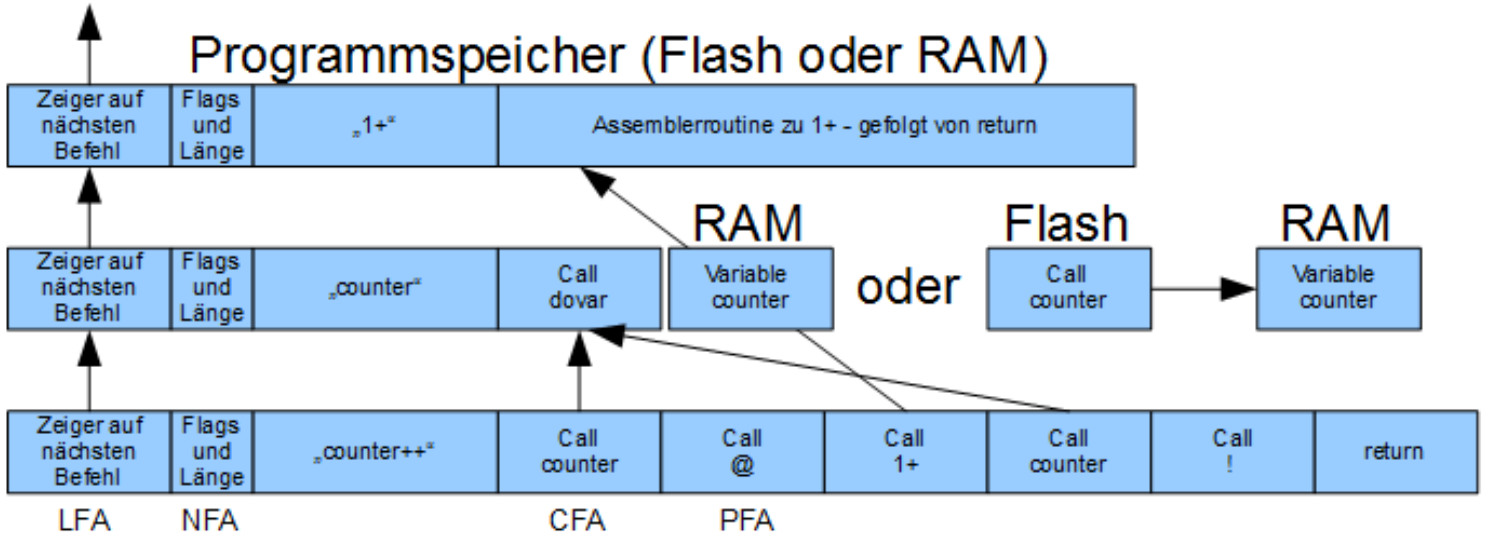
- Beispielprogramm:
  Variable counter
  : counter++ counter @ 1+ counter ! ;
- Aufbau bei **Direct Threaded Code** (DTC):



Hinweis: Benötigt auch ein IP-Register, W nicht notwendig, da PFA auf Returnstack liegt

Aufbau von FORTH (3 - STC)

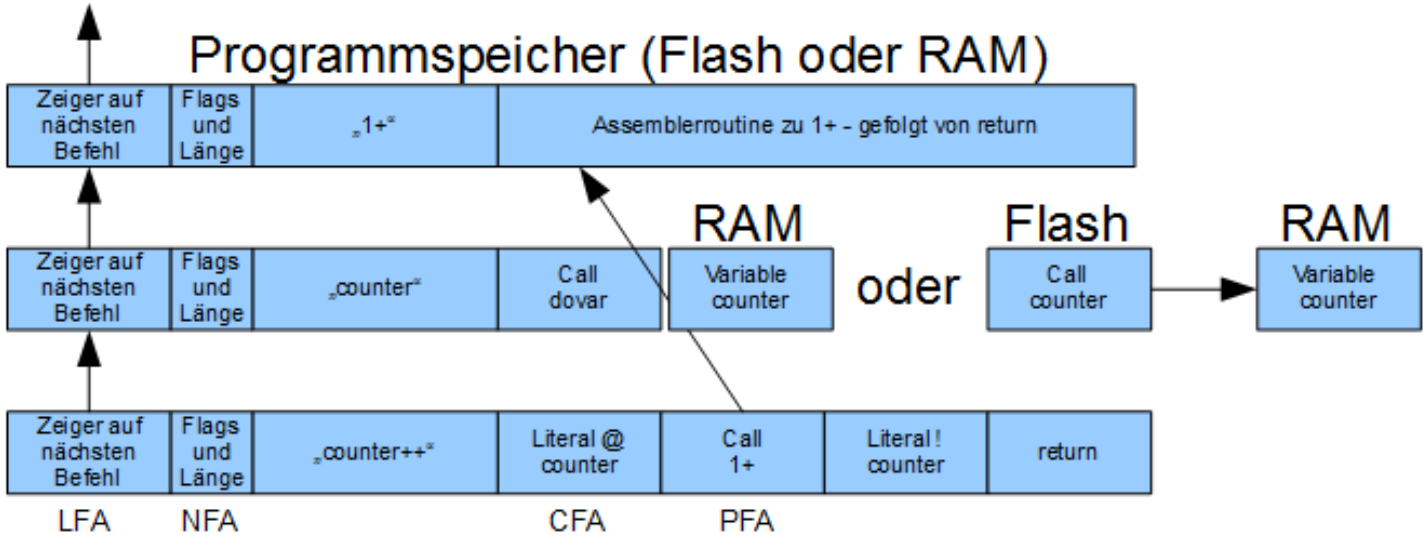
- Beispielprogramm:
  Variable counter
  : counter++ counter @ 1+ counter ! ;
- Aufbau bei **Subroutine Threaded Code** (STC):



Hinweis: Selten genutzt, da auch Inline-Code hier einfach realisierbar ist.

