soll, an der sie zuletzt unterbrochen wurde (bzw. ab der Adresse, die zuletzt als Argument 1 angegeben wurde). Jedesmal, wenn der Datenstring gefunden wird, legt ZETBUG die Startadresse des Strings in der Memory-map ab, und zwar in die Speicherzellen, in denen das Argument 1 steht.

Auf diese Weise kann auf jeden String im Speicher zugegriffen werden, auch wenn dieser an mehreren Stellen gleichzeitig vorkommt.

 # D aaaa bbbb cccc name
(dump data to cassette)

Hiermit lassen sich Datenblöcke (files) auf Kassette aufzeichnen; aaaa ist die Anfangsadresse, bbbb ist die Endadresse, cccc ist der Entrypoint (siehe auch LEVEL II-System-Kommando); name ist der Filename (maximal 6 Zeichen). Aufzeichnungsformat (ZETBUG-Tapes können direkt mit dem Level-II-System-Kommando gelesen werden):

- 255 Byte Headercode
- Synchronisationsbyte (A5)
- filename Headercode (55)
- 6 Byte langer file-name
- Datenblöcke: – data Headercode (3C)
  - Anzahl der Datenbytes in diesem Block (00-FF) (00=256x)
  - Blockstartadresse
  - (LSB/MSB)
  - Daten
  - Checksumbyte
- Entrypoint Headercode (78)
- Entrypoint (LSB/MSB)

 # L aaaa±
(Load data from cassette into memory)

Liest einen File von der Kassette in den Speicher. aaaa± gibt das Displacement an, um das der File versetzt geladen werden soll (kann entfallen).

L 1200+ bedeutet, daß der File auf eine um 1200 höhere Adresse als ursprünglich geladen werden soll.

 # P aaaa bbbb
(Print data)

Druckt den Speicherbereich von aaaa bis bbbb auf dem Lineprinter aus. (Das Format entspricht dem des T-Kommandos.)

 # E aaaa
(Execute machine program)

Führt das Maschinenprogramm ab der Startadresse aaaa aus. Zuerst wird der Breakpoint (siehe B) auf die durch das B-Kommando festgesetzte Adresse gelegt, dann werden sämtliche CPU-Register mit den Daten aus der RSA (siehe auch I) geladen und zur Programmadresse aaaa gesprungen.

 # J aaaa
(Jump machine program)

Wie das E-Kommando, jedoch wird kein Breakpoint gesetzt.

 # G
(Go on / continue execution at PC)

Durch Eingabe von G wird die Programmausführung dort fortgesetzt, wo sie zuletzt (z. B. durch einen Breakpoint) unterbrochen wurde. Nachdem die CPU-Register geladen wurden, wird geprüft, ob die PC-Adresse (PC = Programcounter der Z-80-CPU) auf den Breakpoint zeigt (das nämlich ist genau dann der Fall, wenn der Programmstop durch eben diesen Breakpoint verursacht wurde). Ist dem so, dann wird direkt zur PC-Adresse gesprungen, ansonsten wird zuerst der Breakpoint gesetzt und dann zur PC-Adresse gesprungen.

 # B aaaa
(Breakpoint)

Dies ist wohl einer der hilfreichsten Befehle zum Austesten von Maschinenprogrammen. Er bietet die Möglichkeit, an jeder Stelle im Programm einen Software-Haltepunkt (Breakpoint) zu setzen, ohne das Programm verändern zu müssen. Prinzip: An die für den Break ausgewählte Stelle, die immer die Adresse des *ersten* (!) Bytes eines Maschinenbefehls sein *muß*, wird eine 3 Byte lange „CALL BREAK"-Sequenz (CD 98 48) gelegt. Gelangt das Programm bei der Ausführung an diese Stelle, so springt es zu der Breakpointbehandlungsroutine BREAK in den ZETBUG-Monitor. Diese rettet nun die Inhalte der CPU-Register in die RSA, ersetzt im Speicher die Breakpointsequenz durch die ursprüngliche Opcodefolge, druckt die Meldung: BREAK IN aaaa #– und gibt die Kontrolle wieder zurück an den Benutzer (man kann nun z. B. das Kommando R: eingeben, siehe R).

Anzumerken ist noch, daß die Breakpointsequenz wirklich nur dann im Speicher steht, wenn gerade ein durch E oder G gestartetes Maschinenprogramm von der CPU abgearbeitet wird. Dem Programm merkt man also nichts an, während man z. B. mit dem M-Kommando arbeitet.

 # R
(Register display / modify)

Mit diesem Befehl läßt sich direkt auf die Register Save Area (RSA) zugreifen, ohne die Speicheradressen der einzelnen CPU-Register wissen zu müssen. Die Eingabe # R/XX bewirkt, daß der Inhalt des Register(paares) XX angezeigt wird und im Anschluß an das Promptsymbol die neuen Daten (ähnlich wie im Modify-Modus) eingegeben werden können. Das Kommando # R: druckt die gesamte RSA einschließlich der Breakpointsequenz und der Opcodefolge mit Benennung auf dem Bildschirm aus. Außerdem werden die Flagregister interpretiert und die gesetzten Flags abgekürzt angegeben (F' in Klammern). Auf diese Weise unterstützt der R-Befehl wirkungsvoll die Kommandos E, J, G und B. Beispiel:

```
# R:
BP:534D  BS:212040
SP:4FFE  PC:534D  IY:6CDD  IX:3000
MAIN  HL:21DB  DE:8F53  BC:4433
  AF:3EAA
EXXR  HL:1277  DE:9CCC  BC:E841
  AF:5503
FLAGS:  SN  (NC)
#
#  R/DE'
9CCC # 8888
#  R/DE'
8888
#
```

```
100 DEFINT A-Z
110 PRINT:PRINT"ADRESSE:";:GOSUB1000:Z=D*256:GOSUB1000:Z=Z+D
200 PRINT:D=Z/256:GOSUB1200:D=Z-D*256:GOSUB1200:PRINT CHR$(32);
210 D=PEEK(Z+512):GOSUB1200:PRINT CHR$(32);:GOSUB1000:IF X=33 THEN200
220 IF X=23 THEN280 ELSE IF X=22 THEN100 ELSE IF X=29 END
230 POKE(Z+512),D
280 Z=Z+1:GOTO 200
1000 GOSUB1020:IFX>16 RETURN ELSE D=X*16:GOSUB1020:D=D+X:RETURN
1020 X$=INKEY$:IF X$="" THEN1020 ELSE X=ASC(X$)
1030 PRINT CHR$(X);:IF X<64 THEN X=X-48:RETURN ELSE X=X-55:RETURN
1200 A=D/16:IF A>9 THEN A=A+55 ELSE A=A+48
1210 B=D/16:B=D-B*16:IF B>9 THEN B=B+55 ELSE B=B+48
1220 PRINT CHR$(A);:PRINT CHR$(B);:RETURN
```

Bild 2. Basic-Programm zur Eingabe hexadezimaler Maschinenprogramme. Zeile 100 definiert die verwendeten Variablen als speicherplatzsparende Ganzzahlen. Der INKEY-Befehl entspricht beim PET der GET-Anweisung. ELSE IF X = 29 END müßte beim PET in Zeile 225 als IF X = 29 THEN END stehen