Ulrich Rohde

Computer für Anfänger

Teil 1

Mit dieser Serie soll jedem der Einstieg in die Computerszene leichtgemacht werden, der sich für mehr als nur vordergründige Anwendungen interessiert. Welche Vorlieben man dann entwikkeln möchte, ob man zum Beispiel Hardware-Fan werden möchte, oder ob man kühner Software-Konstrukteur werden möchte – das ist später in erster Linie eine Frage des persönlichen Geschmacks und der persönlichen Vorbildung.

Wie funktionieren Computer?

Nichts ist heute eine verschwommener beantwortete Frage als die nach dem Funktionieren von Computern. Experten sagen, das sei ganz einfach, und erklären dann tagelang. Philosophen neigen zu Negativ-Definitionen: Ein Computer wird den Menschen in seiner Denkfähigkeit nie erreichen! Laien haben Angst und suchen nach Dingen, die ein Computer nie können wird oder sie sind fasziniert und dem Computer und seinen Spielmöglichkeiten verfallen. Auch gestandene Praktiker, Ingenieure oder Programmierer, die die Computer und Mikrocomputer täglich vor Augen haben, weichen der Frage, wie ein Computer funktioniert, gerne aus. Eher wird betont, was man damit machen kann.

Der Ursprung des Wortes Computer

Geboren wurde das Wort Computer im angelsächsischen Sprachraum. „To compute" heißt einfach „rechnen" auf Englisch. Und Computer ist eben der Rechner. Man könnte noch mit Latein anfangen und das Wort computare in die Debatte werfen, aus dem alles abgeleitet ist. In den frühen Veröffentlichungen hießen allerdings alle Apparate, die man zur automatischen Berechnung bauen wollte, „Rechenmaschine". Eine der berühmtesten war die Analytical Engine von Charles Babbage, ein Wunderwerk aus Tausenden von Rädchen und Getrieben, das als der erste frei programmierbare Computer überhaupt angesehen werden kann. Später baute man „calculating machines", bis sich dann nach dem zweiten Weltkrieg das Wort Computer einbürgerte.

Vom Rechnen

Das, was alle Computerkonstrukteure früher (und auch heute) antreibt, ist wohl die Mischung aus dem Wunsch, eine der stursten Sklavenarbeiten, das „mechanische" Rechnen, zu erleichtern, und dem intellektuellen Bemühen, das Wesen des Rechnens überhaupt zu durchdringen. Gerade diese Komponente hat zum Beispiel bei dem Computerpionier Alan M. Turing dazu geführt, daß er sich in England während des Zweiten Weltkrieges erfolgreich am Bau des sagenumwobenen Computers Colossus beteiligte, der wiederum den als einbruchssicher bezeichneten Code der deutschen Schlüsselmaschine Enigma zu brechen geholfen haben soll. Turing hat nämlich vor seinen ganz praktischen Arbeiten an Colossus theoretisch klargelegt, was Rechnen heißt und in welcher Weise Maschinen dazu etwas beitragen können.
Turings Gedanken sind so formuliert, daß sie sich zunächst nur Mathematikern erschließen. Trotzdem kann man die Grundlagen der Funktionsweise und der Leistungsfähigkeit von Computern auch als Newcomer ganz schnell und ohne Qualitätseinbußen verstehen, wenn man sich mit ähnlichen Gedanken beschäftigt, die aber näher an der normalen Schulmathematik (und zwar Grundschule!) liegen. Die Ideen dazu stammen von Prof. Cohors-Fresenborg, Universität Osnabrück.

Zurück in die Schule

Erinnern Sie sich an die Schule. Dort wurde Ihnen, wenn Sie es noch nicht konnten, das Zählen beigebracht. Und addiert haben Sie da vielleicht, indem Sie mit der einen Hand zum Beispiel die Zahl 4 gebildet haben und mit der anderen die Zahl 5. Die Summe beider Zahlen war einfach die Anzahl der Finger beider Hände. Das Zählen allein hat Ihnen damals zum Ergebnis verholfen. Erst bis Vier an der einen Hand, dann weiter noch Fünf dazu an der anderen Hand, was 9 ergibt. Gewissermaßen haben Sie ganz materiell die Zahlen gehandhabt. Es ist der Mühe wert, die Zahlen noch einmal mit unvorgebildeten Augen anzusehen, sie zum Beispiel mit Streichhölzern nachzulegen. Man kann dann, vorausgesetzt, man besitzt genügend Streichhölzer, jede natürliche Zahl „aufbauen", indem man eine solche Zahl zählend durch das Hinzulegen jeweils eines Streichholzes bei jedem Zählschritt materiell konstruiert.

Die Streichholz-Mathematik

Sind zwei Streichholzanzahlen gegeben, dann kann man durch Hinzuzählen von Hölzern auf den einen Haufen und Wegnehmen vom anderen sogar die Summe dieser beiden Anzahlen bilden – wie in der Schule damals mit den Fingern. Als elementare Handlungsweise ist nur das Hinzulegen oder Wegnehmen eines Streichholzes notwendig: von einem Haufen weg und zum anderen hinzu.

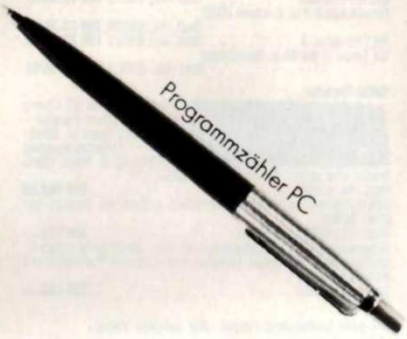
Jetzt müssen Sie sich wieder genau an die Schulmathematik zurückerinnern: Dort hatten Sie durch mehrfaches Addieren ein und derselben Anzahl zum Beispiel auch das Multiplizieren gelernt. Da addiert man den einen Faktor ebensooft zu sich selbst, wie es der andere Faktor angibt.
Auch die Subtraktion zweier Zahlen war in der Schule so lange kein Problem, wie man nur eine kleinere Zahl von einer größeren abziehen wollte. Daß man beim Subtrahieren, wenn der Minuend kleiner als der Subtrahend ist, das, was beim Abziehen sozusagen als nicht durch den Minuenden gedeckt übrigbleibt, mit einem Merkzeichen versehen kann, dem

KNOW TECHNIK GEWUSST WIE HOW

DER KNOW HOW COMPUTER

entwickelt von Wolfgang Back in Zusammenarbeit mit der Zeitschrift mc

PROGRAMM-SPEICHER	PROGRAMM-SPEICHER-ZELLEN-NUMMERN
j 4	1
+ 1	2
– 2	3
0 2	4
j 2	5
Stop	6
	7
	8
	9
	10
	11
	12
	13
	14
	15
	16
	17
	18
	19
	20
	21



Die Befehle:

+ = Addiere 1 zum Inhalt von Datenregister XX und erhöhe PC um 1

– = Subtrahiere 1 vom Inhalt von Datenregister XX und erhöhe PC um 1

j = Setze PC auf XX

0 = Prüfe, ob der Inhalt vom Datenregister XX gleich 0 ist. Wenn ja, dann erhöhe PC um 2; wenn nein, dann erhöhe PC um 1

Stop = Stop

DATENREGISTER	
1	
2	
3	
4	
5	
6	
7	
8	

negativen Vorzeichen, und dann als negative Zahl behandeln kann, das war sicher eine kleine intellektuelle Schwierigkeit beim Rechnenlernen, aber das ist Ihnen heute in Fleisch und Blut übergegangen.

PROGRAMM-SPEICHER	PROGRAMM-SPEICHER-ZELLEN-NUMMERN
i 4	1
- 1	2
- 2	3
0 1	4
j 7	5
j 12	6
0 2	7
j 2	8
Stop	9
+ 1	10
- 2	11
0 2	12
j 10	13
+ 2	14
Stop	15
	16
	17
	18
	19
	20
	21

Bild 2. Das Subtraktionsprogramm

Die Rechenarten sind aus dem „Zählen" entwickelbar

Die Erkenntnis, die hier wichtig ist: Man kommt durch die ganze „berechenbare" Mathematik nur mit Zählen. Genauso, wie das Multiplizieren durch fortgesetzte Addition zu erledigen ist, kann man das Dividieren durch fortgesetzte Subtraktion erledigen. Und wenn ein Rest bleibt, die Division nicht aufgeht, dann könnte man den Rest durch Multiplikation mit Zehn so lange aufblasen, bis man erneut den Divisor fortgesetzt subtrahieren kann. Mit einem Komma und einer Anmerkung, wie oft man schon mit 10 den Rest aufgemöbelt hat, also der Stellenverschiebung nach rechts, kann man so Dezimalbrüche herstellen.

Diese Bemerkungen sollen alle nur dazu dienen, Ihnen das Gefühl zu geben, daß es wirklich reicht, wenn man zählen kann, um durch die kompliziertesten Berechnungen zu kommen. Wobei noch hinzugefügt sei, daß man entscheiden können muß, ob man genügend gezählt hat. Bei der „zählenden Addition" muß man ja wissen, wann alle Streichhölzer des einen Summanden zum anderen hinzugefügt sind.

Ein höheres Beispiel

Wer zum Beispiel wissen will, ob man so die e-Funktion berechnen kann, der muß die Formel

$$\exp(r) = \sum_{v=0}^{\infty} \frac{r^v}{v!}$$ kennen, die die

e-Funktion definiert.

Hier ist zunächst klar, daß die symbolisch dargestellte Reihe nie bis zuletzt ausgerechnet werden kann. Denn man kann nur endlich viele Rechenschritte in begrenzter Zeit durchführen. Aber wenn man sich mit den ersten Gliedern begnügt, dann wird klar, daß man nur eine

Anzahl Ausdrücke der Form $\frac{r^v}{v!}$ auf-

summieren muß, wobei v der Reihe nach die Werte 0, 1, 2, 3... einnimmt.

Jeden einzelnen dieser Ausdrücke kann man wiederum auflösen, zum Beispiel

für v = 3: $\frac{r \cdot r \cdot r}{1 \cdot 2 \cdot 3} = \frac{r^3}{3!}$

Das ist nun alles schon aus den elementaren Rechenarten aufgebaut. Und die wiederum können zählend erledigt werden. Also: auch so komplizierte Funktionen wie exp lassen sich aus den elementarsten Rechenschritten soweit aufbauen, wie man es vernünftigerweise verlangen kann. Es sei betont, daß es hier nur um das Prinzip geht. Versuchen Sie noch andere Beispiele von Funktionen so weit auf die Grundrechenarten zurückzuführen, daß sie im Prinzip beliebig genau damit berechnet werden könnten.

Der Know-how-Computer: Die Registermaschine in Primitiv-Ausführung

Wenn das oben Geschilderte Ihnen ein bißchen Feeling dafür gegeben hat, welche Kraft im „Zählmechanismus" steckt, dann sind Sie reif, einfach einen Sprung in die Programmierung von Computern zu tun. Sie merken schon, es kommt uns hier nicht auf hieb- und stichfest bewiesene Aussagen an, sondern um das Begreifen der Computer. Das Bild auf Seite 41 zeigt Ihnen den Know-how-Computer, der von Wolfgang Back und der mc-Redaktion entwickelt wurde. Es ist ein Papiercomputer, an dem Sie sich im Programmieren üben können. Selbst Profis werden an diesem Computer ihre Freude haben, denn alle Probleme, die beim täglichen Umgang mit Computern auftreten, können daran demonstriert werden.
Übrigens bekommen Sie den „Papier-Computer" auch einzeln als Faltblatt kostenlos vom Verlag. Senden Sie dazu bitte einen an Sie selbst adressierten und mit 1,10 DM frankierten DIN-C4-Umschlag mit der Aufschrift „Drucksache" an den Franzis-Verlag, Abt. ZV, Postfach 37 01 20, 8000 München 37.

Die Bedienungsanleitung zum Know-how-Computer

Dieser Einfachst-Computer soll die Arbeitsweise eines Normal-Computers veranschaulichen – dargestellt an den einzelnen Befehlsvorgängen. Sie können damit aber auch selbst programmieren und dabei erkennen, welche „Gedankengänge" der Computer zu leisten hat, um zum Beispiel eine Multiplikation auszuführen.
Für das erste nebenstehende Programm-Beispiel (eine Addition) benötigen Sie nur einen Bleistift oder Kugelschreiber und eine Anzahl Streichhölzer. Der Stift

dient als Programmzeiger (Program-Counter = PC).
Mit den Streichhölzern (es eignen sich auch Knöpfe oder andere Dinge) werden Sie in die Datenregister Zahlen eingeben, der Computer wird diese Zahlen dann manipulieren.

Noch ein paar allgemeine Hinweise

Legen Sie Ihren Kugelschreiber (als Programmzeiger „PC") am Anfang immer so, daß seine Spitze auf Programmspeicher-Zelle 1 deutet: also Startstellung wie eingezeichnet.

Links neben den Programmspeicher-Zellen-Nummern finden Sie die Inhalte der Zellen und können über das Befehls-Symbol (z. B. j) erkennen, welcher Befehl auszuführen ist. Nach dem Befehls-Symbol folgt eine Zahl. Sie gibt die Nummer des im Befehl angesprochenen Datenregisters oder der angesprochenen Programmspeicherzelle an. In der Mitte unten im Know-how-Computer steht die Beschreibung eines jeden Befehls. Dort ist für die jeweils aktuelle Nummer XX geschrieben. Wenn Ihr Programmzeiger also am Anfang auf 1 zeigt, dann lesen Sie bei unserem Additionsprogramm j4. In der Mitte unten lesen Sie bei j nach, was zu tun ist. Dabei setzen Sie an Stelle von XX diesmal 4 ein.

Wenn Sie eine Eins addieren sollen, dann legen Sie einfach ein zusätzliches Streichholz in das vom Programmbefehl angesprochene Datenregister zu den dort möglicherweise schon vorhandenen. Beim Subtrahieren einer Eins nehmen Sie ein Streichholz weg.

Wenn Sie den Programmzähler PC setzen wollen, dann legen Sie einfach den Kugelschreiber so, daß seine Spitze auf die im Befehl angegebene Programmspeicherzelle zeigt. Beim Erhöhen des Programmzählers lassen Sie die Stiftspitze einfach auf den nächsthöheren Befehl (oder den übernächsten, wenn verlangt) zeigen.

Sind Sie jetzt präpariert? Dann können Sie das Spiel mit dem Computer beginnen

Ein einfaches Additions-Programm haben wir für Sie schon im Programmspeicher vorbereitet. Legen Sie dazu eine Anzahl Streichhölzer in das Datenregister 1 und eine andere Anzahl in das Datenregister 2. Der Einfachheit halber sollten Sie mit kleinen Stückzahlen beginnen.

PC (Stift) muß zu Beginn so eingestellt werden (auf den Papier-Computer gelegt werden), daß seine Spitze auf die Programmspeicher-Zelle 1 zeigt.

Lesen Sie den Inhalt der angezeigten Programmspeicher-Zelle und vergleichen Sie anhand des ersten Symbols, welcher Befehl ausgeführt werden soll.

Der erste Befehl lautet: j 4

Das erste Symbol ist ein j, also führen Sie den Befehl j aus. Sie sollen PC auf Programmspeicher-Zelle 4 setzen. Also lassen Sie bitte die Stiftspitze auf Programmspeicher-Zelle 4 zeigen. Dort steht 0 2.
Der Befehl lautet 0, bitte führen Sie diesen Befehl durch. An Stelle von XX setzen Sie die in der Programmspeicher-Zelle angegebene Zahl ein (in diesem Fall 2). Sie sollen also prüfen, ob der Inhalt des Datenregisters 2 Null ist.
Angenommen es liegt mindestens 1 Stäbchen in Datenregister 2, dann bewegen Sie die Schreibspitze um eine Speicherzelle nach unten. Dort (in der Programmspeicher-Zelle 5) steht j, also Befehl j durchführen, also jetzt PC auf Programmspeicher-Zelle 2 setzen.
Der neue Befehl lautet +. Bitte führen Sie diesen Befehl durch: Sie erhöhen in Datenregister 1 die Anzahl der Stäbchen um 1 Stück und erhöhen PC um 1 Stelle. Der Stift zeigt danach auf 3. Dort steht der Befehl −. Diesen Befehl wiederum durchführen, d. h. Sie nehmen aus Datenregister 2 ein Stäbchen weg und erhöhen PC um 1 Stelle.
In dieser Weise fahren Sie weiter fort, bis Sie zu Programmspeicher-Zelle 6 mit „Stop" gelangen. An dieser Stelle sind der Rechenvorgang und damit die einzelnen „Computer-Gedankengänge" beendet.
Was tut unser Trainingsprogramm? Es addiert einfach die Zahlen in Register 1 und 2. Das Ergebnis steht in Register 1. So „einfach" denkt der Computer. Prüfen Sie das nach, indem Sie verschiedene Summen bilden.

Ein Subtraktions-Programm

Und so arbeitet das Programm (*Bild 2*): Es erwartet zwei (positive) Zahlen in Register 1 und in Register 2 und zieht die Zahl in Register 2 von der in Register 1 ab.
Mit den Befehlen von 1 bis 8 werden die Register 1 und 2 zunächst herabgezählt. Wird Register 2 eher Null als Register 1, dann ist alles in Ordnung, in Register 1 bleibt die Differenz der beiden Zahlen stehen, die Maschine stoppt in 9. Tritt aber der Fall ein, daß der Subtrahend in Register 2 größer oder gleich dem Minuenden in Register 1 ist, dann führt der Befehl in Zeile 6 in den zweiten Programmteil, in dem nun Register 1 ebensooft wieder heraufgezählt wird wie Register 2 bis zu Null herabgezählt werden kann. Danach wird zur Kennzeichnung, daß der sich ergebende Differenzbetrag in Register 1 Null ist oder negativ, das Register 2 um Eins heraufgezählt. Das Programm merkt also in Register 2 das Vorzeichen an.
Rechnen Sie jetzt das Programm mit zwei Zahlen Ihrer Wahl auf dem Papiercomputer durch.

