

### 09 Ausgabe von Zahlen (CK 11/85)

Ein kniffliges Kapitel der Assemblerprogrammierung ist die Ausgabe von Zahlen auf dem Bildschirm. Während in BASIC ein PRINT-Befehl die gesamte Arbeit leistet, ist in Assembler schon etwas mehr geistiges Kapital zu investieren. Jeder, der seinen Atari in Maschinensprache programmiert, wird früher oder später vor diesem Problem stehen: Eine Zahl, sei es nun ein Rechenergebnis, ein Punktestand oder auch nur ein Fehlercode, muss angezeigt werden. Bei Assemblerprogrammen müssen im Regelfall sog. "2-Byte Integers" (s. Kasten) ausgegeben werden, deren Wertebereich die meisten Anwendungsfälle abdeckt. Das Problem liegt aber darin, eine derartige binäre Zahl in eine Folge von Code-Zahlen umzurechnen, die die einzelnen Ziffern repräsentieren. Diese Codes sollen, wenn sie in den Bildschirmspeicher geschrieben werden, die gewünschte Zahl in einer für Menschen gut lesbaren Darstellung (also im Zehnersystem) ergeben.

Obwohl ja bekanntlich viele Wege nach Rom führen, wollen wir hier nur zwei verschiedene Methoden genauer betrachten. Beide Möglichkeiten haben dabei ihre Vorteile für bestimmte Einsatzgebiete.

#### Der fließende Punkt

---

Der erste und einfachere Weg führt über die sogenannte Floatin-Point Package. Im Speicherbereich von \$D800 bis \$DFFF befinden sich eine Anzahl von Rechenroutinen, angefangen bei der Addition über Multiplikation bis hin zur Berechnung einer Potenzreihe. Alle diese Routinen benutzen ein Fließkomma-Format. Der Aufruf dieser Programme erfolgt über (im O. S. Manual) festgelegte Einsprung-Adressen, die tatsächlich in allen Atari-Computern gleich sind. Das heißt, dass ein damit geschriebenes Programm auf einem alten 400er genauso wie auf einem neuen 130XE läuft. Davon könnte sich mancher Computer-Hersteller eine dicke Scheibe abschneiden.

Ebenfalls In diesem ROM sind Wandlungsroutinen enthalten, die es erlauben, eine Intergerzahl in FP-Darstellung umzurechnen (IFP, \$D9AA), sowie eine FP-Zahl in einen ATASCII-String (FASC, \$D8E6) zu konvertieren. Die auszugebende Zahl wird dabei in das FP-Register 0 (FR0, ab \$D4) eingetragen, wobei das LSB nach FR0 und das MSB nach FR0+1 kommt. Ein Aufruf von IFP wandelt den Inhalt von FR0 in FP-Darstellung um, ein weiterer Aufruf von FASC erzeugt die Darstellung der Zahl im ATASCII-Code. FASC teilt uns im Vektor INBUFF (\$F3 \$F4) noch mit, wo der String zu finden ist. Schließlich kann man den String als Text direkt über CIO (Central Input Output, wer erinnert sich noch an Assemblerecke Nr. 1 "Textausgabe"?) auf dem Bildschirm ausgeben. Wichtig ist dabei, daß das 8. Bit der letzten Ziffer gleich eins beträgt, so daß das Ende der Zahl bequem erkennbar ist.

[Listing 1](#) gibt ein Beispiel dieser Methode indem ein Zähler in der Mitte des Bildschirmes so schnell wie möglich zählt.

#### Do it yourself

---

In der Methode Nummer zwei nehmen wir die Sache selbst in die Hand. Ein Blick ins Handbuch der Mathematik bringt Ihnen die Erkenntnis, daß die Umrechnung einer Zahl vom Zweier ins Zehnersystem durch fortgesetzte Division durch 10 zu erreichen ist, wobei der Rest jeder Division eine Ziffer liefert. Schön und gut. aber schon das Wort Division erzeugt etwas Unbehagen, da der 6502 nun mal von Haus aus nicht dividieren kann. Glücklicherweise gibt es da noch einen Trick, der auf folgender Überlegung beruht: Die größte mit 2-Byte Integers darstellbare Zahl ist 65535. wobei die höchste enthaltene Zehnerpotenz 10000 ist. Man subtrahiert diese Zehnerpotenz nun so oft von der zu wandelnden Zahl, bis die Zahl selbst kleiner als 10000 ist.

