Detlef Hartmann

Der 68008 wird entwanzt

Tips zum NDR-Klein-Computer

Der Autor beschäftigt sich seit Juli 1984 mit dem „großen" NDR-Klein-Computer, d.h. dem System mit dem Prozessor 68008. Im folgenden werden einige Fehler im 68008-Grundprogramm (Version 3.1) und deren Behebung besprochen.

Zuerst ein Wort zu den Adressenangaben: Rolf-Dieter Klein hat das Grundprogramm auf einem Z80-System mit einem Cross-Assembler übersetzt. Dabei setzt sich das gesamte Grundprogramm aus drei Teilen (GRUND, ASSEM68 und EDIT68) zusammen, die einzeln übersetzt und später mit einem Linker auf ihre endgültigen Adressen umgesetzt wurden. Das Listing „68008-Grundprogramme" vom Franzis-Software-Service (FSS) enthält die Adressen auf das jeweilige Teilmodul bezogen. Die entsprechenden Offsets, die zu den Listing-Adressen addiert werden müssen, um zu den wahren EPROM-Adressen zu gelangen, findet man auf den letzten Seiten. Sie betragen für GRUND: $0400, für ASSEM68: $468A und für EDIT68: $6918. Das Programm-Listing dieses Artikels bezieht sich auf die wahren EPROM-Adressen.

```
= 0000B052              PCSTAND   EQU   $8052
= 000081DE              MODBER    EQU   $81DE
= 000081F6              BEFCODE   EQU   $81F6
= 000081FA              WORDBYTE  EQU   $81FA
009C00
= 00000400              GROFFSET  EQU    $400  ;OFFSET GRUNDPROGRAMM LINKER
= 0000468A              ASOFFSET  EQU   $468A ;OFFSET ASSEMBLER LINKER
= 0000468A              EXPR1     EQU   $0+ASOFFSET
= 00004B38              PUTWORD   EQU   $1AE+ASOFFSET
= 00005BDC              PUTCAR    EQU   $1552+ASOFFSET
= 0000666A              RANGE07   EQU   $1FE0+ASOFFSET
= 000066B6              RANGEB1CK EQU   $1FFC+ASOFFSET
= 000066D2              RANGEW1CK EQU   $2048+ASOFFSET
= 0000673E              ERRCBER   EQU   $20B4+ASOFFSET

                                  ORG     $19E4+GROFFSET
  001DE4
  001DE4  E202                    ASR.B   #1,D2 ;ETWAS UMSTAENDLICH,
  001DE6  4282                    CLR.L   D2    ;SOLLTE JEDOCH NICHT
  001DE8  D182                    ADDX.L  D2,D0 ;MEHR PLATZ VERBRAUCHEN
```

Bild 1. Definition nötiger Symbole und Korrektur zur richtigen Umwandlung binärer Konstanten

```
002688                          ORG     $2288+GROFFSET
002688
002688  6100 5B54               BSR     MODBER
00268C  40D7                    MOVE    SR,(A7) ;KORREKTUR FUER JSR XXX.L
00268E
00268E
00268E
0026B6                          ORG     $22B6+GROFFSET
0026B6
0026B6  6100 5B26               BSR     MODBER
0026BA  40D7                    MOVE    SR,(A7) ;KORREKTUR TRAP
```

Bild 2. Korrektur zur richtigen Übernahme des Status-Registers bei Einzelschritt-Betrieb

Binäre Konstanten

Der erste Fehler erscheint zunächst harmlos, kann jedoch auch schwerwiegende Folgefehler erzeugen: Symbole, die binär (%) definiert wurden, konnten nur wortgroß richtig definiert werden. Die höchsten 16 Bit enthielten stets unsinnige Daten. Die Fehlersuche führte schnell an die entsprechende Stelle im Grundprogramm, an der %-Konstanten gewandelt werden. Dies geschieht beim Label „FAKT320", das man bei Adresse $0019E2 auf Seite 44 des FSS-Listings findet. Bei $0019E4 wird das niedrigste Bit von Datenregister D2 maskiert, d.h., alle anderen Bits werden auf Null gesetzt. Leider hat R.-D. Klein an dieser Stelle nur „AND.W" statt „AND.L" verwendet. Die Folge ist, daß die vordersten 16 Bit von D2 unverändert erhalten bleiben und beim folgenden Addierbefehl mitaddiert werden. Abhilfe wäre hier also ein „AND.L #$1,D2". Leider braucht dieser Befehl mehr Speicherplatz. Um die Korrektur dennoch an derselben Speicherstelle durchführen zu können, wurde ein etwas ungewöhnlicher Weg eingeschlagen, wie *Bild 1* zeigt.

