Geschwindigkeit durch Maschinencode - so arbeiten Compiler

Programme komfortabel in Basic schreiben, aber mit der Geschwindigkeit von Maschinensprache ausführen lassen — Compiler machen's möglich.

Es ist schon erstaunlich. Da gibt es Programme, die andere Programme als Eingabedaten verwenden und diese in reinen Maschinencode übersetzen, der von der CPU direkt ausgeführt werden kann. Das ist durchaus von Vorteil:

Basic-Programme sind nämlich einfach zu schreiben, aber langsam in der Ausführung. Maschinensprache ist dagegen sehr schnell, aber schwierig zu programmieren. Compiler bilden praktisch die Brücke zwischen den beiden grundsätzlichen Anforderungen nach einfacher Programmerstellung und hoher Ausführungsgeschwindigkeit.

Auf den ersten Blick drängt sich der Vergleich mit einem Assembler auf, der ja auch in Klartext gegebene Befehle in Maschinencode übersetzt. Doch der Vergleich hinkt, denn die Aufgabe eines Assemblers ist vergleichsweise trivial. Zu jedem Klartextbefehl (Mnemonic) wie zum Beispiel LDA gibt es nämlich genau einen Opcode. Der Assembler macht daher im wesentlichen nichts anderes, als in einer zweispaltigen Tabelle das Mnemonic zu suchen und bei Erfolg aus der zweiten Spalte den zugehörigen Opcode zu entnehmen. Zwischen Assembler-Mnemonic und erzeugtem Maschinencode besteht also ein Verhältnis eins zu eins.

Bei sehr maschinennahen Befehlen wie beispielsweise beim Basic-Befehl GOTO hat es der Compiler ähnlich einfach: »GOTO ⟨Zeilennummer⟩« würde von einem 6502-Basic-Compiler übersetzt in »JMP ⟨Adresse⟩«. Bei Befehlen wie »IF X=A+3 THEN Z=5« wird die Sache schon etwas komplizierter. Eine solche Anweisung kann allein schon deshalb nicht direkt übersetzt werden, weil der 6502-Prozessor keinen Maschinenbefehl »IF« kennt.

Hier muß der Compiler also wesentlich mehr leisten, als nur Opcodes zu bestimmten Schlüsselwörtern herauszusuchen. In der Regel entspricht einem Befehl in einer höheren Programmiersprache eine ganze Befehlssequenz auf der Maschinenebene. Die Aufgabe eines Compilers wird daher mit Recht als Übersetzung statt Assemblierung (Zusammenfügung) bezeichnet.

Übersetzt wird immer aus einer höheren Programmiersprache (beispielsweise Basic) in eine maschinennahe »Sprache«, meistens direkt in Maschinencode. Einer Programmiersprache liegt — wie auch jeder natürlichen Sprache — ein Vokabular und eine Grammatik zugrunde.

Auf die Grammatik kommt es an

Vokabeln der Sprache Basic sind zum Beispiel GOTO, PRINT, DATA, aber auch Zahlen und Variablennamen gehören dazu. Daneben gibt es bestimmte »Satzzeichen« wie »=«, »:« etc.

Eine Grammatik ist ja nichts anderes als eine Menge von Regeln, die angeben, wie aus den zur Verfügung stehenden Zeichen und Wörtern korrekte »Sätze« gebildet werden. Folgende drei Regeln bilden zum Beispiel eine einfache Grammatik:

Regel 1: Ein Satz besteht aus einem Subjekt, gefolgt von einem Prädikat.

Regel 2: Ein Subjekt ist eines der Worte HUNDE, COMPUTER, MENSCHEN.

Regel 3: Ein Prädikat ist eines der Worte LESEN, RECHNEN, BELLEN, SCHLAFEN.

In der sogenannten Backus-Naur-Form, die zur Definition der Programmiersprache Algol entwickelt wurde, lassen sich die drei Regeln kürzer und eindeutiger darstellen:

1. ⟨Satz⟩ ::= ⟨Subjekt⟩ ⟨Prädikat⟩

Befehl	Bedeutung
LIT n	Konstante n auf Stack legen
RCL x	Inhalt der Variablen x auf Stack legen
STO x	obersten Stapelwert nach x speichern
ADD	die obersten beiden Stapelwerte addieren
SUB	die obersten beiden Stapelwerte subtrahieren
MUL	die obersten beiden Stapelwerte multiplizieren
DIV	die obersten beiden Stapelwerte dividieren
JMP a	unbedingter Sprung zur Adresse a
JEQ a	Sprung, falls oberstes Stapelelement Null
JNE a	Sprung, falls oberstes Stapelelement nicht Null
JGE a	Sprung, falls oberstes Stapelelement positiv
JLT a	Sprung, falls oberstes Stapelelement negativ
JSR a	Unterprogrammaufruf
RTS	Rückkehr vom Unterprogramm
SQR	Quadratwurzel aus oberstem Stapelelement
SIN	Sinus des obersten Stapelelements
COS	Cosinus des obersten Stapelelements
TAN	Tangens des obersten Stapelelements
ATN	Arcustangens des obersten Stapelelements
LOG	Logarithmus des obersten Stapelelements
EXP	Exponentialfunktion des obersten Stapelelements

