

VPort-50 Monitor Bedienungsanleitung

Hans Rosenfeld

24. Mai 2006

Zusammenfassung

Dieses Dokument beschreibt die Funktionsweise und die Bedienung des VPort-50 Monitors.

Inhaltsverzeichnis

1	Grundlagen	4
1.1	Das Terminal	4
1.2	Der Monitor	4
1.3	Adressen, Zahlen und Strings	5
2	Anwendung des Monitors	6
2.1	Anzeigen und Verändern des Speicherinhalts	6
2.2	Ein- und Ausgabe mit I/O-Ports	6
2.3	Laden und Ausführen von Programmen	9
2.4	Anzeigen und Verändern des Programmzustandes	9
2.5	Breakpoints und Einzelschrittbetrieb	9
2.6	Weitere Möglichkeiten	10
2.7	Monitor-Aufrufe	11
3	Kommandoreferenz	13
3.1	clear	13
3.1.1	clear breakpoint	13
3.1.2	clear current	13
3.1.3	clear registers	13
3.2	continue	14
3.3	disassemble	14
3.4	dump	14
3.5	help	14
3.6	input	14
3.7	move	15
3.8	output	15
3.9	set	15
3.9.1	set breakpoint	15
3.9.2	set memory	16
3.9.3	set memory/intel	16
3.9.4	set registers	17
3.9.5	set trace	17
3.10	show	18
3.10.1	show breakpoints	18
3.10.2	show current	18
3.10.3	show registers	19
3.11	start	19
3.12	reset	19

A	Adressen der VPort-50 Hardware	20
A.1	Speicherbereiche	20
A.2	Peripherie	20
A.2.1	PIO 8255	20
A.2.2	Counter/Timer 8254	20
A.2.3	UART 16450	20
A.2.4	Interrupt Controller 8259A	21

1 Grundlagen

1.1 Das Terminal

Die Kommunikation mit dem Monitor erfolgt prinzipiell durch ein Terminal an einer RS232-Schnittstelle des Rechners. Da echte Terminals inzwischen selten und z.B. zum Laden von Programmen unpraktisch sind, dient als Terminal ein PC mit der Software *Kermit*¹.

Kermit wurde so konfiguriert, daß es nach dem Start sofort als Terminal benutzbar ist. Um in den Kommandomodus umzuschalten, z.B. um eine Hex-Datei zum Monitor zu übertragen oder um DOS-Kommandos auszuführen, verwendet man die Tastenkombination *Alt-X*.

Im Kommandomodus von Kermit stehen u.a. folgende Kommandos zur Verfügung:

exit zum Verlassen von Kermit

connect zum Umschalten in den Terminalmodus

push zum Umschalten zur DOS-Kommandozeile (Rückkehr zu Kermit mit dem DOS-Kommando *EXIT*)

transmit zum Senden einer Datei an den Monitor

Die meisten Kommandos lassen sich abkürzen (z.B. *ex* statt *exit*), ausserdem verfügt Kermit über eine automatische Vervollständigung von Kommando- und Dateinamen (*ESC*-Taste) und Menu-On-Demand (*?*-Taste). Für Informationen über weitere Kommandos gibt es das Kommando *help*.

1.2 Der Monitor

Nach dem Einschalten des Rechners meldet sich der Monitor unter Angabe der Rechnerarchitektur, seiner Versionsnummer sowie Datum und Uhrzeit seiner Übersetzung auf der Console:

VPort-50 Monitor Build 573, 15-Sep-2004 15:17

Der Monitor verwendet den Prompt “\$” um anzuzeigen, daß er eine Kommandoeingabe erwartet. Eine Eingabe darf nicht länger als 78 Zeichen (eine Zeile auf dem Terminal) sein. Bei fehlerhaften Eingaben lässt sich durch die Tasten *Backspace* (bzw. *Ctrl-H*) und *Delete* das letzte Zeichen der Eingabe löschen, *Ctrl-U* löscht die gesamte Zeile. Alle Kommandos, Kommandoschalter und Parameter lassen sich bis auf 2 Zeichen abkürzen, trennende

¹siehe <http://www.columbia.edu/kermit/mskermit.html>

Zwischenräume können aus beliebig vielen Leerzeichen bestehen. Eine Auflistung sämtlicher Kommandos erhält man mit dem Kommando *help*.