Aufbau von FORTH (4 – STC exp.)

- Beispielprogramm:
  Variable counter
  : counter++ counter @ 1+ counter ! ;
- Aufbau bei **STC mit in-line expansion**:



Hinweis: Bei fast allen optimierenden FORTH-Versionen genutzt
(Weitere Abwandlungen des FORTH-Aufbau wie z.B. Token Threaded möglich)

PROBLEME BEI REALE FORTH-SYSTEMEN

Probleme beim Prozessor

- Datenbreite Register/Speicher <> FORTH
- Harvard statt Neumann Architektur
- Alignment bei Wortzugriffe auf Speicher nötig
- Sprung erreicht nicht den ganzen Speicherbereich

Bei vielen Prozessoren wurde deshalb für FORTH ein Aufbau ähnlich ITC gewählt, bei dem der eigentliche FORTH-Code nur Zeiger beinhaltet. Kernroutinen und einige Befehle wie Literal, Sprünge, Speicherzugriffe oder Logic & Arithmetik werden in Assembler realisiert:

- NEST = Nächsten FORTH-Befehl ausführen: (IP++) => W; JMP (W++)
- DOCOL = Highlevel-Routine starten: IP => (--RP); W => IP; NEST
- EXIT = Highlevel-Routine verlassen: (RP++) => IP; NEST
- LIT = Inline-Literal zum Datenstack: (IP++) => (--SP); NEST
- BRANCH = Unbedingter Sprung: (IP++) + IP => IP; NEST

Der Vorteil ist, dass sowohl der Zeiger in der CFA als auch die Daten und CFA-Zeiger in der PFA meist Wortlänge haben und deshalb das Alignment bei Wortzugriff passt.

Probleme bei Speicher

- Flash als Programmspeicher:
  - Teilweise sehr große und wenige Sektoren, die vor Schreiben gelöscht werden müssen
  - Größe des auf einmal zu schreibenden Bereiches auch groß (z.B. 16-64 Bytes)
  - Manche Speicher können nur einmal geschrieben werden (z.B. wegen ECC)
- Oft zu kleines RAM
  - Bei den meisten Controller nur 1/4 bis 1/8 der Flash-Größe (Ausnahme: neue MCU‘s)
  - Variablen, Arbeitsbereiche (TIB, PAD, HERE, BUFFER) und Stacks belegen große Teile
  - z.B. DOS: Großer externer Speicher benötigte eigene Befehle mit erweiterten Adressen
- Harvard-Architektur:
  - Unterschiedliche Befehle bei Programm- und Variablenspeicher erforderlich
  - RAM nicht als Programmspeicher nutzbar

Der MSP430FR ist so beliebt, weil er das FRAM wie ein RAM verwenden und beliebig oft überschreiben kann. In anderen Systemen wird oft das Programm nur ins RAM kompiliert und für Autostart der ganze belegte RAM-Bereich ins Flash kopiert. Beim Start des Systems wird dieser Teil dann wieder vom Flash ins RAM kopiert.

Probleme bei Schnittstellen

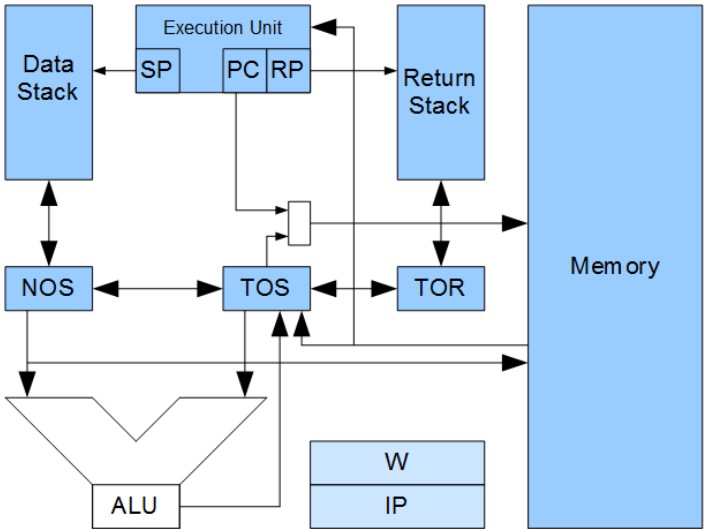
- Nur eine RS232-Schnittstelle, aber kein Filesystem
- Eigene Befehle für I/O erforderlich (z.B. beim PC)
- Schnittstellen benötigen aufwendige Stacks (z.B. USB)