Tabelle 1. Typische Befehle eines einfachen Zwischencodes

Basic-Befehl	erzeugter Zwischencode
A=3+5*X	LIT 3, LIT 5, RCL X, MUL, ADD, STO A
GOTO 123	JMP xxxx
GOSUB 500	JSR xxxx
RETURN	RTS
IF A=0 THEN 50	RCL A, JEQ xxxx

Tabelle 2. Beispiele zur Übersetzung von Basic in Zwischencode

2. ⟨Subjekt⟩ ::= HUNDE | COMPUTER | MENSCHEN
3. ⟨Prädikat⟩ ::= LESEN | RECHNEN | BELLEN | SCHLAFEN

In spitzen Klammern stehen dabei immer Verweise auf andere Regeln, die senkrechten Striche trennen verschiedene Alternativen.

Mit dieser Grammatik können durch einfache Anwendung der Regeln Sätze erzeugt werden wie »HUNDE BELLEN« oder »MENSCHEN LESEN« und einige mehr.

Nach Regeln dieser Art ist auch die Basic-Grammatik aufgebaut. Um alle syntaktisch korrekten Sprungbefehle zu beschreiben, genügen folgende Regeln:

1. ⟨Sprungbefehl⟩ ::= GOTO ⟨Zeilennummer⟩
2. ⟨Zeilennummer⟩ ::= {⟨Ziffer⟩}
3. ⟨Ziffer⟩ ::= 0|1|2|3|4|5|6|7|8|9

Regel 1 bestimmt, daß ein Sprungbefehl aus dem Schlüsselwort GOTO, gefolgt von einer Zeilennummer, besteht. Regel 2 besagt, daß eine Zeilennummer aus einer Folge von Ziffern besteht (die geschweiften Klammern deuten die Wiederholung an). Regel 3 schließlich definiert, was eine Ziffer ist. Bleibt noch zu bemerken, daß eine Zeilennummer 999999 zum Beispiel syntaktisch richtig ist, aber aus anderen Gründen nicht zulässig ist.

Häufig werden anstelle der Backus-Naur-Form Syntax-Graphen als anschaulichere Art der Darstellung gewählt. Bild 1 zeigt den Anfang einer syntaktischen Definition des allgemeinen Begriffes »Basic-Programm«, und zwar die ersten drei Regeln. Die Diagramme werden grundsätzlich von links nach rechts gelesen, jede andere Richtung muß durch entsprechende Markierungen angegeben werden.

Regel 1 aus Bild 1 besagt, daß ein Basic-Programm zunächst aus einer Programmzeile besteht. Dann kann entweder bereits Schluß sein, oder aber eine weitere Programmzeile folgen und so fort. Regel 2 definiert den Begriff der Basic-Zeile, bestehend aus Zeilennummer und einer Anweisung, worauf ein Doppelpunkt und eine Anweisung etc. folgen können.

Was hat das Ganze nun mit Compilern zu tun? Sehr viel, denn eigentlich sind wir schon mittendrin im Thema. Jeder Compiler verfügt nämlich über einen sogenannten »Parser«, ein wichtiges Teilprogramm, das die syntaktische Analyse übernimmt. Dazu ist es notwendig, daß der Parser die Syntax genau »kennt«. Das sieht dann so aus, daß der Parser bei der Programmanalyse bildlich gesprochen den Linien der Syntax-Diagramme folgt. Stößt er dabei auf ein Kästchen mit einer Syntax-Regel, dann wird sofort das entsprechende Unterprogramm zur Analyse oder Übersetzung dieser Regel aufgerufen. Die Eingabe für den Parser ist also der Programm-Quelltext, als Ausgabe liefert er einen Zwischencode, also eine Art vor-übersetztes Programm.

Bei diesem Zwischencode handelt es sich in der Regel — man höre und staune — um den Maschinenbefehlssatz eines Computers, den es gar nicht gibt. Tabelle 1 zeigt, wie der Befehlssatz eines solchen hypothetischen Computers aussehen könnte. Natürlich erzeugt der Compiler intern nicht die ASCII-Zeichenfolge für diese Befehle, sondern legt jeden Befehl in einem Byte codiert ab. Ein solches oft als »Token« bezeichnetes Byte ist im folgenden immer gemeint, wenn von einem vom Compiler erzeugten Zwischencode die Rede ist.

