# S D T - D E B U G G E R

## Hardwarekofiguration

Der Debugger ist in einem System, das eine 8080, 8085 oder Z80 CPU verwendet ablauffähig. Er benötigt ca 4kB Eprom und 512 Byte Ram.

Die Benutzerschnittstelle ist möglichst hardwareunabhängig gestaltet, softwaremäßig wird sie nur über die Subroutinen CI, CO, CSTS angesteuert, sodaß die Implementierung auf einer beliebigen Hardwareumgebung möglich ist.

## Notation

Die verschiedenen Kommandos werden mittels BNF beschrieben. Diese Notationsform wird im folgendne kurz erläutert:

| | |
|---|---|
| [ <x> ] | <x> muß nicht unbedingt angegeben werden. |
| { <x> } | <x> kann beliebig oft angegeben werden |
| <x> | ein hexadezimaler Wert |

Die Zeichen [,],{,},< und > gehören zur Syntaxbeschreibung und DÜRFEN nicht eingegeben werden.

Für

G [<start>] {,<Breakpoint>}

könnte folgendes eingegeben werden:

```
G
G8000
G8000,3010
G,8010
G,8010,1814
.......
```

## Kommandos

Nach dem Start des Debuggers mit RESET, gibt dieser seine Signonmessage aus.

Danach wird das Prompt > angezeigt und auf die Eingabe eines Kommandos gewartet.

Ein Kommando hat immer folgende Form:

<Name> [ <Operand> ] { , <operand> }

Folgende Kommandonamen <Name> sind definiert:





```
A ..... Assemble
B ..... Breakpoints
D ..... Display
F ..... Fill
G ..... Go
L ..... List source of instructions
M ..... Move
N ..... step Next instruction
P ..... access I/O Port
R ..... Realtimesubroutines
S ..... Substitute
T ..... Transparente Zeichenausgabe
X ..... Examine Register
```

Ein Kommandozeile wird mit Return beendet. Text nach einem Strichpunkt ';' wird ignoriert.

## Assemblieren

| | |
|---|---|
| A [ <adr> ] | Assembliere beginnend bei <adr> |

In Assemblerschreibweise eingegebene Instruktionen werden vom Debugger hexadezimal codiert und beginnend bei der angegebenen Adresse im Speicher abgelegt.

z.B.

```
A8000
8000 lxi d,795f
8003 mvi a,59
8005
```

Der Debugger gibt die Adresse, von der ab abgespeichert wird, aus. Anschliebend wartet er auf die Eingabe einer Instruktion, codiert diese und speichert sie ab. Die Speicheradresse wird dabei auf die nächste Instruktion gesetzt. Diese Sequenz wird wiederholt, bis statt einer Instruktion nur RETURN eingegeben wird.

Nach der Exekution des Beispiels stehen folgende Bytes von 8000 bis 8004 im Speicher:

```
8000 11 5f 78 3e 59
```

Wenn beim Aufruf die Adresse nicht angegeben wird, dann wird bei der Adresse begonnen, bei der bei der vorhergehenden Assemblierung beendet wurde.

## Breakpoints

| | |
|---|---|
| B | Anzeigen der Breakpoints |
| B <adr> {,<adr>} | setzen von Breakpoints |
| B {,<adr>} | hinzufügen von Breakpoints |

Ein Breakpoint ist eine Adresse, die auf das erste Byte einer Instruktion (Opcode) zeigt. Wenn bei Programmausführung diese Adresse erreicht wird, wird das Programm angehalten.

Mit dem B Kommando können bis zu 8 Breakpoints definiert werden, die bei jedem Programmstart überwacht werden.

z.B. B6E3A Breakpoint an der Adresse 6E3AH setzen

**Display**

D [<start>] [,<ende>]

Mit dem D Kommando können Speicherteile angezeigt werden. Ausgegeben wird eine hexadezimale Darstellung der Bytes und die entsprechenden ASCII Zeichen.

Angezeigt wird der Bereich zwischen und inclusive <start> und <ende> Adresse. Wenn <start> nicht eingegeben wurde, wird beim letzten Breakpoint oder beim Ende des letzten D Kommandos begonnen. Wenn <ende> nicht eingegeben wurde, werden 64 Bytes ausgegeben.

z.B. D5,9 Speicher von Adresse 5H bis 9H anzeigen

**Fill**

F <start>,<ende>,<filler>

initialisiert den Speicherbereich <start> bis <ende> mit einem vorgegebenen Byte.

z.B. F9000,9200,ff 9000h bis 9200h mit 0ffh füllen

**Go**

G [<start>] {,<Breakpoint>}

Mit dem G Kommando wird die Ausführung eines im Speicher befindlichen Programms gestartet. Wenn die <start> Adresse nicht eingegeben wird, wird beim letzten PC Wert fortgesetzt (PC kann mit dem X-Kommando angeschaut werden). Alle Breakpoints werden gesetzt. Die mit dem G-Kommando angegebenen Abbruchpunkte werden nach einer Unterbrechung wieder gelöscht.

**List**

L [<start>] [,<ende>]

Ausgeben der Instruktionen im Speicherbereich zwischen <start> und <ende> in memnonischer Schreibweise. Die Defaultwerte wie beim D-Kommando.

**Move**

M <start>,<ende>,<dest>

Speicherbereich verschieben. Quell- und Zielbereich dürfen sich beliebig überschneiden.





**Next**

N [<start>]

Schrittweise Ausführung von Instruktionen. Wenn die Startadressenicht eingegeben wird, dann wird beim momentanen PC Stand begonnen.