Das zuletzt ausgeführte Kommando kann durch eine leere Eingabe wiederholt werden. Ausserdem speichert der Monitor auch noch andere Werte, die im Verlauf der letzten Kommandos verwendet wurden. Für eine genaue Beschreibung dieser sogenannten *Current Values* siehe Abschnitt 3.10.2.

1.3 Adressen, Zahlen und Strings

Die vom Monitor verwendeten und erwarteten Adreß- und Zahlenformate sind architekturabhängig. *Strings* sind immer in Anführungszeichen eingeschlossene ASCII-Zeichenketten, in denen selbst keine Anführungszeichen oder Zeilenumbrüche vorkommen dürfen. Am Ende einer Eingabezeile kann ein den String abschliessendes Anführungszeichen auch weggelassen werden.

Auf dem VPort-50 sind alle Zahlen, für die nicht explizit etwas anderes gesagt ist, grundsätzlich als hexadezimale Zahlen zu verstehen. Ist eine eingegebene Zahl grösser als 16 bzw. 8 Bit, wird nur der niederwertigste Teil verwendet (d.h., die Hex-Zahl 12345 entspricht 2345 als Word oder 45 als Byte).

Das Adreßformat des VPort-50 entspricht dem gewohnten 8086-Adreßformat bestehend aus Segment und Offset, getrennt durch “:” (*seg:ofs*). Wird das Segment und der “:” weggelassen, wird das zuvor verwendete Segment weiterverwendet, wird dagegen nur das Segment oder das Offset weggelassen wird es als 0 angenommen.

2 Anwendung des Monitors

2.1 Anzeigen und Verändern des Speicherinhalts

Das Kommando *dump* dient zum Ausgeben eines Speicherauszugs, er erwartet die Adresse des auszugebenden Speicherbereichs als Argument.

Analog dazu dient das Kommando *set memory* zum Verändern des Speicherinhalts, auch er erwartet eine Speicheradresse als Argument. Weitere Argumente beschreiben die Daten, die in den Speicher geschrieben werden sollen, das können einzelne Bytes, Bytes mit einer durch Komma getrennten Anzahl sowie Strings sein. Wird kein solches Argument angegeben, kann der Speicher interaktiv verändert werden: die aktuelle Adresse und ihr derzeitiger Inhalt wird angezeigt und die Eingabe von Daten wird erwartet. Die Eingabe eines einzelnen "x" oder "." beendet das Kommando, keine Eingabe führt zur Anzeige der nächsten Speicherzelle ohne den Inhalt der Vorherigen zu verändern.

Bei beiden Kommandos kann das Adreß-Argument weggelassen werden, in dem Fall wird die zuletzt verwendete Adresse wiederverwendet. Abbildung 1 verdeutlicht die Verwendung der beiden Kommandos, für eine genaue Beschreibung siehe die Abschnitte 3.4 und 3.9.2.

Ausserdem steht noch das Kommando *move* zur Verfügung, mit dem Speicherblöcke von einer Adresse an eine Andere kopiert werden können. *Move* erwartet die Angabe der Quelladresse, der Zieladresse und der Anzahl der zu kopierenden Bytes als Argumente. Ein Beispiel für die Verwendung von *move* findet man in Abbildung 2, eine genaue Beschreibung der Funktion in Abschnitt 3.7.

2.2 Ein- und Ausgabe mit I/O-Ports

Zur Ein- und Ausgabe von Daten von bzw. zu den I/O-Ports des Rechners stehen die Kommandos *input* und *output* zur Verfügung.

Input erwartet eine Portadresse als Argument, wird sie weggelassen wird der zuletzt verwendete Port wieder verwendet. Die Ausgabe von *Input* ist das Byte, welches vom Port gelesen wurde.

