Dipl.-Phys. Peter Kittel

Große Sprünge mit dem 6502

Sollen mehrere Maschinenprogramme (Moduln) zusammengefügt werden, ist es von Vorteil, wenn sie im Speicher frei verschiebbar sind. Sonst müssen nämlich bei einer Umordnung der Programme alle absoluten Adressen mitgeändert werden, was meist sehr mühsam ist.

Um ein Programm frei verschiebbar zu machen, dürfen statt absoluter Adressen nur noch relative (zur derzeitigen Programmadresse) verwendet werden. Im 6502-Befehlssatz ist das aber leider nur bei den kurzreichweitigen (±128 Bytes) Branch-Befehlen möglich. Mit den weiter unten beschriebenen Hilfsroutinen AKTADR und VERSATZ werden dem Benutzer jetzt aber drei neue Pseudo-Befehle zur Verfügung gestellt, die 8 bzw. 5 Bytes lang sind. Mit ihnen lassen sich völlig frei verschiebbare Programme verwirklichen:

1. JMP relativ:
JSR AKTADR
VL ←
VH
JMP VERSATZ
2. JSR relativ:
JSR AKTADR
VL ←
VH
JSR VERSATZ
3. Ermittle derzeitige Programmadresse:
JSR AKTADR
NOP
NOP ←

Die Hilfsroutinen

Das Unterprogramm AKTADR ermittelt die aktuelle Programmadresse, erhöht sie um 2 (um die Versatzbytes einbauen zu können, s. u.) und speichert sie als 16 Bit langen Pointer in der Zero Page ab. Dieser Pointer zeigt jeweils auf das mit einem Pfeil gekennzeichnete Byte. Die Routine benutzt ein ähnliches Verfahren, wie es in [1] und [2] vorgestellt wurde.

VL und VH bilden zusammen einen 16 Bit großen Versatz, der in der Routine VERSATZ zum Pointer addiert wird. Der Versatz ist ab dem gekennzeichneten Byte zu zählen. Da der Versatz 16 Bit groß ist, kann man mit den Sprüngen jeden anderen Befehl im 64-KByte-Speicherbereich erreichen, also auch rückwärts springen.

Der dritte Pseudobefehl von 5 Byte Länge (wobei statt der NOP-Befehle auch beliebig anderes stehen kann) wird gebraucht, um ins Programm eingebaute Tabellen ansprechen zu können (s. u., Ladeprogramm).

Bild 1 zeigt ein Disassembler-Listing der beiden Routinen. Beide sind etwas länger als unbedingt nötig, weil mit einigem Aufwand *alle* Registerinhalte gerettet werden – in VERSATZ auf den Stack, in AKTADR in zwei Zero Page-Speicherstellen. Zusätzlich ist in AKTADR noch ein PET-2001-spezifischer Teil zur Abfrage der Break-Taste eingebaut. So hat man mehr Sicherheit gegen Maschinenprogramme mit „unendlichen" Schleifen. Dieser Programmteil be-

```
949   03B5  08        PHP IMPLIED
950   03B6  85 00     STA ZER PAG
952   03B8  68        PLA IMPLIED
953   03B9  85 0C     STA ZER PAG
955   03BB  18        CLC IMPLIED
956   03BC  68        PLA IMPLIED
957   03BD  69 02     ADC IMMED.
959   03BF  85 01     STA ZER PAG
961   03C1  68        PLA IMPLIED
962   03C2  69 00     ADC IMMED.
964   03C4  85 02     STA ZER PAG
966   03C6  48        PHA IMPLIED
967   03C7  A5 01     LDA ZER PAG
969   03C9  48        PHA IMPLIED
970   03CA  AD 03 02  LDA ABSOLUT
973   03CD  C9 04     CMP IMMED.
975   03CF  D0 03     BNE REL 03D4
977   03D1  4C 8B C3  JMP ABSOLUT
980   03D4  A5 0C     LDA ZER PAG
982   03D6  48        PHA IMPLIED
983   03D7  A5 00     LDA ZER PAG
985   03D9  28        PLP IMPLIED
986   03DA  60        RTS IMPLIED

987   03DB  08        PHP IMPLIED
988   03DC  48        PHA IMPLIED
989   03DD  98        TYA IMPLIED
990   03DE  48        PHA IMPLIED
991   03DF  A5 01     LDA ZER PAG
993   03E1  D0 02     BNE REL 03E5
995   03E3  C6 02     DEC ZER PAG
997   03E5  C6 01     DEC ZER PAG
999   03E7  A0 00     LDY IMMED.
1001  03E9  18        CLC IMPLIED
1002  03EA  B1 01     LDA (IND),Y
1004  03EC  65 01     ADC ZER PAG
1006  03EE  48        PHA IMPLIED
1007  03EF  C8        INY IMPLIED
1008  03F0  B1 01     LDA (IND),Y
1010  03F2  65 02     ADC ZER PAG
1012  03F4  85 02     STA ZER PAG
1014  03F6  68        PLA IMPLIED
1015  03F7  85 01     STA ZER PAG
1017  03F9  68        PLA IMPLIED
1018  03FA  A8        TAY IMPLIED
1019  03FB  68        PLA IMPLIED
1020  03FC  28        PLP IMPLIED
1021  03FD  6C 01 00  JMP ABS IND
```

