Programmierhandbuch

1. Einführung in die Mikroprozessorarchitektur

Computer sind Geräte, welche eine Liste oder auch Folge von Befehlen abarbeiten. Eine solche Folge von Befehlen heißt *Programm*. Diese Programme befinden sich im *Speicher*. Ausgeführt werden diese Programme von einem *Prozessor*. Prozessor und Speicher sind durch *Busse* miteinander verbunden.

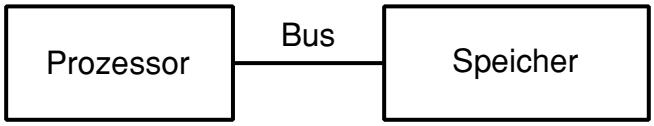

Illustration 1.1: Einfachste Darstellung eines Computers

Man kann sich den Speicher wie eine lange Liste mit Zeilennummern vorstellen. In jeder Zeile steht ein Befehl.

Veranschaulicht sieht das in etwa so aus:

Zeile	**Befehl**
1	Gehe einkaufen!
2	Putze das Fahrrad!
3	Staubsauge das Wohnzimmer!
4	Lese die GAG-News!

Tabelle 1.1

Grundsätzlich funktioniert das so: Der Prozessor nennt dem Speicher die Zeilennummer, deren Befehl er wissen möchte. Der Speicher nennt dem Prozessor diesen Befehl, und der Prozessor führt ihn aus.

Statt Zeile sagt man beim Computer jedoch *Adresse*. Technisch umgesetzt wurde dieser Ablauf mit Hilfe eines *Programmzählers* (englisch *programm counter*, kurz *PC*), einem *Adress-* und einem *Datenbus*.

Im Programmzähler steht die Adresse, welche der Prozessor wissen möchte. Der Programmzähler fängt nach dem Anschalten bei Null an und wird nach jeder Ausführung eines Befehls automatisch erhöht. Der Programmzähler ist über den *Adressbus* mit dem Speicher verbunden. Der Speicher gibt den Inhalt der Adresse, welcher im PC steht und über den Adressbus an den Speicher gemeldet wird, auf den Datenbus aus. Der Datenbus ist ebenfalls mit dem Prozessor verbunden. Der Prozessor "sieht" so, was an der Adresse steht, dessen Adresse er über den Adressbus an den Speicher meldet. Dies ist eine ganz starke Vereinfachung des tatsächlichen Vorgangs. Aber im Prinzip funktioniert es so.

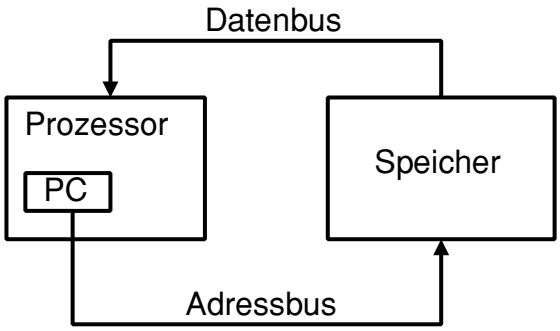

Illustration 1.2: Computer mit Programmzähler (PC), Adress- und Datenbus

Während der Mensch viele verschiedene Zeichen kennt, z. B. die Ziffern 0 bis 9 oder die 26 Buchstaben des Alphabets, arbeitet der Computer für gewöhnlich nur mit zwei verschiedenen "Zeichen". Diese können z. B. mit verschiedenen Spannungswerten verwirklicht werden (es liegt Spannung an oder nicht). Man kann sich das wie einen Schalter vorstellen, der zwei verschiedene Positionen einnehmen kann: an oder aus. Man schreibt dafür auch 0 oder 1. Jeder Schalter, z. B. ein Lichtschalter, ist damit auch automatisch ein Speicher. Er kann sich nämlich die Stellung "merken".

Illustration 1.3: Schalter offen (oben) und Schalter geschlossen (unten). Für einen offenen Schalter schreibt man auch 0 und für einen geschlossenen 1.

Würde nun jede Adresse des Speichers nur über einen einzigen Schalter verfügen, also nur die Zustände an oder aus annehmen können, könnte man nicht allzuviel machen. Denn dann könnte der Computer im besten Fall nur über zwei verschiedene Befehle verfügen. Er könnte kaum etwas unterscheiden.