Ausgabe:

```
3680 mvi a,3e _
```

Es wird die nächste Instruktion disassembliert. Nach Drücken der Blanktaste wird diese ausgeführt. Wird die "R" Taste gedrückt, und die Instruktion ändert den PC, wird ein Echtzeitunterprogramm ausgeführt. Mit jeder anderen Taste wird der Stepmode abgebrochen. Nach der Ausführung einer Instruktion werden die Registerinhalte angezeigt.

```
3680 mvi a,3e   p...a..s a=3e bc=1234 de=3456 hl=7890 sp=8908 i=03
                aaaaaaaa bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb cccc
```

a ... Flags
b ... Registerdump
c ... Interruptmaske (mit RIM gelesen)

**Port_access**

P <I/O adr> Port anzeigen oder setzen

Die I/O Adresse und der vom Port eingelesene Wert werden angezeigt. Wenn der Portinhalt überschrieben werden soll, muß der neue Wert und Return, andernfalls Escape, gedrückt werden.

**Realtimesubroutine**

| | |
|---|---|
| R | anzeigen der Echtzeitunterprogramme |
| R <adr> {,<adr>} | definieren von Echtzeitroutinen |
| R {,<adr>} | hinzufügen von Echtzeitroutinen |

Es gibt Unterprogramme, die nicht einzelschrittweise ausgeführt werden dürfen. Zum Beispiel die Ein- oder Ausgabesubroutinen. Diese müssen als Echtzeitunterprogramme definiert werden. Beim Steppen dieses Unterprogrammaufrufes wird dann das Unterprogramm ohne Unterbrechung ausgeführt und nach dem Rücksprung ein Registerdump ausgegeben.

Mit dem R Kommando können bis zu 8 Adressen von Realtimesubroutinen angegeben werden.

**Substitute**

S [<start>]

Mit dem S Kommando kann der Speicherinhalt angezeigt und geändert werden. Die Adresse, ab der der Speicher geändert werden soll wird angegeben.

z.B.  s4789  4789 6F-_

Der angezeigte Speicherinhalt kann geändert werden, indem man den neuen Wert eingibt. Blank nächstes Byte, Return Abschluß. Wenn kein neuer Wert angegeben wird, bleibt der alte erhalten.

**Transparent**

```
T
[<text>] CTRL-A
```

Jedes nach dem T-Kommando eingegebene Zeichen wird ohne Veränderung an die Konsole geschickt. Als Abschlußzeichen wird CTRL-A=01H verwendet. Dieses Zeichen kann nicht gesendet werden.

**Examine Register**

| | |
|---|---|
| X | ausgeben der Register |
| X <Register-name> | Ändern des Registers |

als Registername sind

| | |
|---|---|
| a,f,b,c,d,e,h,l,i | als Byteregister und |
| bc,de,hl,sp,pc | als Wordregister zulässig. |

z.B. xa
a=76- _

ein Registerwert wird angezeigt, neuer Wert kann eingegeben werden.Aufs nächste Register wird mit Blank umgeschaltet. Das Kommando wird mit Return beendet.

**Fehlermeldungen**

**memory fails at xxxx good = gg bad = bb**

Der Debugger konnte das Byte gg nicht an der Adresse xxxx ins Ram schreiben. Die Speicherzelle enthält das Byte bb. Entweder addressiert xxxx keinen Rambaustein, oder der addressierte Baustein ist defekt.

**too many breaks**

Es wurde versucht mehr als 8 Breakpoints zu definieren.

**too many RTS**

Es wurde versucht mehr als 8 Echtzeitroutinen zu definieren.

**execution halted**

Ein HLT Befehl wurde während Einzelschrittausführung entdeckt.



**SST return c[illegible]usion**

Der Stack des in Einzelschritten ausgeführten Programms konnte nicht korrekt rekonstruiert werden.Es wurde der SP nicht gesetzt. Entweder er kollidierte mit vom Debugger benutzten Speicherbereichen, oder addressiert keinen RAM-Speicher.

**bad instruction**

Die eingegeben Instruktion ist nicht korrekt.

**destination range > 64kB**

Der Zielbereich einer Move-Anweisung liegt nicht innherhalb 64k.

**bad register name**

Das angegebene Register ist falsch.

**unknown command**
**bad parameter count**
**terminator expected**

Das Kommando wurde fehlerhaft eingegeben.

**Einsprungadressen:**

Folgende Einsprungadressen, können benutzt werden:

| | |
|---|---|
| CI | Zeichen von der Tastatur -> a, kein Echo |
| CO | Zeichen vom A auf den Bildschirm |
| GETLIN | Zeileneditor<br>rubout -> letztes Zeichen löschen,<br>^x -> current line löschen<br>im Registerpaar DE wird ein Zeiger auf den Buffer, im Register B die Anzahl der Zeichen zurückgegeben. |
| HXOUT | Das Register A wird hexadezimal ausgegeben. |
| HXIN | Ein Hexadezimalwert wird ins Registerpaar HL eingelesen. Eingabequittierung mit Blank oder Return. |

Die Einsprungpunkte der RST 1 bis RST 6 können nicht benützt werden. Bei der Instruktion RST 7 (IM-1) wird an den Beginn (Adresse 0) verzweigt, der Vektor kann allerdings geändert werden, da er im RAM steht. Der NMI-Vektor kann zum Abbruch eines Programms verwendet werden. Er zeigt auf den Kommando-Decoder.

Ein Programm kann mit einer RST 7 Instruktion abgebrochen werden.