Bild 1. Die Hilfsroutinen AKTADR (03B5-03DA) und VERSATZ (03DB-03FF) ermitteln die aktuelle Adresse des Programms und den relativen Versatz bei Sprung- und Ladebefehlen des Prozessors 6502

steht aus den Speicherstellen 03CA-03D3 und kann auf anderen Rechnern ganz entfallen. Bei CBMs mit neuen ROMs müssen auch andere Adressen eingesetzt werden.
Die beiden Routinen stehen im oberen Teil des Tape Buffers 2 im PET. Sie sind aber selbst schon frei verschiebbar, so daß sie auf anderen Rechnern auch an beliebig anderer Stelle untergebracht werden können. Man muß sich allerdings für eine bestimmte, unveränderliche Stelle entscheiden, um die Routinen jederzeit mit dem normalen, absolut adressierten JSR-Befehl aufrufen zu können.

So arbeiten AKTADR und VERSATZ

AKTADR rettet also zunächst Akku und Statusregister in zwei Speicherstellen, holt dann die vom JSR-Befehl dorthin gebrachte Rücksprungadresse vom Stack, erhöht sie um 2, speichert sie als Pointer in die Zero Page und schiebt sie wieder auf den Stack zurück. Bei gedrückter Break-Taste wird zum Warm Start of Basic (READY-Meldung beim CBM) gesprungen. Sonst werden die Register wieder auf den alten Stand gebracht und per RTS ins Hauptprogramm zurückgekehrt. Dabei werden wegen der Erhöhung der Rücksprungadresse die 2 Bytes hinter dem JSR-Befehl übersprungen, so daß in ihnen auf einfache Weise der 16 Bit lange Versatz für die relative Adressierung übergeben werden kann.
VERSATZ rettet die Register auf den Stack und erniedrigt den Pointer gleich wieder um 1, um auch an das erste (niederwertigere) Versatzbyte heranzukommen. Die beiden Versatzbytes werden mit der indirekt-indizierten Adressierung geholt und zum Pointer hinzuaddiert. Daraufhin werden die alten Registerinhalte wiederhergestellt. Als Abschluß erfolgt kein Return- sondern ein absolut-indirekter Sprungbefehl, der sich sein Sprungziel vom umgerechneten Pointer holt. VERSATZ ist also wegen des Sprunges am Schluß eigentlich keine Subroutine, sondern ein Vorspann vor dem angesprungenen Programmteil, so daß sie gleichermaßen für Sprünge wie für Subroutinen-Aufrufe zu gebrauchen ist.

