Tips und Tricks zu C128

Um jedem C128-Besitzer die Programmierung in Maschinensprache zu erleichtern, schildern wir hier die ersten Erfahrungen mit diesem Computer. Durch diese Erfahrungen werden Sie schneller mit dem C128 zurechtkommen.

Das »Bankswitching«, also das Umschalten zwischen mehreren sich überlagernden Speicherbereichen, ist bereits vom C64 her bekannt, bei dem auf diese Weise RAM-Bereiche unter dem ROM genutzt werden können. Beim C64 war es jedoch nicht unbedingt erforderlich, über Speicherkonfigurationen und deren Änderung Bescheid zu wissen. Viele Assemblerprogrammierer dürften sich – ebenso wie ich selbst – so weit wie möglich um diese etwas »ominöse« Geschichte herumgedrückt haben.

Die Programmierung des C128 in Maschinensprache ist leider ohne Kenntnis des Bankswitching kaum möglich. Wie Sie bereits wissen, verwendet der Basic-Interpreter Bank 0 für das Basic-Programm und Bank 1 für die Variablenspeicherung. Der Interpreter ist daher gezwungen, ständig zwischen diesen 64 KByte RAM-Bänken hin- und herzuschalten.

Das gleiche Problem stellt sich für den Assemblerprogrammierer. Eine beliebte Anwendung von Assemblerprogrammen sind Unterroutinen, die mit dem Basic zusammenarbeiten sollen, zum Beispiel um besonders zeitkritische Programmteile zu beschleunigen, oder aber, um nicht vorhandene Basic-Befehle zu implementieren. Werden solche Routinen auf dem C64 geschrieben, kann Sie der Programmierer problemlos in einen freien, vom Überschreiben durch Basic geschützten Speicherbereich legen (zum Beispiel $C000 bis $CFFF).

Der C128 erfordert auch bei kleineren Assembler-Routinen einen weitaus höheren Aufwand. So stellt bereits die Frage, in welchen Speicherbereich eine Routine gelegt werden soll, größere Probleme. Es stehen zwar 128 KByte zur Verfügung, jedoch kein größerer Speicherbereich, der vom Basic mit Sicherheit verschont bliebe. Bank 0 steht bis $FFFF für den Basic-Text, und Bank 1 ebenfalls bis $FFFF für Variablen zur Verfügung. Weiterhin muß in vielen Fällen eine Assembler-Routine auf den Basic-Text und auch auf die Variablen zugreifen, was ohne Umschaltung der Konfiguration nicht möglich ist. Zum Beispiel muß eine Sortierroutine Parameter aus dem Basic-Text lesen (Name des zu sortierenden Arrays), der eigentliche Sortiervorgang findet anschließend in Bank 1, der Variablenbank, statt.

Die ersten Gehversuche

Da ich zusammen mit einem Programmiererkollegen kürzlich vor der Aufgabe stand, ein Basic-Programm, das verschiedene Assembler-Routinen verwendet, vom C64 auf den C128 umzuschreiben, waren wir gezwungen, uns mit diesen – für uns völlig ungewohnten – Problemen auseinanderzusetzen. In diesem Artikel werde ich beschreiben, welche Lösungswege wir sahen, wieder verwarfen, und wie es uns nach unzähligen Fehlschlägen tatsächlich gelang, alle benötigten Routinen umzuschreiben. Ich bin der Ansicht, daß unsere Vorgehensweise typisch war und dieser Artikel es vielen C128-»Neulingen« erspart, unsere zum Großteil frustrierenden Erfahrungen nachzuvollziehen.

Begonnen hatte alles mit einem C128, dem Handbuch und einem Exemplar des »C128 Intern«. Unser erster Schritt war ein Testprogramm, mit dem das Umschalten zwischen den Speicherbänken erprobt werden sollte. Das Programm sollte eine Speicherstelle in Bank 1 lesen. Wir gaben es mit dem integrierten Monitor in Bank 0 ein, und legten es – wie vom C64 gewohnt – nach $C000. Nach dem Aufruf sollte es durch Beschreiben des Konfigurationsregisters $FF00 auf Bank 1 umschalten, die gewünschte Speicherstelle in den Akku einlesen und mit einem BRK in den Monitor zurückkehren.

