Logic Disassembler

Wer hat nicht schon nach einem guten Einstieg in das Betriebssystem oder in ein selbstgeschriebenes Assembler-Programm gesucht, um dem Verlauf der Programmlogik folgen zu können?

Mit diesem ganz in Basic geschriebenen Programm besteht nun die Möglichkeit, bei einer beliebigen Startadresse einzusteigen und von dort aus das Programm zu verfolgen.

Auch mit einem kompletten Disassembler-Listing war dies bisher eine äußerst aufwendige und zeitraubende Angelegenheit. Bei jedem Sprung-Befehl (sei es BRANCH, JSR oder JMP) ging das große Blättern los. Das Verfolgen von mehrstufigen Unterroutinen mit unterschiedlichen Ausstiegsmöglichkeiten ist schon ganz unmöglich.

Der Logic-Disassembler hingegen wartet bei jedem Sprungbefehl auf die Eingabe von Y, N oder X. Y steht für »yes«, das heißt dem Sprungbefehl folgen, N steht für »no«, also zum Beispiel eine bekannte Unterroutine einfach zu übergehen, damit das Listing nicht zu unübersichtlich wird. X schließlich steht für »exit«, wodurch der Disassembler bei einer neuen Adresse gestartet werden kann.

Das Listing der Basic-Routine »NEW« (Bild 1) zeigt als Beispiel, wie ein solcher Disassembler-Lauf vor sich geht.

Die Einsprungadresse liegt bei $C642.

In Zeile 1 wird der BNE-Befehl mit der Eingabe von »N« übergangen. Der JSR-Befehl in Zeile 14 wird mit »Y« verfolgt, und es erfolgt eine Verzweigung in die Routine $C68E. Mit der Anzeige von STACK 0 wird über die Tiefe der Unterprogramm-Verschachtelung informiert. Bei Erreichen von RTS in Zeile 22 wird in das Hauptprogramm (STACK 0) zurückgesprungen und dieses ab der Adresse $C65C fortgeführt.

Der darauffolgende BRANCH-Befehl in Zeile 24 wird wieder mit der Eingabe von »N« übergangen. In Zeile 25 folgt dann ein RTS, und da der STACK auf 0 steht (man befindet sich also im Hauptprogramm), wird der Disassembler-Vorgang beendet

```
<LOGIC - DISASSEMBLER>

LINE# LOC    CODE          STATEMENT

      0000                 *=$C642

0001  C642  D0 FD          BNE $C641
0002  C644  A9 00          LDA #$00
0003  C646  A8             TAY
0004  C647  91 2B          STA ($2B),Y
0005  C649  C8             INY
0006  C64A  91 2B          STA ($2B),Y
0007  C64C  A5 2B          LDA $2B
0008  C64E  18             CLC
0009  C64F  69 02          ADC #$02
0010  C651  85 2D          STA $2D
0011  C653  A5 2C          LDA $2C
0012  C655  69 00          ADC #$00
0013  C657  85 2E          STA $2E
0014  C659  20 8E C6       JSR $C68E              STACK 1

0015  C68E  18             CLC
0016  C68F  A5 2B          LDA $2B
0017  C691  69 FF          ADC #$FF
0018  C693  85 7A          STA $7A
0019  C695  A5 2C          LDA $2C
0020  C697  69 FF          ADC #$FF
0021  C699  85 7B          STA $7B
0022  C69B  60             RTS                    STACK 0

0023  C65C  A9 00          LDA #$00
0024  C65E  D0 2D          BNE $C68D

0025  C68D  60             RTS

START-ADR. = $C642
END-ADR.   = $C68D

< DISASSEMBLER - END >
```