Das gleiche betrifft auch den Programmzähler (PC): würde dieser nur über einen einzigen Schalter verfügen, könnte er nur die zwei Adressen 0 und 1 vom Speicher unterscheiden. Also nicht weiter zählen als bis eins. Er würde abwechselnd hin- und herschalten (an und aus, bzw. 0 und 1). Das wäre alles.

Genauso wie man die verschiedenen Ziffern 0, 1, 2, 3, 4, 5, 6, 7, 8 9 zu ganzen Zahlen miteinander kombiniert (z. B. 10, 11, 12 usw.), arbeitet man hier deshalb mit einer Vielzahl von Schaltern, mit einer Gruppe von Schaltern an jeder Adresse.

Illustration 1.4: Gruppe von acht Schaltern. Für das hier dargestellte Beispiel schreibt man hierfür auch 0100 1101 (0: offen, 1: geschlossen). Das Leerzeichen dient nur zur besseren Lesbarkeit.

Die aktuelle Adresse steht im Programmzähler. Diese Adresse wird über den Adressbus dem Speicher mitgeteilt. Die Stellungen all der Schalter an der betreffenden Adresse im Speicher werden dann vom Speicher über den Datenbus dem Prozessor rückgemeldet. Ein Bus besteht hier also aus einer Vielzahl von Leitungen und dient einem gemeinsamen Zweck, einem gleichzeitigen Vorgang.

Wieviele verschiedene Kombinationen können nun eine bestimmte Anzahl von Schalter haben? Bei einem Schalter wissen wir: er kann die Stellungen an oder aus (wir schreiben hierfür auch 0 oder 1) haben. Kombiniert man mehrere Schalter, schreibt man die möglichen Kombinationen am besten unter Verwendung eines Übertrages einfach hin. Ein Übertrag bedeutet, daß man einen weiteren Schalter umschaltet, wenn man alle möglichen Kombinationen der bisherigen Schalter durch hat. Man merkt sich so diesen Übertrag. Dann fängt man bei den hinteren Schaltern wieder von vorne an. Hier ein Beispiel mit vier Schaltern:

0000		0	
0001		1	
0010	(1. Übertrag)	2	
0011		3	
0100	(2. Übertrag)	4	
0101		5	
0110	(3. Übertrag)	6	
0111		7	
1000	(4. Übertrag)	8	
1001		9	
1010	(5. Übertrag)	10	(1. Übertrag)
1011		11	
1100	(6. Übertrag)	12	
1101		13	
1110	(7. Übertrag)	14	
1111		15	

Tabelle 1.2

Man hat eine gewisse Anzahl von verschiedenen Zeichen, die man durchzählt. Sind alle Zeichen durch, fügt man eine Stelle hinzu (Übertrag) und fängt wieder von vorne an. Das funktioniert ungeachtet der Anzahl von Zeichen immer gleich. Hat man weniger Zeichen zur Verfügung, findet ein Übertrag früher statt. Der Übertrag steht normalerweise immer davor. Die linke Stelle ist damit die Höchstwertige, die rechte die niedrigwertigste.

Wir sehen also: Mit einem Schalter sind zwei verschiedene Kombinationen möglich. Mit zwei Schaltern vier. Mit drei Schaltern acht. Mit vier Schaltern sind es bereits sechszehn. Die Anzahl der möglichen Kombinationen ist damit 2^n (2^n bedeutet, die Zahl 2 wird n mal mit sich selbst malgenommen, also z. B. für $n = 3$: $2^3 = 2 * 2 * 2 = 8$). Mit acht Schaltern hat man bereits $2^8 = 256$ mögliche Kombinationen. Auf diese Anzahl hat man sich anfangs für sehr

viele Computer geeinigt wie den Commodore 64 oder Schneider CPC.

Weil viele Schalter schnell sehr unübersichtlich werden, kann man diese anders darstellen. In Tabelle 1.2 sehen wir rechts die Folge im Zehnersystem. Dieses System verwendet zehn verschiedene Zeichen (0, 1, 2, 3, 4, 5, 6, 7, 8 und 9) und lernen wir in der Schule. Wir können statt 1111, wie in der ersten Spalte angegeben, also auch 15 schreiben (das hat dann die Bedeutung: vier Schalter sind gesetzt). Es gibt hier eine eindeutige Zuordnung.

