

Assembler

Thomas Beierlein:

Da ich in letzter Zeit eine Reihe von Fragen zur Benutzung des Z80-Assemblers in FORTH83 erhalten habe, möchte ich hier einige Hinweise zu seiner Benutzung geben. Da ich z.Z. unter chronischem Zeitmangel stehe, wird es nur eine recht kurze Barstellung. Zur Sache:

Maschinencoderoutinen werden im FORTH83 im Unterschied zu FIG mit folgenden Worten definiert bzw. abgeschlossen.

CODE <name> END-CODE

Diese beiden Worte sind auch ohne geladenen Assembler verfügbar und können wie früher kurze Coderoutinen als Hexcode direkt ins Wörterbuch „hineinkommaen“. Nach wie vor müssen Coderoutinen mit einem Sprung nach NEXT abgeschlossen werden. Die Adresse von NEXT wird durch die Konstante >NEXT im System bereitgehalten.

Eine zweite Art von Code sind Unterprogramme, die mit der Sequenz

LABEL <name> RET, END-CODE

definiert werden. Ein Beispiel zur Anwendung folgt am Schluß. Bei der Benutzung von Code-Definitionen ist zu beachten, daß die Register BC und IY vom System verwendet werden. Sie sind also bei Bedarf zu sichern.

Nach dem Laden des Assemblers stehen nach Aufruf von CODE <name> alle Befehle des Z80 in ihrer ursprünglichen Mnemonik zur Verfügung (am besten mit ASSEMBLER WORDS einmal anschauen). Bei der Anwendung sind folgende Syntaxregeln zu beachten:

- alle Mnemoniks haben ein , (Komma) am Namensende,
- die Notation der Parameter erfolgt streng nach umgekehrt polnischer Notation. Die Reihenfolge ist also <Quelle Ziel Operand>.

Folgende Regeln gelten für die Kennzeichnung der Parameter:

- Register werden mit ihrem Namen aufgerufen. Folgende gibt es:
A B C D E H L (HL) (IY) I R AF BC DE HL IX IY
- alle Adressen sind durch ein nachfolgendes #) als solche zu kennzeichnen,
- alle direkt in Register zu ladenden Zahlenwerte sind durch # zu kennzeichnen,
- bei Verwendung der Indexregister in der Form (IX+displ) ist statt dessen displ (IX) zu schreiben. Eine Angabe des Displacement ist auch bei displ=0 notwendig.
- Bitoperationen sind wie folgt zu codieren
RES 4,C wird z.B. zu 4 C RES,
- bei Relativsprüngern ist die Distanz mit # als Zahlenwert zu markieren.

Alle Sprünge und Call's, unbedingt und auch bedingt, werden compiliert. Zu beachten ist, daß statt JMP (HL) die Sequenz HL JMP, zu verwenden ist.

Bei der Verwendung der Strukturoperatoren IF, ELSE THEN, BEGIN, UNTIL, WHILE, und REPEAT, werden nur absolute Sprünge compiliert. Die folgenden Entscheidungsbedingungen gibt es dabei:

0= (Zero)
CY (Carry)
PE (Parity)
0< (Minus)

Ein NOT nach der Bedingung negiert seine Bedeutung (0= NOT IF, entspricht so z.B. JPNZ).

Der Assembler unterstützt des weiteren den Abschluß einer Coderoutine mit den Macros NEXT, HPUSH, und DPUSH. Diese compilieren einen Sprung zur Nextroutine bzw. legen vorher noch das Register DE oder DE und HL auf den Stack.

Soweit dazu. Nun noch einige Beispiel für die möglichen Parameterformen.

konventionelle Notation	FORTH-Assembler
EX (SP),HL	HL (SP) EX,
RST 28H	HEX 28 #) RST,
JPC 4236H	4236 #) JPC,
JMP (IX)	IX JMP,
CANZ 1A7H	1A7 #) CANZ,
LD A,17	17 # A LD,
LD (BC),A	A (BC) LD,
LD A,(IX)	0 IX A LD,
SBC HL,DE	DE HL SBC,
OR D	D OR,
AND 0E7H	HEX 0E7 # AND,
IN 23	23 # IN,
OUT L	L OUT,
JRNZ #-5	-5 # JRNZ,
RES 0,(IY+4)	0 4 IY # RES,

Hier waren nun hoffentlich alle Varianten dabei.

Zum Schluss noch ein Beispiel, welches auch die Verwendung von Labels und die strukturierte Programmierung verdeutlichen soll. Folgendes Problem steht:

An einem I/O-Port sollen zwei Steuerbits P0 und P1 überwacht werden. Eine definierte Zeit nach dem Übergang von P0 auf High soll P1 abgetastet und in einem 8-Bit Puffer von rechts eingeschoben werden.