Output erwartet zwei Argumente: einmal die Portadresse, die wie bei *Input* mit dem gleichen Effekt weggelassen werden kann, sowie ein Byte, welches auf dem Port ausgegeben werden soll. Wird auch dieses Argument weggelassen, wird der zuletzt verwendete Wert (*val*) auf dem Port ausgegeben.

Abbildung 3 verdeutlicht die Anwendung der beiden Kommandos. Für eine genaue Beschreibung der beiden Kommandos siehe die Abschnitte 3.6 und 3.8.

```

$ dump/address=100:0
0100:0000    cd 10 cd 11 3d 0a 00 75 f7 cd 14 00 00 00 00 00    ....=..u.....
0100:0010    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    .....
0100:0020    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    .....
0100:0030    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    .....
0100:0040    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    .....
0100:0050    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    .....
0100:0060    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    .....
0100:0070    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    .....
$ set memory/address=100:10
0100:0010 00 "abcdefghijklmnopqrstuvwxy"
0100:002a 00 20,20
0100:004a 00 x
$ dump/address=100:0
0100:0000    cd 10 cd 11 3d 0a 00 75 f7 cd 14 00 00 00 00 00    ....=..u.....
0100:0010    61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70    abcdefghijklmnop
0100:0020    71 72 73 74 75 76 77 78 79 7a 20 20 20 20 20 20    qrstuvwxyz
0100:0030    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0100:0040    20 20 20 20 20 20 20 20 20 20 20 00 00 00 00 00    .....
0100:0050    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    .....
0100:0060    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    .....
0100:0070    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    .....
$

```

Abbildung 1: Beispiele für die Verwendung von *dump* und *set memory*

```

$ dump/address=100:0
0100:0000    cd 10 cd 11 3d 0a 00 75 f7 cd 14 00 00 00 00 00    ....=..u.....
0100:0010    61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70    abcdefghijklmnop
0100:0020    71 72 73 74 75 76 77 78 79 7a 20 20 20 20 20 20    qrstuvwxyz
0100:0030    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0100:0040    20 20 20 20 20 20 20 20 20 20 00 00 00 00 00 00    .....
0100:0050    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    .....
0100:0060    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    .....
0100:0070    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    .....
$ move 100:10 100:11 18
$ dump/address=100:0
0100:0000    cd 10 cd 11 3d 0a 00 75 f7 cd 14 00 00 00 00 00 00    ....=..u.....
0100:0010    61 61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f    aabcdefghijklmno
0100:0020    70 71 72 73 74 75 76 77 78 7a 20 20 20 20 20 20    pqrstuvwxyz
0100:0030    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0100:0040    20 20 20 20 20 20 20 20 20 20 00 00 00 00 00 00    .....
0100:0050    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    .....
0100:0060    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    .....
0100:0070    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    .....
$ move 100:11 100:10 18
$ dump/address=100:0
0100:0000    cd 10 cd 11 3d 0a 00 75 f7 cd 14 00 00 00 00 00 00    ....=..u.....
0100:0010    61 62 63 64 65 66 67 68 69 6a 6b 6c 6d 6e 6f 70    abcdefghijklmnop
0100:0020    71 72 73 74 75 76 77 78 78 7a 20 20 20 20 20 20    qrstuvwxxz
0100:0030    20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
0100:0040    20 20 20 20 20 20 20 20 20 20 00 00 00 00 00 00    .....
0100:0050    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    .....
0100:0060    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    .....
0100:0070    00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00    .....
$

```

Abbildung 2: Beispiel für die Verwendung von *move*

```

$ input/port=1a
5d
$ output/port=1a fd
$ input
fd
$

```

Abbildung 3: Beispiel für die Verwendung von *input* und *output*

2.3 Laden und Ausführen von Programmen

Zum Laden von Programmen in den Speicher des Rechners steht das Kommando *set memory/intel* zur Verfügung. *Kermit* wurde so eingerichtet, daß es beim Senden einer Datei diesen Befehl automatisch ausführt, so daß er unter normalen Umständen nicht direkt eingegeben werden muss. Die genaue Beschreibung des Kommandos sowie eine Schritt-für-Schritt-Anleitung zum Laden eines Programms befindet sich in Abschnitt 3.9.3.