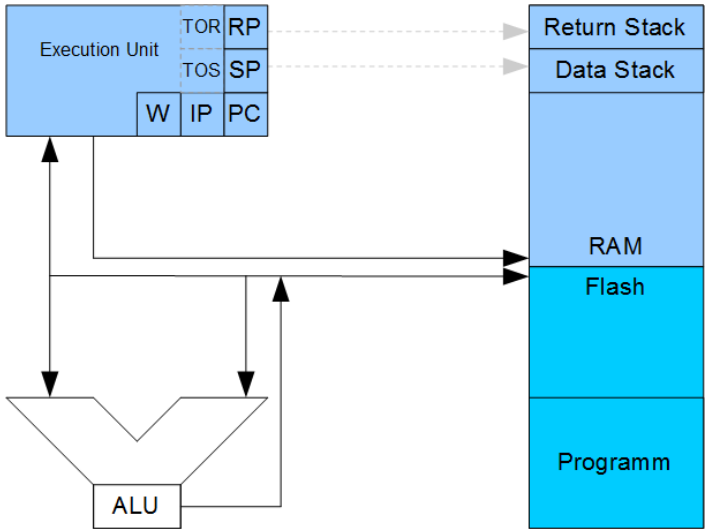
Viele FORTH-Versionen sind in Assembler oder C realisiert, bei dem die Einbindung von Betriebssystemen oder Stacks relativ einfach ist. Aber auch bei direkt in FORTH realisierten Systemen ist die Einbindung von Systemaufrufen gut möglich, wenn die Parameter über den Datenstack übergeben werden.

Gut geeignet für lokale Filesysteme sind (Quad-)SPI, da die Blockgröße mit 512Byte oder 1K optimal auf die übliche Screengröße von 1K angepasst werden können und nur wenig Programmcode zur Ansteuerung benötigen.

FÜR TESTS DES mcFORTH:
DER VIRTUELLER PROZESSOR VP32

Der FORTH-Prozessor VP32





Standard-FORTH-CPU

- Getrennter Speicher für Daten und Returnstack
- TOS/NOS/TOR mit Stacks als Schieberegister
- Befehlscode meist Wortgröße
- FORTH-Befehle meist Single-Cycle-Operationen
- Optional W und IP für Indirekt Nested Code
- Programmspeicher meist nur als RAM ausgelegt

VP32

- Ein Speicher für Programm, RAM und Stacks
- TOS/NOS/TOR oft im RAM (optional im Register)
- Befehlscode 8 Bit evtl. mit 16/32/64 Bit Daten
- Befehle eher als Assembler-Routinen (Macro‘s)
- W und IP verfügbar (W auch als Carry genutzt)
- Speicher meist getrennt als Flash und RAM

VP32-Befehlssatz

Ein-Byte-Befehle mit optionalen 1-Wort-Parameter

Code	FORTH-Befehl	Code	FORTH-Befehl	Code	FORTH-Befehl	Code	FORTH-Befehl
$00	CALL rel	$10	R>	$20	c@	$30	@IO
$01	BRANCH rel	$11	R@	$21	c!	$31	!IO
$02	0BRANCH rel	$12	RDROP	$22	W@	$32	C@X
$03	NBRANCH rel	$13	AND	$23	W!	$33	C!X
$04	LIT value	$14	OR	$24	@	$34	W@X
$05	EXIT	$15	XOR	$25	!	$35	W!X
$06	EXECUTE	$16	+C	$26	C@P	$36	@X
$07	SP@	$17	-C	$27	C!P	$37	!X
$08	SP!	$18	LSHIFT	$28	W@P	$38	FIP@
$09	DROP	$19	RSHIFT	$29	W!P	$39	FIP!
$0A	DUP	$1A	ASHIFT	$2A	@P	$3A	FW@
$0B	OVER	$1B	UM*	$2B	!P	$3B	FW!
$0C	SWAP	$1C	UM/MOD	$2C	C@IO	$3C	NEST
$0D	RP@	$1D	0<	$2D	C!IO	$3D	>CODE
$0E	RP!	$1E	U<	$2E	W@IO	$3E	SYSCALL
$0F	>R	$1F	<	$2F	W!IO	$3F	NOP

Gruppe
Programmlauf
Literal
Datenstack
Returnstack
Arithmetik/Shift/Compare
RAM (Variablen)
Flash (Programm)
IO (Port)
Extended Memory
Indirect Threaded Register
Realer Prozessor

