

PETER'S ASSEMBLERECKE

Texte flott gedruckt

Nachdem der sechsteilige Assemblerkurs im letzten Heft zu Ende ging, dürfen wir sicher einige neue Leser in der Assemblercke begrüßen. Sie können die neu erworbenen Kenntnisse gleich an einem praktischen Beispiel erproben, das Ihnen sicherlich auch bei künftigen Programmierprojekten wertvolle Dienste leisten wird. Aber auch fortgeschrittenere Leser kommen nicht zu kurz, denn das Beispiel dieser Assemblercke ist ganz schön trickreich programmiert.

Thema ist die Ausgabe eines Strings am Bildschirm. Mancher Leser erinnert sich vielleicht noch an die erste Assemblercke vom Februar 1985, in der es ebenfalls darum ging. Im Unterschied zu damals wollen wir diesmal einen wesentlich eleganteren, aber auch kniffligeren Anlauf nehmen.

Ziel sollte es sein, Texte ohne großen Aufwand am Bildschirm ausgeben zu können. Bestes Vorbild wäre die PRINT-Funktion von Basic, denn einfacher als mit PRINT "Text" geht es wirklich nicht. Wer sich schon mit dieser Anwendung befaßt hat, weiß, daß sie in Assembler viel schwieriger ist. Man muß die Anfangsadresse des Strings in einem IOCB eintragen, eine maximale Länge angeben und schließlich die zentrale Ein/Ausgabe-Routine CIO aufrufen. Außerdem ist es erforderlich, den String entweder mit einem JMP-Befehl zu überspringen oder an anderer Stelle im Programm unterzubringen.

Sehen Sie sich dagegen einmal folgende Möglichkeit an:

```
JSR PRINT
ASC "Text"
...
```

Ein solches Unterprogramm PRINT finden Sie im Listing. Wir gehen sogar noch einen Schritt weiter und geben zusätz-

lich an, ab welcher Cursor-Position der String stehen soll. Abgeschlossen wird er – wie beim Atari üblich – mit einem End-of-Line-Zeichen (\$9B). Das Ganze sieht dann folgendermaßen aus:

```
JSR PRINT
DFB <X-Koord.>, <Y-Koord.>
ASC "<String>"
DFB $9B
```

Danach können Sie, wie auch das Beispiel des Listings zeigt, ganz normal weiterprogrammieren, ohne das Datenssegment in irgendeiner Form überspringen zu müssen. Damit steht uns ein sehr praktischer Weg zur Verfügung.

Nun bleiben aber noch einige Fragen offen. Woher weiß das Unterprogramm, wo sich der auszudruckende Text befindet? Schließlich werden ja keinerlei Parameter übergeben. Warum kann der String mitten im Programm stehen und muß nicht per JMP übersprungen werden? Wem so etwas schon einmal aus Versehen passiert ist, der kennt bestimmt die bitteren Konsequenzen.

Manipulieren, aber richtig!

Des Rätsels Lösung ist einfach. Bei einem Unterprogrammaufruf mit JSR merkt sich die CPU immer die Adresse, an welcher der Ablauf nach dem RTS fortzusetzen ist. Genau an dieser Rücksprungsadresse, die vom JSR-Befehl am Stack zurückgelassen wird, befindet sich der Datenblock, bestehend aus Koordinaten und String.

Man kann sich die Rücksprungsadresse einfach mit zwei PLA-Kommandos verschaffen und sie dann auf beliebige Art manipulieren. Es ist jedoch eine Eigenheit des 6502-Prozessors, daß diese nicht auf das nächste

Byte nach dem JSR zeigt, sondern auf das letzte des JSR-Befehls selbst. (Es handelt sich dabei ja um ein 3-Byte-Kommando.)

Nun ist es auch nicht weiter verwunderlich, daß der Datenblock nach dem JSR nicht übersprungen werden muß. Es genügt, lediglich die Rücksprungsadresse am Stack so zu modifizieren, daß das RTS den Programmablauf erst nach dem Datenblock fortsetzt. Wieder ist zu beachten, daß die veränderte Rücksprungsadresse auf das letzte String-Element (EOL) zeigen muß und nicht auf den darauf folgenden Befehl.

Diese Art der Parameterübergabe ist natürlich nicht nur auf die Ausgabe von Strings beschränkt. Sie läßt sich überall einsetzen, wo es gilt, viele Parameter an ein Unterprogramm zu leiten.