(Fortsetzung folgt, mit mehr Programmen und mit Bemerkungen über das Verhältnis zu realen Prozessoren)

Instring in Basic

Unser Literatursuch-Programm (mc 1982, Heft 9 und mc 1983, Heft 3) verwendet ein Maschinenprogramm zur Realisation des INSTR-Befehls, den nicht alle Computer besitzen. Das Maschinenprogramm ist zwar recht schnell, aber leider auch systemgebunden. Deshalb zeigt das *Bild* eine etwas langsamere, aber dafür in Basic geschriebene Routine, die den gleichen Zweck erfüllt. Den Befehl s = @instr (m$, b$) kann man dann durch den Unterprogrammsprung GOSUB 460 ersetzen, der in obengenanntem Programm zweimal vorkommt, nämlich in der Such- und in der Editierroutine (Kommandos F und E). *Fe.*

```
460 s=0:rem" *** Instring ***
470 forl=1tolen(m$)-len(b$)+1:ifmid$(m$,l,len(b$))=b$thens=l:return
472 nextl:return
```

Diese Routine liefert in s die Position des Strings b$ in m$

Ulrich Rohde

Computer für Anfänger

Teil 2

Der erste Teil der Serie, der den Papiercomputer geschildert hat, ist in die verschiedensten Richtungen weiter ausbaubar. Einmal könnte man ganz theoretisch über die sogenannten Algorithmen nachdenken. Das sind nichts anderes als Computerprogramme. Zum anderen könnte man versuchen, diesen Papiercomputer, der Know-how-Computer heißt, weil er in der Sendereihe „Know-how, Technik – gewußt wie" zum ersten Mal gezeigt wurde, wirklich mit elektronischen Mitteln aufzubauen. Das ist möglich, eine Experimentalversion existiert. Aber beide Richtungen sind, extrem betrieben, vielleicht etwas zu weit von der Praxis entfernt. Hier in dieser Serie soll nämlich bald auch über reale Computer gesprochen werden.

Es sei nochmals betont, daß der Papiercomputer jedem, der einmal an ihm programmiert hat, ohne Umschweife zeigt, wie ein Computer arbeitet. Gerade, daß der Befehlssatz so klein ist, daß er mühelos von jedem überblickt werden kann, ist sein großer Vorteil. Sein Nachteil resultiert ebenfalls aus dem sehr kleinen Befehlssatz: Die Programme des Papiercomputers sind extrem umständlich, denn man muß sich mühsam jede einzelne Fähigkeit des Computers aus dem Auf- und Abzählen aufbauen. Das war recht einfach bei der Addition, die im Teil 1 gezeigt wurde. Das war schon recht umständlich bei der Form der Subtraktion, die im ersten Teil vorgeführt wurde. Und das ist noch umständlicher bei der Multiplikation, die jetzt programmiert werden soll.

Die Multiplikation mit dem Know-how-Computer

In der Form, wie sie hier betrachtet werden soll, hat die Multiplikation nur einen Hauptaspekt: Wie kann man sie mit den beschränkten Mitteln des Papiercomputers realisieren? Genau dieser Aspekt ist es, der dieser Aufgabe in Bezug auf echte Computer seine Realistik verleiht. Dort, beim Programmieren der großen oder kleinen Computer, treten ähnliche Fragen immer wieder auf, allerdings bei weit komplizierteren Problemen, denn eine Multiplikation ist da oft schon eingebaut.

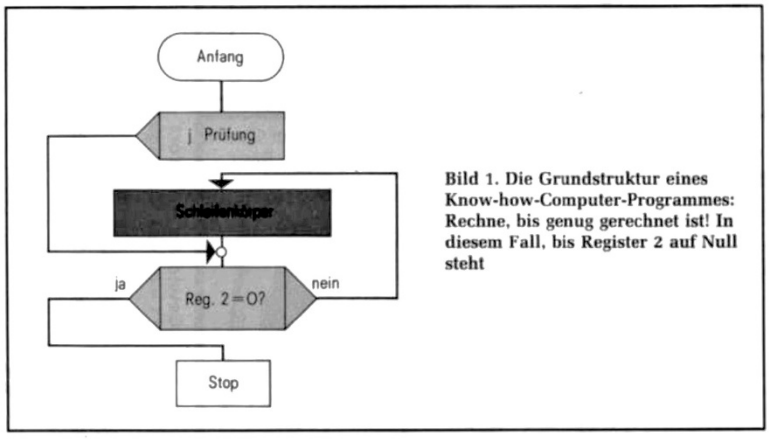


Bild 1. Die Grundstruktur eines Know-how-Computer-Programmes: Rechne, bis genug gerechnet ist! In diesem Fall, bis Register 2 auf Null steht

Aus Additionen aufgebaut

Wenn man sich die Sache genau betrachtet, dann ist die Multiplikation eine fortgesetzte Addition, so haben Sie es bestimmt in der Schule gelernt. Und wenn Sie es recht überlegen, dann können Sie mit dem Papiercomputer programmierend die ganze Grundschulmathematik nacherfinden, denn die basiert auch nur auf dem Zählen. Wenn Sie also einen inneren Plan für die Art, wie sie die Multiplikation entwerfen wollen, sich zurecht legen, dann könnte es nützlich sein, auf die fortgesetzte Addition zurückzugreifen.
Falls Sie sich dazu entschlossen haben, dann entsteht sofort die Frage, welchen der beiden Faktoren Sie fortgesetzt zu sich selbst addieren wollen und welchen Sie zum Abzählen – ob schon genügend Summanden im Produkt aufaddiert worden sind – benutzen wollen. Planen Sie einmal, den ersten Faktor ebensooft zu sich selbst dazuzuaddieren, wie es der zweite Faktor angibt. Planen Sie weiter, daß die Faktoren in Datenregister 1 und Datenregister 2 stehen und daß das Ergebnis in Datenregister 3 stehen soll.
Dann käme jetzt der Gedanke, wie denn solch ein Programm anfangen sollte. Es läge zum Beispiel die Idee nahe, gleich zu Beginn zu überprüfen, ob in Datenregister 2 die Null steht. Dann wüßte man gleich, daß es nichts zu multiplizieren gibt und könnte in diesem Fall schon abbrechen. Weil aber auch später sicher wieder zu überprüfen ist, ob Datenregister 2 schon auf Null steht oder nicht, müßte man diesen Überprüfungsbefehl später in derselben Funktion nochmals niederschreiben. Hier sei deshalb vorgeschlagen, daß zuerst ein Sprungbefehl, ein Jump, wie der Fachmann sagt (deshalb auch das j), auf die Überprüfungsstelle das Programm eröffnen soll.

Schleifen, die Grundstruktur der Papiercomputerprogramme

Es ist zunächst noch unbekannt, wohin der erste Sprung führen soll, aber die Grundstruktur der Angelegenheit ist be-

stimmt klar: Überprüfe, ob etwas getan werden muß und wiederhole solange, bis nichts mehr (oder etwas anderes) getan werden muß. Das war schon bei der Addition so. Ein Programmstück, das immer wieder durchlaufen werden muß, bis das gewünschte Ergebnis errechnet ist, das nennt man eine Schleife. Die Frage also, was innerhalb der Schleife getan werden muß, damit in Register 3 das gewünschte Ergebnis steht (*Bild 1*). Der Schleifenkörper (*Bild 2*) muß auf alle Fälle eine Stelle enthalten, an der Register 2 herabgezählt wird, denn sonst würde das Programm nie enden, wenn der zweite Faktor des zu errechnenden Produktes von Null verschieden ist. Und dann muß jetzt der Inhalt von Register 1 zum Inhalt von Register 3 hinzugezählt werden. Die Schwierigkeit, die entsteht, ist, daß bei einer Addition, wie sie im ersten Teil durchgeführt wurde, der Inhalt beider Summanden-Register verfälscht wurde. Das eine Register wurde auf Null herabgezählt, der Inhalt des anderen auf die Summe hinaufgezählt.

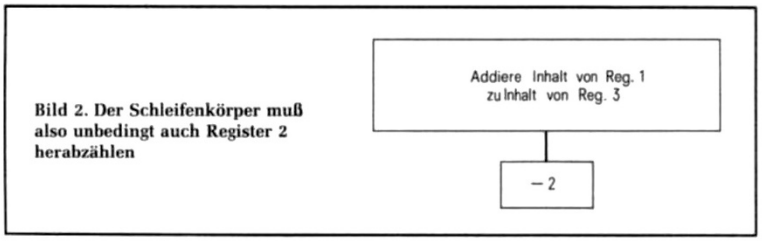


Bild 2. Der Schleifenkörper muß also unbedingt auch Register 2 herabzählen

Eine andere Addition

Das normale Additionsprogramm ist also hier nicht ohne weiteres benutzbar, denn der Inhalt von Register 1 darf nicht verlorengehen, weil er möglicherweise für einen weiteren Durchlauf benötigt wird. Hier sei nun vorgeschlagen, daß zur Rettung des Inhaltes von Register 1 eine Variante des Additionsprogrammes benutzt wird, die sich in einem Hilfsregister den Inhalt von Register 1 merkt und ihn nach Durchlaufen der Addition wieder nach Register 1 zurückzählt. *Bild 3* zeigt den Plan. Er und die Pläne vorher sind als Flußdiagramme gezeichnet, in welchen die Abfolge der einzelnen Schritte besonders anschaulich dargestellt werden können. Es muß dabei betont werden, daß es zwar in den DIN-Vorschriften und -Empfehlungen auch Kapitel über die eindeutige Gestaltung von Datenflußplänen und Programmlaufplänen gibt, daß aber die Darstellung von Programmen in Form von Flußdiagrammen bei Informatikern und Programmier-Praktikern locker gehandhabt wird. Unsere Flußdiagramme sind also nicht streng konstruiert – sie sollen nur den Programmablauf veranschaulichen.

Das Multiplikationsprogramm wird geschrieben

Nach den Vorüberlegungen scheinen die einzelnen Abschnitte des zu schreibenden Programmes klar zu sein. Erst ein Sprung zur Überprüfung der Frage, ob das Programm nicht schon gleich bei Beginn zu Ende ist, weil der zweite Faktor Null ist. Wenn nicht, dann Eintritt in den Schleifenkörper, der zunächst Register 1 nach Register 3 und 4 aufaddiert. Dabei ist noch Folgendes zu sagen: Alle Register des Papiercomputers, die nicht ausdrücklich vor Programmstart mit irgendwelchen vorgegebenen Werten geladen wurden, sollen Null enthalten. Das ist anders als bei den richtigen Computern, wo sich nach dem Einschalten oft die merkwürdigsten Werte in den Speicherzellen finden und wo ein Programmierer selbst für die richtige Voreinstellung der Speicherzellen Sorge tragen muß.
Mit dieser Voraussetzung können Sie nun das folgende Programmstück benutzen:

```
– 1
+ 3
+ 4
0 1
j 2
```

und daran anschließend

```
j 10
– 4
+ 1
0 4
j 8
```

wobei einfach fortlaufend in den Programmspeicherzellen nach dem ersten „Jump" weiterprogrammiert wurde. An diese beiden Abschnitte schließt sich das Herabzählen von Register 2 um jeweils Eins pro Durchlauf an. Und die Überprüfung, ob das Ende der Berechnungen erreicht ist.

Das gesamte Programm zeigt *Bild 4*. Überprüfen Sie, ob das Programm wirklich in allen Fällen das Verlangte auch leistet.

Vom Wert der Papiercomputer-Programme

Wenn Sie anhand unserer Anregungen schon einmal selbständig den Know-how-Computer zu programmieren versucht haben, dann hat er schon seinen Zweck erfüllt und Sie wissen Bescheid über das Programmieren. Zwar können Sie noch nicht professionell programmieren, aber das Prinzip ist klar. Viele Probleme treten auf, wenn man lange und komplizierte Programme schreiben möchte. Wie kann man so programmieren, daß man stets die Übersicht behält, was im einzelnen geschieht? Eine positive Antwort wäre ein Verfahren, das eine Methode liefert, richtige Programme zu entwerfen. Dieses Verfahren gibt es nicht. Beim Programmieren spielen erstaunlicherweise Stilfragen eine Rolle. Gute Programme sind so geschrieben, daß man sie leicht verstehen kann. Genauso wichtig, wie die Frage, mit wel-

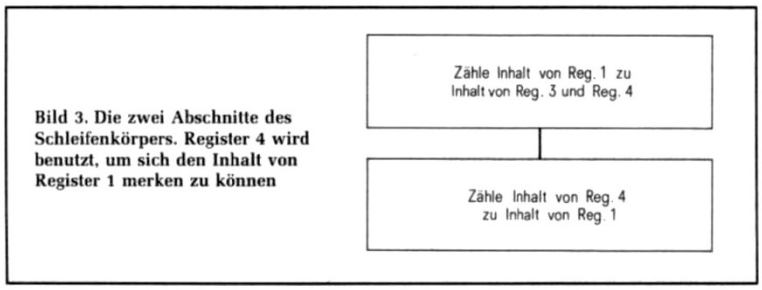


Bild 3. Die zwei Abschnitte des Schleifenkörpers. Register 4 wird benutzt, um sich den Inhalt von Register 1 merken zu können

Programmspeicher	Programm-Speicherzellen-Nummern
j 13	1
−1	2
+3	3
+4	4
01	5
j 2	6
j10	7
−4	8
+1	9
04	10
j8	11
−2	12
02	13
j5	14
Stop	15
	16

Bild 4.
Das Multiplikationsprogramm

chen Methoden man richtige Programme entwerfen kann, ist die Frage, was man alles programmieren kann. Eine Antwort darauf ist nicht leicht, trägt aber wesentlich zur Klärung dessen bei, was Rechenanlagen prinzipiell zu leisten imstande sind.

Zuerst sei betont, daß der Know-how-Computer zwar nur mit natürlichen Zahlen rechnen kann, daß aber das keine Beschränkung gegenüber normalen Computern darstellt, weil diese in einer Speicherzelle in der Regel nur einen kleinen Ausschnitt aus den natürlichen Zahlen darstellen können. Gängige 8-Bit-Mikroprozessoren zum Beispiel können nur bis 255 zählen (von 0 an).

Von ordentlichen Programmen

Vor einer angedeuteten Antwort auf die Frage, was der Know-how-Computer prinzipiell leisten kann, sollen noch einige technische Dinge geklärt werden,

Programmspeicher	Programm-Speicherzellen-Nummern
j 5	1
−1	2
+3	3
+4	4
01	5
j 2	6
j10	7
−3	8
+1	9
03	10
j8	11
j16	12
−2	13
+3	14
+5	15
02	16
j13	17
j21	18
−5	19
+2	20
05	21
j19	22
j26	23
−4	24
+3	25
04	26
j24	27
Stop	28
	29

Bild 5. Ein ordentliches Additionsprogramm

Programmspeicher	Programm-Speicherzellen-Nummern
j5	1
−2	2
+5	3
+3	4
02	5
j 2	6
j10	7
−3	8
+2	9
03	10
j8	11
j24	12
−1	13
+3	14
+4	15
01	16
j13	17
j21	18
−4	19
+1	20
04	21
j19	22
−2	23
02	24
j16	25
j29	26
−5	27
+2	28
05	29
j27	30
Stop	31
	32

Bild 6. Das ist ein ordentliches Multiplikationsprogramm

die den Überblick erleichtern. Wenn Sie zum Beispiel die Addition betrachten, dann finden Sie im Speicher nach einem Programmlauf das Ergebnis, die Summe beider Zahlen in Register 1. Wenn Sie die Multiplikation sich anschauen, dann finden Sie hier nach einem Programmlauf Unordnung im Speicher: Das Register Nr. 1 enthält einen der Faktoren und Register Nr. 3 das Ergebnis.

Es sei jetzt empfohlen, alle Programme so zu schreiben, daß alle Registerwerte, die dem Programm vor einem Lauf als Startwerte mitgegeben werden, in den ersten Registern aufeinander folgend erwartet werden und daß ein Programmlauf diese Inhalte zwar zwischendurch ändern kann, aber am Ende, vor dem Stop, wieder original herstellt. Und daß das Ergebnis oder die Ergebnisse unmittelbar nach den Eingangswerten in den Registern stehen sollen.

Die Addition könnte dann wie in *Bild* 5 aussehen.

Bild 6 zeigt eine ordentliche Multiplikation.

Programme und Funktionen

Wenn Sie den Satz der Eingabewerte einmal mit X bezeichnen, den Satz der Ausgabewerte einmal mit Y und das Programm, das aus den Werten X die Ergebniswerte Y errechnen soll, mit P, dann läßt sich in naheliegender Analogie ganz mathematisch schreiben:

$Y=P(X)$

Im Grunde ist dann ein Programm nichts anderes als eine Funktion, wie man sie in der Schule kennengelernt hat. Und zwar eine Funktion, die einer gewissen Anzahl von Argumenten, das sind die Eingabewerte in den ersten Registern, eine gewisse Anzahl von Funktionswerten, das sind die Ausgabewerte in den nächsten Registern, zuordnet.

So könnten Sie unser Additionsprogramm einmal PLUS nennen und schreiben:

$y=PLUS\ (x_1,x_2)$

Hier werden zwei Argumente von PLUS zu deren Summe y verarbeitet. Nennen Sie das Multiplikationsprogramm MAL und schreiben Sie:

$y=MAL\ (x_1,x_2)$

Hier werden zwei Argumente von MAL zu ihrem Produkt verarbeitet.

Es sei betont, daß unsere Programme PLUS und MAL nur Argumente aus den natürlichen Zahlen (einschließlich 0) vertragen und auch nur Ergebniswerte aus den natürlichen Zahlen abliefern.

Weshalb das alles erzählt wird?

Wer sich mit Rechenmaschinen beschäftigt, der sollte möglichst früh auch über deren prinzipielle Leistungsfähigkeit Bescheid wissen. Und hier ist in der Mathematik schon viel Vorarbeit geleistet worden.

Von der Berechenbarkeit

Dort gibt es eine Forschungsrichtung, die konstruktive Mathematik, in der man die ganze Mathematik nur aus effektiv durchführbaren Konstruktionen und Berechnungen aufbauen will.

Eng verknüpft ist das mit der Grundlagenforschung über Berechenbarkeit und Beweisbarkeit.