---

Dabei merkt man sich, wie oft die Subtraktion stattfand und gewinnt so die Ziffer, die an die Zehntausender Stelle einzutragen ist. Das gleiche Spiel wird mit 1000, 100 und 10 wiederholt, und schließlich bleibt nur noch die Einerstelle übrig.

Simpel, nicht wahr? Genau dieses Schema wird in [Listing 2](#) verwendet. Eine Zahl, die dem Unterprogramm BIN-DEZ im Akku und X-Register übergeben werden muß, wird in dem fünf Byte langen Buffer "ZIFFER" im Bildschirmcode (3. Kasten) abgelegt. Jetzt braucht man nur noch eine Routine (PRINT), die den errechneten Code in den Bildschirmspeicher schreibt. Dies geschieht ohne Zutun des Betriebssystems durch direktes Einbringen der Codes in den Bildschirmspeicher. Zur Demonstration wird vom Programm ein spezieller, nur aus einer GRAPHICS-2 Zeile bestehender Bildschirm aufgebaut (Unterprogramm SETDL). In dieser Zeile läuft dann wieder ein Zähler mit höchster Geschwindigkeit.

Nun haben Sie zwei Methoden zur freien Auswahl nur welche ist günstiger für eine spezielle Anwendung? Wenn Sie beide Methoden eingetippt und ausprobiert haben, dann wissen Sie es bereits: Methode 1 (via FP-ROM) ist die weitaus langsamere von beiden. Wenn es nicht auf Geschwindigkeit ankommt ist diese Methode zweifellos die bessere, da sie einfacher anzuwenden ist und einen ATASCII-String mit Unterdrückung der Nullen vor der ersten Ziffer liefert.

Methode zwei hingegen liefert blitzschnelle Ergebnisse und eignet sich daher besonders für zeitkritische Anwendungen, Typisch wäre hier die Ausgabe eines Punktstandes in einem Spielprogramm oder das Mitführen eines Zählers in einem schnellen Sektorkopierer. Tatsächlich ist Methode 2 um mehr als 15 mal schneller als der Umweg über die Floating-Point Arithmetik. Interessant ist an dieser Stelle vielleicht ein Vergleich der Laufzeiten von verschiedenen Sprachen (jeweils die Zeit, bis der Zähler 5000 erreicht hat):

Atari-BASIC:	63 Sek.
Assembler (Methode 1):	46 Sek.
ACTION! (mit PrintC):	40 Sek.
ACTIONI (Listing 3):	7 Sek.
Assembler (Methode 2):	3 Sek.

Diese Ergebnisse sind doch recht erstaunlich, Unsere Methode 1 ist nicht viel schneller als Basic! Nun ja, Basic verwendet schließlich die gleichen FP-Routinen und diese sind eben langsam. Auf der anderen Seite kommt ACTION! dem Assemblerprogramm schon verdächtig nahe, vorausgesetzt, man programmiert den Algorithmus nach Methode 2. Das ACTION1-Programm dazu, eine Umsetzung von Listing 2, finden Sie in [Listing 3](#) .

Eine gewisse Mitschuld am Schneckentempo der Methode 1 trifft auch die Betriebssystem CIO-Routine, mit der die normale Bildschirmausgabe erledigt wird. Wandelt man den ATASCII-String mit einem Maschinenprogramm in Bildschirmcode um und schreibt ihn direkt in den Videospeicher, so würde das einiges an Tempo bringen. Aber dann kann man eigentlich gleich zur Methode 2 greifen, denn hier haben Sie auch die Zeropage der FP-Package von \$D4 bis \$FF zu Ihrer Verfügung.