Es lohnt sich, auch die Technik der String-Ausgabe kurz zu betrachten. Hier wird einmal nicht mit der CIO-Funktion des Betriebssystems gearbeitet, sondern eine Routine bemüht, die einzelne Zeichen am Bildschirm ausgeben kann. Da diese nicht bei allen Versionen des Betriebssystems an der gleichen Adresse liegt, wird sie über einen Vektor in der sogenannten Editor-Handler-Tabelle aufgerufen. Dazu muß ein indirekter

Unterprogrammssprung durchgeführt werden.

Leider besitzt der 6502 zu diesem Zweck keinen Befehl. Über eine weitere Stapelmanipulation läßt sich das Ziel aber erreichen. Das funktioniert so: Das Unterprogramm SCROUT wird mit einem normalen JSR angesprungen, so daß die Rücksprungsadresse nach PRINT korrekt auf den Stack kommt. In SCROUT wird die Adresse der tatsächlichen Ausgaberroutine mit zwei PHA-Kommandos zusätzlich am Stack abgelegt. Genauer gesagt handelt es sich um die Adresse minus eins. (Die Gründe hierfür wurden bereits erläutert.)

Das folgende RTS führt nun einen indirekten Sprung zu dieser Routine durch, und erst deren abschließendes RTS läßt den Programmablauf wieder nach PRINT zurückkehren. Dabei ist zu beachten, daß das auszugebende Zeichen im Akku stehen muß; folglich ist die Kombination TAX-TXA in SCROUT noch notwendig.

Sie sehen, mit der Stack-Manipulation läßt sich einiges sehr elegant programmieren. Man muß aber gut aufpassen, daß alles richtig läuft, denn der Computer reagiert auf entsprechende Fehler mit einem Absturz.

Peter Finzel

Assemblerlisting

```
*****
*      Ausgabe von Strings      V1.1 *
*                                  *
*Assembler: ATMAS-II    P. Finzel 87 *
*****
*
ZEIGER    EQU $80    ;Zeropage-Register
OFFSET    EQU $82    ;Zaehler
*
RTCLOK    EQU $12    ;Uhr
COLCRS    EQU $55    ;Cursorspalte
ROWCRS    EQU $54    ;Cursorzeile
CRSINH    EQU $2F0    ;1:=Cursor unsichtbar
EOL        EQU $96    ;End-Of-Line Zeichen
*
```

```

* Adressen der Handlertabelle im ROM
*
EDITDV EQU $E400 ;Editor-Handler
*
* Start an Adresse $A800 im Monitor
*

```

```

    ORG $A800

```

```

*
*
BEISPIEL JSR PRINT
          DFB 5,10
          ASC "Das ist ein Beispiel"
          DFB EOL
          JSR WARTEN

```

```

*
* Jetzt revers ausdrucken
*

```

```

    JSR PRINT
    DFB 5,10
    ASC /Das ist ein Beispiel/
    DFB EOL
    JSR WARTEN
    JMP BEISPIEL

```

```

*
* Ein Weilchen warten
*

```

```

WARTEN   LDX #5
WART1    LDA RTCLOCK+2 hat Uhr schon
WART2    CMP RTCLOCK+2 weitergezaehlt?
          BEQ WART2    nein ->
          DEX
          BNE WART1
          RTS

```

```

*****
*      Das PRINT-Unterprogramm
*****

```

```

PRINT    PLA      Aufrufadresse
          STA ZEIGER holen
          PLA      in ZEIGER merken
          STA ZEIGER+1
          LDY #1    Cursor abschalten
          STA CRSINH
          LDA (ZEIGER),Y
          STA COLCRS Anfangsspalte
          INY
          LDA (ZEIGER),Y
          STA ROWCRS Anfangszeile
          STY OFFSET Zaehler merken

```

```

PRINT1   INC OFFSET Zaehler erhoehen
          LDY OFFSET
          LDA (ZEIGER),Y Zeichen holen
          CMP #EOL    schon letztes?
          BEQ PRTEND   ja -->
          JSR SCROUT   sonst ausgeben
          JMP PRINT1   und weiter ==>

```

```

PRTEND   CLC
          LDA ZEIGER   neue Ruecksprung-
          ADC OFFSET   adresse berechnen
          STA ZEIGER   (alte Adresse
          LDA ZEIGER+1 plus Zaehler)
          ADC #0
          PHA          Ruecksprungadresse
          LDA ZEIGER   zurueck auf Stack
          PHA
          RTS

```

```

*****
*      Ein Zeichen ausgeben
*****

```

```

SCROUT   TAX          Zeichen retten
          LDA EDITDV+7
          PHA          Anfangsadresse-1
          LDA EDITDV+6 der SCROUT-Routine
          PHA          im OS-ROM
          TXA          Zeichen zurueck
          RTS          indirekter Sprung

```