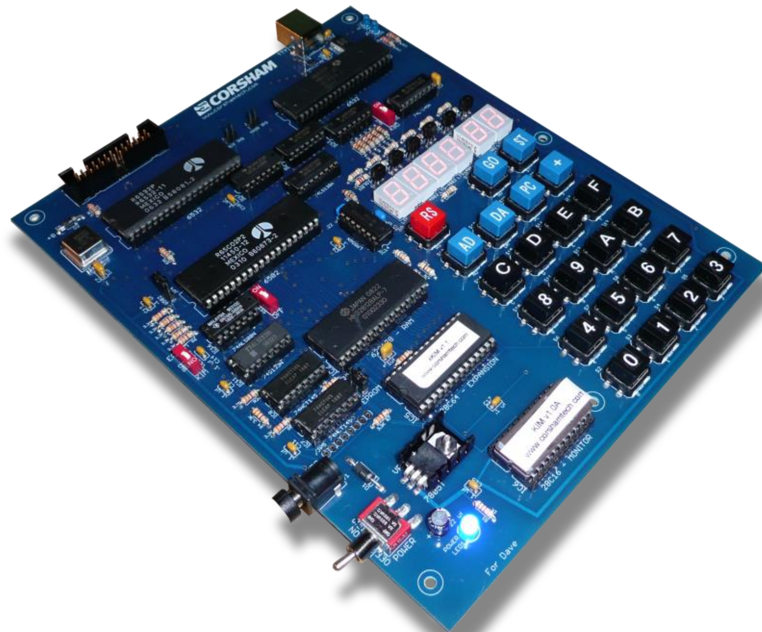
A PROGRAMMER'S GUIDE TO KIM PROGRAMMING

© by Erik Bartmann - Vers. 0.1

2 - ERSTE BEFEHLE



Erste Befehle

Nun wird es aber Zeit, dass wir unser erstes Programm schreiben und wir werden hier einiges über die Maschinensprache bzw. die Assemblerprogrammierung lernen. Es handelt sich zwar nur um sehr wenige Befehle, doch das macht überhaupt nichts. Fangen wir also einfach einmal damit an, einen Wert zu laden und ihn in einer Speicherstelle abzulegen. Hört sich recht simpel an, nicht wahr?! Welche Befehle sind dafür notwendig, welchen Speicherbereich verwenden wir und wie bekommen wir das Programm in den KIM-1 gespeichert? Gehen wir gleich Schrittchen für Schrittchen vor.

Ein sehr wichtiges Register innerhalb des 6502 und auch anderen CPUs ist der sogenannte Akkumulator - kurz *Akku* genannt. Er speichert die Ergebnisse einer Recheneinheit, die ALU (Arithmetic Logic Unit) genannt wird. Natürlich wäre es möglich gewesen, diese Speicherstelle im Arbeitsspeicher anzusiedeln, was jedoch bedeutet hätte, bei jeder Rechenaktion auf den langsamen externen Speicher zuzugreifen. Der Zugriff auf ein internes Register der CPU ist um ein Vielfaches schneller. Die nachfolgende Abbildung zeigt das Zusammenspiel von ALU, Akku und dem Speicher (ROM, RAM) untereinander bzw. über den Datenbus.

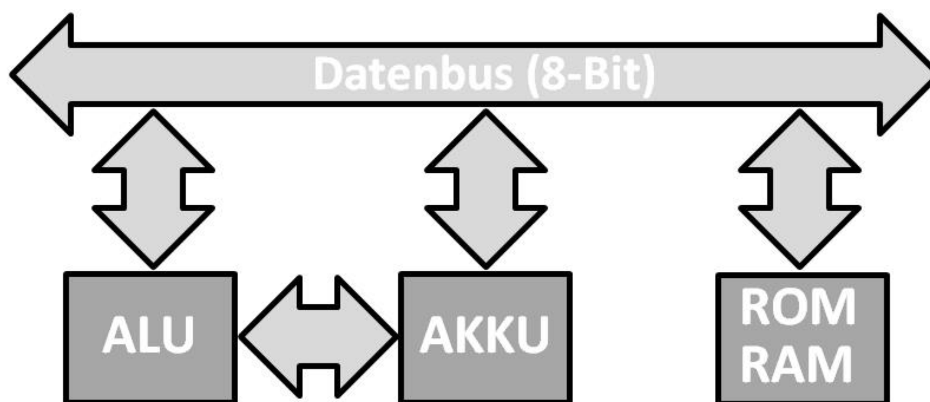


Abbildung 1: Die ALU, der Akku und der Speicher

Im ersten Schritt werden wir die ALU jedoch nicht nutzen, sondern lediglich den Akku. Schauen wir uns die folgenden drei Befehle einmal an. Der erste lädt einen Wert in den Akku, der zweite legt den Inhalt des Akkus im Speicher ab und der dritte unterbricht bzw. beendet das Programm.