Die Fragen, um die es sich dabei dreht, können anhand des Know-how-Computers und der bisherigen Erklärungen angedeutet werden. Und zwar ist anhand von Programmen schon bewiesen worden, daß der Know-how-Computer addieren und multiplizieren kann. Außerdem kann er gut zählen. Alles, was man sich aus diesen elementaren Funktionen, im Grunde also aus den einzelnen Befehlen des Know-how-Computers „zusammenprogrammieren" kann, ist effektiv berechenbar. Das ist sehr selbstverständlich. Ganz aufregend ist aber das hier nicht bewiesene Resultat, daß alle bisher in der Mathematik definierten Begriffe von Berechenbarkeit sich als gleichwertig zum Begriff „Know-how-Computer-berechenbar" erwiesen haben. Keine Angst, das soll hier nicht weiter vertieft werden, wer sich dafür interessiert, der kann das interessante, aber auch anspruchsvolle Buch „Mathematik mit Kalkülen und Maschinen" von Cohors-Fresenborg lesen, das im Vieweg-Verlag erschienen ist. Oder ein anderes Mathematikbuch über die Grundlagen der Berechenbarkeit.

Programme in Programmen: Unterprogramme

In der Schule war es ganz leicht, aus mehreren Funktionen sich neue zusammenzubauen. Jedenfalls dann, wenn die eine Funktion ihre Werte so errechnete, daß die zweite sie als Argumente gleich benutzen konnte. Ein Beispiel, aber so nicht für den Papiercomputer geeignet, wäre

$y=\sqrt{\sin x}$,

in dem von der Wurzelfunktion der Wert der Sinusfunktion an der Stelle x übernommen wird. Das Beispiel ist deshalb nicht für den Papiercomputer geeignet, weil es so wie oben geschildert meist nicht mehr natürliche Zahlen liefert, sondern eine andere Zahlensorte, die im Papiercomputer nicht implementiert ist.

Ein anderes Beispiel sei mit PLUS und MAL aufgebaut. Wenn Sie

$y=PLUS(MAL(x_1,x_2),x_3)$

betrachten, dann sieht man, daß für das erste Argument bei PLUS diesmal das Ergebnis von $MAL(x_1,x_2)$ eingesetzt ist. Es wird einfach $(x_1 \cdot x_2)+x_3$ ausgerechnet. Hier ist es nun so, daß das Programm PLUS das Ergebnis von MAL nutzen soll. Ein anderes Beispiel wäre

$y=MAL(x_1,PLUS(x_2,x_3))$

in dem

$x_1 \cdot (x_2+x_3)$

ausgerechnet wird.

In beiden Fällen ist das jeweilige Gesamtprogramm aus den beiden „Unterprogrammen" PLUS und MAL zusammengesetzt. Allerdings kann man nicht ohne weiteres die Programme PLUS und MAL einfach als Unterprogramme hernehmen. Denn wie schon mit den Indizes an den Argumenten angedeutet, müssen die Programmteile, die als Unterprogramme zusammenspielen sollen, in der Benutzung der Register aufeinander abgestimmt sein. Dies ist ein Problem, das bei Einzelfällen immer gut lösbar ist, indem man sie konkret durchprogrammiert. Tun Sie das doch bis zum nächsten Mal mit den oben angegebenen „Rechenaufgaben". *(Fortsetzung folgt)*

□

Ulrich Rohde

Computer für Anfänger

Teil 3

Bis heute wurde geschildert, wie der Know-how-Computer aus Papier arbeitet, einige Programme wurden geschrieben und ein bißchen angedeutet, was die Computer alles können: nichts weiter als alles das, was man auf dem Papiercomputer programmieren kann. In dieser Folge sollen nun die Techniken besprochen werden, die man benötigt, um auf Computern Unterprogramme bilden zu können.

Hatten Sie beim ersten Einstieg in die Computerei mehr Technik erwartet, als hier im Kurs geschildert? Oder war Ihnen das Thema zu abstrakt und praxisfern durchgeführt? Dann müssen Sie noch etwas Geduld besitzen, denn es soll noch etwas weitergehen mit dem Theoretischen. Der Zweck der Betrachtungen: Je besser Ihr inneres Bild von den Software-Objekten ist, desto weniger müssen Sie sich später um Software-Engineering bemühen oder desto weniger laufen Sie Gefahr, daß Sie Ihre persönliche Softwarekrise nehmen müssen. Modular und strukturiert werden Sie dann ganz automatisch programmieren. Es gibt zwar schwierige Probleme, aber Sie werden diese Probleme dann adäquat angehen können.

Unterprogramme – natürliche Hierarchiestufen

In der letzten Folge wurden dem Programm zur Addition und auch dem Programm zur Multiplikation jeweils ein Name gegeben. PLUS für die Addition und MAL für die Multiplikation. Aus diesen beiden Programmen wurden neue zusammengesetzt: PLUS (x_1, MAL (x_2, x_3)) zum Beispiel. Besser gesagt: Es wäre schön, wenn man die Papiercomputer-Programme so einfach, wie es oben formal getan wurde, zu neuen Programmen zusammensetzen könnte. Denn wie sich so etwas realisieren läßt, dazu wurde bisher nichts gesagt.
Bevor ein Vorschlag zum Umbau des Papiercomputers gemacht wird, damit er auch Unterprogramme beherrschen lernt, sei noch der Vorteil der Unterprogrammtechnik betont. Gewissermaßen vergrößert man durch die Verwendung von Unterprogrammen den Befehlssatz eines Computers. Denn mit einem kurzen Unterprogrammaufruf kann man dann höchst komplizierte Dinge erledigen, ohne sie am Ort programmieren zu müssen. Hat man eine Reihe von Unterprogrammen im Computer, zum Beispiel UPR01, UPR02, UPR03, ..., dann kann man ein komplizierteres Programm daraus aufbauen (*Bild 1*).
Bild 2 zeigt, daß auch Unterprogramme aus Unter-Unter-Programmen aufgebaut sein könnten und diese wieder aus Unter-Unter-Unter-Programmen – und so fort. Oder anders gesagt, über dem Bodensatz der Maschinenbefehle eines Computers kann man, wenn dieser Computer die Unterprogrammtechnik beherrscht, von unten herauf (Bottom up) eine Hierarchie immer komplexerer Programme errichten, die entsprechend komplexe Probleme lösen können.
Im ersten Teil dieser kleinen Serie war zum Beispiel über die Funktion exp(r) sehr kursorisch diskutiert worden. Ein Programm EXP, das ein Argument r zum Start benötigen würde und daraus zum Beispiel die Summe der ersten zehn Reihenglieder, also

$$1+\frac{r}{1}+\frac{r^2}{1\cdot 2}+\frac{r^3}{1\cdot 2\cdot 3}+\frac{r^4}{1\cdot 2\cdot 3\cdot 4}+\frac{r^5}{1\cdot 2\cdot 3\cdot 4\cdot 5}+\frac{r^6}{1\cdot 2\cdot 3\cdot 4\cdot 5\cdot 6}+\frac{r^7}{1\cdot 2\cdot 3\cdot 4\cdot 5\cdot 6\cdot 7}+\frac{r^8}{1\cdot 2\cdot 3\cdot 4\cdot 5\cdot 6\cdot 7\cdot 8}+\frac{r^9}{1\cdot 2\cdot 3\cdot 4\cdot 5\cdot 6\cdot 7\cdot 8\cdot 9}$$

errechnen sollte, müßte mit Sicherheit die Unterprogramme PLUS und MAL gut gebrauchen können. Sicher auch ein Unterprogramm DIVISION und eines, das Fakultäten berechnen kann. Dieses wiederum könnte die Multiplikation als Unter-Unter-Programm gut gebrauchen. Eine solche Überlegung von oben herab (top down) zeigt auf, welche Unterprogramme nützlich wären, um das gesteckte Ziel zu erreichen.
Die etwas lange Formel oben sollte auch zeigen, daß man Unterprogramme innerhalb eines Programmes ebenfalls sehr oft an verschiedenen Stellen einsetzen können möchte. Schauen Sie sich zum Beispiel an, wie oft die Multiplikation hier benötigt wird. Allein unter den Bruchstrichen bis zu achtmal (maximal).
Das Problem beim Papiercomputer liegt in Bezug auf die Unterprogrammtechnik an folgenden Stellen: Wie kann ein Programmstück dazu gebracht werden, die verschiedensten gerade aktuellen Datenregister zu betrachten (um zu den wechselnden Argumenten zu kommen) und wie kann man vermeiden, daß solch ein Unterprogramm jedesmal in einem Hauptprogramm explizit an der Stelle hingeschrieben werden muß, an der es absolviert werden soll. Beim Papiercomputer in der gegenwärtigen Form läßt sich das nicht vermeiden. Er muß gewaltig umgebaut werden, damit Unterpro-

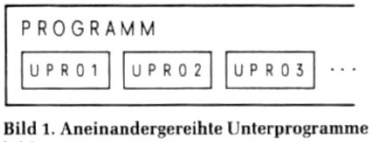
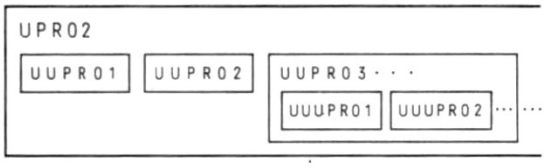


Bild 1. Aneinandergereihte Unterprogramme bilden ein Programm

Bild 2. Unterprogramme können Unterprogramme enthalten



gramme sinnvoll eingesetzt werden können.

Maschinensprache und Papiercomputer

Sie müssen jetzt einen gedanklichen Sprung machen, weg von den Streichholzzahlen. Kein realer Computer wird seine Zahlen so darstellen. Jedermann weiß ja heute schon, daß ein Computer seine Zahlen intern binär (das wird später noch alles sehr genau erklärt werden) darstellt. Wichtig ist daran momentan nur, daß deshalb jetzt beim Papiercomputer in die Datenregister die Zahlen zunächst einmal im Klartext geschrieben werden sollen. Daß man dabei, wenn man den Computer gedanklich laufen läßt, Radierprobleme bekommen kann, das soll uns nicht weiter kümmern, denn der Papiercomputer dient ja im Grunde nur als konkretes Schema für das gedankliche Prinzip, das hinter den Computern steckt.
Ein weiterer gedanklicher Sprung ist notwendig. Die Programmspeicher-Zellen waren bisher getrennt von den Datenregistern gehalten worden. Dort standen ja auch keine Zahlen, sondern Befehle darin. Rein logisch war diese Trennung auf den ersten Blick also auch gut begründet. Nun ist es aber so, daß jeder reale Computer sowohl Zahlen als auch Buchstaben und Befehlssymbole unterschiedslos als Muster aus zweiwertigen Zuständen abspeichert, weil dies technisch besonders einfach ist. Aus diesem Grund ist es sehr vorteilhaft, nur einen einzigen Speichertyp zu haben, und auch die Befehle des Computers als Zahlen abzuspeichern. Und diese Befehle auch, wie die Zahlen, als Daten verfügbar zu machen. Das gibt dem Computer eine einheitliche Linie und hat den großen Vorteil, daß es zum Beispiel die Unterprogrammtechnik sehr leicht möglich macht. Die Menge der in Zahlen codierten Befehle eines Computers nennt man seine Maschinensprache.

Von Neumann, Maschinensprache und Assembler

Wer Spaß an der Geschichte der Rechenmaschinen hat und die Entwicklung der Ideen dazu verfolgen möchte, dem sei das Buch „The Origins of Digital Computers, Selected Papers" empfohlen, das viele Original-Arbeiten enthält. Es ist im Springer-Verlag erschienen. Dort findet man zum Beispiel die berühmte Schrift „Preliminary Discussion of the Logical Design of an Electronic Computing Instrument", in der John von Neumann zusammen mit den Coautoren Burks und Goldstine beschreibt, wie ein geplanter Rechner aussehen soll. Dort findet sich die oben geführte Diskussion, daß man Befehle und Daten innerhalb eines Computers gleichartig abspeichern können sollte und daß der Computer dann selbst während eines Laufes entscheiden muß, ob ein Befehl oder ein Datum vorliegt, in sehr klarer Form wieder.
Als Ergebnis der Diskussion sei hier vorgeschlagen, daß beim Know-how-Computer einfach die Datenregister weggelassen werden und zum Programmspeicher nur noch Speicher gesagt wird. Der Speicher wird dann nicht mehr durch Programmspeicher-Zellen-Nummern eingeteilt, sondern nur durch Speicher-Zellen-Nummern (*Bild 3*).
Außerdem werde festgelegt, daß der Befehl „+" fortan durch die Zahl 1 verschlüsselt werde, „–" durch die Zahl 2, „j" durch die Zahl 3, „0" durch 4 und „Stop" durch 5. Die Adressen der Speicherzellen, in welchen sich die zu verarbeitenden Daten befinden, sollen einfach hinter dem Befehlsschlüssel angefügt werden. Genauso, wenn es sich um ein Sprungziel handelt. Um ganz klar zu machen, was gemeint ist, zeigt *Bild 4* das allererste Additionsprogramm in seiner Maschinenform.
Zu beachten sind hier noch einige zusätzliche Dinge, die auch schon bei realen Mikroprozessoren eine Rolle spielen. Zunächst sind die Befehle alle fünfstellig. Das resultiert daraus, daß ich mir bei der gedanklichen Konstruktion des neuen Computers gesagt habe, daß in jeder seiner Speicherzellen nur fünfstellige Zahlen Platz haben sollen, weil sonst der konstruktive Aufwand für eine Speicherstelle zu hoch sein würde (bei einer – natürlich nicht beabsichtigten – Realisation dieses Computers). Wer mag, kann mehr Stellen vereinbaren – oder auch weniger. Entsprechend größere oder kleinere maximale Zahlen lassen sich darstellen. Außerdem soll der Computer immer noch von vorne anfangen zu rechnen. Das bedeutet, daß in Speicherzelle 1 stets ein Befehl zu stehen hat. Deshalb sind die beiden zu addierenden Zahlen jetzt in den Speicherzellen 7 und 8 zu suchen.
Mit diesen Informationen müßten Sie jetzt in der Lage sein, alle bisher geschriebenen Programme in die neu definierte Maschinensprache zu übersetzen. Sie müssen sich eigentlich nur neue Datenregisternummern ausdenken und jeweils konsequent einsetzen. Beachten Sie, daß Sie bis zu 9999 Speicherzellen adressieren können.
Es sei hier festgehalten, daß die Mikro-Computer-Hersteller stets bei einem Prozessor durch dessen Konstruktion die Codierung der Befehle in „Nummern" festgelegt haben; und auch die Art und Weise, wie an einen Befehl die Adresse der Daten oder Sprungziele angehängt werden müssen. Das ist manchmal komplizierter als hier bei unserer Entwicklung. Genauso wie bei unserer Entwicklung, werden in den Datenbüchern der Hersteller von Mikroprozessoren nicht nur die sogenannten Maschinencodes angegeben, also die Nummern der Befehle, sondern vor allem auch deren sinnfällige Bezeichnungen. Bei uns sollten die Bezeichnungen +, –, j, 0, Stop gleich auch die Wirkungsweise eines Befehls anzeigen. Manche reale Mikroprozessoren besitzen bis über 100 verschiedene Befehle, die in den verschiedensten Mnemonics, wie der Fachmann sagt, im Handbuch dargestellt sind. So, wie Sie die ersten Programme in den

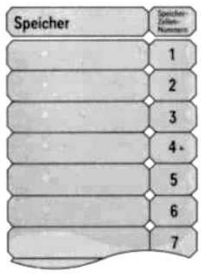


Bild 3. Der Speicher: einheitlich

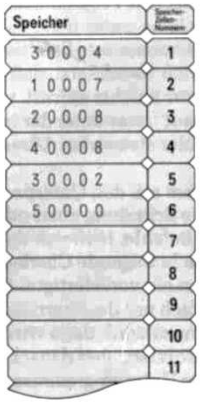


Bild 4. Ein Maschinenprogramm

Mnemonics des Know-how-Computers programmiert haben, werden die meisten Programme für Mikrocomputer in den entsprechenden Kürzeln des jeweiligen Programmiermodelles geschrieben, da sonst niemand längere Programme richtig durchschauen könnte. Es gibt sogar Mikroprozessorprogramme, die automatisch in Mnemonics geschriebene Programme in Maschinensprache übertragen. Ausgehend von den Sprachgewohnheiten in der Groß-EDV nennt man solche Programme „Assembler". Und man sagt im Slang, daß man „in Assembler" programmiert, wenn man mit den symbolischen Befehlsnamen (und meist auch mit symbolischen Adressen) arbeitet.

Zurück zu den Unterprogrammen

Nachdem nun das von Neumannsche Computerkonzept mit „durchgehendem" Speicher vorgestellt worden ist und auch die Struktur der Befehle mit Operations-Teil (der Teil, in dem der Befehl verschlüsselt ist) und Adreß-Teil (die an den Operations-Teil sich anschließenden weiteren vier Stellen) klar ist, kann geschildert werden, wie die Unterprogrammtechnik auf dem Papiercomputer implementiert werden kann. Zuvor aber sei noch gesagt, daß ein Computer unserer Bauart stets, wenn er anläuft und auch im weiteren Betrieb, sozusagen in einer immerwährenden Schleife arbeitet, die wie folgt abläuft: Schaue unter der im Programmzähler angegebenen Adresse (beim Start ist das Eins, im Betrieb ergibt sich die Adresse aus dem vorherigen Programmlauf) nach, welcher Befehl in dieser Speicherzelle steht. Führe ihn durch (wobei sich auch der nächste Programmzählerstand ergibt). Schaue unter der...
Das ist die sogenannte Befehlsschleife, die natürlich voraussetzt, daß der Computer immer Befehle antrifft, wenn er „nachschaut", was unter der im Programmzähler stehenden Adresse zu finden ist.
Das Problem mit den Unterprogrammen ist jetzt die Erfindung eines oder mehrerer neuer Befehle. Hilfreich dabei könnten vielleicht folgende Überlegungen sein: Wenn ein vorgefertigtes Programmstück einfach mit dem normalen jXXX angesprungen wird, dann wird es zwar richtig absolviert, aber danach, bei Beendigung des Unterprogrammes sollte ja noch weiter im Hauptprogramm gerechnet werden. Und genau diese Information, woher nämlich das Unterprogramm aufgerufen worden ist, muß irgendwie bei dessen Beendigung verwertet werden, damit der Computer an der richtigen Stelle im Hauptprogramm weitermachen kann. Das ist genau wie bei der Befehls-Schleife: nach Beendigung eines Befehls muß feststehen, an welcher Stelle, mit welcher Stellung des PC, weiter gerechnet werden soll.