Nachdem wir feststellen mußten, daß dieses einfache Testprogramm abstürzte, war klar, daß uns C64-Programmierern gewaltige Umstellungen bevorstanden. Der Grund für den Absturz war schnell gefunden: nach dem Beschreiben von $FF00 und damit dem Umschalten auf Bank 1 folgte der Befehl »LDA $1000«, der den Inhalt dieser Speicherstelle lesen sollte. Da unser Programm jedoch in Bank 0 lag, sägten wir uns mit dem Umschalten auf Bank 1 gewissermaßen »den eigenen Ast ab«, da unser Programm – und damit auch der Ladebefehl – nach dem Umschalten auf Bank 1 für den Prozessor nicht mehr existent war.

Dieser Mißerfolg machte uns immerhin klar, daß wir zu naiv an den neuen Computer herangegangen waren. Wir hatten keine Chance, ohne nähere Kenntnis des C128, unsere Routinen umzuschreiben. Nachdem wir einige Zeit damit verbrachten, das Handbuch und das C128 Intern zu studieren, war klar, daß es prinzipiell zwei Methoden gibt, auf verschiedene Speicherbänke zuzugreifen, ohne daß ein Programm unmittelbar nach dem Umschalten abstürzt:

1. Benutzung der Routinen FETCH, STASH und CMPARE:

Diese Routinen benutzt auch der Basic-Interpreter. Nach dem Einschalten des Computers ist der Bereich $0000 bis $03FF als gemeinsamer Bereich von Bank 0 und Bank 1 definiert. Unabhängig davon, welche Bank eingeschaltet ist, »sieht« der Prozessor immer die gleichen Programme beziehungsweise Daten in diesem Bereich. Dieses Konzept der gemeinsamen Speicherbereiche kann man sich durch die Vorstellung veranschaulichen, daß jede Änderung des Bereichs $0000 bis $03FF in der einen Bank sofort in die andere Bank kopiert wird, die Inhalte daher in jedem Moment identisch sind. Dieses Konzept entspricht zwar nicht dem tatsächlichen physischen Ablauf, ist jedoch eine für die Praxis völlig ausreichende Vorstellung.

Die genannten Routinen werden nach dem Einschalten des Compu-

ters aus dem ROM in diesen gemeinsamen Bereich kopiert. Ihre Benutzung erfordert eine recht langwierige Parameterübergabe. Unter anderem wird die gewünschte Speicherkonfiguration übergeben, auf die zugegriffen werden soll. Nach der Parameterübergabe und dem Aufruf schaltet die jeweilige Routine die benötigte Konfiguration ein, führt in der ausgewählten Bank die gewünschte Operation aus (Lesen, Schreiben, Vergleichen) und schaltet danach auf die alte Speicherkonfiguration zurück.

Der Einsprung in eine dieser Routinen findet in das Kernel statt. Danach wird jedoch in die Kopie, also in den gemeinsamen Speicherbereich verzweigt. Da das Umschalten in diesem gemeinsamen Bereich durchgeführt wird, findet kein Absturz statt, da die Routine für den Prozessor in jeder Bank existent ist. Diese Routinen sind daher das »Verbindungsglied« zwischen Programmen und Daten, die in verschiedenen Bänken liegen.

Der gemeinsame Speicherbereich

Ich erspare es mir, diese Routinen mit allen übergebenen und rückübergebenen Parametern zu beschreiben, da inzwischen mehrere Bücher kompetenterer Programmierer auf dem Markt sind, die das Innenleben des C128 detailliert beschreiben.

2. Assembler-Routinen in einen gemeinsamen Speicherbereich legen:

Wie erwähnt, wird beim Einschalten des C128 ein bestimmter Speicherbereich als gemeinsam definiert. Über das Register $D506, das im I/O-Bereich liegt, kann der Programmierer jedoch selbst bestimmen, ob er gemeinsame Bereiche definieren will, wo und wie groß diese sein sollen.

Es existieren jedoch mehrere Begrenzungen, zum Beispiel dadurch, daß ein gemeinsamer Bereich an einem Ende des Speichers beginnen muß (entweder ab $0000 aufwärts oder ab $FFFF abwärts) und durch die maximale Größe eines solchen Bereichs von 32 KByte.