Befehl	Bedeutung	Funktion
LDA	Load Akkumulator	Lade einen Wert in den Akku
STA	Store Akkumulator	Speicher den Wert des Akku
BRK	Break	Unterbrechung des Programms

Tabelle 1: Verwendete Befehle

Unser Programm soll einen festen Wert, z.B. 7 in den Akku laden und ihn später im Speicher ablegen. Anschließend soll das Programm unterbrochen werden. Wir sollten uns zudem Gedanken machen, wo unser Programm im Speicher überhaupt beginnen soll. Welche Bereiche im Arbeitsspeicher sind überhaupt für uns zugänglich und werden z.B. nicht vom Betriebssystem genutzt? Im Kapitel über die Speicheraufteilung habe ich das dargestellt. Es sind z.B. die gesamte Page 2 und Page 3 im Bereich von \$0200 bis \$03FF frei verfügbar. Was spräche also dagegen, das Programm bei \$0200 beginnen zu lassen. Im Assemblerprogramm *Ophis* habe ich den folgenden Code eingegeben bzw. in einer Textdatei gespeichert, die wie folgt ausschaut:

```
1 .org $0200           ; Startadresse $0200
2 START:  LDA #$07     ; Lade Wert 7 in Akku
3         STA $0207    ; Speichere Akku
4         BRK          ; Unterbrechung
```

Abbildung 2: Unser erstes Programm

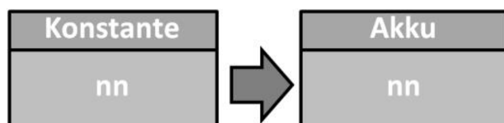
Hier ist noch einiges zu erklären, denn es befinden sich neben den Assemblerbefehlen noch weitere Befehle im Quellcode. Sie haben im eigentlichen Sinn nichts mit unserem Programm zu tun, werden aber trotzdem benötigt, um dieses zu verwalten bzw. zu strukturieren. Zu Beginn in Zeile 1 steht die sogenannte Origin-Anweisung, die dem Assembler von Ophis mitteilt, an welcher Stelle unser Programm im Arbeitsspeicher abgelegt werden soll. Origin bedeutet übersetzt *Ursprung*.

In Zeile 2 steht ganz zu Beginn eine Bezeichnung, die als Label - also Markierung - fungiert. Sie hat mit dem eigentlichen Programm nichts zu tun und ist auch in unserem Fall nicht unbedingt erforderlich. Das Label *START* dient im Moment nur der besseren Orientierung, an welcher Stelle das Programm beginnt. Es ist bei Programmverzweigungen sehr hilfreich, um z.B. bei einem Sprungbefehl an eine bestimmte Stelle im Code zu springen, so dass man die eigentliche Speicheradresse nicht kennen muss, sondern nur den Namen des Labels. In jeder Zeile stehen auf der rechten Seite Kommentare, die mit einem Semikolon eingeleitet werden. Sie sind an manchen Stellen sehr hilfreich, wenn es darum geht, bestimmte Zeilen im Code zu dokumentieren.

Sehen wir uns aber die eigentlichen Codezeilen genauer an. Der Befehl

```
LDA #$07
```

lädt den Wert 7 in den Akku. Das Doppelkreuz # vor der HEX-Zahl $07 besagt, dass es sich um eine unmittelbare (immediate) Adressierung handelt. Die LDA-Anweisung über die unmittelbare Adressierung lädt also die Konstante $07 in den Akku und holt nicht einen Wert aus dem Speicher, um diesen dann zu laden.



Wir kommen später noch auf weitere Adressierungsarten zu sprechen. Dieser Wert $07 soll jetzt in einer bestimmten Speicherstelle im Arbeitsspeicher abgelegt werden, was wir mit dem folgendem Befehl machen:

```
STA $0207
```

Er bewirkt das Lesen des Inhaltes Akkus und die Speicherung an die genannte Speicheradresse. Warum ich die Speicheradresse $0207 verwendet habe, werden wir gleich sehen. Der letzte Befehl

```
BRK
```

bedeutet eine Programmunterbrechung, denn ansonsten würde das Programm versuchen, jeden weiteren Befehl in den darauffolgenden Speicherstellen zu interpretieren und ggf. ausführen, was wir nicht möchten und möglicherweise ungeahnte Auswirkungen haben könnte, da der Code, der dort steht, nicht bekannt ist. Haben wir das Programm z.B. in Notepad++ eingegeben und z.B. unter der