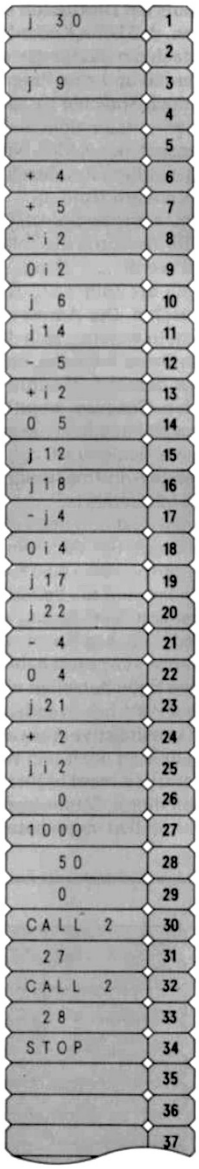


Bild 5. Ein Beispiel zur Unterprogrammtechnik. Erfinden Sie weitere!

Von indirekter Adressierung

Die Computerhersteller haben zur Lösung der anstehenden Frage zunächst eine recht simple Methode gefunden. Jemand, der ein Unterprogramm programmieren sollte, mußte in dem Unterprogramm eine spezielle Speicherzelle für die Rückkehradresse reservieren. Meistens war dies die erste Adresse des Unterprogrammes. Ein eigenständiger „CALL"-Befehl im Hauptprogramm sorgte dafür, daß erstens der momentane Stand des Programmzählers (genauer: die Adresse des CALL-Befehls + 1) in die erste Zelle des Unterprogrammes eingeschrieben wurde und dann zweitens der Programmzähler auf die zweite Speicher-Zellen-Nummern des Unterprogrammes eingestellt wurde. Dort begann der eigentliche Code des Unterprogrammes. Der CALL-Befehl war also ein Sprungbefehl mit einigen Zusatzeffekten. Am Ende des Unterprogrammes mußte ein weiterer Sonderbefehl stehen und zwar ein indirekter Sprungbefehl. Die Adresse dieses indirekten Sprungbefehles, er sei einmal ji genannt, gab nicht das Sprungziel, sondern diejenige Zelle an, unter der die Sprungziel-Adresse zu finden war. ji wirkte also so, daß der Programmzähler mit dem Inhalt der angegebenen Speicherzelle geladen wurde. Wenn man also ein Unterprogramm mit dem Befehl ji XXX abschloß, wobei XXX die Adresse der ersten Zelle des Unterprogrammes war, dann wurde die Rückkehradresse des Unterprogrammes in den PC gegeben und der Computer machte noch im Hauptprogramm der Absprungstelle weiter.
Während die Technik der indirekten Adressierung heute verstärkt benutzt wird, hat sich die Technik, ein Unterprogramm anzuspringen, geändert. Wie, das wird später in dieser Serie am konkreten Mikroprozessor noch eingehend besprochen werden.

Als weitere Beispiele für die indirekte Adressierung sei die Sprache unseres Papiercomputers jetzt noch um + i, − i, 0i und ji ergänzt. Jedesmal bezeichnen bei diesen neuen Befehlen die angegebene Adresse eine Speicherzelle, in der die eigentliche Adresse für den gegebenen Befehl steht, + i 5 würde also bedeuten, daß in der fünften Speicherzelle die Adresse der Speicherzelle steht, deren Inhalt um 1 zu erhöhen ist. Nicht der Inhalt von Zelle 5, sondern der Inhalt der dort angegebenen Zelle wird um 1 erhöht. Für Versuchszwecke sei der Papiercomputer auch noch mit dem vorhin geschilderten antiquierten CALL-Befehl

ausgerüstet. Insgesamt also mit folgendem Befehlssatz:

+ XXX	1
− XXX	2
0 XXX	3
j XXX	4
Stop	5
+i XXX	6
−i XXX	7
0i XXX	8
ji XXX	9
CALL XXX	0

Dann können nämlich alle besprochenen Sachverhalte konkret am Papiercomputer durchprogrammiert werden.

Ein Beispiel

Bild 5 zeigt Ihnen ein relativ einfaches Programmier-Beispiel mit Unterprogramm. Es enthält einige Details, die viele Dinge bei „richtigen" Unterprogrammen berühren. Zunächst steht da, wo der Computer losrechnet, auch ein Befehl, j 30, springe direkt nach 30. In 30 kommt der erste CALL-Befehl und zwar nach Zelle 2. In 2 beginnt also das Unterprogramm mit der Zelle, in die durch CALL die Rücksprungadresse eingeschrieben wird. In unserem Fall ist das 31. In 3 steht der erste Befehl des Unterprogrammes, ein Sprung nach 9. Dort wird indirekt geprüft, ob der Inhalt der in 2 angegebenen Zelle, also der Inhalt von Zelle 31, Null ist. Vergleichen Sie die Schleife mit früheren Programmen. Sie kopiert den Inhalt der unter 2 angegebenen Zelle 4 und 5. Dadurch wird, und jetzt sei der Zweck des Unterprogrammes von 2 bis 25 verraten, der Inhalt der auf den CALL-Befehl folgenden Zelle in das Unterprogramm transportiert, damit dieses ihn als Adresse einer zu löschenden, auf Null zu stellenden Zelle, interpretiert. Das Unterprogramm soll also die Funktion

LÖSCHE ZELLE XXXX

durchführen. Wobei im Hauptprogramm XXXX gleich hinter CALL 2 stehen muß, damit dieser Wert als Argument (oder auch Parameter) in das Unterprogramm übernommen werden kann. Mit dem Inhalt von 5 wird der Inhalt von 31 wieder auf den alten Stand gebracht. Das tun die Befehle in 12 bis 15. Dann wird die in 4 angegebene Zelle (27) auf Null gestellt durch die Befehle von 17 bis 19. Danach wird der Inhalt von 4 auf Null gestellt für nachfolgende Aufrufe. Dann wird der Inhalt von 2 um Eins hochgezählt. Und dann indirekt über 2 zurückgesprungen. Wohin? – Nach 32. Dort folgt? Merken Sie, wie alles funktioniert? Zugegeben, etwas pervers mag diese Weiterentwicklung des Papiercomputers ja sein, aber wenn Sie mitverfolgen können, wie Sie mit dem vorgelegten Unterprogramm und der geschilderten Technik der Parameterübergabe an das Unterprogramm jede beliebige Speicherzelle XXXX auf Null stellen können, dann dürfte es Ihnen auch nicht schwer fallen, ein Unterprogramm ADD XXXX YYYY ZZZZ zu schreiben, das den Inhalt von XXXX und YYYY in ZZZZ aufaddiert. Wobei dem Aufruf von ADD die drei Adreßangaben, wie beim Beispiel LÖSCHE die eine Adresse, direkt hinter dem CALL-Befehl folgen sollen. Das ist sicher nicht einfach, wer's aber gemacht hat, den können indirekte Sprünge und anderes nicht mehr erschrecken.

(Fortsetzung folgt, mit einer Besprechung eines wirklichen Prozessors: Damit gehen wir von grauer Theorie zur Praxis über!)

Übrigens:

Den Know-how-Computer gibt es überall da, wo es mc gibt. Und er kostet dort nichts, denn jeder soll die Computer verstehen lernen. Holen Sie sich Ihren Know-how-Computer!

Nochmals: CP/M-Blocking und Deblocking

In Heft 2/1983 auf Seite 80 wurde ein Fehler beschrieben, der bei CP/M auftreten kann, aber – abhängig davon, wie der BIOS-Teil des Betriebssystems implementiert wird – nicht auftreten muß. Beim mc-CP/M-Computer tritt das Problem nicht auf, weil der von Digital Research empfohlene Algorithmus (der wiederum selbst bestimmte Probleme mit sich bringt) nicht verwendet wird.

Worin besteht nun der „Fehler"? Der Autor von [1] beschreibt das Verfahren nach [2] so, daß nach Übergabe eines logischen 128-Byte-Sektors vom BDOS an das BIOS vor dem tatsächlichen Schreiben auf die Disk zunächst noch abgewartet wird, ob nicht eventuell noch weitere, in den gleichen physikalischen Sektor der Disk gehörende Daten nachfolgen. Also könnte der im BIOS liegende Puffer (für einen physikalischen Sektor) nach dem beendeten Schreiben eines Files u. U. noch Daten dieses Files enthalten, die auf die Disk gehören, aber erst später auf diese geschrieben werden. Nimmt der Bediener in dieser Situation die Diskette heraus, so kann dieses spätere Schreiben nicht mehr geschehen.

Aber: Beim Schließen eines Files (CLOSE-Aufruf) wird der Pufferinhalt bedingungslos sofort auf die Disk geschrieben. Dies geschieht in den Zeilen 296 bis 298. Diese Zeilen, die ja schon im Original [2] vorhanden sind, leisten im Grunde genau das, was die in [1] angegebene „Korrektur" bewirken soll.

Zu bemängeln ist nur, daß das Sourcelisting des Blocking/Deblocking-Algorithmus [2] von Digital Research seine Feinheiten nur bei einer „detektivischen" Betrachtungsweise offenbar werden läßt. Der Vorgang des Schließens eines Files kommt darin explizit überhaupt nicht vor! Die Befehle in den oben erwähnten Zeilen 296 bis 298 bewirken das sofortige Schreiben auf die Disk ausschließlich für den Fall, daß es sich um einen Sektor aus der Directory handelt. Man kann nun kombinieren: Im Normalfall wird beim Schließen eines Files (CLOSE-Aufruf) die letzte Disk-Operation das Schreiben desjenigen Directory-Sektors sein, der den endgültigen Zustand des Files enthält. Da nun der Blocking/Deblocking-Algorithmus von Digital Research bei Directory-Sektoren eine Ausnahme macht, indem er sie stets sofort auf die Disk schreibt, kann nach einem CLOSE-Aufruf der Puffer stets nur leer sein. Daher kann man dann auch die Diskette gefahrlos herausnehmen oder den Rechner abschalten. *Rolf Keller*

Literatur

[1] *Podien, Wolfgang:* Ein Fehler im CP/M-Betriebssystem? mc 1983, Heft 2, Seite 80

[2] Digital Research: CP/M-2 Alteration Guide

Herwig Feichtinger

Computer für Anfänger

Teil 4

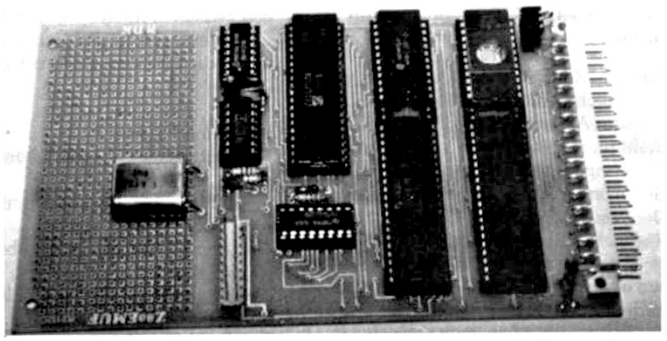


Bild 2. So sieht ein Platinchen aus, das genau dem Blockschaltbild in Bild 1 entspricht. mc hat es als Bauanleitung in Heft 4/1983 veröffentlicht

Wenn Sie bis hierher bei unserem Computer-Kurs mitgemacht haben, dann haben Sie nicht nur Ausdauer bewiesen, was das Verarbeiten von „grauer Theorie" angeht, sondern sich auch mit dem Grundwissen eingedeckt, das nötig ist, um einen richtigen Mikrocomputer ganz zu verstehen. Mit einem solchen wollen wir uns nämlich jetzt befassen.

So nützlich der mc-Papiercomputer auch sein mag, den wir in Heft 5/1983 abgebildet haben, er ist nur ein Denkmodell. Denn man muß ihn manuell bedienen, sein „Speicherplatz" ist recht eingeschränkt, und es stehen ihm nur fünf elementare Befehle zur Verfügung.

Trotzdem werden Sie feststellen, daß der Computer, den wir jetzt besprechen, mit dem Papiercomputer vieles gemein hat. Natürlich müssen wir uns jetzt an ein ganz konkretes Gerät halten, und das hat den Haken, daß uns vielleicht alle jene, die einen anderen Computer besitzen, vielleicht sogar mit einem anderen Mikroprozessor in seinem Inneren, gram werden und uns tausend Leserbriefe schreiben: Warum habt ihr nicht den Prozessor XY 493 genommen...
Für irgendwas muß man sich aber entscheiden. Wir haben uns für die Prozessorfamilie 6502 entschieden, weil sie zwei Bedingungen erfüllt: Sie ist weit verbreitet, und die Zentraleinheit weist einen auch für Anfänger leicht überschaubaren Aufbau auf. Aber wir werden andere Typen nicht ignorieren und hier und da auch auf Unterschiede zwischen den Computer-Familien eingehen.

So recht deutlich wird das allerdings erst in den nächsten Folgen; diesmal bleiben wir noch allgemein, so daß das hier gesagte für alle heute üblichen Mikroprozessor-Familien gleichermaßen zutrifft.

Wieso eigentlich „Familien"? Nun, es gibt meist mehrere CPU-Typen, die den gleichen Befehlssatz ausführen können. So wird etwa der 8080-Befehlssatz auch von den Prozessoren 8085 und Z80 ausgeführt, und man spricht von der 80-Familie. Ebenso besteht die 65XX-Familie aus mehreren Mitgliedern, wie 6502, 6504, 65C02 und einige andere.

Die 80-Familie und die 65XX-Familie stellen zusammen den größten Teil des Mikroprozessor-Marktes dar. Darüber, daß der 6502 gerade 6502 heißt und nicht 4711 oder 0815, sollte man sich keine Gedanken machen. Jedes Kind braucht eben einen Namen.

Register, ROM und RAM

Unser Papiercomputer aus Heft 5 kannte zwei Speicher, nämlich den Daten- und den Programmspeicher. Der 6502 als „richtiger" Mikroprozessor besitzt auf seinem Silizium-Chip nur ein paar Hilfsregister, die man aber nicht zur Programm- oder dauernden Datenspeicherung verwenden kann, dazu würde ihre Kapazität gar nicht ausreichen. Aber immerhin: Es handelt sich bei ihnen auch um eine Art Speicher, und man nennt diese Zellen im Prozessor „Register".

Die beiden anderen Speicherarten, nämlich Daten- und Programmspeicher, sind außerhalb des Prozessors untergebracht. In vielen Fällen steht dabei das Pro-

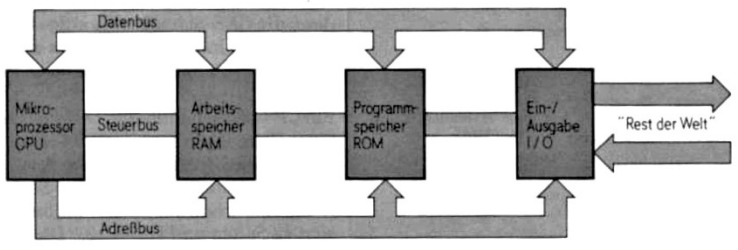


Bild 1. Blockschaltbild eines Mikrocomputer-Minimalsystems: I/O-Baustein, Programmspeicher (ROM), Arbeitsspeicher (RAM) und Mikroprozessor (CPU) sind über drei Gruppen von Leitungen miteinander verbunden

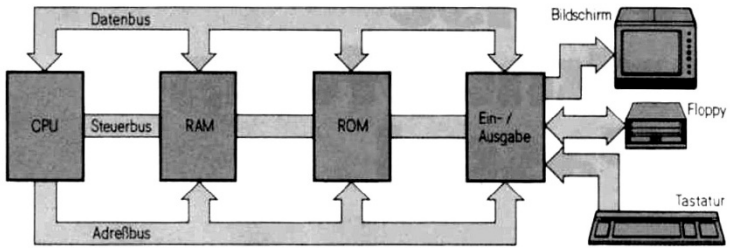


Bild 4. Bei einem richtigen Tischcomputer sind noch zusätzliche I/O-Bausteine zur Verbindung mit Bildschirm, Schreibmaschinen-Tastatur und Floppy-Disk-Speicher nötig

gramm in einem sogenannten Festwertspeicher (ROM = Read-Only Memory), so daß es beim Ausschalten der Betriebsspannung nicht verloren geht. Und die Daten, die das System zu verarbeiten hat, stehen im Arbeitsspeicher (RAM = Random Access Memory). Der Mikroprozessor kann aus dem ROM nur lesen, das RAM dagegen beschreiben oder lesen. Der RAM-Inhalt geht allerdings beim Abschalten der Betriebsspannung verloren.
Ein kompletter Computer bestünde also aus ROM (Programmspeicher), RAM (Arbeitsspeicher) und Mikroprozessor (CPU = Central Processing Unit). Nur: Wie bekommt man in dieses System überhaupt Daten hinein, die der Computer verarbeiten soll? Und wie erfährt man, welche Ergebnisse der Computer errechnet hat?

Ein- und Ausgabe

Ohne solche Möglichkeiten zur Ein- und Ausgabe geht gar nichts, der Computer wäre reiner Selbstzweck. Bestenfalls könnte er gegen sich selbst Schach spielen (aber niemand würde erfahren, ob seine linke oder rechte Hälfte gewonnen hat!).

Denkbar wäre zum Beispiel der Anschluß einer einfachen Ziffern-Tastatur und eines Siebensegment-LED-Displays (LED = Light Emitting Diodes, Leuchtdioden). Damit könnte man zumindest numerisch Werte eingeben und das Ergebnis auf der Anzeige bestaunen. *Bild 1* zeigt, wie ein solcher kleiner Computer aufgebaut ist. Prozessor (CPU), Programmspeicher (ROM), Arbeitsspeicher (RAM) und Ein/Ausgabe-Baustein (I/O = Input/Output) sind über drei Gruppen von Leitungen miteinander verbunden: Der Adressenbus dient zur Auswahl einer bestimmten Zelle in ROM oder RAM oder bestimmt auch, ob eine Ein- oder Ausgabe stattfinden soll. Der Datenbus dient zum Transfer von Informationen zwischen dem Mikroprozessor und den übrigen Bausteinen des Systems. Der dritte Bus, der Steuerbus (Control Bus), kümmert sich um den richtigen zeitlichen Ablauf: Eine Leitung bestimmt, wann der nächste Befehl geholt werden soll, eine andere teilt dem Arbeitsspeicher mit, ob aus ihm gelesen oder in ihn geschrieben werden soll.