Einzelschritt

Der nächste Fehler trat sehr störend beim Austesten von Programmen im Einzelschritt-Betrieb in Erscheinung. Eine schöne Eigenschaft der Einzelschritt-Verarbeitung ist der schnelle Durchlauf (ohne Trace) von Programmteilen, die durch „JSR XXX.L" oder „TRAP #X" aufgerufen werden. Solche Programmteile verändern sicher das Status-Register bzw. geben Informationen über das Status-Register zurück (z.B. Carry-Flag gesetzt/gelöscht). Durch einen Programmfehler im Trace-Programm wird jedoch das Status-Register nicht aktualisiert, d.h. nach Durchlaufen von Programmteilen durch „JSR XXX.L" oder „TRAP #X" hat das Status-Register immer noch denselben Inhalt wie vor Ausführung dieser Routinen. Dies hat nun zur Folge, daß man ein Programm nicht korrekt im Einzelschritt-Betrieb testen kann.

Die Korrektur dieses Fehlers besteht darin, daß der neue Inhalt des Status-Registers auf den Stack gebracht werden muß, wo ihn die Register-Anzeigeroutine erwartet (dort ist er automatisch jedoch nur nach einem Trace-Trap, weil der Prozessor bei einem Trap das Status-Register und den Programmzähler auf den Stack rettet). Die Korrektur kann im Detail dem *Bild 2* entnommen werden.

Speicher auf Kassette

Ein weiterer (harmloser) Fehler führt dazu, daß man auf Kassette nicht mehr als 64 KByte Daten speichern kann. Ein Versuch, dies zu tun, führt zu einer Endlosschleife, aus der man sich nur noch mit einem Reset befreien kann. Die Korrektur erreicht man mit demselben Platzbedarf durch eine Long-Addition anstelle einer Word-Addition (*Bild 3*).
Soweit zu den Fehlern im ersten Modul GRUND. Auch im zweiten Modul ASSEM68 finden sich einige kleine Fehler.

```
003032                          ORG     $2C32+GROFFSET
003032
003032  5288                    ADDQ.L  #1,A0 ;LONG ADDITION
```

Bild 3. So können auch mehr als 64 KByte auf Kassette abgespeichert werden

Aus 8 mach 0

Zunächst zur Befehlsfamilie „ADDQ #...“ und „SUBQ #...“. Als Direkt-Operand ist hier laut Prozessor-Beschreibung ein logischer Wertebereich von 1 bis 8 zugelassen, wobei der Prozessor im Opcode die 8 als 0 dargestellt haben möchte. Diese Umwandlung könnte eigentlich der Assembler ausführen, denn welcher Programmierer denkt schon beim Lesen des Befehls „ADDQ #0,D1“ daran, daß hier eine 8 zum Datenregister D1 addiert werden soll? Diese Umrechnung wird bei den Rotier- und Schiebebefehlen vom Assembler automatisch durchgeführt. Die entsprechende Korrektur für die anderen Befehle findet man in *Bild 4*. Ein Haken soll hier nicht verschwiegen werden: Durch diese Änderung sind Source-Texte nicht mehr kompatibel! Da beide Versionen jedoch bei der jeweils anderen Schreibart eine Fehlermeldung erzeugen, können entsprechende Stellen leicht gefunden und korrigiert werden.

```
00594E                          ORG     $12C4+ASOFFSET
00594E
00594E  2A1F                    MOVE.L  (A7)+,D5        ;REG. D5 FREI
005950  2F00                    MOVE.L  D0,-(A7)        ;RETTEN
005952  2005                    MOVE.L  D5,D0
005954  5380                    SUBQ.L  #1,D0           ;1...8 -> 0...7
005956  6100 0D12               BSR     RANGE07         ;TEST BEREICH 0...7
00595A  4E71                    NOP
00595C  6100 0DE0               BSR     ERRCBER
005960  0205 0007               AND.B   #7,D5           ;1...7,8 -> 1...7,0
005964  201F                    MOVE.L  (A7)+,D0        ;RESTORE D0
005966  EE5D                    ROR     #7,D5
```