Die zweite Möglichkeit besteht somit darin, einen Speicherbereich als gemeinsam zu deklarieren, der groß genug ist, alle benötigten Assembler-Routinen aufzunehmen.

Da die Routinen FETCH etc. langwierige Parameterübergaben erfordern, ein Programm daher länger wird und zusätzliche Fehlerquellen entstehen, waren mein Kollege und ich uns einig, die zweite Möglichkeit zu verwenden. Die Frage war nun, welcher Speicherbereich als gemeinsam deklariert und für die Routinen verwendet werden sollte. Wir entschieden uns dafür, den oberen Speicherbereich bis $FFFF zu verwenden. Voraussetzung dafür war natürlich, den Zeiger auf das Ende des Basic-Textes herabzusetzen, um die Routinen vor dem Überschreiben zu schützen. Zusätzlich war es erforderlich, die Zeiger auf den Anfang des Stringbereichs ebenfalls herabzusetzen. Der Stringstack beginnt ab $FFFF in Bank 1. Wenn nun zum Beispiel der Speicherbereich $F000 bis $FFFF als gemeinsamer Bereich definiert wird, führt jedes Anlegen eines Strings dazu, daß die Assembler-Routinen in dem entsprechenden Bereich von Bank 0 durch den String überschrieben würden.

Nach den frustrierenden Erfahrungen mit dem erstem Testprogramm versuchten wir nun, möglichst alle Konsequenzen unseres Plans zu überdenken. Dabei stellte sich ein neues Problem: Im gleichen Bereich befindet sich das Kernel, das zum Zugriff auf unsere Routinen ausgeschaltet werden mußte. Diese Routinen benötigten jedoch selbst mehrere Kernel-Routinen.

Zur Benutzung von ROM-Routinen trotz ausgeblendetem ROM stehen die Routinen JRSFAR und JMPFAR zur Verfügung. Bei der Benutzung dieser Routinen muß das Kernel-ROM eingeschaltet sein. Die Routinen selbst befinden sich zwar nicht im Kernel, verzweigen jedoch zu einer Kernel-Routine, was bei ausgeblendetem Kernel natürlich zu einem Absturz führen würde.

JSRFAR stellt daher leider für Programme, die sich unter dem Kernel befinden und dessen Routinen verwenden wollen, keine brauchbare Lösung dar. Denkbar ist jedoch ein Umweg, mit dem das Ziel gewissermaßen über mehrere Ecken angesteuert wird:

Zwei gemeinsame Bereiche werden definiert, einer am unteren und einer am oberen Ende des Speichers. In den unteren Bereich, der nicht vom ROM überlagert wird, wird eine kleine Routine gelegt, die den Aufruf einer vom Hauptprogramm benötigten Kernel-Routine übernimmt. Das Hauptprogramm übergibt alle Parameter (zum Beispiel die Adresse der Kernel-Routine) und ruft die Routine im unteren Bereich auf. Diese blendet das Kernel ein und dadurch zwangsläufig das Hauptprogramm aus. Nun wird JSRFAR aufgerufen. Nach Durchführung der jeweiligen Kernel-Routine wird mit einem RTS zu der Routine im unteren gemeinsamen Bereich zurückgekehrt. Diese blendet nun das Kernel wieder aus und das darunterliegende Hauptprogramm ein; anschließend erfolgt die Rückkehr in dieses Hauptprogramm.

Wie Sie sehen – und auch wir einsehen mußten – ist es ohne größeren Aufwand nicht möglich, ein Assemblerprogramm, das Kernel-Routinen verwendet, unter eben dieses Kernel zu legen. Die Umschaltung zwischen Kernel und darunterliegendem Programm führt zu sehr umständlichen Routinen.

Die einzige von uns gefundene Möglichkeit, kleinere Assembler-Routinen auf dem C128 zu schreiben, ohne an derartigen Problemen zu scheitern, besteht darin, die Programme in einen Bereich zu legen, der nicht vom ROM - das eventuell benötigt wird - überlagert und als gemeinsamer Bereich von Bank 0 und Bank 1 deklariert wird.