Details zum VP32-Befehlssatz

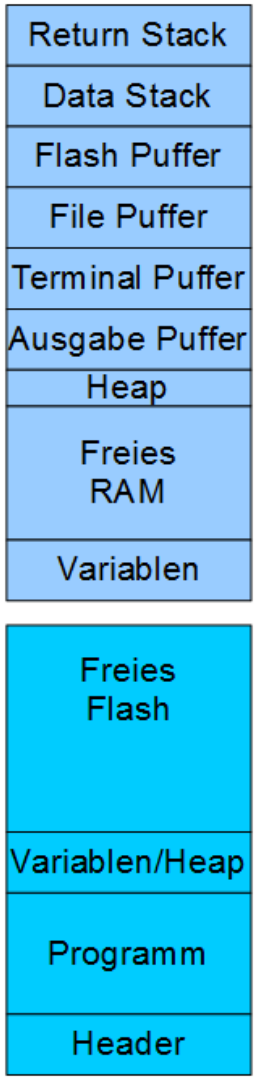
- Speicher wird als Byte (hier tatsächlich 8-Bit) Adressiert
- Vor Ausführung des Befehles wird der PC um 1 oder 5 Bytes erhöht
- Sprungbefehle – erwarten nach dem 8-Byte-Opcode eine 32-Bit-Relativadresse
  - CALL legt Rücksprungadresse auf den Returnstack
  - BRANCH (unbedingt) und 0BRANCH (wenn TOS=0) für Strukturen (IF..ELSE..THEN …)
  - NBRANCH (für FOR…NEXT-Struktur) erniedrigt (mit Sprung) oder entfernt TOR (bei 0)
- Literals sind 32Bit-Inline-Werte, die auf den Datenstack kommen
- Daten- und Returnstackbefehle dürften bekannt sein
- Arithmetik: Carry bei +C, -C, LSHIFT, RSHIFT und ASHIFT kommt in Bit 0 von W (Rest 0)
- Getrennte Speicherbefehle für RAM, Programm, I/O und externer Speicher (8/16/32-Bit)
- Befehle für den Zugriff auf die zusätzlichen Register IP und W mit dem NEST-Befehl
- >CODE zum Aufruf des realen Prozessors, dessen Opcode danach im Speicher steht
- SYSCALL erwarten Befehlsnummer und Parameter auf Datenstack
- NOP = NOOP als Platzhalter z.B. für Warteschleifen
- $40..$FF führen zum Abbruch des Simulators mit Fehlermeldung

Speicher des VP32

- Flash (Standard: 2MByte ab $0000.0000
  - Anfangsadresse und Größe änderbar (**+p**aaaaaaaa ssssssss oder **-p**)
  - Löschbar in Segmente zu 256 Byte, schreibbar in 16Byte-Blöcke (wird nicht getestet)
- RAM (Standard: 1MByte ab $2000.0000
  - Anfangsadresse und Größe änderbar (**+v**aaaaaaaa ssssssss)
  - Darf bei Harvard-Architektur auch gleiche Adresse wie Flash haben
- Sonstiges
  - Alignment für 16-Bit- und 32-Bit-Zugriffe optional zuschaltbar (**+a** – ausschalten mit **-a**)
  - Neumann-Architektur wählbar (**-h –** Harvard einschalten mit **+h**)
  - Returnstack und Datenstack werden im RAM untergebracht
  - TOS, NOS und TOR nicht realisiert (direktes Handling über Stackpointer)
  - NBRANCH verwendet den TOR (kein getrenntes Register für Schleifenzähler)
  - Debugger zuschaltbar (über \ bzw. /Filename – oder über Befehl DEBUG!)