Soll ein Programm vom Beginn ausgeführt werden, empfiehlt sich das Kommando *start*. Er erwartet als Argument die Startadresse des Programms und richtet ausserdem noch den Stack für das Programm ein.

Um ein Programm nach einer Unterbrechung fortzusetzen verwendet man das Kommando *continue*, das keine Änderungen am gespeicherten Programmzustand vornimmt und das Programm dort fortsetzt, wo die Unterbrechung aufgetreten ist.

Die Anwendung beider Kommandos wird in Abbildung 6 verdeutlicht, eine genaue Beschreibung ihrer Funktionsweise findet man in den Abschnitten 3.2 und 3.11.

2.4 Anzeigen und Verändern des Programmzustandes

Der gespeicherte Programmzustand kann mit den Kommandos *show registers*, *set registers* und *clear registers* angezeigt, verändert oder auf Null gesetzt werden.

Als einziges dieser Kommandos erwartet *set registers* Argumente, die die Form einer Zuweisung eines Zahlenwertes an ein Register haben.

Die Abbildung 4 zeigt ein Beispiel für die Verwendung dieser Kommandos, ihre genaue Funktionsweise ist in den Abschnitten 3.9.4, 3.10.3 und 3.1.3 beschrieben.

2.5 Breakpoints und Einzelschrittbetrieb

Bei der Fehlersuche in Programmen ist es oft nötig, den Programmablauf an bestimmten Stellen zu unterbrechen oder das Programm in Einzelschritten ablaufen zu lassen. Dazu kann der Monitor im Programm bis zu 10 Breakpoints setzen und außerdem das Trace-Flag des Rechners verwenden.

Der Einzelschrittbetrieb wird durch das Kommando *set trace* ein- oder ausgeschaltet. Eine genaue Beschreibung findet man im Abschnitt 3.9.5.

Breakpoints setzt man mit dem Kommando *set breakpoint*, welches die Adresse des Breakpoints als Argument erwartet. Eine Liste aller Breakpoints kann mit *show breakpoints* angezeigt werden, aus der Liste gelöscht werden

```

$ show registers
ax = 0061, bx = 0000, cx = 0000, dx = 0000
si = 0000, di = 0000, bp = 0000, sp = 07ec
cs = 0100, ds = 0000, es = 0000, ss = 0040
ip = 0000, Flags = f112, - - A - - T - - -
$ set registers ax=0 bx=1 cx=2 dx
$ show registers
ax = 0000, bx = 0001, cx = 0002, dx = 0002
si = 0000, di = 0000, bp = 0000, sp = 07ec
cs = 0100, ds = 0000, es = 0000, ss = 0040
ip = 0000, Flags = f112, - - A - - T - - -
$ clear registers
$ show registers
ax = 0000, bx = 0000, cx = 0000, dx = 0000
si = 0000, di = 0000, bp = 0000, sp = 0000
cs = 0000, ds = 0000, es = 0000, ss = 0000
ip = 0000, Flags = 0000, - - - - - - - -
$

```

Abbildung 4: Beispiel für die Verwendung von *set/show/clear registers*

kann ein Breakpoint mit *clear breakpoint* unter Angabe seiner Nummer. Die Anwendung dieser Kommandos wird in Abbildung 5 verdeutlicht, in den Abschnitten 3.9.1, 3.10 und 3.1.1 wird ihre Funktionsweise genau beschrieben.

Beim Erreichen eines Breakpoints oder nach dem Ausführen eines Einzelschritts wird vom Monitor automatisch der aktuelle Programmzustand ausgegeben, danach erwartet der Monitor die Eingabe weiterer Kommandos. Ein Beispiel der Anwendung von Breakpoints ist in Abbildung 6 zu sehen.