Der Computer ist komplett

Ein System wie das gerade beschriebene läßt sich recht preiswert aufbauen; Computer wie die in unseren Heften 2/1981 oder 4/1983 (*Bild 2*) für die Mikroprozessoren 6504 und Z80 sind zwei Beispiele dafür, daß so etwas unter 100 DM machbar ist. Solche Systeme eignen sich zum Beispiel dafür, Ihre häusliche Heizung zu regeln, als Herz eines Spielautomaten zu dienen oder das Typenrad einer Schreibmaschine zu steuern. Derartige Einplatinen-Computer gibt es auch mit LED-Display und numerischer Tastatur (*Bild 3*). Richtige Tischcomputer sind im Prinzip genauso aufgebaut, verfügen aber über mehr Speicherplatz und enthalten spezielle Ein/Ausgabe-Bausteine z. B. zur Abfrage einer großen Schreibmaschinen-Tastatur und zum Anschluß eines Datensichtgeräts, eines sogenannten Video-Monitors. *Bild 4* zeigt das Blockschaltbild eines solchen Computers.
Im nächsten Heft wollen wir uns damit befassen, wie man die Speicherkapazität mißt, was Wortbreite heißt und was beim Ausführen eines Befehls im Mikroprozessor passiert. Dann erfahren Sie auch, warum die Bezeichnung „4 K" irreführend, ja falsch ist, und was „4 KByte" bedeutet. *Fortsetzung folgt*

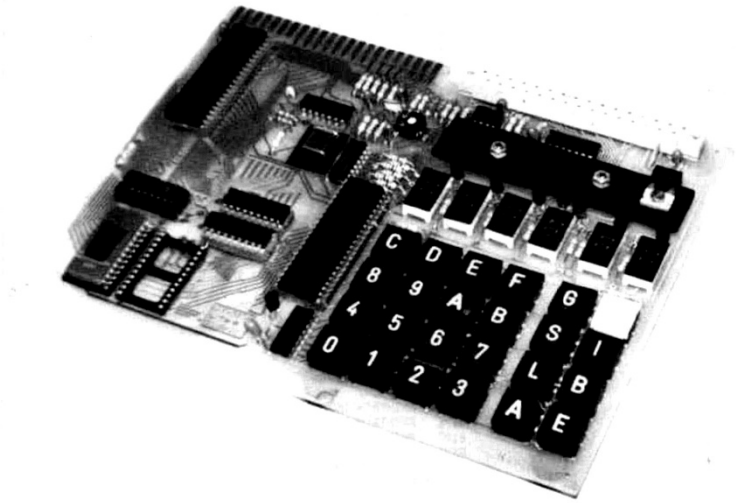


Bild 3. Ergänzt man ein einfaches Mikrocomputersystem mit einer Zifferntastatur und einem LED-Display, so kommt man zu dieser Konfiguration (CT-65, Testbericht in mc 4/1983)

Herwig Feichtinger

Computer für Anfänger

Teil 5

Wie ein Mikrocomputer prinzipiell aufgebaut ist, haben wir schon gesehen. Hier geht es nun um die Schreibweise von Adressen und Daten, um hexadezimale Zahlen, Bits, Bytes und Nibbles.

Die vier wesentlichen Bausteine eines Computers – ROM, RAM, CPU und I/O – sind untereinander über den Adressen-, Daten- und Steuerbus verbunden. In den heutigen Mikrocomputern werden überwiegend sogenannte 8-Bit-Mikroprozessoren eingesetzt: Das bedeutet, daß 8 Bits parallel verarbeitet und über den Datenbus transferiert werden können.

Ein Bit: Ja oder nein

Was aber ist überhaupt ein Bit? So nennt man die Informationsmenge, die man allein durch Ein- oder Ausschalten der Spannung auf einer einzigen Leitung darstellen kann. Die beiden Zustände ein und aus werden auch mit 1 und 0, High und Low oder – als Entscheidung – ja oder nein bezeichnet.
Dementsprechend kann man mit acht Datenbus-Leitungen gleichzeitig acht Bits zum Beispiel vom Speicher zum Mikroprozessor übertragen. Diese Gruppe von acht Bits nennt man ein Byte.
Nun will man für die Berechnungen, die der Computer ausführen soll, natürlich nicht „ja" und „nein", sondern Zahlenwerte auf dem Datenbus transferieren. Also weist man einfach jeder denkbaren Zustands-Kombination der acht Datenbus-Leitungen je eine Zahl zu, etwa so wie in *Tabelle 1*.

256 Zahlen mit acht Leitungen

Statt „ja" und „nein" hätte man natürlich in dieser Tabelle auch „1" oder „0" schreiben können. Dann würden zum Beispiel die Zustände der acht Leitungen für die Darstellung der Zahl 6 so aussehen: 00000110.

Warum ist die Tabelle aber gerade so aufgebaut? Man könnte doch auch die „Codes" für die darzustellenden Zahlen beliebig vertauschen, solange nur immer genau eine Kombination zu einer Zahl gehört und keine Verwechselungen möglich sind!
Nun, die Computer-Freaks sind größtenteils faule Leute und haben die Tabelle von vornherein so aufgebaut, daß man rein mathematisch den Zusammenhang zwischen Zahl und Ja-Nein-Darstellung ausrechnen kann.
Dazu schreiben wir statt „ja" immer 1, statt „nein" immer Null und für jede Leitung eine Zweierpotenz (deshalb haben wir trickreicherweise die Leitungen auch nicht von 1 bis 8, sondern von 7 bis 0 durchnumeriert!). Für die Zahl 6 (Code 00000110) ergäbe sich dann:

$$\begin{aligned}\text{Zahl} &= 0\cdot 2^7 + 0\cdot 2^6 + 0\cdot 2^5 + 0\cdot 2^4 + 0\cdot 2^3 + 1\cdot 2^2 + 1\cdot 2^1 + 0\cdot 2^0\\ &= 0 + 0 + 0 + 0 + 0 + 4 + 2 + 0\\ &= 6\end{aligned}$$

Die Leitung Nr. 7 wird also als 2^7 gerechnet, die Leitung 6 als 2^6 und so fort; und die Ja-Nein-Kombination (0 oder 1), die auf den acht Leitungen des Datenbus anliegt, bestimmt, welche Zweierpotenzen aufsummiert werden (1) und welche nicht (0).
Sie sollten jetzt nicht überflüssigerweise versuchen, irgendwelche langen 0-1-Kolonnen durch Aufsummieren von Zweierpotenzen in dezimale Zahlen umzurechnen. Wichtig ist nur, daß Sie verstehen, daß man mit acht Leitungen (und damit mit acht Bits oder einem Byte) 256 verschiedene Zahlenwerte darstellen kann, nämlich alle natürlichen Zahlen von Null (alle Leitungen auf Null bzw. „nein") bis 255 (alle Leitungen auf „ja"). Darauf kommt man auch, ohne die Tabelle auf 256 Zahlen zu vervollständigen: Bei acht Leitungen ergeben sich $2^8 = 256$ Kombinationsmöglichkeiten.
Die meisten 8-Bit-Mikrocomputer besitzen nun nicht nur acht, sondern 16 Adressenbus-Leitungen. Das bedeutet, daß maximal $2^{16} = 65\,536$ einzelne Adressen mit diesem Bus darstellbar sind. Also kann man 65 536 verschiedene Speicherzellen adressieren. Da jede Speicherzelle wiederum aus acht Bits (einem Byte) besteht, ist der maximale Speicherumfang 65 536 Byte, den der Mikroprozessor mit 16 Adressenbus-Leitungen adressieren kann.

Binäre Darstellung

Schreibt man die Zahlen, die auf dem Bus transferiert werden, mit „0" für „nein" und „1" für „ja", so nennt man das binäre Darstellung. Und die so ge-

Tabelle 1: Darstellung von Zahlen mit acht Leitungen

Zahl	*Ltg. 7*	*Ltg. 6*	*Ltg. 5*	*Ltg. 4*	*Ltg. 3*	*Ltg. 2*	*Ltg. 1*	*Ltg. 0*
0	nein	nein	nein	nein	nein	nein	nein	nein
1	nein	nein	nein	nein	nein	nein	nein	ja
2	nein	nein	nein	nein	nein	nein	ja	nein
3	nein	nein	nein	nein	nein	nein	ja	ja
4	nein	nein	nein	nein	nein	ja	nein	nein
5	nein	nein	nein	nein	nein	ja	nein	ja
6	nein	nein	nein	nein	nein	ja	ja	nein
7	nein	nein	nein	nein	nein	ja	ja	ja
8	nein	nein	nein	nein	ja	nein	nein	nein
9	und so weiter...							
253	ja	ja	ja	ja	ja	ja	nein	ja
254	ja	ja	ja	ja	ja	ja	ja	nein
255	ja	ja	ja	ja	ja	ja	ja	ja

schriebene Zahl 00000110 (statt 6) nennt man Dualzahl. Denn das duale Zahlensystem verwendet nur die beiden Ziffern 0 und 1. Im Gegensatz dazu stellen wir normalerweise, wie wir es von der Schule her gewohnt sind, unsere Zahlen mit den zehn Ziffern 0...9 im Dezimalsystem dar.
Eine solche dezimale Ziffer kann man mit einem halben Byte, nämlich mit vier Bits codieren; zu diesem Halbbyte sagt man auch „Nibble". Für die zehn Dezimalziffern ergibt sich dann folgende Zuordnung:

Ziffer	Code	Ziffer	Code
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

Das funktioniert doch ganz gut! Jetzt tappen Sie aber nicht in eine bereitgestellte Falle: Wie stellen Sie die dezimale Zahl 12 mit acht Bits dual dar? Ja, werden Sie sagen, ganz einfach, für die 1 nehme ich den Code 0001 und für die 2 den Code 0010, zusammen also 00010010. Haha! Weit gefehlt. Denn diese 8-Bit-Kombination ergäbe die Dezimalzahl $2^4 + 2^1 = 17$ und nicht etwa 12. Man kann eine längere Dualzahl nicht dadurch herstellen, indem man die binären Codes für die einzelnen Dezimalziffern aneinanderhängt. Außerdem liefert die obige Tabelle mit den Ziffern 0...9 noch ein Problem: Codes wie 1101 oder 1111 werden Sie darin vergeblich suchen. Sie entsprechen nämlich zweistelligen Dezimalzahlen, hier 13 und 15. Damit wäre also die Regel, man könne ein Nibble (4 Bits) als eine Ziffer darstellen, durchbrochen – es geht eben nur umgekehrt: Eine Dezimalziffer kann man stets als ein Nibble darstellen.

Hexadezimale Ziffern

Die Nichtumkehrbarkeit dieser Regel hat natürlich die Computeristen geärgert. Deshalb haben sie sich ein Zahlensystem ausgedacht, das so viele Ziffern aufweist, wie vier Bits Kombinationsmöglichkeiten bieten, nämlich $2^4 = 16$. Weil dazu die zehn Zeichen 0...9 nicht ausreichen, haben sie noch die sechs Buchstaben A...F hinzugefügt. Und das Resultat nennt sich Hexadezimal-, Sedezimal- oder Sechzehner-System. Die Zuordnung für die Codes bei 0...9 ist wie oben unverändert, nur für A...F ergibt sich jetzt zusätzlich:

A	1010	D	1101
B	1011	E	1110
C	1100	F	1111

Wenn Sie jetzt gefragt werden, wie die hexadezimale Zahl A1 binär codiert wird, so ist das wirklich einfach: Im Hexadezimalsystem darf man nämlich die Binärcodes aneinanderketten. Für A ergibt sich 1010, für 1 schreiben wir 0001, und A1 ist binär also 10100001. Wenn Sie bisher mitgemacht haben, ist es für Sie kein Kunststück mehr, herauszufinden, was diese Bitfolge 10100001 nun dezimal „wert" ist. Damit Ihnen dabei das Aufaddieren der Zweierpotenzen leichter fällt, zeigt *Tabelle 2* noch eine Aufstellung der Zweierpotenzen bis 2^7 – das reicht für ein Byte! Und *Tabelle 3* zeigt eine kleine Gegenüberstellung der dezimalen und hexadezimalen Schreibweise.

Tabelle 2: Zweierpotenzen bis 2^7

$2^0 = 1$	$2^4 = 16$
$2^1 = 2$	$2^5 = 32$
$2^2 = 4$	$2^6 = 64$
$2^3 = 8$	$2^7 = 128$

Tabelle 3: Dezimale und hexadezimale Zählweise

dez. hex	dez. hex	dez. hex
0 00	12 0C	34 12
1 01	13 0D	usw.
2 02	14 0E	99 63
3 03	15 0F	100 64
4 04	16 10	usw.
5 05	17 11	254 FE
usw.	usw.	255 FF
9 09	31 1F	
10 0A	32 20	
11 0B	33 21	

Das große K

Noch einmal zurück zu den 65536 Adressen, die man mit 16 Adressenbus-Leitungen ansprechen kann. „65536" ist ein Zungenbrecher, also hat man sich Gedanken gemacht, wie man 65536 Speicherzellen anders ausdrücken kann. Für 2 kg Leberkäse sagt man ja normalerweise beim Einkaufen auch nicht 2000 g, obwohl man das noch besser aussprechen kann als 65536 Byte. Die Computeristen haben deshalb die Einheit KByte eingeführt (beachten Sie das große K!). Ein KByte bedeutet 1024 Byte, und das sind genau 2^{10} Byte, weil die Informatiker eben so gern in Zweierpotenzen rechnen. Und 65536 Byte sind demzufolge genau 64 KByte (= 65536/1024). Das kann man sich schon besser merken als 65536 Byte oder, was auch richtig wäre, 65,536 kByte, denn das kleine k steht ja nach wie vor für 1000. Das K kann man natürlich auch vor Bit schreiben: Ein KBit sind 1024 Bit. Ein Speicher mit 256 KBit hat also eine Kapazität von 256 · 1024 Bit oder auch 256 · 1024/8 = 32 768 Byte. Und das wiederum sind 32 768/1024 = 32 KByte! Lassen Sie sich also nicht durch ein großes K verwirren, höchstens, wenn es einmal alleine steht: Was ein Speicher mit 64 K ist, das kann Ihnen niemand sagen. Denn keiner weiß, ob 64 KBit oder 64 KByte gemeint sind. Ein K darf also niemals allein stehen, sonst ist es mehrdeutig. Noch ein Tip: Sollten Sie bei Ihrem Metzgermeister sehen, daß er 1 Kg Leberkäse für fünf Mark anbietet, so können Sie ihm guten Gewissens 1024 g abnehmen. Und wenn er beim Blick auf die Waage sagt „Darf's ein wenig mehr sein?", dann werden Sie höchstens verständnislos den Kopf schütteln...

Fortsetzung folgt

Computer in Deutschland – ein paar Zahlen

In der deutschen Computer-Hardware-Industrie sind rund 60 000 Personen beschäftigt, weitere 23 000 in der Bürotechnik-Industrie.
Für Datenverarbeitungs-Software wurden 1980 rund 590 Mio. DM ausgegeben, weitere 1,39 Mrd. für DV-Dienstleistungen (z. B. Aufträge an Rechenzentren) und eine Milliarde DM für Beratung und Ausbildung, zusammen also fast 3 Mrd. DM. Inzwischen schätzt man diesen Wert auf mehr als 3,5 Mrd. DM.
1980 gab es für Standard-Software in Deutschland 2600 Anbieter, 1982 mehr als 3000. Allerdings schreiben viele Computeranwender ihre Software selbst und erreichten damit eine Wertschöpfung von 8 Milliarden DM.

(Quelle: ZVEI)

Herwig Feichtinger

Computer für Anfänger

Teil 6

Im letzten Teil ging es etwas theoretisch zu; besprochen wurden solche Dinge wie Hexadezimal-Darstellung, binärer Datentransfer, duale Zahlen, Bits und Bytes. Diesmal wollen wir uns ansehen, wie die einzelnen Teile eines Computers zusammenspielen.

Wir haben schon erwähnt, daß bei den meisten üblichen 8-Bit-Mikrocomputern der Adressenbus 16 Leitungen umfaßt. Das bedeutet, daß man $2^{16} = 65\,536$ unterschiedliche Speicherzellen von je einem Byte Kapazität adressieren kann. Die Adresse einer jeden Speicherzelle könnte man mit einer Dezimalzahl (z. B. 13) oder mit dem binären Bitmuster als Dualzahl (z. B. 0000 0000 0000 1101) darstellen. Üblicherweise tut man das aber hexadezimal, in diesem Falle also als Hex-Zahl 000C. Denn dann ist es ganz einfach, aus dem auf dem Adressenbus anliegenden Bitmuster durch Einteilen der Dualzahl in Vierergruppen die hexadezimale Kurzdarstellung zu erhalten: Vier Bits (ein Nibble) ergeben ja stets genau eine Hex-Ziffer.
Sie sollten nicht krampfhaft versuchen, jede hier genannte Hex-Zahl in das Dezimalsystem umzurechnen. Denn es interessiert im Grunde niemanden, was zum Beispiel die Adresse FA13 dezimal bedeutet. Wichtig ist nur, daß man genau zwischen dezimal und hexadezimal unterscheidet, denn „20" kann ja sowohl eine Hex-Zahl sein, die dem Dezimalwert 32 entspricht, oder eben eine Dezimalzahl.

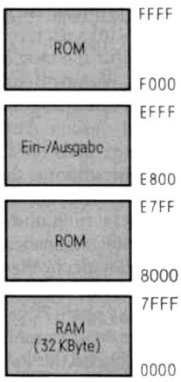


Bild 1. Aufteilung des Adressenraumes beim Tischcomputer CBM-4032

20_{16}, 20H oder $20

Zur Unterscheidung, welches Zahlensystem gemeint ist, gibt es verschiedene Möglichkeiten. Ist eine Zahl gar nicht gekennzeichnet, ist es vermutlich eine Dezimalzahl (es sei denn, es ist ganz offensichtlich eine Hex-Zahl, weil sie eine der Ziffern A...F enthält). Man kann eine Dezimalzahl auch explizit als solche kennzeichnen, indem man ihr den Index 10 verleiht, z. B. 20_{10}.
Ebenso kann man eine Zahl als Hexadezimalzahl kennzeichnen, indem man sie mit dem Index 16 versieht, z. B. 20_{16}. Die Anhänger der Prozessortypen der 80-Familie (8080, Z80 usw.) verwenden auch ein der Zahl folgendes H, z. B. 20H. Und in den CPU-Familien 68XX und 65XX stellt man Hex-Zahlen ein Dollarzeichen voraus, also $20.