Ladeprogramm und Beispiele

Bild 2 zeigt ein (natürlich frei verschiebbares) *Ladeprogramm*, das als Vorspann eines im oberen Teil des RAM liegenden Maschinenprogramms die beiden Routinen an eine *feste* Stelle speichert. Dazu wird zunächst eine Einfachversion von AKTADR „zu Fuß" erzeugt (*Bild 3*) und gleich aufgerufen, um wieder mit der indirekt-indizierten Adressierung die komplett ausgebauten Routinen aus einer dahinter folgenden Tabelle (1C40-1C8A) zu holen und umzuspeichern. Der Programmabschnitt 1C2E-1C3F zeigt, wie mit AKTADR umzugehen ist, aber Vorsicht: hier wurde der Pointer *nicht* um 2 erhöht wie in der späteren Version! Die Speicherstellen 1C8B und 1C8C sollen das folgende Hauptprogramm symbolisieren.
Bild 4 zeigt kurze Beispiele zu den immer 8 Bytes langen Pseudobefehlen für relative Sprünge und Subroutine-Aufrufe. Der Disassembler interpretiert auch die Versatzbytes als Befehle, was bitte ignoriert werden möge. Pfeile zeigen die ausgeführten Sprünge. – Im ersten Programm wird lediglich der NOP-Befehl mit einem relativen Sprung übergangen. (Das wäre auch mit einem normalen Branch-Befehl möglich, hier geht es nur ums Prinzip.) Der Versatz ist hier nur 6 Bytes groß, gezählt wird vom Byte aus, das die 06 enthält. – Im zweiten wird die Subroutine in Speicherstelle 0399, die lediglich aus einem RTS-Befehl besteht, relativ angesprungen. (Das ist mit den normalen Befehlen schon nicht mehr möglich!) – Das dritte Beispielprogramm, das in Speicherstelle 039D beginnt (nicht vorher!), zeigt einen relativen Sprung rückwärts nach 039C. Man beachte, daß bei der Addition dieses großen Versatzwertes zur Programmadresse eigentlich ein Wert von über 64 K herauskommt, der Übertrag wird jedoch ignoriert.
Im letzten Beispielprogramm mit Startpunkt in 03A7 wird die Subroutine in 03A6 rückwärts relativ aufgerufen, das Programm endet in 03AF.

```
7168  1C00  A9 68     LDA IMMED.
7170  1C02  8D B5 03  STA ABSOLUT
7173  1C05  8D B8 03  STA ABSOLUT
7176  1C08  A9 85     LDA IMMED.
7178  1C0A  8D B6 03  STA ABSOLUT
7181  1C0D  8D B9 03  STA ABSOLUT
7184  1C10  A2 01     LDX IMMED.
7186  1C12  8E B7 03  STX ABSOLUT
7189  1C15  8E BA 03  STX ABSOLUT
7192  1C18  E8        INX IMPLIED
7193  1C19  8E BA 03  STX ABSOLUT
7196  1C1C  A9 48     LDA IMMED.
7198  1C1E  8D BB 03  STA ABSOLUT
7201  1C21  8D BE 03  STA ABSOLUT
7204  1C24  A9 A5     LDA IMMED.
7206  1C26  8D BC 03  STA ABSOLUT
7209  1C29  A9 60     LDA IMMED.
7211  1C2B  8D BF 03  STA ABSOLUT
7214  1C2E  20 B5 03  JSR ABSOLUT
7217  1C31  A0 10     LDY IMMED.
7219  1C33  B1 01     LDA (IND),Y
7221  1C35  99 A5 03  STA ABS,Y
7224  1C38  C8        INY IMPLIED
7225  1C39  C0 58     CPY IMMED.
7227  1C3B  D0 F6     BNE REL 1C33
7229  1C3D  18        CLC IMPLIED
7230  1C3E  90 4B     BCC REL 1C8B

7232  1C40  08 85 00 68 85 0C 18 68
7240  1C48  69 02 85 01 68 69 00 85
7248  1C50  02 48 A5 01 48 AD 03 02
7256  1C58  C9 04 D0 03 4C BB C3 A5
7264  1C60  0C 48 A5 00 28 60 08 48
7272  1C68  98 48 A5 01 D0 02 C6 02
7280  1C70  C6 01 A0 00 18 B1 01 65
7288  1C78  01 48 C8 B1 01 65 02 85
7296  1C80  02 68 85 01 68 A8 68 28
7304  1C88  6C 01 00 EA 60 00 00 00

7307  1C8B  EA        NOP IMPLIED
7308  1C8C  60        RTS IMPLIED
```

Bild 2. Ladeprogramm für die beiden Hilfsroutinen in Bild 1. Es ist selbst wiederum frei verschiebbar (relokatibel)

```
949   03B5  68        PLA IMPLIED
950   03B6  85 01     STA ZER PAG
952   03B8  68        PLA IMPLIED
953   03B9  85 02     STA ZER PAG
955   03BB  48        PHA IMPLIED
956   03BC  A5 01     LDA ZER PAG
958   03BE  48        PHA IMPLIED
959   03BF  60        RTS IMPLIED
```

