Assembler ist keine Alchimie

Bisher haben wir uns mit dem Innenleben unserer Computer auseinandergesetzt und die wichtigsten Teile der Hardware kennengelernt. Jetzt kommen wir wie versprochen zur Software, nämlich zum Assembler.

Sollten Sie einen Assembler haben, dann schalten Sie ihn jetzt bitte ein. Falls Sie keinen haben, schaffen wir Abhilfe. Wir werden in den nächsten Ausgaben ein komplettes Software-Paket; Assembler, Disassembler und Monitor veröffentlichen. Dieser Assembler wird uns dann den ganzen Kurs hindurch begleiten. Meistens meldet der Assembler sich mit einer Registeranzeige. Sollte Ihrer das nicht tun, dann müssen Sie wohl noch den Monitor anschalten oder speziell einen Befehl für die Registeranzeige eingeben (häufig ist das ein R). Jedenfalls wird nun auf dem Bildschirm der Inhalt der Register angezeigt, (Bild 1).

Die angezeigten Werte sind Beispiele, wie sie beim C 64 auftreten können. PC ist der Programmzähler, der immer auf den nächsten zu holenden Befehl zeigt. (Der Wert $E147 rührt vom SYS-Aufruf, mit dem ich meinen Assembler starte). IRQ zeigt uns an, auf welche Adresse der sogenannte Interrupt-Vektor gestellt ist. Das ist das Byte-Paar 788 (LSB) und 789 (MSB). Auf den Wert $EA31 zeigt es im Normalfall.

Die nächsten acht Angaben beziehen sich auf das Prozessorstatusregister, das wir in der letzten Folge P genannt haben. Die Bedeutung der einzelnen »Flaggen« zeigt Ihnen (Bild 2).

AC ist der aktuelle Inhalt des Akkus.
XR zeigt an, was im X-Register und YR was im Y-Register enthalten ist.
SP (von Stack-pointer = Stapelzeiger) gibt uns Auskunft über den freien Platz im Stapelregister. Damit wissen wir genau, was in diesem Moment in unserem Computer vorgeht. So fremd Ihnen das alles im Augenblick noch vorkommt, bald werden Sie mit dieser Registeranzeige auf vertrautem Fuß stehen.

Wie sieht ein Assemblerprogramm aus?

Das menschliche Gehirn hat dem des Computers vieles voraus. Dazu gehört es beispielsweise, daß ein Mensch allerlei Dinge gleichzeitig tun kann: gehen, sprechen, Musik hören, lächeln, Handbewegungen vollführen, womöglich dabei auch noch etwas kauen und so weiter. Ein Computer ist dazu nicht imstande. Er erledigt eine kleine Aufgabe nach der anderen. Weil er das so schnell macht, hat es für uns den Anschein, es geschähe alles gleichzeitig. Das Maschinenprogramm ist eine Kette solcher kleiner Aufgaben. Das erste Glied daraus, das wir kennenlernen wollen ist der Befehl

LDA.

Das bedeutet: Lade den Akkumulator. Alle Assembler-Befehlsworte bestehen aus drei Buchstaben wie dieser hier auch. Wir haben in der ersten Folge schon gesagt, daß einem solchen Befehl eine 8-Bit-Codezahl entspricht. Das ist hier $A9 oder binär 1010 1001 oder schließlich dezimal 169. Die Code-Zahl muß in einem Speicherplatz stehen, zum Beispiel in $1500 (entspricht dez. 5376). Assemblerlistings sehen dann so aus:

1500 LDA

Hier tritt also die Speicherplatznummer mit einem nachfolgenden Befehl anstelle der vom Basic gewohnten Zeilennummer.

Hier fehlt noch etwas Entscheidendes: Was soll den in den Akku geladen werden? Genauso wie es in Basic Befehle gibt, die für sich alleine stehen können wie CLR oder LIST, gibt es auch im Assembler solche Befehle. Weitaus häufiger aber sind hier Befehle, die ein Argument erfordern (in Basic zum Beispiel PEEK(100)). Dabei ist 100 das Argument). In Assembler gibt es zwei Sorten von Argumenten. Solche, die in einem Speicherplatz unterzubringen sind und andere, die zwei Bytes brauchen. Mit dem Befehlswort (hier also LDA) zusammen gezählt, existieren in Assembler also 1-Byte-Befehle, 2-Byte-Befehle und 3-Byte-Befehle.