Bild 4. Bei ADDQ und SUBQ ist jetzt der Wertebereich 1...8 erlaubt; die Zahl 8 wird automatisch in Null umgesetzt

Fehlermeldung fehlt

Ein recht schwerwiegender Fehler im ASSEM68 ist das Fehlen einer Fehlermeldung, wenn nichtlauffähiger Code erzeugt wird. Als Beispiel soll ein Unterprogramm geschrieben werden, das ein oder zwei Leerzeichen ausgibt. Zweckmäßigerweise sieht ein solches Programm so aus:

```
PRTZWEI: BSR PRTEIN
PRTEIN: MOVE #' ',D0
BRA CO2
```

Dieses Programm funktioniert. Ersetzt man nun das „BSR PRTEIN“ durch „BSR.S PRTEIN“, da dieses zwei Byte spart, so übersetzt der Assembler dieses anstandslos, das erzeugte Objektprogramm läuft jedoch nicht mehr. Der Grund liegt darin, daß als relative Sprungweite Null im Code erzeugt wird. Der Prozessor „denkt“ jedoch bei

```
005B54                          ORG     $14CA+ASOFFSET
005B54
005B54  0C79 0001       CODBRA1: CMP    #1,WORDBYTE
005B58  000081FA
005B5C  672C                    BEQ.S   CODBRA1          ;LONG BRANCH
005B5E
005B5E  6100 EB2A               BSR     EXPR1
005B62  90B9 00008052           SUB.L   PCSTAND,D0
005B68  5580                    SUBQ.L  #2,D0
005B6A  6100 0B1A               BSR     RANGEB1CK
005B6E  6100 0BCE               BSR     ERRCBER
005B72  0240 00FF               AND.W   #$00FF,D0
005B76  660A                    BNE.S   CODBRA0         ;OK, WIE BISHER WEITER
005B78  5300                    SUBQ.B  #1,D0           ;D0.W:=$FF
005B7A  6100 0AEE               BSR     RANGE07         ;ERZ. BEI PASS 2 CARSET
005B7E  6100 0BBE               BSR     ERRCBER         ;FEHLERMELD. BEI CARSET
005B82  8079 0000B1F6   CODBRA0:OR      BEFCODE,D0
005B88  6052                    BRA.S   PUTCAR          ;=BSR PUTWORD,BRA CARRES
005B8A
005B8A  3039 0000B1F6   CODBRA1:MOVE    BEFCODE,D0
005B90  6100 ECA6               BSR     PUTWORD
005B94  6100 EAF4               BSR     EXPR1
005B98  90B9 00008052           SUB.L   PCSTAND,D0
005B9E  6100 0B32               BSR     RANGEW1CK
005BA2  6100 0B9A               BSR     ERRCBER
005BA6  6034                    BRA.S   PUTCAR
```

Bild 5. Bei Short-Branch-Befehlen wird nun bei der Sprungweite Null eine Fehlermeldung ausgegeben

```
0064AA                          ORG     $1E20+ASOFFSET
0064AA
0064AA  6100 01DA               BSR     RANGEB1CK
0064AE
0064AE                          END

0000    Fehler entdeckt
008B2A  Ende-Symboltabelle
```

Bild 6. Der Wertebereich bei MOVEQ wird jetzt auf −128 bis +127 begrenzt

Sprungweite Null an einen langen Branch-Befehl und holt sich die nächsten zwei Bytes als Sprungweite. Dadurch wird der Befehl „MOVE #' ',D0“ als Sprungweite interpretiert! Diese problematische Übersetzung sollte der Assembler vorher melden. Durch eine entsprechende Korrektur meldet der Assembler in diesem Fall jetzt „Wertebereich falsch“ (*Bild 5*).

MOVEQ mit Vorzeichen

Ein weiterer kleiner Schönheitsfehler ist die Zulassung des Wertebereichs −256 bis +255 beim Befehl „MOVEQ #“. Der Befehl „MOVEQ #...“ wirkt nämlich stets sign-extended, d. h., vorzeichenerweiternd. Daher ist (um unfreiwillige, schwer zu findende Fehler zu vermeiden) nur ein Wertebereich von −128 bis +127 sinnvoll. Durch die Änderung in *Bild 6* verhält sich der Assembler nun so, daß er außerhalb des Bereichs von −128 bis +127 die Fehlermeldung „Wertebereich falsch“ erzeugt.

In einem der nächsten Hefte werden einige nützliche Erweiterungen des 68008-Grundprogramms geschildert.