---

### Zahlen, Codes und ATMAS-II

---

Wenn man in Assembler programmiert, hat man es im Regelfall mit "2-Byte Integers", also ganzen Zahlen im Bereich von 0 bis 65.535 zu tun. Diese Zahlen liegen im Rechner in einem binären Format (im Zweiersystem) vor. Zur Darstellung werden dabei 16 Bit benötigt die in zwei Bytes Platz finden. Bei 6502-Rechnern ist es hier üblich, zuerst das Byte mit den niederwertigen Bits (das LSB) und in der nächsthöheren Adresse das Byte mit den höherwertigen 8 Bits (MSB) abzulegen.

Als Floating-Point (FP) bezeichnet man eine kompliziertere Art der Zahlendarstellung, die nicht nur ganze, sondern auch gebrochene Zahlen erlaubt. Die sechs Byte lange rechnerinterne Darstellung ist in eine Mantisse und einen Exponenten unterteilt, so daß auch der Zahlenbereich wesentlich erweitert wird (beim ATARI: +/-10E-98 bis +/-10E+98). Floating-Point Arithmetik wird z.B. vom BASIC benutzt, während viele BASIC-Compiler oder auch der ACTION!-Compiler mit einer wesentlich schnelleren Integer-Arithmetik arbeiten.

Unter ATASCII-Code versteht man eine Adaption des ASCII-Codes für Atari-Computer, der u.a. neben den ursprünglichen Zeichen auch noch Grafik und Invers-Zeichen enthält. Bildschirmcode dagegen ist ein hardwarespezifischer Zeichencode, bei dem jeder Code-Zahl ein Bitmuster des Zeichensatzes zugeordnet wird. Wenn Sie daher einen ASCII-String auf den Bildschirm bringen wollen, so müssen Sie diesen zuerst in den Bildschirmcode umwandeln. Diese Arbeit nimmt Ihnen jedoch im Regelfall das Betriebssystem ab. Die Ziffern beginnen dabei mit der Null ab Code \$10.

Die Assemblerlistings wurden diesmal mit dem ATMAS-II Makroassembler erstellt. Optisch auffälligstes Kennzeichen ist das Fehlen der Zeilennummern, die wegen des bildschirmorientierten Editors nicht mehr benötigt werden. Beide Programme können natürlich auch für Assembler-Cartridge oder für MAC/65 umgesetzt werden. In diesem Fall sollten Sie eine andere Anfangsadresse der Maschinenprogramme (z. B. \$7000) wählen, da bei ATMAS-II der spezielle reservierte Bereich ab \$A800 benutzt wird. Beide Programme können Sie dann im ATMAS-Monitor mit "G" A800 starten und mit RESET abbrechen, da es sich um Endlosschleifen handelt.

---

```

*****
*  Zahlenausgabe in Assembler      *
*  LISTING 1 - FLOATING POINT      *
*  mit der FLOATING-POINT PACKAGE  *
*  ATMAS-II PETER FINZEL 1985      *
*****
*
*  Konstante fuer Floating-Point
*

FASC      EQU  $D8E6      FP nach ASCII
IFP       EQU  $D9AA      Integer nach FP
FR0       EQU  $D4        FP-Akku 0
INBUFF    EQU  $F3        Zeiger auf ASCII-Ergebnis

*
*  IOCB-Konstante
*

CIOV      EQU  $E456
ICCOM     EQU  $342
ICBAL     EQU  $344
ICBAH     EQU  $345
ICBLL     EQU  $348
ICBLH     EQU  $349

*
*  CIO-Befehlscode
*
CPBIN     EQU  11  Get Binary Record
*
*  Cursorsteuerung
*

ROWCRS    EQU  $54        Cursor-Zeile
COLCRS    EQU  $55        Cursor-Spalte
CRSINH    EQU  $2F0      Cursor ein=0/aus=1

*****
*  Testprogramm:Zaehler am Bildschirm
*****

          ORG  $A800      im res. Bereich

TEST     LDA  #0          Zaehlung beginnt
          STA  ZAHL        bei Null
          STA  ZAHL+1
          LDA  #1          Cursor aus
          STA  CRSINH

ZAEHLER  LDA  ZAHL        zu wandelnde
          LDX  ZAHL+1      Zahl uebergeben
          JSR  BINASC      umwandeln
          LDX  #18         in die Mitte
          LDY  #11        des Bildschirmes
          JSR  PRINT      ausgeben
          INC  ZAHL        und weiterzaehlen
          BNE  ZAEHLER
          INC  ZAHL+1     auch das MSB
          JMP  ZAEHLER     Endlose Schleife=>