SPEICHERAUFBAU DES mcFORTH

Speicher des mcFORTH

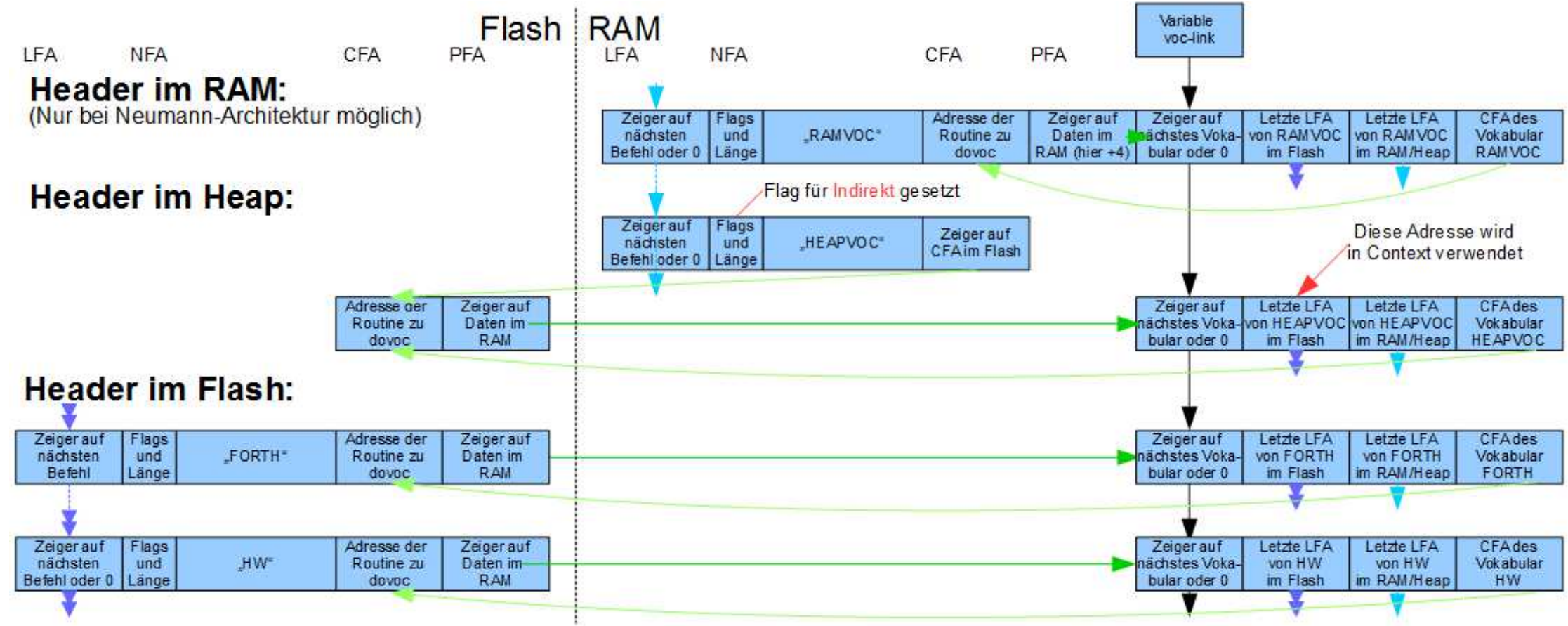


- Flash Puffer: Zwischenablage für die Programmierung des Flash
- Heap: Speichert die Header von headerlosen Befehle
- Variablen: Kann bei Neumann-Architektur auch Befehle beinhalten

- Variablen/Heap: Bei SAVE kopierter Variablen- und Heap-Bereich aus RAM (Ende des Flash-Löschblockes noch Information über das letzte SAVE)
- Programm: Jeweils Header der Befehle mit dem Befehl
- Header: Enthält Sprung zu Init und Informationen über das Grundsystem

Da die Zeiger der headerlosen Befehle immer in den Programmbereich zeigen, funktioniert dies sowohl bei Harvard (Programme nicht im RAM möglich, dafür evtl. überlappende Speicher) als auch bei Neumann (Programme im RAM möglich, aber keine überlappende Speicher).

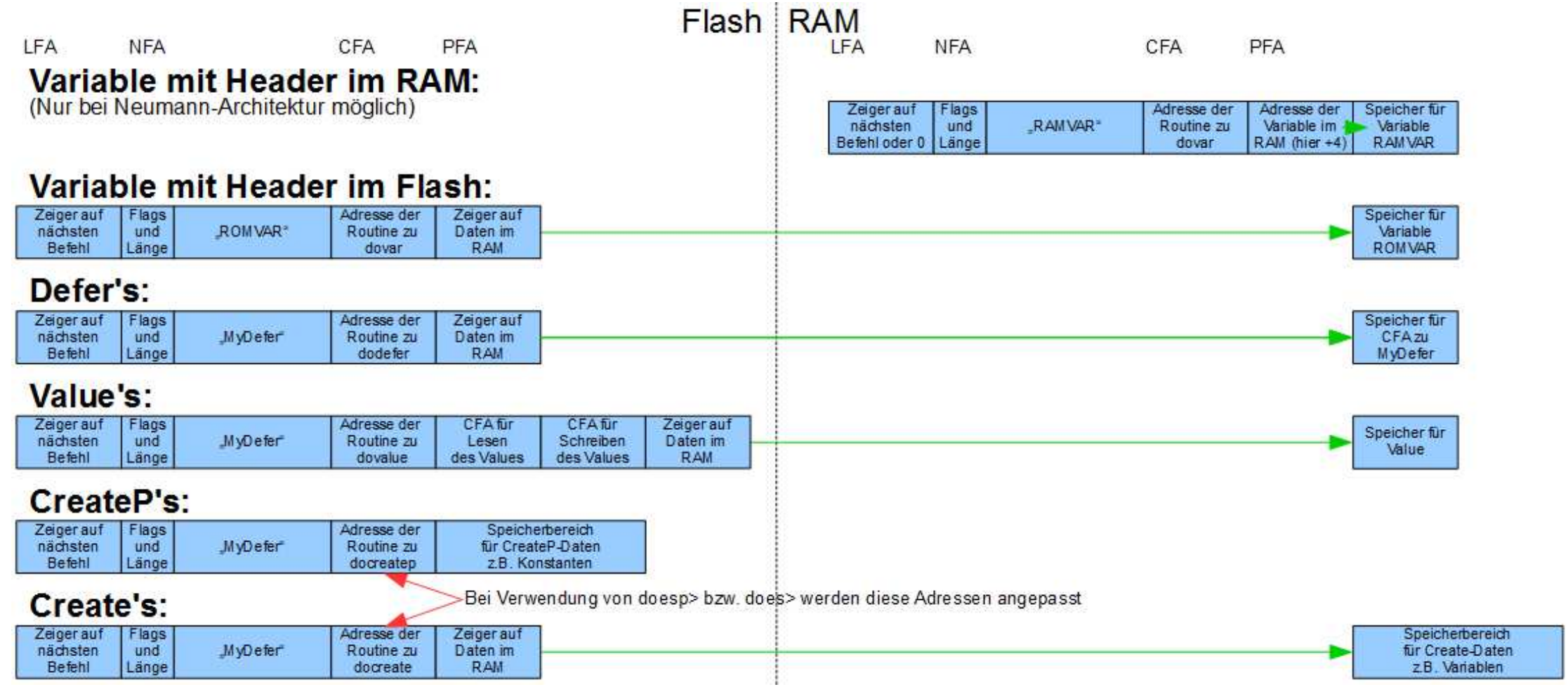
Vokabulare und Befehlsheader



Die Header der Vokabulare sind immer im Flash (Ausnahme Neumann-Architektur), aber die Verkettung der Vokabulare und Zeiger auf letzte LFA getrennt für Flash und sind im RAM. Damit der Name eines Vokabulars gesucht werden kann, gibt es im RAM auch einen Zeiger zur CFA.