Weil es sich besser aufgeht, hat man dem Zehnersystem die sechs weiteren Zeichen A, B, C, D, E und F hinzugefügt. Es handelt sich dabei um das Hexdezimal- oder Sechszehnersystem. Ergänzen wir die Tabelle 1.2 damit und schreiben wir das ganze noch einmal hin:

		Dezimalsystem (Zehnersystem)		Hexdezimalsystem (Sechszehnersystem)	
0 0000		0		0	
0 0001		1		1	
0 0010	(1. Übertrag)	2		2	
0 0011		3		3	
0 0100	(2. Übertrag)	4		4	
0 0101		5		5	
0 0110	(3. Übertrag)	6		6	
0 0111		7		7	
0 1000	(4. Übertrag)	8		8	
0 1001		9		9	
0 1010	(5. Übertrag)	10	(1. Übertrag)	A	
0 1011		11		B	
0 1100	(6. Übertrag)	12		C	
0 1101		13		D	
0 1110	(7. Übertrag)	14		E	
0 1111		15		F	
1 0000	(8. Übertrag)	16		10	(1. Übertrag)

Tabelle 1.3

Wir haben dem ganzen noch eine weitere Stelle bzw. Zeile hinzugefügt. Wie man in Tabelle 1.3 sieht, findet im Hexdezimalsystem der erste Übertrag erst an 17-ter Stelle oder Zeile statt. Damit entsprechen vier Schalter mit den möglichen Kombinationen an (0) und aus (1) immer *einer* Stelle dem System in der dritten Spalte. Wenn man sechszehn verschiedene Zeichen verwendet, läßt sich mit nur einem dieser Zeichen immer eindeutig eine mögliche Kombination von vier Schaltern darstellen. Das ist von Vorteil.

Natürlich lassen sich diese Systeme leicht verwechseln, da man ja mitunter die gleichen Zeichen hinschreibt. Ohne weitere Angabe ist unklar, was 10 sein soll. Man muß deshalb immer angeben,was gemeint ist!

In der ersten Spalte von Tabelle 1.3 verwenden wir *zwei* verschiedene Zeichen. Deshalb heißt dieses System Dualsystem (von lat. duo = zwei).

In der zweiten Spalte von Tabelle 1.3 verwenden wir *zehn* verschiedene Zeichen. Deshalb heißt dieses System Dezimal- oder Zehnersystem.

In der dritten Spalte von Tabelle 1.3 verwenden wir *sechszehn* verschiedene Zeichen. Deshalb heißt dieses System Sechszehner oder Hexdezimalsystem. Zur Markierung schreibt man oft ein & vor die Zahl. Also z. B. &1F.

Statt von Schaltern spricht man in der Fachsprache jedoch von *Bits*. Spricht man von Bitbreite, ist damit gemeint, mit wievielen Bits das System gleichzeitig arbeitet. Typisch sind 8 Bit, 16 Bit, 32 Bit oder mittlweile 64 Bit.

Ein *Byte* sind acht Bit. Viele frühe Computer wie der Commodore 64 oder Amstrad CPC arbeiten mit einem Byte bzw. acht Bit. Dazu zählt aber auch der relativ neue Mega65 von Trentz Elektronik, welchen man durchaus als Nachfolger des nie auf den Markt gekommenen Commodore 65 sehen kann. Der Acorn Archimedes arbeitet mit 32 Bit oder 4 Byte. Diesen 32 Bits gibt man auch die Einheit *Word* (englisch für Wort). Ein Word sind also 4 Byte oder 32 Bit.

$$32\ \text{Bit} = 4\ \text{Byte} = 1\ \text{Word}$$

Dies gilt jedoch nicht immer. Auf anderen Systemen kann ein Word auch 2 Bytes oder 16 Bit umfassen.

$2^{10}\,Bytes = 1024\,Bytes$ entsprechen einem Kilobyte [Kb]. $2^{10}\,Kilobytes = 1024\,Kilobytes$ entsprechen einem Megabyte [MB]. $2^{10}\,Megabytes = 1024\,Megabytes$ entsprechen einem Gigabyte. $2^{10}\,Gigabytes = 1024\,Gigabytes$ entsprechen einem Terrabyte. Man sieht also, daß die Einheiten nicht immer um den Faktor Tausendfach, sondern um den Faktor Tausendvierundzwanzig steigen.