Die mit den 16 Adressenbits darstellbaren Hex-Zahlen reichen von 0000 bis FFFF. Innerhalb dieses Adressenraums muß der Mikroprozessor natürlich irgendwie zwischen den einzelnen Bausteinen unterscheiden, die an ihn angeschlossen sind. Jedem Baustein (Speicher, Ein/Ausgabe) ist also ein bestimmter Adressenbereich zugewiesen. In einem typischen 6502-System sieht das so wie in *Bild 1* aus.

Wir schalten den Computer ein

Die in Bild 1 gezeigte Konfiguration entspricht einem verbreiteten Tischcomputer, nämlich dem CBM-4032. Wenn wir dieses Gerät einschalten, passiert folgendes:

1. Beim Einschalten sieht der Mikroprozessor in den beiden Speicherzellen FFFC und FFFD nach, an welcher Adresse er mit der Programmausführung beginnen soll (Reset-Vektor).

2. Der Inhalt von FFFC und FFFD (2 × 8 Bit) ergibt eine 16-Bit-Adresse, an der die CPU nun mit der Programmausführung beginnt.

Wie macht sie das eigentlich genau? Zuerst legt sie die Adresse FFFC auf den Adressenbus (FFFC ist vom CPU-Hersteller willkürlich festgelegt und vorprogrammiert). Die Speicherzelle mit der Adresse FFFC fühlt sich persönlich angesprochen und sendet bereitwillig ihre acht gespeicherten Bits (hex D1) über den Datenbus zum Mikroprozessor. Übrigens muß natürlich dieser Reset-Vektor bei FFFC/FFFD stets innerhalb des Festwertspeichers (ROM) stehen; wären dies Speicherzellen des Arbeitsspeichers (RAM), so wäre ihr Inhalt beim Einschalten ja noch undefiniert.

Zwei Bytes ergeben eine Adresse

Jetzt speichert der Mikroprozessor diese ersten acht Bits und legt „FFFD" auf den Adressenbus. Die Speicherzelle namens FFFD sendet dann ihre acht Bits (hex FC) über den Datenbus zur CPU. Und der Mikroprozessor verbindet diese Bits mit den vorher zwischengespeicherten zu 16 Bits, die er wiederum auf den Adressenbus legt. Die so adressierte Speicherzelle FCD1 enthält bereits den ersten Befehl des auszuführenden Programms.
Was anschließend passiert, bestimmt noch weitgehend der Computerhersteller und nicht der Anwender: Das im

ROM stehende Betriebsprogramm löscht den Bildschirm, führt Selbsttest-Programmstücke aus und fragt dann erst über die Tastatur (die über einen Ein/Ausgabe-Baustein angeschlossen ist) ab, was der Benutzer eigentlich will.

Selbstgeschriebene Programme

Oben war immer die Rede davon, daß das Programm, das der Computer ausführt, im ROM steht. Dort steht es unveränderlich, man könnte also selbst gar keine eigenen Programme schreiben?! Das kann man aber doch, nämlich in den Arbeitsspeicher (RAM). Das im ROM stehende Betriebsprogramm hilft einem sogar dabei, indem es die Tastatur abfragt, gedrückte Tasten auf dem Bildschirm wiedergibt und die gewünschten Befehle oder Daten in den Arbeitsspeicher schreibt.

Und eine bestimmte über die Tastatur eingegebene Anweisung (in Basic RUN) sorgt dafür, daß das ROM-Betriebsprogramm einen Sprungbefehl ins RAM ausführt, so daß nun das in den Arbeitsspeicher geschriebene Programm ausgeführt wird. Ist das geschehen, trifft der Prozessor also auf einen STOP-Befehl oder etwas ähnliches, so erfolgt ein Rücksprung in das ROM-Programm. Das hört sich doch eigentlich ganz einfach an – und es ist auch tatsächlich so einfach. Wer natürlich selbst programmieren will, der muß schon etwas mehr wissen. Zum Beispiel, was für Befehle überhaupt ausgeführt werden können. Und um diese Befehle begreifen zu können, muß man sich den inneren Aufbau des Mikroprozessors etwas näher ansehen.

Register statt Kugelschreiber

Auch auf dem Siliziumchip des Mikroprozessors sind einige Speicherzellen untergebracht; man nennt sie Register. Die CPU 6502 verfügt nur über ein einziges 16-Bit-Register, nämlich den Programmzähler, und einige weitere 8-Bit-Register. Der Programmzähler entspricht dem Kugelschreiber bei unserem Papiercomputer aus Heft 5/1983: Er zeigt auf die Adresse des nächsten auszuführenden Befehls. Immer dann, wenn ein Byte aus dem Programm zur Verarbeitung geholt wurde, erhöht die CPU den Programmzähler (Program Counter, PC) um 1. Die einzige Ausnahme bilden Sprungbefehle, bei denen das Programm an einer ganz anderen Adresse fortgeführt wird, mit der der Programmzähler dann neu geladen wird.

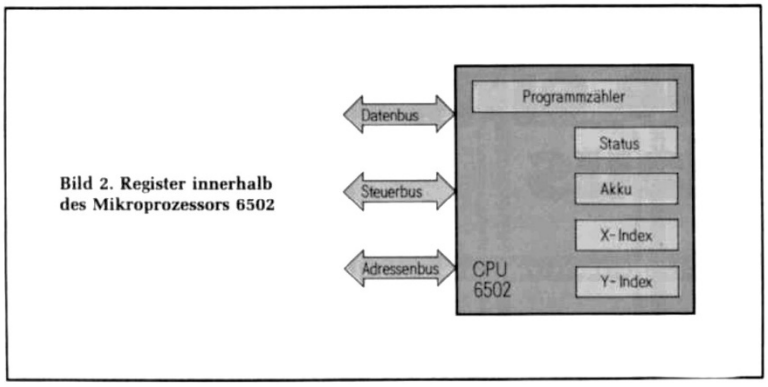


Bild 2. Register innerhalb des Mikroprozessors 6502

Bild 2 zeigt die internen Register der CPU 6502. Neben dem Programmzähler ist wohl der Akkumulator das wichtigste. Der Akku ist 8 Bit lang, und in ihm kann der Prozessor alle arithmetischen Operationen ausführen, wie Addition, Subtraktion und Transfer von Daten von und zum externen Speicher. Dann gibt es noch zwei Indexregister X und Y; mit ihnen werden wir uns erst später befassen. Nicht gezeichnet ist hier der „Stackpointer"; dieses 8-Bit-Register werden wir im Zusammenhang mit Unterprogrammen besprechen.

Das Statusregister

Das Statusregister erfüllt eine wichtige Aufgabe, die wir uns schon anhand des Papiercomputers klarmachen können. Wenn dieser nämlich auf den Befehl „0" stößt (Test auf Null), dann muß er hintereinander zwei Dinge tun: Zuerst muß er testen, ob ein bestimmtes Register den Inhalt Null aufweist. Das muß er sich für einen Moment merken, bis er den zweiten Befehlsteil ausführen kann, nämlich entweder zur nächsten oder zur übernächsten Programmspeicherzelle weiterzugehen.
Ein richtiger Computer braucht zum „Merken" natürlich auch irgendeine Merkzelle. Das Statusregister enthält „Merker" (englisch Flags) für unterschiedliche Aufgaben, wie Test auf Null, Test auf Übertrag bei Addition, Test auf negatives Ergebnis und so weiter. Darauf werden wir auch in Zusammenhang mit den einzelnen Befehlen noch zu sprechen kommen.

Verbindungswege im Prozessor

Die einzelnen Register der CPU sind untereinander über steuerbare Wege verbunden. Je nach Art des gerade auszuführenden Befehls wird ein Register zu einem anderen oder auch zum Adressen- oder zum Datenbus durchgeschaltet. So ist zum Beispiel der Programmzähler normalerweise mit dem Adressenbus verbunden; beim Ausführen eines Sprungbefehls wird er aber zweimal kurz mit dem Datenbus verbunden, damit er sich die Sprungziel-Adresse (16 Bit) nacheinander als zwei 8-Bit-Bytes aus dem Programmspeicher holen kann. Was in Bild 2 nicht gezeichnet ist, das ist das eigentliche Rechenwerk, das Schaltungen für arithmetische Operationen enthält. So ist es im Gegensatz zu unserem Papiercomputer möglich, zwei Bytes mit nur einem einzigen Befehl zu addieren oder voneinander zu subtrahieren. Natürlich gibt es auch Befehle, die einzelne Register sowie den Datenbus in geeigneter Weise mit diesem Rechenwerk (ALU = Arithmetic and Logic Unit) innerhalb des Prozessors verbinden. Und es gibt einen sogenannten Befehlsdecoder, der bestimmt, welcher Befehl welche Verbindungswege öffnet und schließt.

Jetzt geht's richtig los

Übrigens gilt das bisher gesagte weitgehend für alle Prozessortypen. Nur in der Zahl der CPU-internen Register gibt es Unterschiede. So besitzt zum Beispiel der 6800 nur ein Indexregister, aber dafür gleich zwei Akkus. An der prinzipiellen Funktionsweise ändert das aber nichts.
So, genug der Theorie. Beim nächsten Mal wollen wir nämlich schon ein richtiges kleines Programm schreiben, das eine Leuchtdiode an einem Ein/Ausgabe-Baustein zum Blinken bringt!

Fortsetzung folgt

Herwig Feichtinger

Computer für Anfänger

Teil 7

Jetzt geht es richtig los! Wir schreiben ein kleines Maschinenprogramm, um an einem Anschluß des Computers eine Leuchtdiode blinken zu lassen. Dazu verwenden wir eine kleine Treiberschaltung mit einem Transistor.

Das folgende Programmbeispiel ist übrigens für Computer der PET- und CBM-Serie mit dem Mikroprozessor 6502 ausgelegt. Rein prinzipiell ist so etwas aber mit jedem Computer realisierbar, wenn er wenigstens über einen Ein/Ausgabe-Baustein verfügt, an dem man eine Leuchtdiode anschließen kann.

Die Treiberschaltung

Leider kann das aber nicht direkt geschehen; die üblichen Ein/Ausgabe-Bausteine sind an ihren Ausgängen nicht genügend belastbar, um direkt eine Leuchtdiode mit einem Strom von 10 mA oder mehr treiben zu können. Also brauchen wir eine Hilfsschaltung (*Bild 1*). Sie besteht nur aus einem Transistor zur Stromverstärkung und einem Widerstand, der den Strom auf den für die Leuchtdiode (LED = Light Emitting Diode) zulässigen Wert begrenzt.
Die CBM-Computer besitzen alle einen sogenannten User-Port. Diese Schnittstelle zur Außenwelt kann frei programmiert werden: Jeder der acht mit PA 0...7 bezeichneten Anschlüsse kann wahlweise als Ein- oder Ausgang deklariert werden.
Zu diesem Zweck sind dem Ein/Ausgabe-Port PA zwei Adressen im Speicherbereich des CBM zugeordnet, nämlich hex E843 als Datenrichtungs-Register und hex E84F als Datenregister. Wir wählen jetzt beliebig irgendeinen Anschluß, an dem die Leuchtdioden-Treiberschaltung angeschlossen wird, zum Beispiel PA 2. Diese I/O-Leitung (Input/Output) soll als Ausgang dienen. Zu diesem Zweck muß entsprechend die Bitposition 2 im Datenrichtungsregister auf log. 1 gesetzt werden (*Bild 2*). Denn eine 1 in einem Bit des Datenrichtungsregisters definiert die zugehörige I/O-Leitung als Ausgang, eine Null würde sie als Eingang deklarieren. Der erste Befehl in unserem Blink-Programm müßte also lauten: Setze Bit 2 im Datenrichtungsregister (Adresse E843) auf 1. Dazu brauchen wir schon den Akkumulator des Mikroprozessors: Ihn laden wir mit dem entsprechenden Bitmuster 0000 0100 (Bit 2 = 1) als Wert und speichern seinen Inhalt dann an die Adresse E843. Natürlich versteht die CPU nicht Befehle wie „LADE AKKU MIT WERT" oder „SPEICHERE AKKUINHALT AN ADRESSE". Vielmehr entspricht jeder Befehl einem bestimmten Operationscode, der im Mikroprozessor-Handbuch nachgesehen werden kann. „Lade Akku mit Wert" wäre hex A9, „speichere Akkuinhalt an Adresse" hex 8D als Operationscode. Außerdem werden Programme meist hexadezimal geschrieben, also müssen wir aus dem Bitmuster eine Hex-Zahl machen. Das ist weiter kein Problem: Aus 0000 0100 wird hex 04, weil Bit 2 die Wertigkeit $2^2 = 4$ hat.

Die ersten Befehle

Der Programmanfang, der lediglich PA 2 als Ausgang deklariert, sähe dann so aus:

```
2000   A9 04      ; Akku mit hex 04
                    laden
2002   8D 43 E8   ; Akku an Adresse
                    E843 speichern
```

Als Anfangsadresse unseres Mini-Programms haben wir willkürlich hex 2000 gewählt. A9 und 8D sind die Operationscodes der beiden Befehle. 04 ist das Bitmuster mit Bit 2 = 1, und 43 E8 sind die beiden (in der Reihenfolge vertauschten!) Bytes, die zusammen die Adresse E843 ergeben. Diese Vertauschung ist bei fast allen CPU-Familien üblich (8080, Z80, 6502 usw.), nur beim 6800 nicht. Vielleicht verwirrt Sie jetzt, daß der erste Befehl bei Adresse 2000, der zweite aber erst bei 2002 steht. Aber das muß er ja auch. Wenn wir nämlich das Programm einfach anders schreiben, so daß neben jeder Adresse der zugehörige Speicherinhalt steht, dann sähe das so aus:

```
2000   A9    ; Operationscode
2001   04    ; Bitmuster, Bit 2 = 1
2002   8D    ; Operationscode
2003   43    ; Niederwertige Adr.-Hälfte
2004   E8    ; Höherwertige Adr.-Hälfte
```

Der erste Befehl (Akku mit Zahl laden) ist nämlich ein sogenannter Zwei-Byte-Befehl, und der zweite ist ein Drei-Byte-Befehl. Der Mikroprozessor weiß anhand des jeweiligen Operationscodes selbst, um wieviele Bytes er den Programmzähler bis zum nächsten Befehl erhöhen muß. (Übrigens gibt es auch 1-Byte-Befehle, aber dazu später.)
Bei unseren Programmlistings trennen wir der Übersichtlichkeit halber unsere Kommentare, die den jeweiligen links danebenstehenden Befehl erläutern, aber nicht zum Programm gehören (also nicht im Speicher stehen), durch einen Strichpunkt ab.

Die Leuchtdiode blinkt...

Der Ein/Ausgabe-Baustein weiß jetzt also, daß PA 2 als Ausgang dienen soll. Jetzt müssen wir nur noch die Leuchtdiode abwechselnd ein- und ausschalten. Das kann mit zwei Operationscodes geschehen, die wir schon kennen, nämlich A9 (Akku mit Wert laden) und 8D (Akku an Adresse speichern). Wir müssen nämlich nur im Datenregister des Ein/Ausgabe-Bausteins (Adresse E84F beim CBM) das zu PA 2 gehörende Bit 2 abwechselnd auf 0 und 1 setzen. Zusätzlich brauchen wir noch einen Sprungbefehl, der dafür sorgt, daß sich dieser Vorgang immerwährend wiederholt; sein Operationscode ist hex 4C, und ihm

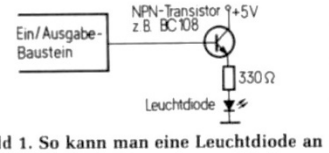


Bild 1. So kann man eine Leuchtdiode an den Ein/Ausgabe-Baustein anschließen

Bild 2. Jedem I/O-Port-Anschluß entspricht ein Bit im Datenregister und im Datenrichtungs-Register des Ein/Ausgabe-Bausteins

PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	Port-PA-Anschlüsse
7	6	5	4	3	2	1	0	Datenrichtungs-Bit Nr.
7	6	5	4	3	2	1	0	Datenbit Nr.
128	64	32	16	8	4	2	1	Bit-Wertigkeit (dezimal)

müssen die zweite Bytes mit der Adresse des Sprungziels folgen. Das erweiterte Programm sieht dann so aus:

```
2000  A9 04     ; PA 2 als
2002  8D 43 E8  ; Ausg. definieren
2005  A9 00     ; Bit 2 = 0
2007  8D 4F E8  ; in PA
200A  A9 04     ; Bit 2 = 1
200C  8D 4F E8  ; in PA
200F  4C 05 20  ; Sprung zu 2005
```

Wenn Sie ein sogenanntes „Monitorprogramm" haben, ein Programm, mit dem man hexadezimal Maschinenprogramme wie dieses in den Speicher schreiben und starten kann, dann können Sie gleich ausprobieren, was das Programm tut. Wenn man es startet, verringert sich nämlich nur die Helligkeit der Leuchtdiode auf den halben Wert, statt daß die LED zu blinken beginnt. Was ist passiert? Haben wir etwas falsch gemacht?

...aber zu schnell!

Nein. Bevor das Programm gestartet wird, sind (nach dem Einschalt-Reset des Computers) alle PA-Anschlüsse als Eingang geschaltet und liegen über interne Widerstände im Ein/Ausgabe-Baustein an +5 V; die LED leuchtet also. Nach dem Programmstart blinkt die LED zwar, aber das tut sie so schnell, daß wir es mit dem Auge nicht verfolgen können. Statt dessen nehmen wir nur eine Abnahme der mittleren LED-Helligkeit wahr. (Mit der Break- oder Stop-Taste des CBM können wir das Programm nicht anhalten. Wir müssen den Computer zur Beendigung des Programmlaufs ausschalten.)
Jetzt sollten wir uns überlegen, wie man das Blinken so verlangsamen kann, daß es auch sichtbar ist. Bedenken Sie, daß ein Mikroprozessor einen Befehl in wenigen Mikrosekunden (das sind millionstel Sekunden) ausführt. Die Leuchtdiode geht also mehrere tausend Male in der Sekunde an und aus. Das muß sich ändern.
Da wir aber nicht verhindern können, daß der Mikroprozessor die Befehle innerhalb von Mikrosekunden ausführt (er kann nicht per Programm z. B. für eine halbe Sekunde angehalten werden), müssen wir zwei Verzögerungsschleifen einbauen: Eine, die nach dem Ausschalten der LED eine Verzögerung von einem Bruchteil einer Sekunde bewirkt, und eine zweite nach dem Einschalten der Leuchtdiode.