2.6 Weitere Möglichkeiten

Oft ist es nötig, sich den Maschinencode eines Programms anzeigen zu lassen. Zu diesem Zweck hat der Monitor einen integrierten Disassembler, der mit dem Kommando *disassemble* aufgerufen wird. Eine genaue Beschreibung von *disassemble* befindet sich in Abschnitt 3.3.

Um dem Benutzer die wiederholte Eingabe von Adressen, Zahlenwerten und ähnlichem zu ersparen, merkt sich der Monitor die letzten Eingaben in sogenannten *Current Values*. Diese *Current Values* können mit den Kommandos *show current* und *clear current* angezeigt bzw. auf Null gesetzt werden. Eine genaue Beschreibung der Kommandos und aller *Current Values* findet sich in den Abschnitten 3.10.2 und 3.1.2, Abbildung 7 verdeutlicht ihre Funktion.

Sollte es einmal nötig sein, den Monitor zu reinitialisieren, kann man das

```

$ show breakpoints
no breakpoints set
$ set breakpoint/address=100:0
$ set breakpoint/address=100:7
$ set breakpoint/address=100:4
$ show breakpoints
breakpoint 0: 0100:0000
breakpoint 1: 0100:0007
breakpoint 2: 0100:0004
$ clear breakpoint 1 2 4
no such breakpoint - 4
$ show breakpoints
breakpoint 0: 0100:0000
$

```

Abbildung 5: Beispiel für die Verwendung von *set/show/clear breakpoint*

Kommando *reset* verwenden, welches in der Funktion einem Hardware-Reset entspricht.

Ausserdem gibt es das Kommando *help*, welches alle Kommandos des Monitors auflistet.

2.7 Monitor-Aufrufe

Zur Ein- und Ausgabe von Zeichen auf der Console durch Programme unterstützt der Monitor zwei Aufrufe, *getchar* und *putchar*. *Putchar* schreibt das Zeichen in AX auf die Console. *Getchar* gibt in AX ein Zeichen zurück, war kein Zeichen verfügbar wird stattdessen -1 zurückgegeben. Mit *getcharw* existiert auch eine blockierende Variante von *getchar*, die erst dann zum Programm zurückkehrt wenn auch wirklich ein Zeichen von der Console gelesen wurde. *Putchar* ist immer blockierend.

Ausserdem existiert der Aufruf *terminate*, der zum Beenden des Programms verwendet werden muss.

Auf dem VPort-50 werden Aufrufe durch folgende Software-Interrupts realisiert:

getchar int 10

putchar int 11

getcharw int 12

terminate int 14

```

$ start/address=100:0
abcdefghijklmnopqrstuvwxyz

terminated
ax = 000a, bx = 0001, cx = 0002, dx = 0002
si = 0000, di = 0000, bp = 0000, sp = 07fe
cs = 0100, ds = 0000, es = 0000, ss = 0040
ip = 000b, Flags = f046, - P - Z - - - -
$ set breakpoint/address=100:4
$ start 100:0

breakpoint reached
ax = 000a, bx = 0001, cx = 0002, dx = 0002
si = 0000, di = 0000, bp = 0000, sp = 07fe
cs = 0100, ds = 0000, es = 0000, ss = 0040
ip = 0004, Flags = f046, - P - Z - - - -
$ start 100:0
a
breakpoint reached
ax = 0061, bx = 0001, cx = 0002, dx = 0002
si = 0000, di = 0000, bp = 0000, sp = 07fe
cs = 0100, ds = 0000, es = 0000, ss = 0040
ip = 0004, Flags = f046, - P - Z - - - -
$ cont
b
breakpoint reached
ax = 0062, bx = 0001, cx = 0002, dx = 0002
si = 0000, di = 0000, bp = 0000, sp = 07fc
cs = 0100, ds = 0000, es = 0000, ss = 0040
ip = 0004, Flags = f012, - - A - - - - -
$ clear break 0
$ cont
aaa

terminated
ax = 000a, bx = 0001, cx = 0002, dx = 0002
si = 0000, di = 0000, bp = 0000, sp = 07fa
cs = 0100, ds = 0000, es = 0000, ss = 0040
ip = 000b, Flags = f046, - P - Z - - - -
$