Bild 3. Einfachstversion für das Unterprogramm AKTADR, wie sie in Bild 2 verwendet wird

```
900   0384  20 B5 03  JSR ABSOLUT
903   0387  06 00     ASL ZER PAG
905   0389  4C DB 03  JMP ABSOLUT
908   038C  EA        NOP IMPLIED
909   038D  60        RTS IMPLIED

910   03BE  20 B5 03  JSR ABSOLUT
913   0391  08        PHP IMPLIED
914   0392  00        BRK IMPLIED
915   0393  20 DB 03  JSR ABSOLUT
918   0396  60        RTS IMPLIED
919   0397  EA        NOP IMPLIED
920   0398  EA        NOP IMPLIED
921   0399  60        RTS IMPLIED

924   039C  60        RTS IMPLIED
925   039D  20 B5 03  JSR ABSOLUT
928   03A0  FC        --- IMPLIED
929   03A1  FF        --- IMPLIED
930   03A2  4C DB 03  JMP ABSOLUT

934   03A6  60        RTS IMPLIED
935   03A7  20 B5 03  JSR ABSOLUT
938   03AA  FC        --- IMPLIED
939   03AB  FF        --- IMPLIED
940   03AC  20 DB 03  JSR ABSOLUT
943   03AF  60        RTS IMPLIED
```

Bild 4. Vier kurze Programmbeispiele mit den jeweils acht Bytes langen Pseudobefehlen für relative Sprünge und Unterprogramm-Aufrufe

Bild 5 zeigt einen Trace-Lauf dieses Programms, bei dem alle Befehle echt im Einzelschrittbetrieb ausgeführt und gleichzeitig disassembliert gelistet werden. An strategisch wichtigen Stellen habe ich per Tastendruck zusätzlich die Registerinhalte (im Trace-Betrieb eigener Stack mit eigenem Stack-Pointer!) abgerufen. Der in Bild 4 durch Pfeile angedeutete Ablauf wird tatsächlich so ausgeführt.

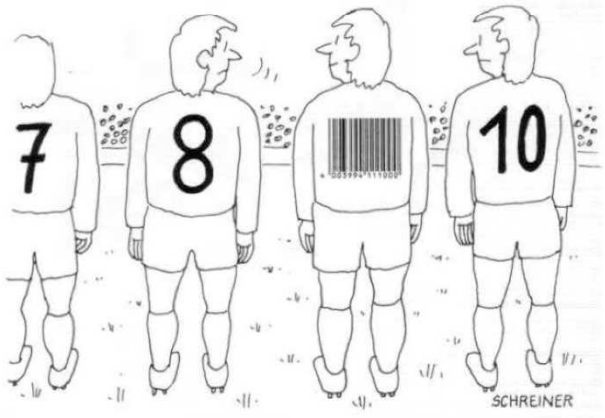
Adaption auf andere Systeme

Noch einmal: Sollen diese Routinen auf andere 6502-Rechner übertragen werden, können sie dort an eine *beliebige* Speicherstelle, sogar ins ROM (vielleicht eine Anregung für Firmen wie Commodore?) gelegt werden. Die vier in der Zero Page benötigten Speicherstellen können dort auch frei gewählt werden, solange die beiden Pointer-Bytes direkt nebeneinander liegen. Die anderen beiden Stellen müssen nicht einmal unbedingt in Zero Page liegen! Die *hier* gewählten vier Stellen in der Zero Page müßten für PETs und CBMs gleichermaßen verträglich sein [4].