Nun enthält der Speicher nicht nur *Befehle* für den Prozessor, sondern auch *Daten*. Und der Prozessor kann den Inhalt einer Adresse nicht nur *lesen*, sondern auch *schreiben*. Mit Schreiben ist gemeint, daß er die Schalter umstellen, die Bits ändern kann. Über einen dritten, nämlich dem *Steuerbus*, teilt der Prozessor dem Speicher mit, ob er die Adresse lesen oder schreiben, also den Inhalt ändern möchte.

Ob der Inhalt einer Adresse als Befehl oder als Daten verstanden werden muß, hängt von der Logik der Befehle und vom Programm ab. Hierbei können Fehler passieren. Diese führen dazu, daß ein Programm nicht richtig, nicht wie gedacht funktioniert. Denn der Prozessor kann das Programm, welches er gerade abarbeitet, auch überschreiben und damit verändern oder löschen.

Über den Speicher kommuniziert der Prozessor aber auch mit anderer Elektronik. Man kann

sich das so vorstellen: Wird auf der Tastatur[1] eine Taste gedrückt, ändert sich an einer ganz bestimmten Adresse des Speichers der Inhalt. Von der Tastatur wird an dieser Adresse ein ganz bestimmtes Bitmuster, ein ganz bestimmter Wert gesetzt, welcher eindeutig einer Taste zugeordnet werden kann. Die Schalter werden nach einer Tabelle der gedrückten Taste entsprechend umgelegt. Der Prozessor kann diesen Wert an dieser Adresse auslesen und weiß damit, welche Taste gerade eben gedrückt worden ist. Der Wert an dieser Adresse darf in diesem Fall eben nicht als Befehl verstanden werden.

Auf der Tastatur findet man die 26 Buchstaben des Alphabets, die Ziffern 0 bis 9 sowie diverse Sonderzeichen. Diese Zuordnung von Zeichen und Wert hat man in der ASCII[2]-Tabelle festgelegt. Der ursprüngliche ASCII-Code ist sieben Bit ›lang‹, das heißt er besteht aus einer Folge von sieben Bit. Ein um Sonderzeichen wie den deutschen Umlauten oder dem scharfen S erweiteter ASCII-Code ist acht Bit lang. Dies dürfte der Grund für die gleichzeitige Verarbeitung von acht Bits früherer Maschinen sein oder warum man sich auf acht Bit festgelegt hatte. Sieben Bits wären auch unpraktisch gewesen wegen der Umrechnung der Zahlensysteme. Sieben ist eben kein Vielfaches von zwei. Und mit weniger Bits hätte man einfach nicht genug Zeichen von der Tastatur abbilden, unterscheiden können.

Das Gesagte gilt auch für andere Geräte wie dem Monitor, dem Lautsprecher oder irgendwelche Ein- oder Ausgänge. So kann der Prozessor an einer ganz bestimmten Adresse einen ganz bestimmten Schalter umlegen und so einen Ausgang plötzlich auf Spannung umschalten. In diesem Fall wirkt so ein Bit tatsächlich wie ein Schalter.

Es ist natürlich eine ganz blöde Idee, an einer solchen Stelle, also an einer solchen Adresse Information oder einen Befehl hinterlegen zu wollen. Information oder Befehl wären schließlich weg, sobald eine Taste gedrückt würde. Deshalb ist es so wichtig, den Speicher des Systems, das man programmieren will, also die Hardware ganz genau zu kennen!

Bei Mikrokontrollern ist es oft so, daß dort nur Befehle ablaufen, welche man selbst für dieses System eingegeben hat. Man muß dort also keine Rücksicht auf andere schon bereits im Speicher vorhandenen Programme oder Befehle nehmen. Das macht es etwas leichter. Allerdings heißt das auch, daß sich solche Mikrokontroller nur von anderen Computern aus programmieren lassen. Sie selbst sind ohne Programm ja nicht arbeitsfähig. Ohne Programm können sie aber auch nicht programmiert werden.