Zählschleifen

Die CPU 6502 besitzt außer dem Akku noch zwei Register, die sich gut für solche Verzögerungsschleifen eignen, nämlich das X- und Y-Register. Jedes ist 8 Bit lang, kann also Zahlen von dezimal 0...255 bzw. hexadezimal 00...FF darstellen. Eine einfache Methode, um eine Zeitverzögerung zu erreichen, ist, z. B. das X-Register mit dem Hex-Wert FF zu laden und dann in einer Schleife bis Null herunterzuzählen. Programmtechnisch sähe das so aus (die Adressen haben wir hier weggelassen):

```
A2 FF     ; X-Reg. mit FF laden
CA        ; X-Reg. dekrementieren
D0 FD     ; Bei X≠0 drei Bytes zurück
```

Drei neue Operationscodes haben wir hier kennengelernt: A2 lädt das X-Register mit dem nachfolgenden Wert. CA erniedrigt den Wert im X-Register um 1; man nennt das „dekrementieren". Und D0 ist ein „bedingter relativer Sprung": Er wird ausgeführt, wenn das Ergebnis der vorhergehenden Operation (hier X-Register dekrementieren) nicht Null ist. Als Sprungziel wird hier aber keine absolute, sondern eine relative Adresse angegeben. Das hat erstens den Vorteil, daß dieser Sprungbefehl insgesamt nur zwei Bytes belegt, und zweitens kann man das gesamte Programmstück in einen anderen Adressenbereich verschieben, ohne den Sprungbefehl korrigieren zu müssen! Wie kommt man nun zu der „relativen" Adresse FD? Nun, dieser „Branch", wie man relative Sprungbefehle auch nennt, springt rückwärts bei Relativadressen von hex 80...FF und vorwärts bei hex 00...7F.
Ein Beispiel: Bei Adresse 2030 stehe ein Branch-Befehl (Operationscode D0). XX ist die relative Branch-Adresse, um eine bestimmte abolute Adresse zu erreichen:

```
202D        ; XX = FB
202E        ; XX = FC
202F        ; XX = FD
2030  D0    ; XX = FE
2031  XX    ; XX = FF
2032        ; XX = 00
2033        ; XX = 01
```

Wir sehen: Um das Byte für die relative Branch-Adresse zu ermitteln, braucht man nur, bei seiner eigenen Adresse mit FF beginnend, hexadezimal vorwärts (00, 01, 02 usw.) oder rückwärts (FE, FD, FC usw.) weiterzuzählen. Soll der Sprung nach 202D führen, müßte der Befehl bei 2030/2031 also D0 FB lauten. Nun aber zurück zu unserer Zeitschleife. Wenn man im Mikroprozessor-Handbuch nachsieht, stellt man fest, daß (bei 1 MHz CPU-Taktfrequenz) der Operationscode CA genau zwei und der Sprungbefehl D0 genau drei Mikrosekunden braucht. Ein Schleifendurchlauf benötigt also fünf Mikrosekunden. Da

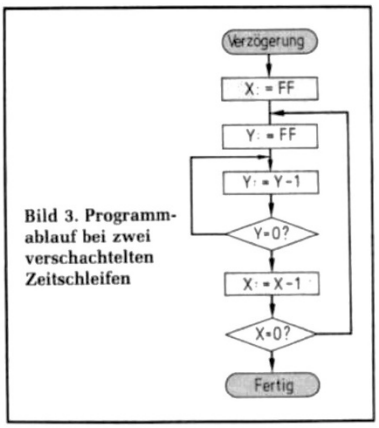


Bild 3. Programmablauf bei zwei verschachtelten Zeitschleifen

maximal 255 Schleifendurchläufe möglich sind, kann man mit dieser Programmschleife eine Verzögerungszeit von 255 × 5 µs erreichen, das sind 1275 µs oder 0,001275 Sekunden. Um da ein Blinken zu sehen, ist das menschliche Auge noch viel zu träge. Also brauchen wir eine längere Verzögerungszeit.

Verschachtelte Schleifen

Wir können natürlich zwei Schleifen ineinander verschachteln, indem wir auch das Y-Register der CPU verwenden. Dazu brauchen wir zwei neue Operationscodes, nämlich A0 für „Y-Register mit Wert laden" und 88 für „Y-Register dekrementieren".
Bild 3 macht den Vorgang als Flußdiagramm deutlich. Und das resultierende Verzögerungsprogramm, wieder ohne Adressen geschrieben, sieht so aus:

```
A2 FF     ; X mit FF laden
A0 FF     ; Y mit FF laden
88        ; Y dekrementieren
D0 FD     ; Branch zu „88"
CA        ; X dekrementieren
D0 F8     ; Branch zu „A0 FF"
```

Innerhalb der mit dem X-Register gebildeten Schleife wird eine Schleife mit dem Y-Register (Dauer: 1275 µs) abgearbeitet. Ein X-Schleifendurchlauf dauert also 1275 µs. Da die X-Schleife ebenfalls hex FF = dezimal 255mal durchlaufen wird, ergibt sich eine Gesamtverzögerung von 255 × 1275 µs = 325 125 µs = 0,325125 Sekunden. Und das ist gerade richtig für eine blinkende Leuchtdiode. Überlegen Sie, wie ein Blink-Programm aussehen könnte. Alles, was Sie dazu benötigen, müßten Sie jetzt kennen.

(*Fortsetzung folgt*)

Herwig Feichtinger

Computer für Anfänger

Teil 8

Haben Sie die Programmieraufgabe vom letzten Mal gelöst? Eine von mehreren möglichen Lösungen finden Sie hier. Außerdem geht es in diesem Teil um Unterprogramme und darum, wie Sie sich die einzelnen Befehle besser merken können.

Im letzten Teil haben wir versucht, ein Programm zu schreiben, das eine Leuchtdiode am Port PA 2 des CBM-Computers zum Blinken bringt. Die nötigen Teilroutinen zur Zeitverzögerung um etwa eine Drittel Sekunde und zum Ein- und Ausschalten der LED wurden dort ja schon veröffentlicht.
Das *Bild 1* zeigt nun das komplette Programm. Zuerst wird PA 2 als Ausgang definiert, dann wird die LED ausgeschaltet, und eine Verzögerung folgt (bestehend aus zwei ineinander verschachtelten Schleifen mit dem X- und Y-Register des Mikroprozessors 6502). Nach Ablauf der Verzögerungszeit schaltet das Programm die LED wieder ein, und wieder folgt eine Verzögerung. Und dann geht das Ganze per Sprungbefehl von vorne los, und zwar nicht etwa bei Adresse 2000, sondern erst bei 2005, denn PA 2 ist nun ein für allemal als Ausgang deklariert, das braucht also nicht nochmal zu geschehen.

```
Adr. Befehl    Kommentar
2000 A9 04     ;PA 2 =
2002 8D 43 E8  ;Ausgang
2005 A9 00     ;LED aus-
2007 8D 4F E8  ;schalten
200A A2 FF     ;Verzög.-
200C A0 FF     ;schleife
200E 88
200F D0 FD
2011 CA
2012 D0 F8
2014 A9 04     ;LED ein-
2016 8D 4F E8  ;schalten
2019 A2 FF     ;Verzög.-
201B A0 FF     ;schleife
201D 88
201E D0 FD
2020 CA
2021 D0 F8
2023 4C 05 20  ;Von vorn
```

Bild 1. So könnte das Maschinenprogramm aussehen, das beim letzten Mal gefragt war: Es läßt eine Leuchtdiode blinken

Kürzere Programme...

Wer genau hinsieht, der erkennt, daß ein Programmteil in Bild 1 unverändert zweimal vorkommt, nämlich der von Adresse 2007...2013 bzw. 2016...2022. Für solche Fälle gibt es die Möglichkeit, diesen Programmteil als Unterprogramm (Subroutine) zu schreiben und ihn aus dem „Hauptprogramm" so oft aufzurufen, wie man ihn eben braucht, in diesem Fall also zweimal.
Dazu müssen wir uns zwei neue Operationscodes merken: hex 20 als Sprungbefehl zu einem Unterprogramm, und hex 60 als Rücksprungbefehl zum Hauptprogramm. Der Trick dabei ist, daß beim Rücksprung keine Adresse angegeben werden muß. Der Befehl „20" sorgt nämlich dafür, daß sich der Prozessor merkt, wo er das Hauptprogramm verlassen hat, eben just die Adresse des 20-Befehls. Und wenn er am Ende des Unterprogramms dann auf den Rücksprungbefehl „60" trifft, dann holt er die alte Adresse aus seinem Gedächtnis und fährt mit dem nächsten Befehl des vorher verlassenen Hauptprogramms fort.
Als Gedächtnis verwendet ein Computer natürlich Speicherzellen. Für die Speicherung von Unterprogramm-Rücksprungadressen ist ein besonderer Speicherbereich im CBM-Computer reserviert, der „Stack" von hex 0100...01FF. Schon beim Einschalten sorgt das im ROM gespeicherte Betriebsprogramm automatisch dafür, daß der „Stackpointer" (ein 8-Bit-Register in der CPU 6502) auf hex FF gesetzt wird. Der Prozessor denkt sich als höherwertigen Adressenteil hex 01 dazu; das ist vom CPU-Hersteller willkürlich festgelegt. Der Stackpointer „zeigt" jetzt also auf die Adresse 01FF.

...durch Unterprogramme

Stößt der Prozessor auf einen Unterprogramm-Sprung (hex 20), so legt er die nötige Rücksprungadresse in den Speicherzellen 01FF und 01FE ab und erniedrigt den Stackpointer um 2, so daß er jetzt auf 01FD zeigt. Wenn innerhalb des Unterprogramms wieder ein Unterprogramm-Sprungbefehl steht (verschachtelte Subroutinen), so würde die neue Rücksprungadresse bei 01FD und 01FC abgelegt. (Zur Erinnerung: Eine Adresse besteht aus 16 Bits und muß daher in zwei 8-Bit-Bytes abgespeichert werden.) Trifft der Prozessor auf den Unterprogrammrücksprung-Befehl hex 60, so inkrementiert er den Stackpointer um 1, holt das erste Adressenbyte, inkrementiert den Stackpointer nochmals um 1, holt das zweite Adressenbyte, setzt beide Adressenhälften zusammen und fährt mit der Programmausführung an der so erhaltenen Rücksprungadresse fort.
Der Stack (auch Stapel- oder Kellerspeicher genannt) ist beim 6502 genau 256 Bytes lang (Adressen 0100...01FF). Also

```
Hauptprogramm:
2000 A9 04
2002 8D 43 E8
2005 A9 00
2007 20 20 20
200A A9 04
200C 20 20 20
200F 4C 05 20

Unterprogramm:
2020 8D 4F E8
2023 A2 FF
2025 A0 FF
2027 88
2028 D0 FD
202A CA
202B D0 F8
202D 60
```

Bild 2. Deutlich kürzer wird das Programm, wenn man die beiden identischen Programmstücke in Bild 1 als Unterprogramm zusammenfaßt

kann er, da jede Rücksprungadresse zwei Bytes belegt, maximal 128 verschachtelte Unterprogramm-„Ebenen" bearbeiten.
Bild 2 zeigt unser Blinkprogramm, durch Verwendung der Unterprogramm-Technik deutlich kürzer geworden. Das (willkürlich) bei Adresse hex 2020 beginnende Unterprogramm speichert den Wert im Akku an die Adresse des Port-Datenregisters und bewirkt anschließend eine Zeitverzögerung von rund einer Drittel Sekunde. Dann erfolgt der Rücksprung zum Hauptprogramm.
Vom Hauptprogramm wird das Unterprogramm zweimal aufgerufen, nämlich von den Adressen hex 2007 und 200C aus.

Die „Mnemonics"

Allmählich dürfte Ihnen der Kopf vor lauter Operationscodes rauchen: Das kann sich ja kein Mensch merken! Die Mikroprozessor-Hersteller wissen das natürlich auch und haben deshalb Namen für die einzelnen Befehle eingeführt, die man sich leichter merken kann als die hexadezimalen Operationscodes. Diese Namen bestehen in der 6502-Schreibweise immer aus drei Buchstaben. Die *Tabelle* gibt eine Übersicht über die von uns bisher verwendeten Befehle; sie sind mit Operationscodes und Mnemonics aufgeführt. „LDA" für „Load Accu" kann man sich schon besser merken als „A9". Und ebenso erkennt man die Arbeitsweise eines Programms besser, wenn es nicht in Form hexadezimaler Bytes, sondern mit Mnemonics geschrieben wurde. Nur eines darf man nicht vergessen: Der Mikroprozessor versteht nur die Bytes, nicht die Mnemonics! Wenn man also ein Programm als Folge mnemonischer Befehle vorliegen hat, dann muß man es erst in Hex-Bytes übersetzen. Diesen Vorgang nennt man „assemblieren". Es gibt auch Hilfsprogramme, die einen mit Mnemonics geschriebenen Programmtext automatisch in Bytes umsetzen, sie heißen „Assembler".

Assembler und Disassembler

Umgekehrt kann man auch (unter Verwendung einer Befehlstabelle) ein in Form von Bytes als Speicherauszug vorliegendes Programm in mnemonische Befehlsworte rückübersetzen, „disassemblieren". Auch dafür gibt es Programme, die das automatisch tun, die „Disassembler". Wendet man einen solchen auf das Programm in Bild 2 an, so sieht das Resultat so aus wie in *Bild 3* abgebildet. Und das kann man schon eher verstehen als eine Folge von nichtssagenden Hex-Bytes! Der Disassembler verfügt auch über eine weitere angenehme Eigenschaft: Die relativen Branch-Adressen nach BNE (hex D0) rechnet er automatisch in absolute Adressen um, so daß man sofort sieht, wohin der bedingte Sprung führt.
Unklar ist Ihnen jetzt vielleicht noch, wie man das nun fertige Programm überhaupt in den Speicher des Computers hineinbekommt und startet. Dazu braucht man ein sogenanntes Monitorprogramm. Aber mit „Monitor" ist hier nicht etwa ein Bildschirm gemeint, sondern ein Hilfsprogramm, um sich Speicherzellen-Inhalte anzusehen, sie zu ändern und um ein Maschinenprogramm starten zu können.

```
2000 LDA # 04
2002 STA E843
2005 LDA # 00
2007 JSR 2020
200A LDA # 04
200C JSR 2020
200F JMP 2005
...

2020 STA E84F
2023 LDX # FF
2025 LDY # FF
2027 DEY
2028 BNE 2027
202A DEX
202B BNE 2025
202D RTS
```

Bild 3. So sieht das Programm aus Bild 2 aus, wenn man es in mnemonischer Form schreibt

Das Monitorprogramm

Zahlreiche Computer haben solch ein Monitorprogramm schon fest in ihrem ROM stehen, bei anderen muß man es vom Hersteller hinzukaufen oder selbst schreiben (in Heft 4/1983 haben wir eines auf Seite 96 veröffentlicht).
Der CBM besitzt ein eingebautes Monitorprogramm, das man z. B. mit SYS 1024 starten kann. Will man die Speicherzellen mit den hexadezimalen Adressen 2000...200F auf dem Bildschirm anzeigen, so tippt man ein: M 2000, 200F und drückt auf die Return-Taste (mit ihr werden alle Eingaben abgeschlossen). Mit dem Cursor (jenem blinkenden Etwas, das die momentane Schreibposition auf dem Schirm markiert) fährt man dann an die hexadezimal dargestellten Datenbytes, die man ändern möchte, gibt neue Werte ein und drückt zum Abschluß jeder Zeile wieder die Return-Taste.
Auf diese Weise können Sie zum Beispiel das in Bild 2 wiedergegebene Programm in den CBM eintippen. Der Start des Programms erfolgt dann mit dem Befehl G 2000 (G für Go, gefolgt von der Startadresse). Allerdings: Das Programm können Sie jetzt nur noch dadurch anhalten, indem Sie den CBM abschalten, weil er keine Reset-Taste besitzt.

Fortsetzung folgt

Tabelle: Einige 6502-Operationscodes und Mnemonics

Operationscode	Mnemonic	Bedeutung englisch	Bedeutung deutsch	Befehlslänge (Bytes)
A9	LDA #	Load Accu Immediate	Lade Akku mit Wert	2
8D	STA	Store Accu Absolute	Speichere Akku an absolute Adresse	3
A2	LDX #	Load X Register Immediate	Lade X-Register mit Wert	2
A0	LDY	Load Y Register Immediate	Lade Y-Register mit Wert	2
CA	DEX	Decrement X Register	Dekrementiere X-Register	1
88	DEY	Decrement Y Register	Dekrementiere Y-Register	1
D0	BNE	Branch if Not Equal	Relativer Sprung, wenn ungleich (Null)	2
4C	JMP	Jump Absolute	Sprung zu absoluter Adresse	3
20	JSR	Jump to Subroutine	Sprung zu einem Unterprogramm	3
60	RTS	Return from Subroutine	Rücksprung vom Unterprogramm	1

Herwig Feichtinger

Computer für Anfänger

Teil 9 (Schluß)

Beim letzten Mal ging es um Mnemonics, also um einigermaßen verständliche Schreibweisen für maschinensprachliche Befehle. Jetzt wenden wir uns den unterschiedlichen Adressierungsarten zu – wieder am Beispiel des Prozessors 6502.

In der Assemblerschreibweise geht aus dem Mnemonic-Wort LDA noch nicht eindeutig ein Operationscode hervor. Den Code A9 haben wir schon kennengelernt. *Tabelle 1* zeigt auch die anderen Möglichkeiten für den Befehl LDA. Sie unterscheiden sich ausschließlich dadurch, mit welcher Speicherzelle der Akku geladen wird.

Die Zero Page

Der Begriff „Zero Page" muß noch erklärt werden, wörtlich übersetzt „Seite Null". Darunter versteht man den Speicherbereich 0000...00FF. Überhaupt wird jener Speicherausschnitt, bei dem sich das höherwertige Adressenbyte nicht verändert, als „Page" bezeichnet. Page 4 ist also beispielsweise der Bereich 0400...04FF.
Die Zero Page hat bei vielen Prozessoren eine besondere Bedeutung, was die Adressierungsarten angeht. Denn auf sie kann man (mit geeigneten Zero-Page-Operationscodes) mit nur einem Adressenbyte zugreifen, während für alle höherliegenden Adressen ja immer zwei Byte erforderlich sind. Um den Akku mit dem Inhalt der Zelle 003F zu laden, kann man (hexadezimal) also entweder AD 3F 00 oder, speicherplatzsparend mit nur zwei statt drei Befehlsbytes, A5 3F schreiben.