Befehle werden zuerst wird RAM gesucht (Default - | |).
Dies kann aber auch mit + | | (einmalig mit | |) auf Flash umgestellt werden.

Aufbau weiterer Befehle



Alle Daten, die im RAM abgelegt werden, haben einen entsprechenden Zeiger im Programmbereich.
Dies gilt auch für Defer‘s, Value‘s und Create-Definitionen.
Create bzw. CreateP kann auch ohne does> bzw. doesp> verwendet werden.

mcFORTH und Header

- Headerlose Befehle sind immer möglich:
  `-Headers` legt alle nachfolgende Befehlsheader im Heap ab – Ende bei `+Headers`
  `|` legt nur den Header des nächsten Befehls auf den Heap
  `CLEARH` löscht den Heap und korrigiert die Zeiger im Header der Befehle
- Bei Neumann-Architektur können Programme auch im RAM sein:
  `>VAR` kompiliert alle nachfolgende Befehle (evtl. mit Header) in den Variablenbereich
  `>PRG` kompiliert alle nachfolgende Befehle wieder in das Flash
  `UNIFIED?` liefert TRUE, wenn Programme auch ins RAM kompiliert werden dürfen
  `>VAR?` liefert TRUE, wenn Programme ins RAM kompiliert werden
  `PRG?` liefert TRUE, wenn Flash verfügbar ist (also keine reine RAM-Version)
- `MARKER` und `(MARKER` (Löschen von Programmteilen)
  `MARKER Name` speichert aktuelle Position (Flash, Variablen, Heap)
  (der Befehl NAME wird dabei an den Anfang des nächsten löschbaren Sektors gelegt)
  `Name` restauriert alten Zustand und korrigiert auch die Vokabular-Verkettungen
  (`MARKER` erwartet Flash- und Variablen-Ende und Heap-Anfang auf dem Stack
  `'MARKER` ist eine DEFER-Routine zur Einbindung eigener Routinen

Flash-Handling des mcFORTH

`flash-se ( addr len – addr2 len2 )`
Liefert Anfangsadresse und Länge eines löschbaren Sektors (enthält addr…addr+len-1)

`flash-sw ( addr len – addr2 len2 )`
Liefert Anfangsadresse und Länge eines schreibbaren Blocks (enthält addr…addr+len-1)

`flash-e ( addr len –)`
Löscht angegebenen Speicherbereich des Flash (flash-se beachten)

`flash-w ( ram flash len – )`
Schreibt Flash mit Daten aus RAM (flash-sw beachten; freie Bereiche möglichst $FF)

`flash-e? ( addr len – flag )`
Prüft, ob Flashbereich gelöscht ist (normalerweise $FF)

Deshalb ist ein Zwischenpuffer der Flash-Daten im RAM notwendig. Dieser Puffer Muss mindestens die Größe eines Schreib-Sektors haben, ist aber typisch auf 512 Bytes bis 1K gelegt, weil die meisten Befehlswörter kleiner sind.

Normalerweise wird jeder FORTH-Befehl getrennt in einen Schreib-Block abgelegt. Deshalb sollte bei sehr großen Schreib-Blöcke Befehlsgruppen in `BEGIN-CREATE` und `END-CREATE` eingeschlossen werden, solange diese sich nicht selbst aufrufen (da solange unsichtbar).

mcFORTH und Befehle im Flash

- Immediate und Restrict ist vor Befehl notwendig
  `(i : Name` oder kurz `i: Name` für Immediate-Befehle
  `(r : Name` oder kurz `r: Name` für Restrict-Befehle (nur kompilierbar)
  `(ir : Name` oder kurz `ir: Name` für Immediate- und Restrict-Befehle
  `(>` setzt Flag für Indirekt und wird implizit bei `‘ Name1 Alias Name2` verwendet
- Ablage in Flash/RAM in eigener Verantwortung
  HERE liefert immer RAM-Adresse und HEREP die Programmadresse (Flash oder RAM)
  Alle Befehle mit …p verändern Programmzeiger (ALLOTP , … , P)
  Bei Variablen und Defer‘s wird immer ein Zeiger auf die RAM-Adresse im Flash abgelegt
- RAM-Zwischenpuffer für Flash-Programmierung genutzt:
  `BEGIN-CREATE` markiert den Anfang einer Befehls für Flash
  `END-CREATE` schließt ein Befehl ab und schreibt Puffer ins Flash
  `CREATE` , `CREATEP` , `HEADER` , `:` und `:noname` verwenden `BEGIN-CREATE`
  `;` beinhaltet `END-CREATE` (damit brauchen Standard-Programme keine Änderungen)
  `DOES>` und `DOESP>` beinhalten `END-CREATE`
  (nur `CREATE` und `CREATEP` ohne `DOES>` bzw. `DOESP>` benötigt `END-CREATE`)
  `(c@p` bis `(!p` prüfen, ob Flash-Puffer genutzt wird (und sind in `c,p` bis `$,p` genutzt)