Die Computer mit Tastatur und Bildschirm sind üblicherweise kurz nach dem Einschalten arbeitsfähig. Das heißt, man kann irgendwas mit ihnen machen. Man kann auf der Tastatur Tasten drücken und sieht irgendwas auf dem Bildschirm. Damit das so funktioniert, müssen diese Computer nach dem Einschalten bereits ein oder mehrere Programme gestartet haben. Diese Programme befinden sich dann bereits im Speicher. Es handelt sich dabei meist um das Betriebssystem. Es ist eine schlechte Idee, diese Programme im Speicher zu überschreiben. Denn dann funktioniert ja irgendwann der Computer nicht mehr.

Man muß als Programmierer neben der Speicherbelegung durch die Hardware auch noch die Speicherbelegung durch das Betriebssystem kennen und wissen, wie man seine Programme so

[1] Die Tastatur dürfte so ziemlich das erste Eingabegerät gewesen sein.
[2] ASCII: American Standard Code for Information Interchange (1968)

für das Betriebssystem gestaltet, daß es sich reibungslos in das System einfügt. Allerdings kann man von seinem Programm aus dann auch auf schon vorhandene Folgen von Befehlen (*Betriebssystemroutinen*) des Betriebssystems zurückgreifen.

Prozessoren verfügen immer über mehrere Einheiten von gemeinsamen Schaltern. Diese Einheiten heißen *Register*. Ein Register entspricht von der Struktur her dem Inhalt einer Zeile oder Adresse im Speicher (siehe auch Illustration vier). Dort wird Information hinterlegt. Damit arbeitet und rechnet der Prozessor. Der Programmzähler ist ein spezielles Register. In diesem wird mit jedem Takt des Systems der Inhalt um eins erhöht, also um eins weitergezählt oder die Schalter bzw. Bits entsprechend umgeschaltet wie in Tabelle 1.3 aufgeführt.

Man kann auch den Inhalt in diesem Register namens Programmzähler verändern. Dafür gibt es einen Befehl. Der Prozessor holt sich den nächsten Befehl dann vom Speicher, dessen Addresse im PC steht. Damit können Sprünge im Speicher oder im Programm realisiert werden.

2. Einführung in die ARM und RISC OS

Maschinencode

Dateien, welche ausführbare Maschinenprogramme enthalten, haben unter RISC OS den Dateityp *Absolute*.

In dem Programmverzeichnis !StrongED findet man eine Datei namens !RunImage. Diese hat den Dateityp *Absolute* und enthält Maschinencode. Das Programmverzeichnis läßt sich öffnen, indem man eine der Umschalttasten gedrückt hält und gleichzeitig einen Doppelklick mit der Maus darauf anwendet.

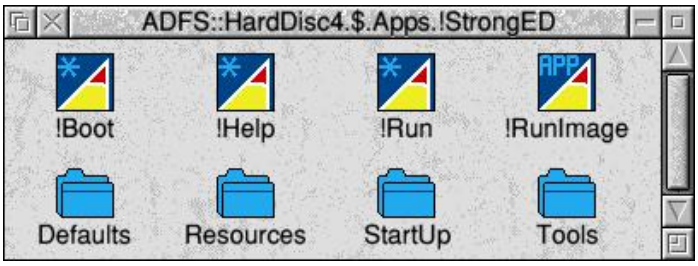

Illustration 2.1: Die Datei mit dem Namen !RunImage hat hier den Dateityp *Absolute*.

Den Inhalt einer solchen Datei kann man sich sinnvollerweise mit einem der mächtigen Editoren !StrongED oder Zap anzeigen lassen. In !StrongED schaltet am besten auf den Modus ASM um.