```

```

ZAHL      DFW 0          Zaehl-Register

*****
* Wandlung Binaer nach ATASCII
* <A>: LSB, <X>:MSB
*****

BINASC    STA FR0        Zahl in FP-Register
          STX FR0+1      Nr. 0 eintragen
          JSR IFP        in FP umwandeln
          JSR FASC       FP in ATASCII
          RTS           umwandeln

*****
* Ausgabe der Zahl auf Bildschirm
* ueber CIO-Interface
* <X>:Spalte <Y>:Zeile
*****

PRINT     STX COLCR5     POSITION ausfuehern
          STY ROWCR5
          LDX #0
          STX COLCR5+1   MSB der Spalte
          STX ICBLL      nur 1 Zeichen
          STX ICBLLH     ausgehen
          STX INBPTR     INBUFF-Zeiger auf Null
          LDA #CPBIN     Zeichenausgabe
          STA ICCOM

NXTCHR    LDY INBPTR     Zeiger in INBUFF
          LDA (INBUFF),Y Zeichen aus Buffer
          BMI PRTEnd     letztes Zeichen
          JSR CIOV       ausgehen
          INC INBPTR     Zeiger weiter
          JMP NXTCHR     und naechstes->

*
* Letztes Zeichen: 8.Bit loeschen u. ausgehen
*

PRTEnd    AND #$7F      Hi-Bit loeschen
          JSR CIOV       und Zeichen 'raus
          RTS

INBPTR    DFB 0          Index fuer INBUFF

```

```

*****
*  Zahlenausgabe in Assembler  *
*  LISTING 2 - Subtraktionsmethode*
*  ATMAS-II PETER FINZEL 1985  *
*****

VDLSTL  EQU $230  Display-List Zeiger
BZAHL   EQU $D4   Hilfsregister in Zeropage
          ORG $A800 im reservierten Bereich

*****
* Testprogramm: Zaehler am Bildschirm
*****

TEST     CLD           wir rechnen binaer
          LDA #0       Zaehlung beginnt
          STA ZAHL     bei Null
          STA ZAHL+1
          JSR SETDL    Display-List

ZAEHLER  LDA ZAHL     Zaehlerinhalt
          LDX ZAHL+1  an Wandelpogramm
          JSR BINDEZ  umwandeln
          LDY #13     zentrieren
          JSR PRINT   ausgeben
          INC ZAHL    und weiterzaehlen
          BNE ZAEHLER
          INC ZAHL+1  auch das MSB
          JMP ZAEHLER

ZAHL     DFW 0        Zaehler-Register

*****
* Wandlungsroutine Binaer in Dezimal
* <A>: LSB, <X>:MSB
*****

BINDEZ   STA BZAHL    Parameter speichern
          STX BZAHL+1
          LDX #4
VORBES   LDA #$10     Ausgabebuffer
          STA ZIFFER,X mit B.-Code fuer
          DEX          Null vorbesetzen
          BPL VORBES
          LDX #0      Stellenzaehler

STELLE   LDA BZAHL+1  Zehner-Potenz
          CMP DECHI,X  groesser als
          BNE TSTHI   restliche Zahl?
          LDA BZAHL
          CMP DECLO,X

TSTHI    BCC KLEINER  Potenz zu gross

*
* Zehner-Potenz abziehen, Ziffer erhoehen
*
          SEC
          LDA BZAHL    die aktuelle
          SBC DECLO,X  Zehnerpotenz
          STA BZAHL    abziehen
          LDA BZAHL+1
          SBC DECHI,X
          STA BZAHL+1
          INC ZIFFER,X Ziffer erhoehen
          JMP STELLE   gleiche Stelle nochmal