Bild 1. Logische Disassemblierung der Basic-Routine »NEW«

Das Listing zum »Logic Disassembler«

```
100 POKE36879,233:PRINT"■■ *** DISASSEMBLER ***"
110 PRINT"■■ LOGIC - DISASSEMBLER"
120 PRINT"■   VC 20 - VERSION"
130 PRINT"■          BY          "
140 PRINT"■■   FRED HAMMER       "
150 PRINT"■   OBERE BERGSTR. 12"
160 PRINT"■   5419 LEUTEROD     "
170 PRINT"■   TEL. 02602/60372 "
180 PRINT"■■■   PRESS  ANY  KEY    ";
190 GETA$:IFA$=""THEN190
200 CLR:POKE36879,25:DIMSS(100):SS%=0:DIMAC$(255)
210 PRINT"■K DISASSEMBLER-START >■"
220 FORI=0TO255:READAC$(I):NEXT
230 POKE36879,25:PRINT"■K DISASSEMBLER-START >■"
240 H$="0123456789ABCDEF"
250 GOTO520
260 REM HEX-UMRECHNUNG
270 DE=0:FORZ0=1TO4:N0=0:FORZ1=1TO16
280 IFMID$(X$,Z0,1)=MID$(H$,Z1,1)THENN0=Z1-1:Z1=16
290 NEXT:DE=DE+16↑(4-Z0)*N0:NEXT:RETURN
300 REM DEZ-UMRECHNUNG
310 X$="":FORZ0=4TO1STEP-1:Z1=INT(DE/16↑(Z0-1))
320 IFZ1=0THENX$=X$+"0":GOTO350
330 IFZ1<10THENX$=X$+RIGHT$(STR$(Z1),LEN(STR$(Z1))-1):GOTO350
340 X$=X$+CHR$(Z1+55)
350 DE=DE-Z1*16↑(Z0-1):NEXT:RETURN
360 REM DER 6502-BEFEHLSSATZ
370 DATA BRK,ORA (X,,,,ORA $,ASL $,,PHP,ORA #,ASL,,,ORA,ASL?,,BPL,ORA (Y,,,
380 DATA ORA $X,ASL $X,,CLC,ORA Y,,,,ORA X,ASL X,,JSR?,AND (X,,,BIT $,AND $
390 DATA ROL $,,PLP,AND #,ROL,,BIT?,AND?,ROL?,,BMI,AND (Y,,,,AND $X,ROL $X,
400 DATA SEC,AND Y,,,,AND X,ROL X,,RTI,EOR (X,,,,EOR $,LSR $,,PHA,EOR #
410 DATA LSR,,JMP?,EOR?,LSR?,,BVC,EOR (Y,,,,EOR $X,LSR $X,,CLI,EOR Y,,,,EOR X
420 DATA LSR X,,RTS,ADC (X,,,,ADC $,ROR $,,PLA,ADC #,ROR,,JMP (,ADC?,ROR?,
430 DATA BVS,ADC (Y,,,,ADC $X,ROR $X,,SEI,ADC Y,,,,ADC X,ROR X,,,STA (X,,
440 DATA STY $,STA $,STX $,,DEY,,TXA,,STY?,STA?,STX?,,BCC,STA (Y,,,STY $X
450 DATA STA $X,STX $Y,,TYA,STA Y,TXS,,,STA X,,,LDY #,LDA (X,LDX #,,LDY $
460 DATA LDA $,LDX $,,TAY,LDA #,TAX,,LDY?,LDA?,LDX?,,BCS,LDA (Y,,, LDY $X
470 DATA LDA $X,LDX $Y,,CLV,LDA Y,TXS,,LDY X,LDA X,LDX Y,,CPY #,CMP (X,,
480 DATA CPY $,CMP $,DEC $,,INY,CMP #,DEX,,CPY?,CMP?,DEC?,,BNE,CMP (Y,,,
490 DATA CMP $X,DEC $X,,CLD,CMP Y,,,,CMP X,DEC X,,CPX #,SBC (X,,,CPX $
500 DATA SBC $,INC $,,INX,SBC #,NOP,,CPX?,SBC?,INC?,,BEQ,SBC (Y,,,,SBC $X
510 DATA INC $X,,SED,SBC Y,,,,SBC X,INC X,
520 FE$=AD$:INPUT"■START (HEX)";A$
530 IFA$="END"THENCLR:POKE36879,29:END
540 IFLEN(A$)<>5ORLEFT$(A$,1)<>"$"THEN520
550 X$=RIGHT$(A$,4):GOSUB260:A=DE
560 PRINT"■PRINTER ? Y/N":POKE198,0:WAIT198,1:GETAN$:IFAN$<>"Y"THEN600
570 INPUT"COMMENT ";FF$
580 OPEN4,4:CMD4
590 PRINTCHR$(14);FF$;CHR$(15):PRINT
600 PRINT"■KLOGIC - DISASSEMBLER>":PRINT:PRINT"LINE# LOC    CODE            STATE
MENT
610 PRINT:PRINT"      0000                *="A$:PRINT:N=0
620 AD=A-1
630 AD=AD+1:N=N+1:DE=AD:GOSUB300
640 PRINTRIGHT$("0000"+RIGHT$(STR$(N),LEN(STR$(N))-1),4)"  "X$"  ";
650 P=PEEK(AD):DE=P:EA=P:GOSUB300:S$=RIGHT$(X$,2):RESTORE
660 MN$=AC$(P):IFLEN(MN$)=3THEN880
670 IFMN$=""THEN940
680 IFLEN(MN$)=4THEN830
690 IFRIGHT$(MN$,1)="#"THENMN$=MN$+"$":GOTO770
```

```
<LOGIC - DISASSEMBLER>

LINE# LOC    CODE          STATEMENT

      0000                 *=$FFE1

0001  FFE1  6C 03 28       JMP ($0328)

0002  F770  A5 91          LDA $91
0003  F772  C9 FE          CMP #$FE
0004  F774  D0 07          BNE $F77D
0005  F776  08             PHP
0006  F777  20 CC FF       JSR $FFCC             STACK 1

0007  FFCC  6C 03 22       JMP ($0322)

0008  F3F3  A2 03          LDX #$03
0009  F3F5  E4 9A          CPX $9A
0010  F3F7  B0 03          BCS $F3FC
0011  F3F9  20 04 EF       JSR $EF04             STACK 2
0012  F3FC  E4 99          CPX $99
0013  F3FE  B0 03          BCS $F403
0014  F400  20 F6 EE       JSR $EEF6             STACK 2
0015  F403  86 9A          STX $9A
0016  F405  A9 00          LDA #$00
0017  F407  85 99          STA $99
0018  F409  60             RTS                   STACK 0

0019  F77A  85 C6          STA $C6
0020  F77C  28             PLP
0021  F77D  60             RTS

START-ADR. = $FFE1
END-ADR.   = $F77D

< DISASSEMBLER - END >
```