```

Abbildung 6: Beispiel für die Verwendung von *start*, *continue* und *set breakpoint*

```

$ show current
src = 0100:0000, dst = 0100:0028, start = 0100:0000
iport = 001a, oport = 001a
val = 0002, cnt = 0018, bcnt = 0080, reg = 4, num = 2
trace is off
$ clear current
$ show current
src = 0000:0000, dst = 0000:0000, start = 0100:000
iport = 0000, oport = 0000
val = 0000, cnt = 0000, bcnt = 0000, reg = 0, num = 0
trace is off
$

```

Abbildung 7: Beispiel für die Verwendung von *show/clear current*

3 Kommandoreferenz

3.1 clear

3.1.1 clear breakpoint

Syntax:

```
clear breakpoint [num]
```

Mit diesem Kommando können Breakpoints aus der Monitor-internen Tabelle gelöscht werden. *Num* ist immer die dezimale Nummer des Breakpoints, die durch *show breakpoints* erfahren werden kann. Es können beliebig viele Argumente angegeben werden, wird kein Argument angegeben, wird der Wert des *Current Values num* verwendet.

3.1.2 clear current

Syntax:

```
clear current
```

Dieses Kommando setzt alle *Current Values* auf Null.

3.1.3 clear registers

Syntax:

```
clear registers
```

Dieses Kommando setzt alle Register auf Null.

3.2 continue

Syntax:

`continue`

Mit *continue* kann die Ausführung des Programms fortgesetzt werden. Es werden keine Register verändert, vor dem eigentlichen Start des Programms werden jedoch die Breakpoints im Programm gesetzt.

3.3 disassemble

Syntax:

`disassemble [/address=] [address]`

Disassemble disassembliert die nächsten 8 Maschinenbefehle ab der angegebenen Adresse. Wird keine Adresse angegeben, wird der *Current Value src* verwendet. Nach der Ausführung dieses Kommandos enthält *src* die Adresse des nächsten Maschinenbefehls.

3.4 dump

Syntax:

`dump [/address=] [address]`

Das Kommando *dump* gibt den Inhalt der nächsten 128 Speicherzellen ab Adresse *address* aus. Wird keine Adresse angegeben, wird die Summe der *Current Values src* und *bcnt* verwendet. *Dump* setzt *bcnt* immer auf 128, *src* wird nicht verändert.

3.5 help

Syntax:

`help`

Help gibt eine Liste aller Kommandos aus.

3.6 input

Syntax:

`input [/port=] [iport]`

Input liest ein Byte vom angegebenen I/O-Port. Wird kein Port angegeben, so wird der Wert des *Current Value iport* verwendet. Das gelesene Byte wird ausgegeben und ausserdem im *Current Value val* gespeichert.

3.7 move

Syntax:

```
move [src [dst [cnt]]]
```

Mit dem Kommando *move* lässt sich ein Speicherbereich der Länge *cnt* von *src* nach *dst* kopieren. Sollten die Speicherbereiche überlappen, wird so kopiert daß keine Daten verloren gehen. *Cnt*, *dst* und *src* können in dieser Reihenfolge weggelassen werden, wobei sie durch ihre jeweiligen *Current Values* ersetzt werden. Nach der Ausführung enthalten die *Current Values* *src* und *dst* die Adressen jeweils hinter den Speicherbereichen und *cnt* die angegebene Anzahl kopierter Bytes.

3.8 output

Syntax:

```
output [[[/port=]oport] val]
```

Output schreibt *val* auf den angegebenen I/O-Port. Wird kein Wert für *val* oder *oport* angegeben, werden die jeweiligen *Current Values* verwendet. *Val* und *oport* werden nach Ausführung in den *Current Values* gespeichert.