Das Argument von LDA ist also das, was in den Akku soll. Laden wir also mal eine 1 in den Akku:

1500 LDA #$01

Wir haben jetzt einen 2-Byte-Befehl erzeugt. Was aber bedeuten # und $ dabei? $ ist leicht zu erklären. Die große Mehrzahl der Assembler nimmt bei Zahlenangaben Hexadezimalzahlen an. Bei einigen muß man das durch das $-Zeichen kennzeichnen. Manche Assembler lassen auch Binärzahlen, Dezimalzahlen und sogar ASCII-Zeichen als Argumente zu. Für jede Eingabeart steht dann vor dem Argument ein Zeichen, das die Art des Argumentes angibt, zum Beispiel häufig »!« für Dezimalzahlen oder % für Binärzahlen. Nun zum #-Zeichen. Es gibt viele Arten, den Akku zu laden. Direkt mit einer Zahl — wie wir hier —, aber zum Beispiel auch mit dem Inhalt eines anderen Speichers und so weiter. Man spricht von der sogenannten Adressierung.

Es gibt eine ganze Menge davon und jede wird auf eindeutige Weise gekennzeichnet. Wenn wir in unseren Akku eine Zahl laden, dann ist das die »unmittelbare« Adressierung, und die kennzeichnet man mit dem #-Zeichen.

Wenn in Speicherstelle $1500 die Codezahl für LDA steht, dann muß die 1 in der Speicherstelle $1501 stehen, wie es sich für einen 2-Byte-Befehl gehört. Wenn Sie nun die Assemblerzeile eingegeben haben und ⟨RETURN⟩ drücken, dann taucht auf dem Bildschirm ... eine Fehlermeldung auf (bei vielen Assemblern). Wir müssen vorher nämlich noch unserem Software-Instrument sagen, jetzt zu assemblieren. Wie das geschieht, ist auch wieder von Assembler zu Assembler verschieden. Die meisten erwarten, daß man vor der Zeile noch ein A eingibt:

A 1500 LDA #$01

Wenn Sie jetzt ⟨RETURN⟩ drücken, zeigt der Bildschirm:

A 1500 LDA #$01 A 1502

und meistens einen blinkenden Cursor, der auf die nächste Eingabe wartet. $ 1502 ist die nächste freie Speicherstelle, und wenn beim Programmablauf der Programmzähler nach dem LDA #$01 auf $1502 deutet, dann erwartet er dort den nächsten Befehl. Wenn dort Unsinn steht, dann stürzt der Computer im allgemeinen ab, je nachdem, welcher Code dann hier zufällig enthalten ist. Wir haben ja 256 Möglichkeiten dafür: $00 bis $FF. Im Gegensatz zu Basic, wo man durch den Interpreter die Möglichkeit hat, Zeilennummern zu bauen wie man will, muß hier das Progamm eine ununterbrochene Perlenschnur von Befehlen in Speicherstellen sein. Durch einige

Befehle läßt sich dieses Prinzip allerdings durchbrechen.

Damit wir die Wirkung von Befehlen sehen können, greife ich auf einen Befehl vor, der ähnlich dem STOP in Basic einen Programmabbruch bewirkt: BRK. Die genaue Funktion soll erst später erklärt werden, aber wir sehen jedenfalls dann, wenn ein Maschinenprogramm auf einen BRK-Befehl läuft, die Registerinhalte angezeigt. Das ist in den meisten Assemblern eingebaut. Wir ergänzen jetzt:

A 1502 BRK

Damit erstmal genug. Steigen Sie aus dem Assembler aus, und starten Sie das Programm. In den meisten Assemblern geht das mit

G 1500

oder sonst von Basic aus mit SYS 5376. Jetzt werden wieder die Register angezeigt. Der Programmzähler steht auf 1503, im Akku steht 01, alle Flaggen außer der Breakflagge sind Null (die unbenutzte Flagge steht immer auf 1). Jetzt ändern wir das Argument:

A 1500 LDA #$00
A 1502 BRK

Wir starten wieder und sehen uns die Register an: Programmzähler 1503, Akku jetzt 00, aber bei den Flaggen hat sich etwas verändert: Die Zero-Flagge ist auf 1 gesetzt. Wir sehen also: Diese Flagge bleibt solange ungesetzt, solange nicht eine Null im Akku auftaucht, erst dann wird sie 1.