SCHNITTSTELLEN DES mcFORTH

mcFORTH bei VP32

SYSCALL zum Aufruf der Betriebssystem-Befehle genutzt:
- $00xx für Systembefehle (z.B. $0000 für BYE)
- $01xx für Verwaltung des externen Speicher und Flash
- $02xx für Timer und Zeitabfrage
- $03xx für Terminal (Zeichenabfrage, Löschen des Bildschirm, Farbe)
- $04xx für Filesystem

- Beim Aufruf von SYSCALL sind unter der Befehlsnummer die Parameter auf dem Stack.
- Bei Rückkehr ist die Fehlernummer (0=OK) auf dem Stack – darunter evtl. weitere Daten.
- Mit Ausnahme von $0000 (BYE) ist $xx00 die (De-)Initialisierung des Befehlsbereiches (erwartet TRUE für Initialisierung und FALSE zum Schließen der Befehlsgruppe)

Bei mcFORTH ohne Betriebssystem werden Zeichen direkt über Terminal ausgegeben bzw. empfangen. SYSCALL wird dann bei geeigneten Terminal-Programme an das entsprechende Betriebssystem weitergeleitet.

mcFORTH-Befehlsgruppe $00xx

Command	Assembler	FORTH	Beschreibung
$0000	**(bye**	**BYE**	**\ Programm beenden (–)**
$0001	**(debug**	**DEBUG**	**\ Setzt Debug-Funktion (addr flag – 0)**
$0002	**(getpar**	**PAR@**	**\ Befehlszeile holen (– addr 0)**
$0003	**(irqoff**	**IRQOFF**	**\ Interrupt aus**
$0004	**(irqon**	**IRQON**	**\ Interrupt ein**
$0005	**(irqset**	**IRQSET**	**\ Interruptvektor ändern/abfragen**
$0006	**(v**	**V**	**\ Editor mit Fehlerposition aufrufen**

Bei Betriebssystem wie Windows oder Linux braucht man die Rückkehr zum System (mit BYE).
Für den Simulator ist auch eine Funktion definiert, die Singlestep oder Breakpoints festlegt.
Meist werden Parameter beim Aufruf von FORTH übergeben, die mit PAR@ abgefragt werden.
Interrupt-Vektoren und Ein-/Ausschalten des Interrupts erfolgt hier auch über Betriebssystem.
Falls kein Editor im FORTH integriert ist, könnte man dies auch über externe Funktionen ersetzen.

mcFORTH-Befehlsgruppe $01xx

Command	Assembler	FORTH	Beschreibung
$0100	(xinit	xinit	\ Externer Speicher initialisieren
$0101	(xalloc	xalloc	\ Reserviert Speicher
$0102	(xrelloc	xrelloc	\ Verändert Speichergröße
$0103	(xfree	xfree	\ Gibt Speicher frei
$0104	(>x	>x	\ Ermittelt Pointer aus RAM-Adresse
$0105	(p>x	p>x	\ Ermittelt Pointer aus ROM-Adresse
$0106	(s>x	s>x	\ Ermittelt Pointer aus Stack-Adresse (oder 0)
$0107	(r>x	r>x	\ Ermittelt Pointer aus Returnstack-Adresse (oder 0)
$0108	(flash-se	flash-se	\ Liefert Sektoradresse und Größe beim Löschen
$0109	(flash-sw	flash-sw	\ Liefert Sektoradresse und Größe beim Schreiben
$010A	(flash-e	flash-e	\ Löschen eines Flash-Bereiches
$010B	(flash-w	flash-w	Schreiben eines Flash-Bereiches

Handling des externen Speichers und Flash ist ebenfalls vorgesehen.

mcFORTH-Befehlsgruppe $02xx

Command	Assembler	FORTH	Beschreibung
$0200	**(cinit**	**cinit**	**\ Timer/Counter/RTC initialisieren**
$0201	**(systick**	**SYSTICK@**	**\ Liefert Systemticker und Frequenz in Hz (tick freq)**
$0202	**(time@**	**TIME@**	**\ Liefert aktuelle Zeit (ms s m h)**
$0203	**(time!**	**TIME!**	**\ Setzt aktuelle Zeit (ms s m h) - nicht im Simulator**
$0204	**(date@**	**DATE@**	**\ Liefert aktuelles Datum (wd t m y)**
$0205	**(date!**	**DATE!**	**\ Setzt aktuelles Datum (wd t m y) - nicht im Simulator**
$0206	**(waitms**	**WAITMS**	**\ wartet n Millisekunden**

Ein Systemtimer ist oft in Hardware realisiert (z.B. bei alle Cortex-M) und liefert meist einen Wert, der mit einer (hier angegebenen) Frequenz hochgezählt wird.
Dagegen wird Uhrzeit und Datum eher über ein Hardware-Uhr (RTC) realisiert.