Bild 2. Die Betriebssystemroutine zur Abfrage der STOP-Taste disassembliert.

```
A$   -  Tabelle fuer Sprungadressen
AC%  -  Tabelle fuer 6502-Mnemonics
SS%  -  Tabelle fuer Stack

SS%  -  Index fuer Stack
I    -  Index fuer Mnemonic-Tabelle
H$   -  Werte fuer Hexa-Umrechnung
FE$  -  Zwischenspeicher fuer Anfangsadresse
A$   -  Startadresse in Hexa
X$   -  Hexa-Wert bei Umrechnung
DE   -  Dezimal-Wert bei Umrechnung
Z0 ]
N0 ] -  Indexfelder fuer Umrechnung
Z1 ]
A    -  Anfangsadresse in Dezimal
AN$  -  KZ fuer Druckerausgabe
N    -  Zeilennummer
AD   -  aktuelle Adresse bei der Disassemblierung
P    -  Zwischenspeicher fuer Adressrechnung
EA   -  Dezimal-Wert des laufenden Befhles
S$   -  Hexa-Werte der laufenden Disassembler-Zeile
MN$  -  laufende Disassembler Zeile
K    -  Anzahl der Bytes fuer das Befehlswort
AA$  -  Eingabefeld bei Sprungbefehlen
FF$  -  Ueberschrift bei Druckerausgabe
A1   -  Arbeitsfeld zur Adressberechnung bei JMP ind.
A2   -  wie A1, jedoch zur Kontrolle ob Memory vorhanden
```

Variablen-Liste vom »Logic Disassembler«

und auf die erneute Eingabe einer Startadresse gewartet.