3.9 set

3.9.1 set breakpoint

Syntax:

```
set breakpoint [address=] [address]
```

Dieses Kommando setzt einen Breakpoint an die angegebene Adresse. Wird keine Adresse angegeben, wird der Inhalt des *Current Values* *src* verwendet. Der Breakpoint wird durch dieses Kommando nur in eine Monitor-interne Tabelle eingetragen, im Programm eingetragen wird er erst bei der Ausführung mit *start* oder *continue*. Es können maximal 10 Breakpoints gesetzt werden. Die Adresse des Breakpoints wird im *Current Value* *src* gespeichert.

3.9.2 set memory

Syntax:

```
set memory [/address=address] [{string | val[,cnt]}]
```

Mit diesem Kommando lassen sich Daten in den Speicher schreiben. Wird keine Adresse angegeben, wird der *Current Value src* verwendet. Ein *string* ist eine ASCII-Zeichenkette wie in Abschnitt 1.3 beschrieben, *val* ist ein Byte, das in den Speicher geschrieben werden soll, *cnt* gibt an, wie oft hintereinander dies geschehen soll. Es können beliebig viele derartige Argumente angegeben werden, die hintereinander in den Speicher geschrieben werden. Wird kein Argument angegeben, gibt der Monitor die aktuelle Adresse und ihren Inhalt aus und erwartet die Eingabe eines neuen Inhalts. Eine leere Eingabe kann verwendet werden, um zur nächsthöheren Adresse zu schalten, ohne den Inhalt der aktuellen Adresse zu verändern, durch die Eingabe eines einzelnen "x" oder "." kann zur Kommandoeingabe des Monitors zurückgekehrt werden. Dieses Kommando setzt die *Current Values src* auf die nächste, nichtveränderte Adresse und *val* und *cnt* auf die jeweils zuletzt verwendeten Werte.

3.9.3 set memory/intel

Syntax:

```
set memory/intel [/start]
```

Dieses Kommando dient zum Laden einer Intel-Hex Datei in den Speicher. Zu diesem Zweck ändert der Monitor seinen Prompt in ">". Kermit ist so konfiguriert, daß es beim Senden einer Datei mit dem Kommando *transmit* genau diesen Prompt erwartet, bevor es die nächste Zeile sendet. Die Vorgehensweise zum Laden einer Intel-Hex Datei ist folgende:

1. dem Monitor das Kommando *set memory/intel* geben
2. sobald der Prompt ">" erscheint, mit *Alt-X* in den Kermit-Kommandomodus umschalten
3. Kermit das Kommando *transmit* geben, z.B. *transmit prog.hex*
4. warten, bis die Datei übertragen ist, dann *Enter* drücken, um zum Kermit-Kommandomodus zurückzukehren
5. Kermit das Kommando *connect* geben, um zum Monitor zurückzukehren

6. am Monitor weiterarbeiten, z.B. mit dem Kommando *start* um das geladene Programm auszuführen

Wem das zu umständlich ist, der kann die Taste *F2* drücken, die ein Kermit-Skript ausführt um den Vorgang zu automatisieren. Dieses Skript erwartet die Eingabe eines Dateinamens und schickt die Datei an den Monitor.

Mit der Variante *set memory/intel/start* wird ein übertragenes Programm unmittelbar nach der Übertragung automatisch ausgeführt.

Enthält die Intel-Hex Datei einen sogenannten “Start Address Record”, der dem Monitor mitteilt an welcher Adresse der Eintrittspunkt in das geladene Programm steht, wird diese Adresse in den *Current Values src* und *start* gespeichert.

3.9.4 set registers

Syntax:

```
set registers [reg=[val]]
```

Mit diesem Kommando lassen sich die Registerinhalte ändern. Das Argument besteht aus dem Namen eines Registers und dem Wert, der diesem Register zugewiesen werden soll. Es können beliebig viele Register mit einem Kommando gesetzt werden, wird kein Argument angegeben wird auch kein Register verändert. Beim letzten Argument kann der Wert weggelassen werden, in dem Fall wird der Inhalt des vorher gesetzten Registers verwendet. Das jeweils zuletzt veränderte Register und der ihm zugewiesene Wert werden in den *Current Values reg* und *val* gespeichert.