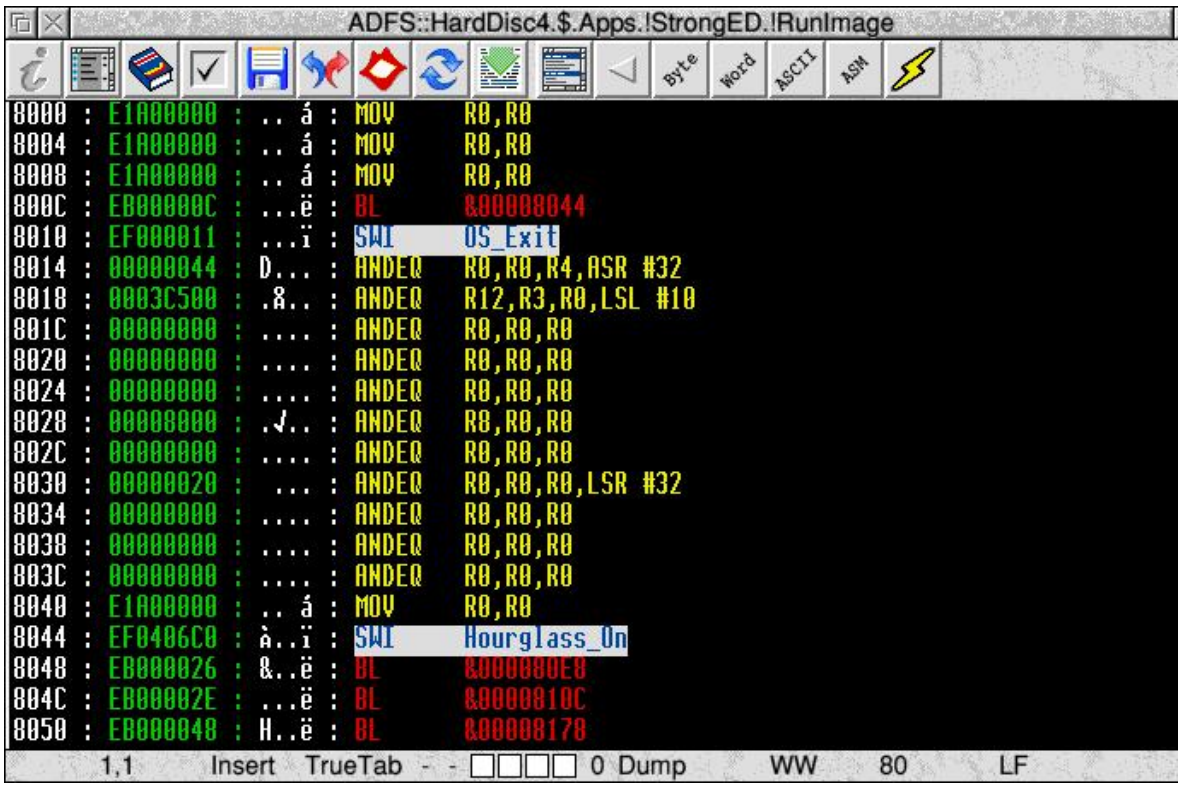

Bild 2.2: Die Datei !RunImage aus dem Verzeichnis !StrongED in !StrongED angezeigt.

In der ersten Spalte sieht man in weiß die Speicheradressen. Hierbei fällt auf, daß jede Zeile mit jeder vierten Adresse anfängt. Alle acht Bits haben eine gemeinsame Adresse. Der Speicher ist damit byteorientiert. Jede Adresse spricht ein Byte oder eine Gruppe von acht Bit an. Auf einem ARM-Prozessor, der mit 32-Bit arbeitet, sind die Befehle jedoch 32 Bits oder vier Bytes lang. Deshalb beginnt jede Zeile mit jeder vierten Adresse.

Der Programmzähler war auf der ARM ursprünglich in Register 15 enthalten. Register 15 war damals aber *nicht* dem Programmzähler gleichzusetzen. Der Programmzähler war nur 26 Bit lang. Die oberen 6 Bits wurden für etwas anderes genutzt.[3] Das hat sich in der Zwischenzeit geändert und dürfte das Problem dafür sein, warum RISC OS und diverse Programme angepaßt werden mußten.

Mit 26 Bits konnten die alten ARM-Prozessoren auch nur $2^{26} = 67108864$ *Bytes* oder *64 Megabyte* Speicher ansprechen. Mit 32 Bits sind $2^{32} = 4$ *Gigabyte* möglich.

Die untersten zwei Bits des Programmzählers sind immer null. Deshalb springt der Programmzähler immer 4 Bytes weiter.