Noch einmal ändern wir das Programm:

A 1500 LDA #$FF
A 1502 BRK

Nach erneutem Start steht das Erwartete in den Registern, nur bei den Flaggen ist etwas Merkwürdiges passiert: Die Vorzeichenflagge steht auf 1. Das bedeutet, im Akku soll eine negative Zahl stehen! Nun wissen wir aber, daß $FF = dez. 255 ist. Dieses Rätsel wird uns noch eine Weile begleiten. Es sei hier nur bemerkt, daß kein Fehler vorliegt: Immer wenn in einer Zahl das Bit 7 gleich 1 ist, geht die Vorzeichenflagge auf 1. Die Lösung des Rätsels werden wir bei den negativen Binärzahlen finden.

Wir schließen aus alledem: Der LDA-Befehl beeinflußt die Vorzeichen- und die Zeroflagge.

Der zweite Assemblerbefehl: STA

STA heißt »store accumulator«, also »lege Akkuinhalt ab«. Wie Sie sich denken können, muß auch hier ein Argument auftauchen: Nämlich wohin abgelegt werden soll. Wir legen unseren Akkuinhalt in die erste Bildschirmspeicherstelle (C 64:$0400, VC 20 Grundversion: $1E00, VC 20 mit Erweiterung: $1000). Unser Programm muß also so aussehen:

A 1500 LDA #$01
A 1502 STA $0400 oder die entsprechende Adresse (siehe oben).

Mit diesem STA-Befehl lernen wir eine neue Adressierungsart kennen: Die »absolute« Adressierung. Sie ist daran zu erkennen, daß kein besonderes Merkmal verwendet wird. Die Adresse $ 0400 ist nicht in einem Byte darstellbar, sondern wird aufgeteilt auf zwei Bytes. Im Speicher steht jetzt:

1500 LDA #
1501 $ 01
1502 STA
1503 $ 00 »das ist das LSB«
1504 $ 04 »das ist das MSB«

Hier liegt also ein 3-Byte-Befehl vor, und die nächste freie Speicherstelle ist $ 1505.

Vom Basic her wissen Sie, daß 1 der Bildschirmcode für den Buchstaben A ist, und daß man jeder Bildschirmspeicherstelle auch eine Bildschirmfarbspeicherstelle zuordnet. Um ein eingeschriebenes Zeichen vom Hintergrund abzuheben, muß man dort dann eine Farbinformation eingeben. Der Start dieses Bildschirmspeichers liegt so:

C 64 :$ D800
VC 20 (Grundv.): $9400
VC 20 (Erw. Vers.): $9600.

Der Farbe Schwarz entspricht die Code-Zahl 0. Wir ergänzen unser Programm durch:

A 1505 LDA #$00
A 1507 STA $D800 (oder entsprechender Speicher, siehe oben). Die nächste freie Adresse ist nun $150A. Unser Programm soll jetzt abgeschlossen sein. Damit der Computer aber beim Programmzählerstand $150A nicht Unsinn vorfindet, muß — ähnlich wie bei END in Basic — das Programm auf irgendeine Weise beendet werden. Das kann durch BRK geschehen. Wir wollen aber den dritten Assembler-Befehl kennenlernen:

RTS

Das heißt »return from subroutine«, also »Rückkehr aus Unterprogramm«. In unserem Fall bewirkt das eine Rückkehr zum Basic. Wie Sie sehen, ist das ein 1-Byte-Befehl, also ohne Argument. Auch hier spricht man von einer Adressierungsart, nämlich der »impliziten«-Adressierung. Man erkennt sie am Fehlen des Argumentes. Die Adresse ist implizit, das heißt im Befehl selbst enthalten. Dies ist nämlich ein Befehl, der immer an den Programmzähler gerichtet ist. Der Computer holt sich vom Stapel-Speicher die dort zuoberst liegende Adresse, das ist die, bei der der Computer in ein Unterprogramm gesprungen ist oder aber die, bei der der Computer Basic verlassen hat. Wir ergänzen also noch:

A 150A RTS

PC	IRQ	NV-BDIZC	AC	XR	YR	SP
E147	EA31	10110000	00	00	00	F8

Bild 1. Eine Registeranzeige

N	V	—	B	D	I	Z	C
Negativ-Flagge	Überlauf-Flagge	unbenutzt	Abbruch-Flagge	Dezimal-Flagge	Interrupt-Flagge	Zero-(Null) Flagge	Carry-(Übertrag) Flagge

Bild 2. Das Prozessor-Status-Register P: die Flaggen

Befehlswort	Adressierung	Byteanzahl	Code HEX	Code DEZ	Dauer in Taktzyklen	Beeinflussung von Flaggen
LDA	unmittelbar	2	A9	169	2	N, Z
	absolut	3	AD	173	4	N, Z
LDX	unmittelbar	2	A2	162	2	N, Z
	absolut	3	AE	174	4	N, Z
LDY	unmittelbar	2	A0	160	2	N, Z
	absolut	3	AC	172	4	N, Z
STA	absolut	3	8D	141	4	keine
STX	absolut	3	8E	142	4	keine
STY	absolut	3	8C	140	4	keine
RTS	implizit	1	60	96	6	keine

Bild 3. Die ersten sieben Befehle

und starten das Programm, zum Beispiel von Basic aus mit SYS 5376. Natürlich taucht dann in der linken oberen Ecke des Bildschirmes ein schwarzes A auf. Hier noch der Basic-Lader:

```
10 FOR I=5376 TO 5386:READ A:POKE I,A:NEXT I:END
20 DATA 169,1,141,0,4*,169,0,141,0,216*,96.
```

Die mit * markierten Zahlen müssen für den VC 20 verändert werden: Grundversion: 30 und 148 Erweiterung: 16 und 150.

Eine Kombination von LDA mit STA ist vergleichbar mit dem POKE-Befehl in Basic. Man kann in Assembler nicht direkt eine Zahl in einen Speicher einschreiben, sondern muß den Umweg über den Akku machen. Außer dem Akku eignen sich dazu aber auch das X-Register und das Y-Register. Hierfür gibt es die Befehle LDX (lade X-Register), STX (lege X-Register-Inhalt ab), LDY lade Y-Register) und schließlich STY (lege Y-Register-Inhalt ab). Sie können das übungshalber an unseren kleinen Programm ausprobieren. An dem folgenden Programm sehen Sie noch eine Eigenart der drei Register (Akku, X-Register, Y-Register):

```
A 1500 LDA #$01
A 1502 LDX #$00
A 1504 LDY #$02
A 1506 STA $0400
A 1509 STX $D800
A 150C STY $0401
A 150F STX $D801
A 1512 STA $0402
A 1515 STX $D802
A 1518 RTS
```

Für den VC 20 werden die entsprechenden Speicherstellen für Bildschirm- und Bildschirmfarbspeicher eingesetzt. Dieses Programm druckt — wie erwartet — »ABA« in die linke obere Ecke des Bildschirms. Dabei ist das X-Register dreimal ausgelesen worden und der Akku zweimal. Sie sehen also, daß die Registerinhalte durch die STA-, STX-, STY-Befehle nicht verändert werden.

Wir wollen noch etwas ausprobieren. Bisher haben wir den LDA-Befehl nur mit der »unmittelbaren« Adressierung kennengelernt. LDA, LDX, LDY können auch »absolut« adressiert werden.

```
A 1518 LDA $D800
```

Damit laden wir den Inhalt der Speicherstelle $ D800 (beim VC 20 die anderen Adressen des Bildschirmfarbspeichers) in den Akku. Der Inhalt ist seit $1509 eine Null. Jetzt weiter:

```
A 151B STA $0403
A 151E STX $D803
A 1521 RTS
```

Das müßte beim Ablauf des Programms noch einen Klammeraffen (@mit Bildschirmcode 0) an die vierte Stelle plazieren, was Sie durch SYS 5376 leicht nachprüfen können. Sie sehen, daß man mit diesen sieben Befehlen schon eine Menge anfangen kann.