Bei der Eingabe von »END« statt einer Startadresse wird das Programm beendet.

```
700 IFRIGHT$(MN$,1)="$"THEN770
710 IFEA=108THEN840
720 IFMID$(MN$,5,1)="("THEN790
730 IFMID$(MN$,5,1)="$"THEN810
740 GOSUB750:MN$=LEFT$(MN$,4)+"$"+A$(2)+A$(1)+","+RIGHT$(MN$,1):GOTO940
750 FORK=1TO2:AD=AD+1:P=PEEK(AD):DE=P:GOSUB300:A$(K)=RIGHT$(X$,2)
760 S$=S$+" "+RIGHT$(X$,2):NEXT:RETURN
770 AD=AD+1:P=PEEK(AD):DE=P:GOSUB300:S$=S$+" "+RIGHT$(X$,2)
780 MN$=MN$+RIGHT$(X$,2):GOTO940
790 AD=AD+1:P=PEEK(AD):DE=P:GOSUB300:S$=S$+" "+RIGHT$(X$,2)
800 MN$=LEFT$(MN$,5)+"$"+RIGHT$(X$,2)+"),"+RIGHT$(MN$,1):GOTO940
810 AD=AD+1:P=PEEK(AD):DE=P:GOSUB300:S$=S$+" "+RIGHT$(X$,2)
820 MN$=LEFT$(MN$,5)+RIGHT$(X$,2)+","+RIGHT$(MN$,1):GOTO940
830 GOSUB750:MN$=LEFT$(MN$,3)+" $"+A$(2)+A$(1):GOTO940
840 AD=AD+2:P=PEEK(AD):DE=P:GOSUB300:S$=S$+" "+RIGHT$(X$,2)
850 MN$=LEFT$(MN$,5)+"$"+RIGHT$(X$,2)
860 P=PEEK(AD-1):DE=P:GOSUB300:S$=S$+" "+RIGHT$(X$,2)
870 MN$=MN$+RIGHT$(X$,2)+")":GOTO940
880 REM BRANCH-BEFEHL
890 IFMN$="BRK"ORLEFT$(MN$,1)<>"B"THEN940
900 AD=AD+1:P=PEEK(AD):DE=P:GOSUB300:S$=S$+" "+RIGHT$(X$,2)
910 IFP>=130THENA0=AD-(255-P):GOTO930
920 A0=(AD-1)+P+2
930 DE=A0:GOSUB300:MN$=MN$+" $"+X$:KZ=1
940 IFEA=32THENMN$=MN$+"                STACK"+STR$(SS%+1)
950 IFEA=96ANDSS%>0THENMN$=MN$+"                STACK"+STR$(SS%-1)
960 PRINTLEFT$(S$+"                ",16)MN$:IFEA=96THENPRINT
970 IFKZ=0THEN1030
980 GETAA$:IFAA$=""THEN980
990 IFAN$="Y"THENCMD4,CHR$(15);
1000 KZ=0:IFAA$="N"THEN1030
1010 IFAA$="X"GOTO1260
1020 PRINT:AD=A0-1:KZ=0
1030 IFEA=32OREA=76OREA=108OREA=96THEN1050
1040 GOTO630
1050 IFEA<>96THEN1090
1060 REM "BEFEHL: RTS"
1070 IFSS%=0THEN1260
1080 SS%=SS%-1:AD=SS(SS%):GOTO630
1090 REM "BEFEHL: JSR"
1100 GETAA$:IFAA$=""THEN1100
1110 IFAN$="Y"THENCMD4,CHR$(15);
1120 IFAA$="N"THEN630
1130 IFAA$="X"GOTO1260
1140 IFEA<>32THEN1180
1150 SS(SS%)=AD:SS%=SS%+1:PRINT
1160 AD=PEEK(AD)*256+PEEK(AD-1)-1
1170 GOTO630
1180 REM "BEFEHL: JMP ABS."
1190 IFEA<>76THEN1210
1200 AD=PEEK(AD)*256+PEEK(AD-1)-1:PRINT:GOTO630
1210 REM "BEFEHL: JMP IND."
1220 A1=PEEK(AD)*256+PEEK(AD-1)
1230 A2=PEEK(AD)*256+PEEK(AD-1)
1240 IFA1<>A2THENPRINT:PRINT"LOGIC FLOW ERROR":GOTO1260
1250 AD=PEEK(A1+1)*256+PEEK(A1)-1:PRINT:GOTO630
1260 DE=AD:GOSUB300
1270 PRINT:PRINT"START-ADR. = "A$:PRINT"END-ADR.   = $"X$
1280 PRINT:PRINT"< DISASSEMBLER - END >"
1290 IFAN$="Y"THENPRINT#4,:CLOSE4
1300 GOTO520
```

Listing zum »Logic Disassembler« (Schluß)

```
 100 -  250  Programmstart und Tabelleninitialisierung
 260 -  290  Umrechnung Hexa in Dezimal
 300 -  350  Umrechnung Dezimal in Hexa
 360 -  510  DATA-Statements mit den 6502-Mnemonics
 520 -  550  Eingabe der Startadresse
 560 -  620  Abfrage auf Druckerausgabe
 630 -  870  Disassembler-Schleife
 880 -  970  BRANCH-Befehle
 980 - 1040  Eingabe (Y)es, (N)o, e(X)it
1050 - 1080  RTS-Befehl mit Stack-Abfrage
1090 - 1170  JSR-Befehl mit Stack-Erhoehung
1180 - 1200  JMP-Befehl absolut
1210 - 1250  JMP-Befehl indirekt
1260 - 1300  Ende
```

Die wichtigsten Programmteile vom »Logic Disassembler«

Dabei wird jede Stackänderung angezeigt.

Es können alle Sprungbefehle verfolgt werden, sogar solche, die sich auf eine indirekte Adressierung beziehen. Als Beispiel hierfür kann das Disassembler-Protokoll der Abfrage der Stop-Taste dienen (Bild 2).

In der Zeile 1 ist der Befehl JMP ($0328) gefunden worden, jetzt wird der Inhalt der Speicherstelle $0328 und $0329 ausgelesen und die Sprungadresse errechnet. Dann erfolgt ein Speichertest an der errechneten Sprungadresse, und erst, wenn dieser Test positiv verläuft, wird ab dieser Adresse weiter disassembliert. Verläuft der Test jedoch negativ, erfolgt ein »LOGIC FLOW ERROR«, und es wird zum Ende der Disassemblierung gesprungen.

Das Programm »Logic-Disassembler« ist in der vorliegenden Form für den VC 20 geschrieben, es ist jedoch nach Änderung der Zeilen 100 und 530 (die POKE-Befehle ändern lediglich die Farben des Bildschirms) sofort auf dem Commodore 64 und den anderen CBM-Computern lauffähig.

(Fred Hammer)