In der zweiten Spalte sieht man grün den Maschinencode in hexdezimaler Form. In der dritten Spalte, wieder weiß, sieht man die Werte aus Spalte zwei als ASCII-Zeichen gedeutet und dargestellt. Bei den letzten beiden Spalten handelt es sich noch einmal um eine andere

[3] siehe RISC OS 3 Programmer's Reference Manual Volume 1, Seite 1-14.

Darstellung der Werte aus Spalte zwei. Hier wird der Speicherinhalt als Mnemonics dargestellt. Mnemonics sind nur eine andere Darstellung von Maschinencode. Und zwar in einer Art und Weise, die der Mensch besser lesen kann. Es handelt sich um Assemblerbefehle. Diese hat uns !StrongED aus den Werten in Spalte zwei errechnet. Es bedeutet genau dasselbe.

Startet man diese Datei namens !RunImage mit dem Dateityp *Absolute* durch einen Doppelklick mit der Maus, so lädt RISC OS diese Datei und hinterlegt sie im Speicher ab der Adresse & 8000. Anschließend wird der Programmzähler auf diese Adresse gesetzt. Der Prozessor holt sich jetzt von dort den ersten Befehl und arbeitet das gerade eben geladene und gestartete Programm ab.

Wenn man sich jetzt viele verschiedene solcher Dateien mit dem Dateityp Absolute ansieht, wird man feststellen, daß *jedes* dieser Programme bei der hexdezimalen Adresse &8000 beginnt.

Das ist insofern verwunderlich, weil unter RISC OS mehrere Programme gleichzeitig laufen können. Denn das hieße ja, daß jedes Programm das andere überschreiben würde.

Daß dem nicht so ist, nicht sein kann, sollte klar sein. In früherer Zeit befand sich zwischen dem Prozessor und dem Speicher noch ein weiterer Chip namens MEMC[4], welcher den Speicher verwaltete. Dieser Chip wies dem Programm dann den tatsächlichen Speicherort zu. Die verschiedenen Programme können über eine Tabelle eingeblendet werden. Für den Prozessor sieht es immer so aus, wie wenn sich nur ein einziges Programm im Speicher befände. Inzwischen wurde diese Funktion vom MEMC in den Prozessor selbst integriert.

So ein Programm im Maschinencode läßt sich mittels !Zap oder !StrongED analysieren. Es startet immer mit der Adresse &8000.

Bevor der Prozessor das Programm anspringt, schreibt es noch den aktuellen Wert des Programmzählers in Register 14. Will man nun das Programm beenden, muß man nur den Programmzähler auf die Adresse setzen, welche im Register 14 hinterlegt wurde. Damit sind wir auch schon beim ersten notwendigen Befehl: `E1A0 F00E` oder als Mnemonik[5] geschrieben: `MOV PC, R14`.

Bei Mnemonics handelt es sich um eine andere Schreibweise der Befehle. Diese können mittels einem Assembler wieder ein Maschinensprache rückübersetzt werden. Eingeführt hat man diese als Mnemonik bezeichnete Befehle als Gedächtnisstütze.

Wir können nun in !StrongED ein neues (leeres) Dokument erzeugen, indem wir auf das Symbold von !StrongED auf der Symbolleiste klicken. Dieses leere Dokument speichern wir unter einem Dateinamen ab. Dabei geben wir ihm gleichzeitig den Dateityp *Absolute*. Alternativ können wir den Dateityp auch später über das Dateisystem ändern. Auf jeden Fall sollten wir dann die Datei schließen und wieder neu öffnen (laden), indem wir sie auf das Symbol von

---

[4] siehe RISC OS 3 Programmer's Reference Manual Volume 1, Seite 1-10.

[5] Mnemotechnik: Kunst, das Gedächtnis durch Lern- oder Gedächtnishilfen zu stärken, Gedächtniskunst (Bertelsmann Universal Lexikon, 1996 Lexikographisches Institut, München, ISBN 3-577-10625-5)

!StrongED auf der Symbolleiste fallen lassen. Alternativ können wir sie öffnen, indem wir einen Doppelklick mit der Maus bei gleichzeitig gedrückter Umschalttaste anwenden.

Jetzt brauchen wir den BaseMode Dump. Wir finden ihn im Menü von !StrongED (zum Öffnen des Menüs mittlere Maustaste oder Scrollrad drücken) unter dem Eintrag `BaseMode -> Change mode -> Dump`. Dann klicken wir in der Werkzeugleiste auf `ASM`.