Bankswitching

Ein solcher Bereich, der für die meisten Assembler-Routinen ausreichend Platz bietet und unbenutzt ist, ist der Bereich von $1300 bis $17FF. Um Probleme mit dem Umschalten zwischen den Bänken zu vermeiden, wird ein gemeinsamer Bereich definiert, der von $0000 bis $1FFF reicht. Da der Basic-Text jedoch ab $1C00 beginnt, muß der Zeiger auf den Basic-Anfang auf $2000 verstellt werden, um Überschneidungen zwischen dem Basic-Text in Bank 0 und den Variablen in Bank 1 zu vermeiden.

Zusätzlich muß auch der Zeiger auf den Beginn der Variablentabelle verstellt werden – auch auf $2000 –, da diese ab $0800 beginnt und ebenfalls ab $0800, jedoch in Bank 0, der Bildschirmspeicher liegt. Überschneidungen zwischen Variablentabelle und Bildschirmspeicher führen sonst bei jeder Ausgabe auf dem Bildschirm zu Änderungen der Variablentabelle, also zu ihrer Zerstörung.

Ich möchte Ihnen nun zwei kleine Initialisierungs-Routinen vorstellen, die die gewünschten Aufgaben erledigen, eine Basic- und eine

Assembler-Routine:
1. Beginn des Basic-Textes und der Variablentabelle verschieben:

```
10 poke 46,dec("20"):poke 48,dec
("20")
20 poke dec("2000"),0
30 clr
40 new
```

Wenn Sie dieses Basic-Programm eingeben und starten, beginnt sowohl der Basic-Text als auch die Variablentabelle ab $2000.
2. Definition eines gemeinsamen Speicherbereiches:

```
a 00b00  ad  00 ff  lda $ff00
a 00b03  48         pha
a 00b04  a9  00     lda #$00
a 00b06  8d  00 ff  sta $ff00
a 00b09  ad  06 d5  lda $d506
a 00b0c  09  06     ora #$06
a 00b0e  8d  06 d5  sta $d506
a 00b11  68         pla
a 00b12  8d  00 ff  sta $ff00
a 00b15  60         rts
```

Dieses kleine Maschinenprogramm rettet zuerst die aktuelle Speicherkonfiguration, blendet nun den I/O-Bereich ein und definiert im RAM-Konfigurationsregister den Bereich $0000 bis $1FFF als gemeinsamen Bereich von Bank 0 und Bank 1. Vor dem Rücksprung aus der Routine wird die gerettete Speicherkonfiguration wieder hergestellt. Diese Routine liegt im Kassettenpuffer. Sollten Sie mit der Datasette arbeiten, legen Sie sie bitte nach $0D00, in den RS232-Eingabepuffer.

Wenn Sie beide Programme eingegeben und gespeichert haben, können Sie mit der Assemblerprogrammierung beginnen. Bis Sie selbst eine komfortablere Lösung gefunden haben, gehen Sie bei der Assemblerprogrammierung bitte wie folgt vor:
1. Basic-Programm laden und starten.
2. Maschinenprogramm laden und mit »SYS DEC("0B00")« aufrufen.

Für Ihre Maschinenprogramme steht Ihnen nun der Bereich $1300 bis $17FF (ab $1800 beginnt der für die Funktionstastenbelegung benötigte Bereich) zur Verfügung. Eventuell können Sie auch weitere Bereiche nutzen, zum Beispiel $0C00 bis $0DFF, wenn Sie die RS232-Schnittstelle, oder den Bereich von $0B00 bis $0BFF, wenn Sie den Kassettenpuffer nicht benötigen.

Die Nutzung aller Bereiche unterhalb $2000 ist theoretisch möglich. Probleme durch das »Absägen des eigenen Astes« werden dank dem bis $1FFF reichenden, gemeinsamen Speicherbereich nicht auftreten.