mcFORTH-Befehlsgruppe $03xx

Command	Assembler	FORTH	Beschreibung
$0300	(tinit	TINIT	\ Initialisieren des Terminals
$0301	(key	KEY	\ Auf Taste warten
$0302	(emit	EMIT	\ Zeichen ausgeben
$0303	(key?	KEY?	\ Testen, ob Taste gedrückt (möglichst nicht nutzen)
$0304	(emit?	EMIT?	\ Testen, ob Ausgabe möglich (möglichst nicht nutzen)
$0305	(emit?	EMIT?	\ Testen, ob Ausgabe möglich (möglichst nicht nutzen)
$0306	(page	PAGE	\ Bildschirm löschen
$0307	(atxymax	ATXYMAX	\ Bildschirmgröße abfragen
$0308	(atxy?	ATXY?	\ Bildschirmposition setzen
$0309	(atxy	ATXY	\ Bildschirmposition abfragen
$030A	(color@	COLOR@	\ Farbe abfragen (bzw. Attribut)
$030B	(color!	COLOR!	\ Farbe setzen (bzw. Attribut)
$030C	(at@	AT@	\ Zeichen und Farbe abfragen (aktuelle Position)
$030D	(at!	AT!	\ Zeichen und Farbe setzen (aktuelle Position)

Grundroutinen für Ein-/Ausgabe mit Erweiterungen zum Löschen, Positionieren und Einfärben.

mcFORTH-Befehlsgruppe $04xx

Command	Assembler	FORTH	Beschreibung
$0400	(finit	FINIT	\ Initialisieren das Fileinterface
$0401	(fcreate	FCREATE	\ File anlegen (csa/0 -- id 0 | error)
$0402	(frename	FRENAME	\ Fileumbenennen (csa1/0 csa2/0 -- error)
$0403	(fdelete	FDELETE	\ File löschen (csa/0 -- id 0 | error)
$0404	(fopen	FOPEN	\ File öffnen (csa/0 flag -- id 0 | error)
$0405	(fclose	FCLOSE	\ File schließen (id -- error)
$0406	(fsize@	FSIZE@	\ Filegröße abfragen (id -- d 0 | error)
$0407	(fsize!	FSIZE!	\ Filegröße setzen (d id -- error)
$0408	(fpos@	FPOS@	\ Fileposition abfragen (id -- d 0 | error)
$0409	(fpos!	FPOS!	\ Fileposition setzen (d dir id -- 0 | error)
$040A	(fread	FREAD	\ File auslesen (xptr len id --len2 0 | error)
$040B	(fwrite	FWRITE	\ File schreiben (xptr len id --0 | error)

…

Abweichend vom ANS-Standard werden hier immer doppeltgenaue Positionen/Länge verwendet. Deshalb wurden die Befehlsnamen nicht an den Standard angepasst.

STATUS DES mcFORTH

Noch in Arbeit

- USER-Bereich und Systemkonstanten:
  - Multitasking für mcFORTH und Interrupts im Simulator
  - Parameterfeld im Header zur Modifikation des Flash- und RAM-Bereiches
- Minimalversion:
  - Reduktion auf unbedingt notwendige Befehle (evtl. ohne Harvard)
  - Vereinfachtes Download- und Save-Interface
- Weitere Portierungen für Tests
  - 16-Bit-DOS-Version mit getrennten Programm und RAM-Bereich (je 64K)
  - 8051-Version für Test der Harvard-Funktionen
  - Anpassung des Simulators an Linux und Microcontroller
  - Weitere ARM-Versionen (evtl. in Co-Operation mit Matthias Koch (mecrisp)
  - Weitere Prozessoren (MSP430, RL78, RX, PowerPC)

Aktuell verfügbares mcFORTH

- Aktuell zwei VP32-Versionen verfügbar:
  - SCT exp. mit Inlines bei 2M Flash und 1M RAM (Neumann ohne Align)
  - ICT bei 2M Flash und 1M RAM (Harvard mit Align – Test für Cortex-M0)
- Portabilität
  - Target-Compiler auch in mcFORTH realisiert (nicht veröffentlicht)
  - VP32-Bytecode-Interpreter für C und Assembler verfügbar (öffentlich)
- Leicht erlernbar und nutzbar
  - annähernd ANS-Standard mit nur wenigen Besonderheiten
  - Standard-Terminal genügt für Tests und Programmierung
  - mcFORTH-Terminal erlaubt vollen Filezugriff beim PC

Auch eine Cortex-M0-Version für einige Infineon XMC1xxx-Kits ist verfügbar.

Lizenz und Kosten

Aus langjähriger Erfahrung wird der Targetcompiler und dessen Sourcen nicht herausgegeben. Die Sourcen des mcFORTH werden vermutlich nur in Teilen veröffentlicht. Der Support erfolgt meist über EMail und Internet.

- Für private Anwendungen und Tests
  - bei <10 Systeme ist das mcFORTH frei verwendbar
  - bei offengelegte Sourcen der Applikation ist das mcFORTH ebenfalls frei verwendbar
- Für kommerzielle Zwecke
  - Lizenz möglich, bei der Sourcen nicht offengelegt werden müssen (je nach Stückzahlen bei 0.50€/CPU bis herunter zu 0.20€/CPU)
  - Targetcompiler und sogar Sourcen des Targetcompiler verfügbar
  - Vor-Ort-Support möglich

Happy mcFORTH‘eln

Vielen Dank für Ihr Interesse

Bei Fragen, Wünsche oder Hinweise auf Fehler bitte die EMail-Adresse auf der ersten Seite verwenden. Unterstütze auch gerne die Implementierungen auf weitere Prozessoren, Boards und anderen FORTH-Versionen.