Wir kommen noch einmal zur Adressierung. Ich hatte Ihnen gesagt, daß LDA #$01 ein 2-Byte-Befehl mit unmittelbarer Adressierung ist (ein Byte für LDA und eines für 01), LDA $D800 ist ein 3-Byte-Befehl (ein Byte für LDA, je eines für das LSB und das MSB von $D800) mit absoluter Adressierung. Da werden Sie sich doch sicher schon gefragt haben, wo bleibt die Adressierung! Wenn aber kein Byte für die Adressenmarkierung (zum Beispiel #) reserviert ist, muß die Kennzeichnung irgendwie anders sein. Wenn Sie einen Disassembler zur Verfügung haben, dann sehen Sie sich damit unser Programm an. Fast jeder Disassembler gibt neben dem Assemblertext auch Byte für Byte in Hexadezimalzahlen die Codes an. Wenn Sie nun die beiden Befehle LDA #$01 und LDA $d800 von den Codes her untersuchen, sehen Sie folgendes:

```
1500 A9 01      LDA #$01
und
1518 AD 00 D8   LDA $D800
```

Offensichtlich gehört jeweils das erste angezeigte Byte zu LDA. Sie sind aber verschieden! Wir sehen daraus, daß die Codezahl für einen Befehl gleich zwei Informationen enthält: Das Befehlswort selbst (LDA) und die Adressierungsart.

Genauso wie man LDA sowohl unmittelbar als auch absolut ausführen kann, ist das auch mit LDX und LDY möglich. Bei den Befehlen STA, STX, STY ist eine unmittelbare Adressierung sinnlos. Für RTS kennt man nur eine implizite Adressierung. Wir fassen das alles zusammen in Bild 3.

In den letzten Spalten von Bild 3 ist noch angegeben, inwieweit durch diese Befehle das Prozessorstatusregister beeinflußt wird, so wie wir es für den Befehl LDA schon ausprobiert haben. In der vorletzten Spalte sehen Sie, wie lange die Ausführung eines Befehls dauert. Wenn sie für einen Taktzyklus etwa eine Mikrosekunde rechnen, dann müßten Sie jetzt ausrechnen können, wie lange unser letztes Programm zur Bearbeitung braucht: 48 Mikrosekunden. Ein vergleichbares Basic-Programm braucht dazu etwa hundertmal so lange: zirka 0,05 Sekunden.

Ein bißchen von Assembler-Alchimie verstehen Sie jetzt schon mit diesen sieben Befehlen. Wir wollen uns nun die Zahlen ansehen, die hier Verwendung finden: Das Binärsystem und das Hexadezimalsystem.

Die einzigen Ziffern, die unser Computer kennt, sind 0 und 1. Sie stehen für »Strom an« oder »Strom aus«, oder für »keine magnetische Erregung« oder »magnetische Erregung«. Deswegen ist es für uns als angehende Assembler-Alchimisten von großer Bedeutung — wir arbeiten ja ganz eng an der Hardware — dieses binäre Zahlensystem handhaben zu können. Das Hexadezimalsystem kennt der Computer eigentlich gar nicht. Wir verwenden es, weil es in einem besonders engen Zusammenhang mit Binärzahlen und dem Aufbau unseres Computers steht: Die größte einstellige Hex-Zahl ist $F, das entspricht genau 1111 im Binärsystem, also dem maximalen Füllungsgrad eines halben Bytes, das Nibble genannt wird. Ein ganzes Byte kann maximal $FF enthalten (binär 1111 1111) und der gesamte Speicheradressenbereich unseres Computers geht bis $FFFF (dezimal 65535). Eine einstellige Hex-Zahl paßt also in ein Nibble, eine zweistellige in ein Byte und eine dreistellige oder vierstellige in zwei Bytes, weshalb man solche Hex-Adressen auch recht leicht in das LSB und das MSB aufteilen kann:

```
$  D8  00
   MSB LSB
```

Rechnen werden wir mit Hexadezimalzahlen nicht, dazu benutzen wir dann das Dezimalsystem oder — wenn es sich um computerinterne Vorgänge handelt — das Binärsystem.