Um wirklich problemlos in Assembler arbeiten zu können, benötigen Sie zwei weitere Routinen, die zwischen Bank 0 und Bank 1 umschalten. Diese Routinen wurden von mir in das Initialisierungs-Programm integriert.
Initialisierung + Umschaltroutinen:

```
a 00b00  ad  00 ff  lda $ff00
a 00b03  48         pha
a 00b04  a9  00     lda #$00
a 00b06  8d  00 ff  sta $ff00
a 00b09  ad  06 d5  lda $d506
a 00b0c  09  06     ora #$06
a 00b0e  8d  06 d5  sta $d506
a 00b11  68         pla
a 00b12  8d  00 ff  sta $ff00
a 00b15  60         rts
a 00b16  08         php
a 00b17  48         pha
a 00b18  a9  00     lda #$00
a 00b1a  8d  00 ff  sta $ff00
a 00b1d  68         pla
a 00b1e  28         plp
a 00b1f  60         rts
a 00b20  08         php
a 00b21  48         pha
a 00b22  a9  7f     lda #$7f
a 00b24  8d  00 ff  sta $ff00
a 00b27  68         pla
a 00b28  28         plp
a 00b29  60         rts
```

$0B16 ist die Einsprungadresse zum Umschalten auf Bank 0, $0B20 der Einsprung zum Umschalten auf Bank 1 (nur RAM), die wichtigere der beiden Routinen. Wichtiger, da Sie Bank 0 wohl nur zum Lesen von Parametern aus dem Basic-Text verwenden werden, wozu problemlos die entsprechenden Routinen des Basic-Interpreters verwendet werden können.

Assemblerprogrammierung

Im Laufe der Zeit werden sicherlich auch elegantere Lösungen der genannten Probleme entdeckt werden. Dieser Artikel sollte nur zeigen, worauf Sie bei der Assemblerprogrammierung des C128 achten müssen:
1. Probleme beim Umschalten zwischen den Speicherbänken.
2. Probleme mit Überschneidungen zwischen als gemeinsam definierten Speicherbereichen, zum Beispiel Bildschirm und Variablentabelle.
3. Probleme bei dem Zugriff auf Betriebssystem-Routinen, deren Lösung davon abhängt, in welchen Bereich die Programme gelegt werden, unter das ROM oder in einen reinen RAM-Bereich.

Wie Sie sehen, ist es bei der Programmierung des C128 in Assembler außerordentlich wichtig, alle Zugriffe, die im späteren Programm sowohl auf verschiedene RAM-Bänke als auch auf ROM-Routinen erfolgen sollen, von vornherein einzuplanen. Ich möchte C128 Besitzer keinesfalls frustrieren, aber meiner Ansicht nach ist dieser Computer bei der Programmierung in Maschinensprache außerordentlich unkomfortabel und stellt weit höhere Anforderungen an den Programmierer als zum Beispiel der C64.

Bildschirmausgabe

Zum Abschluß noch ein Rat: Sie werden sich vor weitere Probleme gestellt sehen, wenn Sie Programme, die mit einer bestimmten Art der Bildschirmdarstellung arbeiten, zum Beispiel dem 40-Zeichenmodus, auf einen anderen Ausgabemodus umstellen wollen.

Der Grund ist der unterschiedliche Videocontroller. Für den 40-Zeichenmodus ist der VIC, für den 80-Zeichenmodus der VDC zuständig. Beide Controller arbeiten unterschiedlich. Während der VIC den direkten Zugriff auf das Video-RAM gestattet, ist beim VDC nur ein indirekter Zugriff möglich.

Beim C64 ist es üblich, daß Assembler-Routinen direkt – unter Umgehung der Betriebssystem-Routinen – in das Video-RAM schreiben beziehungsweise daraus lesen. Bei einer eventuellen Umstellung der Zeichendarstellung sind in solchen Routinen umfangreiche Änderungen nötig.

Ich empfehle Ihnen daher, ausschließlich die Betriebssystem-Routinen - zum Beispiel BSOUT - zu verwenden. Wenn diese standardisierten Schnittstellen verwendet werden, können Sie sicher sein, daß eine spätere Programmumstellung auf eine andere Art der Bildschirmdarstellung nicht einem Neuschreiben des Programms gleichkommt.

Daß mit BSOUT Zeichen auf dem Bildschirm ausgegeben werden können, dürfte allgemein bekannt sein. Woran jedoch nicht jeder Programmierer denkt, ist die Möglichkeit, ohne Umgehung des Betriebssystems auch Zeichen vom Bildschirm zu lesen, indem dieser als logische Datei eröffnet und die Eingabe auf diese Datei gelegt wird.

Zweifellos eine umständliche Methode, die jedoch bei späteren Programmänderungen viel Ärger ersparen kann.

(S. Baloui / ah)