3.9.5 set trace

Syntax:

```
set trace [{on | off | toggle}]
```

Set trace dient zum Architekturunabhängigen verändern des Trace-Flags des Rechners. Die Funktion der Argumente ist eindeutig, kein Argument ist gleichbedeutend mit “toggle”. Zusätzlich zum Trace-Flag des Rechners wird ein Monitor-internes Trace-Flag gesetzt, dessen Wert durch *show current* angezeigt werden kann.

3.10 show

3.10.1 show breakpoints

Syntax:

```
show breakpoints
```

Show breakpoints gibt die Monitor-interne Breakpoint-Tabelle aus.

3.10.2 show current

Syntax:

```
show current
```

Dieses Kommando gibt die *Current Values* aus. Die Bedeutungen der *Current Values* sind wie folgt:

src ist die zuletzt verwendete (Quell-) Adresse

dst ist die zuletzt verwendete Zieladresse

start ist die zuletzt verwendete Startadresse des Programms

iport ist der zuletzt verwendete I/O-Port für *input*

oport ist der zuletzt verwendete I/O-Port für *output*

val ist der zuletzt verwendete Wert (z.B. nach *input*)

cnt ist die zuletzt verwendete Anzahl (z.B. nach *move*)

bcnt ist die zuletzt verwendete Anzahl Bytes (z.B. durch *dump*)

reg ist die dezimale Nummer des zuletzt verwendeten Registers

num ist die zuletzt verwendete dezimale Nummer (z.B. durch *clear breakpoint*)

Ausserdem wird noch der Zustand des Monitor-internen Trace-Flags ausgegeben.

3.10.3 show registers

Syntax:

```
show registers
```

Mit diesem Kommando können die aktuellen Registerinhalte angezeigt werden. Die Flagzustände des Flag-Registers werden ausserdem noch aufgeschlüsselt angezeigt.

3.11 start

Syntax:

```
start [/address=] [address]
```

Start ist ein Spezialfall von *continue*, mit dem die Ausführung des Programms begonnen werden kann. Vor der Ausführung setzt *start* die Startadresse des Programms auf die angegebene Adresse oder verwendet den *Current Value start*. Zusätzlich wird der Stack des Programms auf einen vernünftigen Wert gesetzt. Im Gegensatz zu anderen Befehlen wird dieser Befehl bei einer nachfolgenden leeren Eingabe (z.b. nach Erreichen eines Breakpoints) nicht wiederholt, stattdessen wird dann direkt der Befehl *continue* ausgeführt ohne den Programmzustand zu modifizieren.

3.12 reset

Syntax:

```
reset
```

Reset startet den Monitor einschliesslich aller Initialisierungen neu. Das ist nützlich, falls ein Hardware-Reset nötig ist, man selbst aber zu faul ist aufzustehn um den Reset-Knopf zu betätigen.

A Adressen der VPort-50 Hardware

A.1 Speicherbereiche

00000 – 0fff 64k statisches RAM (U12/U13)

10000 – 1fff 64k statisches RAM (U14/U15)

c0000 – dfff 128k EPROM oder EEPROM (U14/U15)

e0000 – ffff 128k EPROM oder EEPROM (U14/U15)

A.2 Peripherie

A.2.1 PIO 8255

Daten A 11h

Daten B 13h

Daten C 15h

Control 17h

A.2.2 Counter/Timer 8254

Counter 0 30h

Counter 1 32h

Counter 2 34h

Mode 36h

A.2.3 UART 16450

Register 0 40h

Register 1 42h

Register 2 44h

Register 3 46h

Register 4 48h

Register 5 4ah

Register 6 4ch

Register 7 4eh

A.2.4 Interrupt Controller 8259A

Register 0 18h

Register 1 1ah