Aber Tabelle 1 zeigt bei weitem noch nicht alle möglichen Adressierungsarten des Mikroprozessors 6502. Zwei andere haben wir ja schon kennengelernt, nämlich „relativ" (Branch-Befehle) und „Implied" (der ohnehin nur 1 Byte lange Befehl enthält die Adressierung selbst, z. B. DEX, DEY, RTS). Also haben wir in *Tabelle 2* noch die restlichen Adressierungsarten jeweils mit einem Beispiel-Befehl zusammengestellt, an dem die Wirkung sichtbar wird.

Einige der dort genannten Adressierungsarten klingen vielleicht zunächst sehr kompliziert, sind aber nichtsdestotrotz äußerst nützlich. Wenn Sie sich zum Beispiel das kleine Programm in mc 1983, Heft 6, Seite 30, ansehen (es dient zum Kopieren eines Datenblocks in einen anderen Speicherbereich), dann wird Ihnen gleich klar, wie sich die nachindizierte Adressierung einsetzen läßt.

Die Befehls-Tabelle

Natürlich gibt es nicht jede Adressierungsart für jeden Befehl. Das *Bild* enthält daher eine Übersicht, was bei der CPU 6502 für Operationscodes existieren. Jetzt werden Sie sich nur noch fragen, was solche Zeichen wie X, Y, A, ∨ oder ∧ zu bedeuten haben. Also: X = Indexregister X, Y = Indexregister Y, A = Akkumulator, M = adressierte Speicherzelle, Ms = durch Stackpointer adressierte Speicherzelle (0100...01FF), + = Addition, − = Subtraktion, ∧ = Und-Verknüpfung, ∨ = Oder-Verknüp-

Tabelle 1: Adressierungsarten beim LDA-Befehl

Hex-Code	Ass.-Schreibw.	Adressierung	Wirkung
A9 02	LDA #2	Immediate	Akku mit Zahl 2 laden
AD 34 12	LDA 1234	Absolut	Akku mit Inhalt von Speicherzelle 1234 laden
A5 12	LDA 12	Zero Page	Akku mit Inhalt von Speicherzelle 0012 laden
A1 12	LDA (12,X)	Vorindiziert	Zur Zahl 0012 wird der Inhalt des X-Registers addiert. Das Ergebnis ist die Adresse einer Speicherzelle, z. B. 0054. Der Akku wird nun mit dem Inhalt jener Speicherzelle geladen, deren Adresse in 0054 und 0055 steht.
B1 12	LDA (12),Y	Nachindiziert	Zu dem 16-Bit-Wort in den Zellen 0012 und 0013 wird der Inhalt des Y-Registers addiert. Das Ergebnis ist die Adresse der Speicherzelle, mit der der Akku geladen wird.
BD 34 12	LDA 1234,X	X-indiziert	Der Akku wird mit jener Speicherzelle geladen, deren Adresse sich aus der Summe von 1234 und dem Inhalt des X-Registers ergibt.
B9 34 12	LDA 1234,Y	Y-indiziert	Wie X-indiziert, aber mit Y- statt X-Register.
B5 12	LDA 12,X	X-indiziert, Zero Page	Wie X-indiziert, jedoch nur Adressenbereich 0000...00FF möglich.

Tabelle 2: Weitere Adressierungsarten der CPU 6502

Code-Beispiel	Ass.-Schreibw.	Bezeichnung	Wirkung
60	RTS	Implied	Unterprogramm-Rücksprung; die Zieladresse wird vom Stack geholt.
0A	ASL A	Akku	Akkuinhalt um ein Bit nach links schieben.
D0 05	BNE	Relativ	5 Bytes nach vorn springen, wenn Zero-Flag = 0.
B6 12	LDX 12,Y	Y-indiziert, Zero Page	Wie X-indiziert, Zero Page, jedoch Y-Register.
6C 34 12	JMP (1234)	Indirekt	Sprung zu der Adresse, die in den Zellen 1234 und 1235 steht.
B2 12	LDA (12)	Indirekt, Zero Page	Der Akku wird mit jener Speicherzelle geladen, deren Adresse in 0012 und 0013 steht (nur bei den CMOS-CPUs R65C02 und G65SC02)

fung, ⊻ = Exklusiv-Oder-Verknüpfung, n = Zeitbedarf in Taktzyklen, # = Befehlslänge in Bytes, M_6 = Speicherbit Nr. 6, S = Stackpointer, P = Statusregister. Außerdem sind zur Spalte „Operation" noch einige Fußnoten erforderlich:

1) Steht der Befehl auf einer Page-Grenze, so ist n um 1 zu erhöhen.

2) Wenn der Branch-Befehl ausgeführt wird, ist n um 1 bei gleicher Page oder um 2 bei Verzweigung in eine andere Page zu erhöhen.

		IMME-DIATE			ABSO-LUTE			ZERO PAGE			IM-PLIED			(IND, X)			(IND), Y			ZPG,X			ABS,X			ABS,Y			RELA-TIVE			INDI-RECT			ZPG,Y			PROCESSOR STATUS CODE	
MNE-MONIC	OPERATION	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	OP	n	#	7 6 4 3 2 1 0 N V B D I Z C	MNE-MONIC
ADC	A + M + C → A (1,3)	69	2	2	6D	4	3	65	3	2				61	6	2	71	5	2	75	4	2	7D	4	3	79	4	3				72	5	2				N V • • • Z C	ADC
AND	A∧M → A (1)	29	2	2	2D	4	3	25	3	2				21	6	2	31	5	2	35	4	2	3D	4	3	39	4	3				32	5	2				N • • • • Z •	AND
ASL	C ← [7 0] ←0				0E	6	3	06	5	2	0A	2	1							16	6	2	1E	7	3													N • • • • Z C	ASL
BCC	BRANCH IF C=0 (2)																												90	2	2							• • • • • • •	BCC
BCS	BRANCH IF C=1 (2)																												B0	2	2							• • • • • • •	BCS
BEQ	BRANCH IF Z=1 (2)																												F0	2	2							• • • • • • •	BEQ
BIT	A ∧ M	89	2	2	2C	4	3	24	3	2										34	4	2	3C	4	3													M₇M₆• • • Z •	BIT
BMI	BRANCH IF N=1 (2)																												30	2	2							• • • • • • •	BMI
BNE	BRANCH IF Z=0 (2)																												D0	2	2							• • • • • • •	BNE
BPL	BRANCH IF N=0 (2)																												10	2	2							• • • • • • •	BPL
BRA	BRANCH ALWAYS (2)																												80	2	2							• • • • • • •	BRA
BRK	BREAK										00	7	1																									• • 1 0 1 • •	BRK
BVC	BRANCH IF V=0 (2)																												50	2	2							• • • • • • •	BVC
BVS	BRANCH IF V=1 (2)																												70	2	2							• • • • • • •	BVS
CLC	0 → C										18	2	1																									• • • • • • 0	CLC
CLD	0 → D										D8	2	1																									• • • 0 • • •	CLD
CLI	0 → I										58	2	1																									• • • • 0 • •	CLI
CLV	0 → V										B8	2	1																									• 0 • • • • •	CLV
CMP	A-M (1)	C9	2	2	CD	4	3	C5	3	2				C1	6	2	D1	5	2	D5	4	2	DD	4	3	D9	4	3				D2	5	2				N • • • • Z C	CMP
CPX	X-M	E0	2	2	EC	4	3	E4	3	2																												N • • • • Z C	CPX
CPY	Y-M	C0	2	2	CC	4	3	C4	3	2																												N • • • • Z C	CPY
DEC	DECREMENT				CE	6	3	C6	5	2	3A	2	1							D6	6	2	DE	7	3													N • • • • Z •	DEC
DEX	X-1 → X										CA	2	1																									N • • • • Z •	DEX
DEY	Y-1 → Y										88	2	1																									N • • • • Z •	DEY
EOR	A ⊻ M →A (1)	49	2	2	4D	4	3	45	3	2				41	6	2	51	5	2	55	4	2	5D	4	3	59	4	3				52	5	2				N • • • • Z •	EOR
INC	INCREMENT				EE	6	3	E6	5	2	1A	2	1							F6	6	2	FE	7	3													N • • • • Z •	INC
INX	X + 1 → X										E8	2	1																									N • • • • Z •	INX
INY	Y + 1 → Y										C8	2	1																									N • • • • Z •	INY
JMP	JUMP TO NEW LOC				4C	3	3							7C	6	3																6C	6	3				• • • • • • •	JMP
JSR	JUMP SUB				20	6	3																															• • • • • • •	JSR
LDA	M → A (1)	A9	2	2	AD	4	3	A5	3	2				A1	6	2	B1	5	2	B5	4	2	BD	4	3	B9	4	3				B2	5	2				N • • • • Z •	LDA
LDX	M → X (1)	A2	2	2	AE	4	3	A6	3	2																BE	4	3							B6	4	2	N • • • • Z •	LDX
LDY	M → Y (1)	A0	2	2	AC	4	3	A4	3	2										B4	4	2	BC	4	3													N • • • • Z •	LDY
LSR	0 → [7 0] → C				4E	6	3	46	5	2	4A	2	1							56	6	2	5E	7	3													0 • • • • Z C	LSR
NOP	NO OPERATION										EA	2	1																									• • • • • • •	NOP
ORA	AVM → A (1)	09	2	2	0D	4	3	05	3	2				01	6	2	11	5	2	15	4	2	1D	4	3	19	4	3				12	5	2				N • • • • Z •	ORA
PHA	A →Ms S-1→S										48	3	1																									• • • • • • •	PHA
PHP	P →Ms S-1→S										08	3	1																									• • • • • • •	PHP
PHX	X →Ms S-1→S										DA	3	1																									• • • • • • •	PHX
PHY	Y →Ms S-1→S										5A	3	1																									• • • • • • •	PHY
PLA	S+1→S Ms→A										68	4	1																									N • • • • Z •	PLA
PLP	S+1→S Ms→P										28	4	1																									N V 1 D I Z C	PLP
PLX	S+1→S Ms→X										FA	4	1																									N Z • • • • •	PLX
PLY	S+1→S Ms→Y										7A	4	1																									N Z • • • • •	PLY
ROL	[← [7 0] ←C]				2E	6	3	26	5	2	2A	2	1							36	6	2	3E	7	3													N • • • • Z C	ROL
ROR	[→C→ [7 0]]				6E	6	3	66	5	2	6A	2	1							76	6	2	7E	7	3													N • • • • Z C	ROR
RTI	RTRN INT										40	6	1																									N V 1 D I Z C	RTI
RTS	RTRN SUB										60	6	1																									• • • • • • •	RTS
SBC	A-M-C̄→A (1,3)	E9	2	2	ED	4	3	E5	3	2				E1	6	2	F1	5	2	F5	4	2	FD	4	3	F9	4	3				F2	5	2				N V • • • Z C	SBC
SEC	1 → C										38	2	1																									• • • • • • 1	SEC
SED	1 → D										F8	2	1																									• • • 1 • • •	SED
SEI	1 → I										78	2	1																									• • • • 1 • •	SEI
STA	A → M				8D	4	3	85	3	2				81	6	2	91	6	2	95	4	2	9D	5	3	99	5	3				92	5	2				• • • • • • •	STA
STX	X → M				8E	4	3	86	3	2																											96	4	2	• • • • • • •	STX
STY	Y → M				8C	4	3	84	3	2										94	4	2																	• • • • • • •	STY
STZ	00 → M				9C	4	3	64	3	2										74	4	2	9E	5	3													• • • • • • •	STZ
TAX	A → X										AA	2	1																									N • • • • Z •	TAX
TAY	A → Y										A8	2	1																									N • • • • Z •	TAY
TRB	Ā∧M → M				1C	6	3	14	5	2																													• • • • • Z •	TRB
TSB	A∨M → M				0C	6	3	04	5	2																													• • • • • Z •	TSB
TSX	S → X										BA	2	1																									N • • • • Z •	TSX
TXA	X → A										8A	2	1																									N • • • • Z •	TXA
TXS	X → S										9A	2	1																									• • • • • • •	TXS
TYA	Y → A										98	2	1																									N • • • • Z •	TYA

Mnemonics, Adressierungsarten, Operationscodes und Statusregister-Beeinflussung bei der CMOS-CPU G65SC02

3) Im Dezimalmodus ist n um 1 zu erhöhen.

Die Operationscode-Tabelle berücksichtigt auch die zusätzlichen Befehle, die von den neueren CMOS-CPUs von GTE (G65SCXX) ausgeführt werden. Die rechte Spalte stellt dar, wie die einzelnen Statusregister-Flags von den Befehlen beeinflußt werden, wie Carry, Zero-Flag und so weiter. Sollten Sie einen Computer mit der „alten" NMOS-CPU 6502 haben, so können Sie die Befehle BRA, PHX, PHY, PLX, PLY, STZ, TRB und TSB nicht sowie die Adressierungsart „Indirect" nur bei JMP verwenden. Außerdem gelten bei einigen Befehlen, z. B. BIT, Einschränkungen bezüglich der möglichen Adressierungsarten. Das können Sie aber aus dem Programmierhandbuch zu Ihrem Computer ersehen. Haben Sie dagegen einen Computer mit der neuen Rockwell-CMOS-CPU R65C02, so haben Sie gegenüber unserer Tabelle noch einige Befehle mehr zur Verfügung.

Und wie geht es weiter?

So, jetzt haben Sie schon mal einiges zum Verdauen. Bitte erwarten Sie jetzt nicht, daß wir in einer Serie, die „Computer für Anfänger" heißt, tausend Programmbeispiele für Fortgeschrittene bringen. Schließlich finden Sie ja fast in jedem mc-Heft mehrere Assemblerprogramme für unterschiedliche Mikroprozessor-Typen. Auch wollen wir mit dieser Serie nicht dicke Programmierhandbücher ersetzen. Das Programmieren ist in jedem Fall in erster Linie reine Übungssache. Man kann es nur dadurch lernen, indem man es selbst immer wieder tut, und auch, indem man die Programme anderer Leute zerpflückt.

Betrachten Sie also die zahlreichen Assembler-Programme in mc als Fortsetzung dieser Serie – die, die schon erschienen sind, wie auch diejenigen, mit denen wir Sie in Zukunft versorgen werden.

Deutschland vorn

Oft wird behauptet, daß die deutsche Industrie technologisch nicht mehr überall mithalten könne. Die Firma BASF zeigt, daß sie auf ihren Arbeitsgebieten international anerkannte Spitzenprodukte herstellen kann. Mit Recht ist man dort stolz, daß man alle Technologie, von der Mechanik bis zur Rezeptur, im Hause verfügbar habe. Das zeigt auch das Festplatten-Laufwerk BASF 6188 (*Bild 1*), das in Slim-Line-Technik ausgeführt ist. Der Festplattenspeicher, der die Winchester-Technologie beinhaltet, erreicht mit nur halber Bauhöhe mit zwei Speicherplatten eine Kapazität von 15,0 MBytes (unformatiert).

Die geringe Bauhöhe erlaubt es, zwei Geräte in dem Platz eines herkömmlichen Mini-Disk-Speichers unterzubringen. Der Festplattenspeicher BASF 6185/86 (Winchester-Technologie) hat mit drei 130-mm-Magnetplatten eine Geräte-Speicherkapazität von max. 27,5 MBytes (unformatiert). Die Außenabmessungen entsprechen denen eines herkömmlichen Mini-Disk-Speichers (*Bild 2*). Alle wichtigen Elemente, wie Lese-Schreib-Köpfe mit Positioniereinrichtung sind in dem hermetisch abgekapselten Bereich untergebracht. Der Druckausgleich findet über einen Filter statt, ein weiterer Filter reinigt ständig die im abgeschlossenen Bereich zirkulierende Luft. Als Interface steht eine Mini-Disk-Speicher-ähnliche Schnittstelle zur Verfügung. Dieses Gerät ist mit einem Mikroprozessor ausgerüstet, der einen besonders schnellen Datenzugriff ermöglicht (ramped/buffered seekmode) und außerdem verschiedene Funktionen des Datenspeichers ständig überwacht. Die eingebaute Antriebsmotorbremse und die Steppermotorverriegelung garantieren eine hohe Sicherheit bei Transporten. Mit dem Universal-Testgerät Exerciser 20XX (*Bild 3*) können alle Drive Parameter der Floppy Disk Drives getestet, gemessen und in Verbindung mit der CE-Diskette auch eingestellt werden. Außerdem können die BASF-Festplattenspeicher mit ST 506/412-Interface getestet werden. Ro.

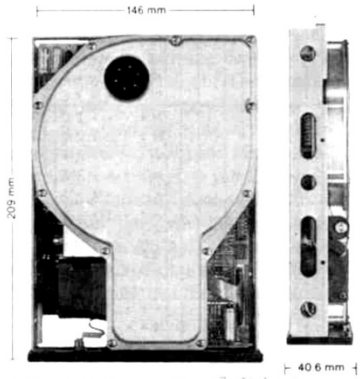


Bild 1. BASF-6188-Slim-Line-Festplattenspeicher

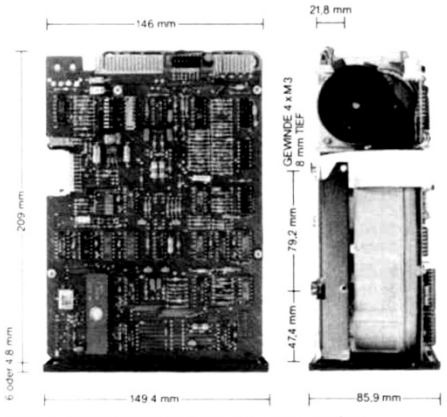


Bild 2. BASF-6185/6186-Festplattenspeicher

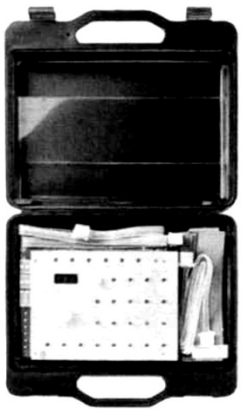


Bild 3. Der Exerciser 20XX hilft Floppies testen