Das Rechnen mit Binärzahlen funktioniert genauso wie das mit Dezimalzahlen. Es gilt also

```
0+0=0
0+1=1
1+0=1
1+1=10
```

wobei binär 10 gleich dezimal 2 ist. Als Beispiel können wir mal 2+1=3 im Binärsystem rechnen:

```
 10 entspricht dez. 2
+01 entspricht dez. 1
---
 11, was ja dezimal 3 ergibt.
```

Die Addition erfolgt also spaltenweise wie beim gewohnten dezima-

Fortsetzung Seite 179

Assembler ist keine Alchimie

Fortsetzung von Seite 152

len Addieren. Auch mit dem Übertrag läuft es wie im dezimalen. Beispiel: 2+2=4:

10 entspricht dez. 2
+10 entspricht dez. 2

100, was dezimal eine 4 ergibt.

In der zweiten Spalte wurde nach der Regel verfahren: 1+1=10. Rechnen wir noch 3+3=6:

11 entspricht dez. 3
+11 entspricht dez. 3

110, was dezimal eine 6 ergibt.

In der ersten Spalte wurde gerechnet 1+1=10, wobei nach dem alten Motto: 0 hin, 1 im Sinn die 0 unter den Strich gesetzt wurde. In der zweiten Spalte wird dann so verfahren: 1+1+1 (das ist die 1, die wir »im Sinn« hatten)=11. Ich meine, daß Sie ohne Probleme die folgenden Übungsaufgaben lösen und dann jeweils dezimal das Ergebnis nachprüfen können: 10+5, 7+1, 16+16, 240+16, 62+65.

In der nächsten Folge werden wir eine Anzahl neuer Assembler-Befehle kennenlernen und erfahren, wie der Computer Zahlen voneinander abzieht. Bei der Gelegenheit lernen Sie dann noch einige Flaggen kennen, und wir werden das Rätsel, warum bei LDA # $FF eine negative Zahl angezeigt wird, lösen. Und vor allen, Sie erhalten einen komfortablen Assembler, Disassembler und einen Monitor.

(Heimo Ponnath/aa)

Inserentenverzeichnis

Einem Teil dieser Ausgabe liegen Prospekte der Firmen Interest-Verlag, Kissing und Microcomputerladen, Berlin, bei.

Impressum

Herausgeber: Carl-Franz von Quadt, Otmar Weber

Chefredakteur: Michael M. Pauly (py)
Stellv. Chefredakteur: Michael Scharfenberger (sc)
Redakteure: aa = Albert Absmeier, leitender Redakteur (130), ev = Volker Everts (278), kg = Karin Gößlinghoff (269), gk = Georg Klinge (169), rg = Christian Rogge (278)
Redaktionsassistenz: Dagmar Zednik-Djadja (237)
Fotografie: Janos Feitser, Titelfoto: Alex Kempkens

Layout: Leo Eder (Ltg.), Dagmar Berninger, Willi Gründl, Walter Höß, Cornelia Weber

Auslandsrepräsentation:
Schweiz: Markt&Technik Vertriebs AG, Alpenstrasse 14, CH-6300 Zug, Tel. 042-22 31 55/56, Telex: 862329 mut ch
USA: M&T Publishing, 2464 Embarcadero Way, Palo Alto, CA 94303; Tel. 001-424 0600; Telex 752 351

Manuskripteinsendungen: Manuskripte und Programmlistings werden gerne von der Redaktion angenommen. Sie müssen frei sein von Rechten Dritter. Sollten sie auch an anderer Stelle zur Veröffentlichung oder gewerblichen Nutzung angeboten werden, so muß dies angegeben werden. Mit der Einsendung von Manuskripten und Listings gibt der Verfasser die Zustimmung zum Abdruck in von der Markt & Technik Verlags AG herausgegebenen Publikationen und zur Vervielfältigung der Programmlistings auf Datenträger. Honorare nach Vereinbarung. Für unverlangt eingesandte Manuskripte und Listings wird keine Haftung übernommen.