Bezeichnung *example001.asm* gespeichert, kann Ophis seine Arbeit mit der folgenden Kommandozeile beginnen:

C:\Program Files (x86)\Ophis\ophis.exe example001.asm -l exmaple001.bin

Im Anschluss liegt eine Datei mit dem Namen *example001.bin* vor, deren Inhalt wie folgt ausschaut:

```
1  0200  A9 07     LDA  #$07
2  0202  8D 07 02  STA  $0207
3  0205  00        BRK
```

Abbildung 3: Die Maschinensprache

Auf der linken Seite befinden sich die Adresszeilen und rechts davon der entsprechende Maschinencode, der rechts außen mit dem jeweiligen Assemblerbefehl kommentiert wird. Wir sehen z.B. in Zeile 1 den Wert A9, welcher der CPU mitteilt, dass es sich um eine unmittelbare Adressierung handelt und der nachfolgende Wert 07 in den Akku geladen werden soll. In Zeile 2 teilt der Wert 8D der CPU mit, dass es sich um eine absolute Adressierung handelt, die auf den Speicher zugreift. Die Werte 07 und 02 sind hier in der Reihenfolge vertauscht, denn es wird immer zuerst das LOW-Byte und dann das HIGH-Byte der Adresse ($0207) dargestellt bzw. gespeichert. Die verschiedenen Elemente einer Zeile haben unterschiedliche Bezeichnungen und lauten wie folgt:

Befehl	OP-Code	Operand
LDA #$07	A9	07
STA $0207	8D	0207
BRK	00	

Tabelle 2: Befehlselemente

Dann wollen wir mal sehen, wie wir das Programm in den Speicher des KIM-1 bekommen.

Die Programmierung des KIM-1

Die Programmierung des KIM-1 Clone erfolgt über eine Tastatur bzw. einer 6-stelligen Siebensegmentanzeige, die wie folgt ausschaut:

Abbildung 4: Die Tastatur bzw. die Anzeige des KIM-1 Clone

Welche Funktionen besitzen die einzelnen Tasten. Schauen wir uns das in der folgenden Tabelle genauer an:

Taste	Funktion
0 bis 9 A bis F	Diese 16 Tasten werden zur Eingabe der Daten bzw. Adressen im HEX-Format benötigt.
AD	Es wird die Eingabe der Werte im Address-Mode aktiviert.
DA	Es wird die Eingabe der Werte im Data-Mode aktiviert.
+	Hierüber wird die angezeigte Adresse um den Wert 1 erhöht, wobei jedoch nicht der Eingabemode AD bzw. DA geändert wird.
PC	Es wird die Adresse abgerufen, die im Program-Counter PC gespeichert ist und zur Anzeige gebracht. Die Reihenfolge ist PCH, PCL.
RS	Es wird ein kompletter System-Reset durchgeführt und die Kontrolle an das Betriebssystem zurückgegeben.
GO	Startet die Programmverarbeitung ab der angezeigten Adresse.
ST	Es wird das laufende Programm unterbrochen und die Kontrolle an das Betriebssystem zurückgegeben.

Tabelle 3: Die Tasten und ihre Funktionen

Wie wir das im Einzelnen alles handhaben, werden wir zu gegebener Zeit sehen.

Die KIM-Vektoren

Bevor wir jedoch beginnen, hier einige Einstellungen, die vorgenommen werden müssen, damit unser KIM-1 Clone auch richtig funktioniert. Für den KIM-Uno ist das schon vorkonfiguriert und muss nicht mehr vorgenommen werden. Es gibt im KIM-1 bzw. den KIM-1 Clone Speicherstellen, die wie mit bestimmten Werten versehen müssen, damit die Programme richtig funktionieren. Das hat weniger mit den eigentlichen Programmen zu tun, als mit der Tatsache, dass das Betriebssystem bei bestimmten Ereignissen auch richtig reagieren kann. Die folgenden Adressen müssen mit den angegebenen Werten versehen werden:

Adresse	Daten
$17FA	00
$17FB	1C
$17FE	00
$17FF	1C

Warum ist das aber erforderlich? Nun, es gibt beim KIM-1 einen sogenannten *SST* (Single-Step-Mode), der eine schrittweise Ausführung des Programms erlaubt und die *RS*-Taste, die eine Programmunterbrechung bewirkt. Beide Ereignisse müssen in richtiger Weise vom Betriebssystem behandelt werden, wofür die ersten beiden Adressen $17FA uns $17FB verantwortlich sind. Des Weiteren muss einer Programmunterbrechung durch die BRK-Anweisung ebenfalls in richtiger Weise begegnet werden, wofür die letzten beiden Adressen $17FE und $17FF verantwortlich sind. Die Adressen werden Vektoren genannt, weil sie auf etwas zeigen, nämlich auf Speicheradressen. Im Betriebssystem schaut es an der entsprechenden Speicherstelle $1C00 wie folgt aus:

```
1C00   85 F3       SAVE     STA    ACC         KIM ENTRY VIA STOP (NMI)
1C02   68                   PLA                OR BRK (IRQ)
```