Mit diesen Hilfsroutinen lassen sich also leicht beliebige Maschinenprogramme miteinander verbinden. Ich könnte mir sogar ein Betriebssystem ähnlich CP/M vorstellen, das Maschinenprogramme in irgendeinen freien RAM-Bereich lädt, anstartet und verwaltet. Diese Maschinenprogramme müßten nur ein paar Konventionen einhalten: Sie müßten frei verschiebbar sein (die Adressen der Hilfsroutinen im Betriebssystem lägen fest), müßten die standardisierten Ein/Ausgabe-Schnittstellen benutzen und müßten sich irgendwie über die Belegung der Zero Page einigen. Nur Letzteres erscheint schwierig, alles andere eigentlich simpel!

```
935   03A7  20 B5 03   JSR ABSOLUT
949   03B5  08         PHP IMPLIED
950   03B6  85 00      STA ZER PAG
952   03B8  68         PLA IMPLIED
953   03B9  85 0C      STA ZER PAG
955   03BB  18         CLC IMPLIED
956   03BC  68         PLA IMPLIED
 A9 00 00 338 100000 0000 00
 AC  X  Y  SP NZCIDV  EA  EI
957   03BD  69 02      ADC IMMED.
 AB 00 00 338 100000 0000 02
 AC  X  Y  SP NZCIDV  EA  EI
959   03BF  85 01      STA ZER PAG
 AB 00 00 338 100000 0001 30
 AC  X  Y  SP NZCIDV  EA  EI
961   03C1  68         PLA IMPLIED
 03 00 00 339 000000 0000 00
 AC  X  Y  SP NZCIDV  EA  EI
962   03C2  69 00      ADC IMMED.
 03 00 00 339 000000 0000 00
 AC  X  Y  SP NZCIDV  EA  EI
964   03C4  85 02      STA ZER PAG
966   03C6  48         PHA IMPLIED
967   03C7  A5 01      LDA ZER PAG
969   03C9  48         PHA IMPLIED
970   03CA  AD 03 02   LDA ABSOLUT
973   03CD  C9 04      CMP IMMED.
975   03CF  B0 03      BNE REL 03D4
980   03D4  A5 0C      LDA ZER PAG
982   03D6  48         PHA IMPLIED
983   03D7  A5 00      LDA ZER PAG
985   03D9  28         PLP IMPLIED
986   03DA  60         RTS IMPLIED
 00 00 00 339 000000 0000 00
 AC  X  Y  SP NZCIDV  EA  EI
940   03AC  20 DB 03   JSR ABSOLUT
987   03DB  08         PHP IMPLIED
988   03DC  48         PHA IMPLIED
989   03DD  98         TYA IMPLIED
990   03DE  48         PHA IMPLIED
991   03DF  A5 01      LDA ZER PAG
993   03E1  D0 02      BNE REL 03E5
997   03E5  C6 01      DEC ZER PAG
999   03E7  A0 00      LDY IMMED.
1001  03E9  18         CLC IMPLIED
1002  03EA  B1 01      LDA (IND),Y
 FC 00 00 334 100000 03AA FC
 AC  X  Y  SP NZCIDV  EA  EI
1004  03EC  65 01      ADC ZER PAG
 A6 00 00 334 101000 0001 AA
 AC  X  Y  SP NZCIDV  EA  EI
1006  03EE  48         PHA IMPLIED
 A6 00 00 333 101000 0000 00
 AC  X  Y  SP NZCIDV  EA  EI
1007  03EF  C8         INY IMPLIED
 A6 00 01 333 001000 0000 00
 AC  X  Y  SP NZCIDV  EA  EI
1008  03F0  B1 01      LDA (IND),Y
 FF 00 01 333 101000 03AB FF
 AC  X  Y  SP NZCIDV  EA  EI
1010  03F2  65 02      ADC ZER PAG
 03 00 01 333 001000 0002 03
 AC  X  Y  SP NZCIDV  EA  EI
1012  03F4  85 02      STA ZER PAG
 03 00 01 333 001000 0002 03
 AC  X  Y  SP NZCIDV  EA  EI
1014  03F6  68         PLA IMPLIED
1015  03F7  85 01      STA ZER PAG
1017  03F9  68         PLA IMPLIED
1018  03FA  A8         TAY IMPLIED
1019  03FB  68         PLA IMPLIED
1020  03FC  28         PLP IMPLIED
1021  03FD  6C 01 00   JMP ABS IND
 00 00 00 337 000000 03A6 00
 AC  X  Y  SP NZCIDV  EA  EI
934   03A6  60         RTS IMPLIED
 00 00 00 339 000000 0000 00
 AC  X  Y  SP NZCIDV  EA  EI
943   03AF  60         RTS IMPLIED
STACK-OVERFLOW 827
```

Bild 5. Trace-Lauf des letzten Beispielprogramms aus Bild 4. Die Meldung „Stack Overflow" am Ende ist eine normale Meldung des verwendeten Monitorprogramms und nicht durch einen Fehler bedingt



Literatur

[1] Franke, E. H.: PRTSTR – Einfache Textausgabe in Assemblerprogrammen. FUNKSCHAU-Sonderheft „Hobbycomputer 2", Franzis-Verlag.

[2] Joepgen, H. G.: 6502 simuliert neue Adressierungsart. FUNKSCHAU-Sonderheft „Programme für Kleincomputer und Taschenrechner", Franzis-Verlag.

[3] MOS Technology Inc.: 6502 Programming Manual (auch von Rockwell und Commodore erhältlich)

[4] Martin, R. und Smode, D.: ROM und RAM in PET und CBM. FUNKSCHAU-Sonderheft „Mikrocomputer-Anwendungen", Franzis-Verlag.