Herstellung: Klaus Buck (180), Leo Eder (181)

Anzeigenleitung: Peter Schrödel (156)
Anzeigenverkauf: Alfred Reeb (211)

Anzeigenverwaltung und Disposition: Michaela Hörl (171)

Anzeigenformate: 1/1-Seite ist 266 Millimeter hoch und 185 Millimeter breit (3 Spalten à 58 mm oder 4 Spalten à 43 Millimeter). Vollformat 297 x 210 Millimeter. Beilagen und Beihefter siehe Anzeigenpreisliste.

Anzeigenpreise: Es gilt die Anzeigenpreisliste Nr. 1 vom 1. März 1984.
Anzeigengrundpreise: 1/1 Seite sw: DM 7400,-. Farbzuschlag: erste und zweite Zusatzfarbe aus Europaskala je DM 1000,-. Vierfarbzuschlag DM 3000,-. Plazierung innerhalb der redaktionellen Beiträge: Mindestgröße 1/1-Seite
Anzeigen im Einkaufs-Magazin: Die ermäßigten Preise im Einkaufs-Magazin gelten nur innerhalb des geschlossenen Anzeigenteils, der ohne redaktionelle Beiträge ist. 1/1-Seite sw: DM 5400,-. Farbzuschlag: erste und zweite Zusatzfarbe aus Europaskala je DM 1000,-. Vierfarbzuschlag DM 3000,-. **Anzeigen in der Fundgrube: Private Kleinanzeigen** mit maximal 5 Zeilen Text DM 5,- je Anzeige. **Gewerbliche Kleinanzeigen:** DM 10,- je Zeile Text.
Auf alle Anzeigenpreise wird die gesetzliche MwSt. jeweils zugerechnet.

Vertriebsleitung, Werbung: Hans Hörl (114)

Vertrieb Handelsauflage: Inland (Groß-, Einzel- und Bahnhofsbuchhandel) sowie Österreich und Schweiz: Pegasus Buch- und Zeitschriften-Vertriebs GmbH, Plieninger Straße 100, 7000 Stuttgart 80 (Möhringen), Telefon (07 11) 72 04-0

Erscheinungsweise: 64'er, Magazin für Computerfans, erscheint monatlich, Mitte des Vormonats.

Bezugsmöglichkeiten: Leser-Service: Telefon 089/46 13-1 19. Bestellungen nimmt der Verlag oder jede Buchhandlung entgegen. Das Abonnement verlängert sich zu den dann jeweils gültigen Bedingungen um ein Jahr, wenn es nicht zwei Monate vor Ablauf schriftlich gekündigt wird.

Bezugspreise: Das Einzelheft kostet DM 6,-. Der Abonnementspreis beträgt im Inland DM 72,- pro Jahr für 12 Ausgaben. Darin enthalten sind die gesetzliche Mehrwertsteuer und die Zustellgebühren. Der Abonnementspreis erhöht sich um DM 18,- für die Zustellung im Ausland, für die Luftpostzustellung in Ländergruppe 1 (z.B. USA) um DM 38,-, in Ländergruppe 2 (z.B. Hongkong) um DM 58,-, in Ländergruppe 3 (z.B. Australien) um DM 68,-.

Druck: Druckerei E. Schwend GmbH, Schmollerstr. 31, 7170 Schwäbisch Hall

Verantwortlich: Für redaktionellen Teil: Michael M. Pauly.
Für Anzeigen: Peter Schrödel.

Vorstand: Carl-Franz von Quadt, Otmar Weber

Anschrift für Verlag, Redaktion, Vertrieb, Anzeigenverwaltung und alle Verantwortlichen:
Markt&Technik Verlag Aktiengesellschaft, Hans-Pinsel-Straße 2, 8013 Haar bei München, Telefon 089/46 13-0, Telex 522 052

Telefon-Durchwahl im Verlag:

Wählen Sie direkt: Per Durchwahl erreichen Sie alle Abteilungen direkt. Sie wählen 089-46 13 und dann die Nummer, die in Klammern hinter dem jeweiligen Namen angegeben ist.