```

```

*
* kleinere Zehnerpotenz anwaehlen
*
KLEINER   INX           schon vier Stellen
          CPX #4        bearbeitet?
          BNE STELLE    nein ->

*
* nur noch Einerstelle da
*
          CLC           Einerstelle
          LDA BZAHL     in Ausgabebuffer
          ADC ZIFFER+4  addieren
          STA ZIFFER+4
          RTS

*
* Tabellen der Zehnerpotenzen
* getrennte Tabellen f. LSB und MSB
*
DECL0     DFB 10000,1000,100,10
DECL1     DFB 10000/256,1000/256,0,0

*
* Ausgabebuffer
*
ZIFFER    ASC %00000%
*****
* Gewandelte Zahl aus ZIFFER in den
* Bildschirm kopieren
* <Y>:letzte Spalte
*****
PRINT     LDX #4           5 Zahlen
NZIF      LDA ZIFFER,X     aus Buffer
          STA BLDRAM,Y     in Bildschirm
          DEY
          DEX
          BPL NZIF        naechste -->
          RTS

*****
* Spezial-Displaylist aus einer Zeile
*****
SETDL     LDA #DLIST       Zeiger auf
          STA VDLSTL       neue Display-
          LDA #DLIST/256    list eintragen
          STA VDLSTL+1
          RTS

*
* Display-List
*
DLIST     DFB $70,$70,$70,$70
          DFB $70,$70,$70,$70
          DFB $47           GRAPHICS 2
          DFW BLDRAM
          DFB $41           Sprung auf
          DFW DLIST         Anfang der D.-List

*
* Video-Ram (20 Bytes lang)
* ASC erzeugt hier Bildschirmcode!
*
BLDRAM    ASC % SCORE:           %

```

Peter's Assemblerecke - 09 Ausgabe von Zahlen (CK 11/85)

```

;*****
;          Zahlenausgabe
;
;LISTING 3: Subtraktionsmethode
;
;ACTION!-Version  Peter Finzel 1985
;*****

DEFINE Null="$10"

BYTE ARRAY BldRam(20)="

;*****
;Wandelt CARD in Bildschirmcode
;gibt Zeiger auf Ausgabebuffer zur.
;*****

CARD FUNC Bin_Dez(CARD Wert)
BYTE i
BYTE ARRAY Ziffer(5)=[16 16 16 16 16]
CARD ARRAY Zehner(5)=[10000 1000 100 10 1]

SetBlock(Ziffer,5,Null)
FOR i=0 TO 3
    DO
        WHILE Wert>=Zehner(i)
            DO
                Wert      ==-Zehner(i)
                Ziffer(i)==+1
            OD
        OD
        Ziffer(4)==+Wert ;Einerstelle
    RETURN (Ziffer)

;*****
;Spezial-Display-List (eine Zeile)
;*****
PROC Set_Dlist()
BYTE ARRAY DList(0)=[$70 $70 $70 $70
                    $70 $70 $70 $70 $70 $47]

CARD Lms_1=[ 0 ]
BYTE L_2  =[ $41 ]
CARD DL_Jmp=[ 0 ]
CARD Vdlistl=$230

Lms_1=BLDRAM+1
DL_Jmp=DList
VDlistl=DList
RETURN

;*****
;Test-Programm: Zaehler
;*****
PROC TEST()
CARD ZAHL,OUT

Zero(BldRam+1,20) ;Video-Ram loeschen
Set_Dlist()      ;Display-List aktiv
ZAHL=0           ;Zaehler init.

    DO
        Out=Bin_Dez(Zahl)
        MoveBlock(BLDRAM+8,OUT,5)
        ZAHL==+1
    OD

RETURN

```