Die Lösung würde bei mir etwa so aussehen:

```
HEX
23 CONSTANT PORT          \ die Portadresse
0  CONSTANT P0             \ die Bitnummer der Steuer-
1  CONSTANT P1             \ leitungen im Port
45 CONSTANT ZK             \ Zeitkonstante der Verzög.
VARIABLE PUFFER           \ der 8-Bit Puffer
```

```

LABEL WAIT REVEAL          \ realisiert Verzögerung
    ZK # HL LD,           \ indem HL bis auf 0
                           \ heruntergezählt wird
    BEGIN, H A LD, L OR, 0= NOT WHILE, HL DEC, REPEAT,
    RET,

CODE ABTASTUNG            \ macht die Arbeit
    BEGIN, PORT # IN, P0 A BIT, 0= NOT UNTIL,
                           \ warte bis P0 High wird
    WAIT #) CALL,          \ rufe Verzögerung auf
    PORT # IN, P1 A BIT,    \ teste P1 und wandle in
    0= IF, A OR, ELSE, SCF, THEN, \ Carry-Flag um
    PUFFER #) A LD,        \ lade A mit dem Inhalt von
    RLA,                  \ Puffer und schiebe Cy ein
    A PUFFER #) LD,        \ lege den Wert wieder zurück
    NEXT END-CODE         \ weiter mit FORTH

```

So, viel Spaß und nicht verzagen, wenn es nicht gleich klappt.

Thomas Beierlein, 1990

Beispiel

Ich habe obiges Beispiel getestet. Dabei ist mir aufgefallen, dass hinter LABEL noch REVEAL fehlte, sonst ist das Label gar nicht sichtbar. (Oder aber in LABEL muss das HIDE weggepatcht werden).

Folgender Code wird durch den Assembler erzeugt. Man beachte die Umsetzung der Schleifenkonstrukte.

```

;
4685 4673                      VARIABLE PUFFER
4687 86
4688 'PUFFER' +#80
468E 0497                      DOVAR
4690 0000
;
4692 3897                      LABEL WAIT
4693 84
4695 'WAIT' +#80
4699 0497                      DOVAR
469B 21 45 00      LD   HL,0045H      ZK # HL LD,
469E     M1                   BEGIN,
469E 7C       LD   A,H           H A LD,
469F B5       OR   L             L OR,
46A0 CA A7 46    JP   Z,46A7H      0= NOT WHILE,
46A3 2B       DEC  HL           HL DEC,
46A4 C3 9E 46    JP   469EH      REPEAT,
46A7 C9       M2   RET          RET,
;

```

46A8	3A8A		CODE ABTASTUNG
46AA	89		
46AB	'ABTASTUNG' +#80		
46B4	\$+2	CODE	
46B6	M3		BEGIN,
46B6	DB 23	IN A,(23H)	PORT # IN,
46B8	CB 47	BIT 0,A	P0 A BIT,
46BA	CA B6 46	JP Z,46B6H	0= NOT UNTIL,
46BD	CD 9B 46	CALL 469BH	WAIT #) CALL,
46C0	DB 23	IN A,(23H)	PORT # IN,
46C2	CB 4F	BIT 1,A	P1 A BIT,
46C4	C2 CB 46	JP NZ,46CBH	0= IF,
46C7	B7	OR A	A OR,
46C8	C3 CC 46	JP 46CCH	ELSE,
46CB	37	M4 SCF	SCF,
46CC		M5	THEN,
46CC	3A 90 46	LD A,(4690H)	PUFFER #) A LD,
46CF	17	RLA	RLA,
46D0	32 90 46	LD (4690H),A	A PUFFER #) LD,
46D3	C3 22 04	JP 0422H	NEXT
			END-CODE

Strukturooperatoren

Die Strukturooperatoren sind forth-gewohnt zu verwenden:

flag 0=, CY, PE oder 0<

IF...ELSE...THEN

```
flag [NOT] IF,           ; JP /flag M1
...
[ELSE,]
...
THEN,                  ; M2:
```

BEGIN ...flag UNTIL

```
BEGIN,                 ; M1:
...
flag [NOT] UNTIL,      ; JP /flag M1
```

BEGIN...AGAIN

```
BEGIN,                 ; M1:
...
AGAIN,                ; JP M1
```

BEGIN...flag WHILE...REPEAT

```
BEGIN,          ; M1:  
...  
flag [NOT] WHILE,    ; JP /flag M2  
...  
REPEAT,         ; JP M1  
                ; M2:
```

Literatur

Der Assembler ist auch in „Vack, Gert Ulrich: Programmieren mit Forth. VEB Verlag Technik Berlin, 1990“ Seite 263-271 beschrieben.

From:

<https://hc-ddr.hucki.net/wiki/> - **Homecomputer DDR**



Permanent link:

<https://hc-ddr.hucki.net/wiki/doku.php/forth/fgforth/assembler>

Last update: **2013/05/01 13:26**