Nun geben wir dort den Befehl

```
E1A0 F00E
```

ein. (Das Leerzeichen dient nur zur besseren Lesbarkeit und darf nicht mit eingegeben werden.) Mittels einem Doppelklick auf das Symbol der Datei im Dateisystem können wir das Programm starten.

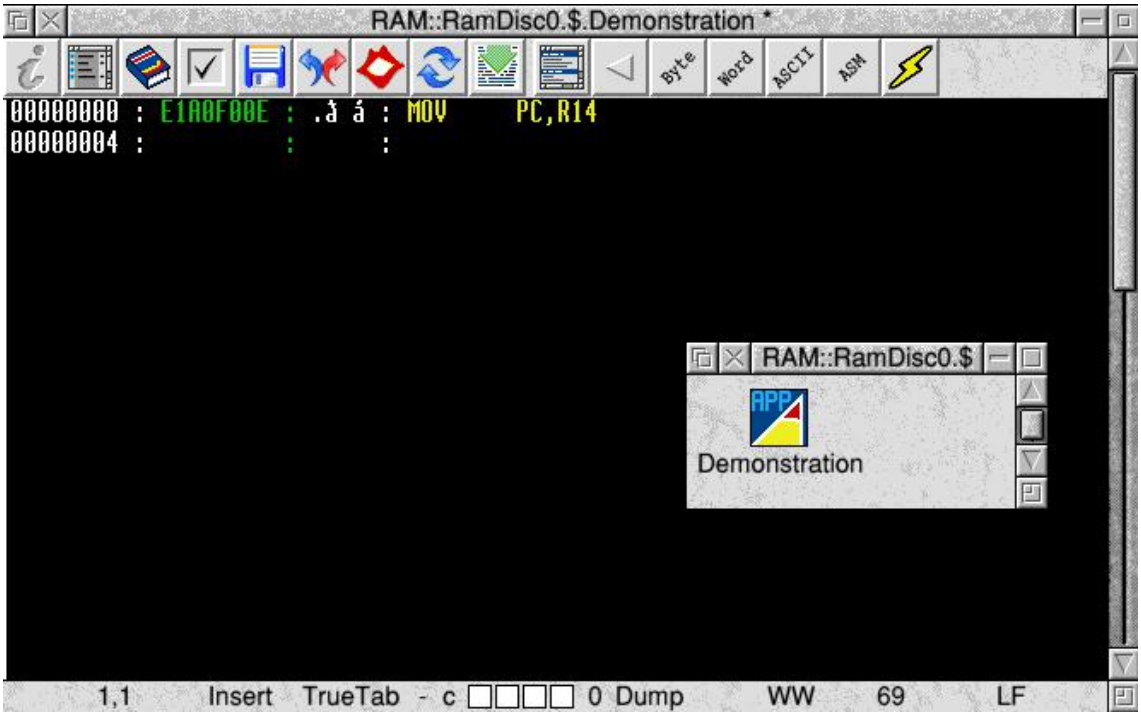

Bild 2.3: Das erste lauffähige Programm in Maschinensprache

Es scheint sich nichts zu tun. Es tut aber doch etwas: Es gibt die Kontrolle sofort wieder ans Betriebssystem zurück.

Wir können den Befehl im Editor ändern in

```
E1A0 E00F
```

Wie aus der Darstellung in der rechten Spalte ersichtlich ist, sind jetzt Quell- und Zielregister

vertauscht. Bei &E und &F handelt es sich also um die beiden Register 14 und 15. Dieser Befehl schreibt den aktuellen Wert von Register 15 ins Register 14. Diesen Befehl sollten wir an dieser Stelle jedoch tunlichst vermeiden.

Wir können dem Befehl `E1A0 F00E` einen weiteren Befehl voranstellen und so unser Programm erweitern:

```
E1A0 0000
E1A0 F00E
```

Der hinzugefügte Befehl `E1A0 0000` schreibt den aktuellen Wert von Register 0 ins Register 0. Das ist Unsinn. Denn dort steht ja bereits dieser Wert! Der Befehl beschäftigt die Maschine, tut aber tatsächlich nichts. Er sollte hier aber zur Übung dienen.