Wie versehen wir also diese vier Speicherstellen mit den entsprechenden Werten. Dafür gehen wir folgendermaßen vor. Die Anzeigen XX als Daten in der Tabelle bedeuten, dass sie keine Rolle spielen, denn es können sich dort Werte befinden, die jedoch überschrieben werden:

Taste	Anzeige
AD	XXXX XX
1 7 F A	17FA XX
DA	17FA XX
0 0	17FA 00
+ 1 C	17FB 1C
+	
+	
+ 0 0	17FE 00

+	1 C	17FF 1C
AD		

Tabelle 4: Einstellen der KIM-Vektoren

Die Umsetzung des Programms

Wenn die Einstellungen vorgenommen wurden, können wir zur Programmierung des eben gezeigten Programms schreiten. Die Tasten AD, DA und + sind ja nun geläufig und stellen sicherlich kein allzu großes Problem mehr dar. Zur Erinnerung hier noch einmal das zu übertragende Programm:

```
1  0200  A9 07     LDA  #$07
2  0202  8D 07 02  STA  $0207
3  0205  00        BRK
```

Die folgenden Eingaben sind dazu erforderlich. Beachte, dass durch die Plus-Taste die Adresse automatisch um den Wert 1 hochgezählt wird und eine erneute Eingabe einer Adresse dadurch entfällt:

Taste		Anzeige
AD		XXXX XX
0 2 0 0		0200 XX
DA		0200 XX
	A 9	0200 A9
+	0 7	0201 07
+	8 D	0202 8D
+	0 7	0203 07
+	0 2	0204 02
+	0 0	0205 00
AD		

Tabelle 5: Die Eingabe des Programms

Das Programm - also der erste Befehl - befindet sich jetzt ab der Speicherstelle $0200 im Arbeitsspeicher, also im RAM und kann dort gestartet werden. Beachte, dass wir uns beim Drücken der DA-Taste im Data-Mode befinden und wir darin solange verbleiben, bis wieder die AD-Taste für den Address-Mode gedrückt wird. Durch das Hochzählen der Adresse durch die Plus-Taste ist also kein erneuter Druck auf die DA-Taste zur Eingabe der Daten erforderlich.

Das Programm starten

Wir müssen jetzt jedoch wieder die Startadresse eingeben, ab der das Programm ausgeführt werden soll. Also geben wir folgende Tastenfolgen ein:

Taste	Anzeige
AD	XXXX XX
0 2 0 0	0200 A9
GO	0207 07

Tabelle 6: Das Starten des Programms

Nach der Ausführung des Programms durch die GO-Taste, sollte in der Anzeige 0207 07 stehen, was bedeutet, dass dort das Programm stehen geblieben ist und an dieser Adresse $0207 genau der Wert 07 zu sehen ist, der den Inhalt des Akkus zeigt. Die Anweisung

```
STA $0207
```

hat genau dies bewirkt. Somit hat alles wunderbar funktioniert. Falls diese Adresse mit dem Inhalt nicht angezeigt werden sollte, wurde entweder das Programm fehlerhaft eingegeben oder die KIM-Vektoren stimmen nicht. Es ist also eine Überprüfung notwendig. Warum steht aber im Display nach dem Start des Programms die Adresse 0207? Bei der letzten Anweisung in Adresse 0205 handelt es sich um den BRK-Befehl (Programmunterbrechung). Der KIM reagiert beim Erreichen dieser Anweisung immer mit dem Erhöhen des Programmzählers +2. Die Addition $0205 + 2 = $0207.

Das Programm überprüfen

Wir können das Programm sehr einfach kontrollieren, in dem die folgenden Tasten gedrückt werden:

Taste	Anzeige
AD	XXXX XX
0 2 0 0	0200 A9
+	0201 07
+	0202 8D
+	0203 07
+	0204 02
+	0205 00

Tabelle 7: Die Überprüfung des Programms

Wenn diese Werte angezeigt werden und die KIM-Vektoren stimmen, sollte alles in Ordnung sein.

Viel Spaß dabei...

Erik Bartmann

http://erik-bartmann.de/