Intern arbeitet der »Ghost-Computer«, dessen Befehlssatz gerade der Zwischencode ist, mit einem Stack, wie er jedem Forth-Programmierer, aber auch jedem Benutzer von Hewlett-Packard-Taschenrechnern vertraut sein dürfte. Diesen Rechenstack kann man sich bildlich als Papierstapel vorstellen. Während der Berechnung eines mehr oder minder komplizierten arithmetischen Ausdrucks können Zettel mit Zwischenergebnissen auf den Stapel gelegt oder aber von oben weggenommen werden.

Natürlich kann man mit den Befehlen aus Tabelle 1 arithmetische Ausdrücke nicht in der gewohnten Schreibweise berechnen, sondern es muß zuvor eine Umwandlung in die sogenannte »umgekehrte polnische Notation« (UPN) erfolgen. Ein Ausdruck wie »2 + 3 x 5« würde beispielsweise folgenden Zwischencode ergeben (Stackinhalt in Klammern):

```
LIT 2    (2)
LIT 3    (2,3)
RCL X    (2,3,5)
MUL      (2,15)
ADD      (17)
```

Diese Übersetzung in Zwischencode ist nun relativ einfach zu automatisieren, denn auch für arithmetische Ausdrücke gibt es eine Art Grammatik, die auch die »Punkt-vor-Strich« - Regel berücksichtigt (Bild 2). Der Compiler »kennt« diese Regeln und wendet sie an. Sobald er beispielweise innerhalb einer Formel auf eine Zahl (Konstante) stößt, erzeugt er den Zwischencode »LIT ⟨Konstante⟩«, bei einer Variablen den Zwischencode »RCL ⟨Variablenadresse⟩« und so fort.

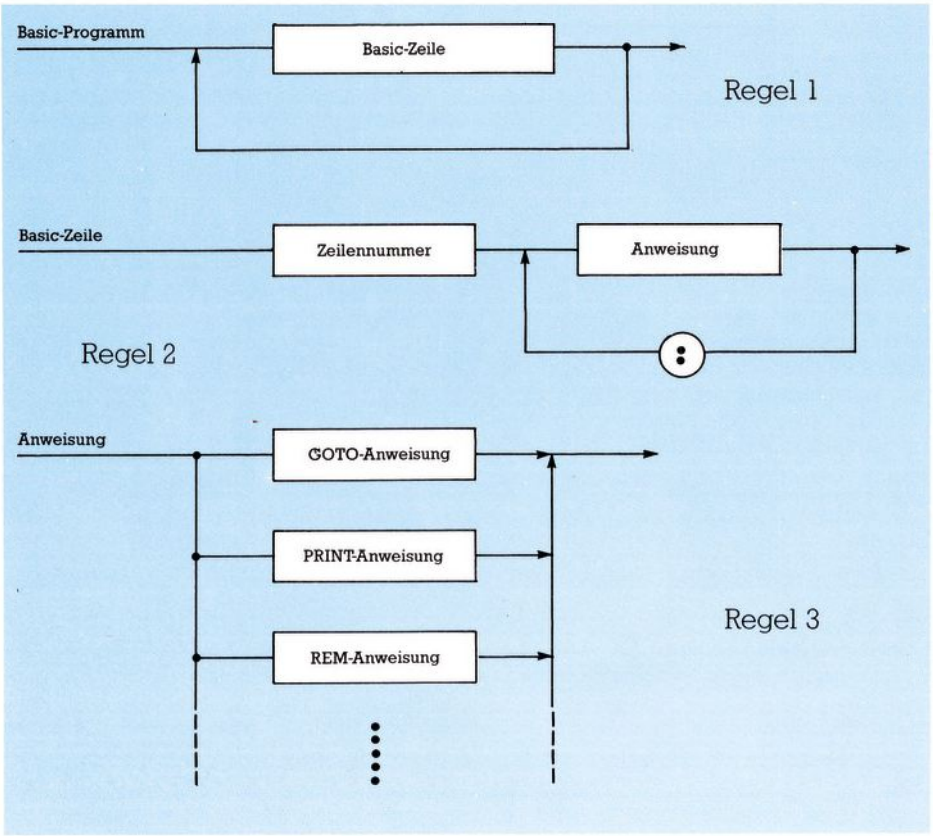

Bild 1. Syntax-Diagramme zur Definition des Begriffs »Basic-Programm«

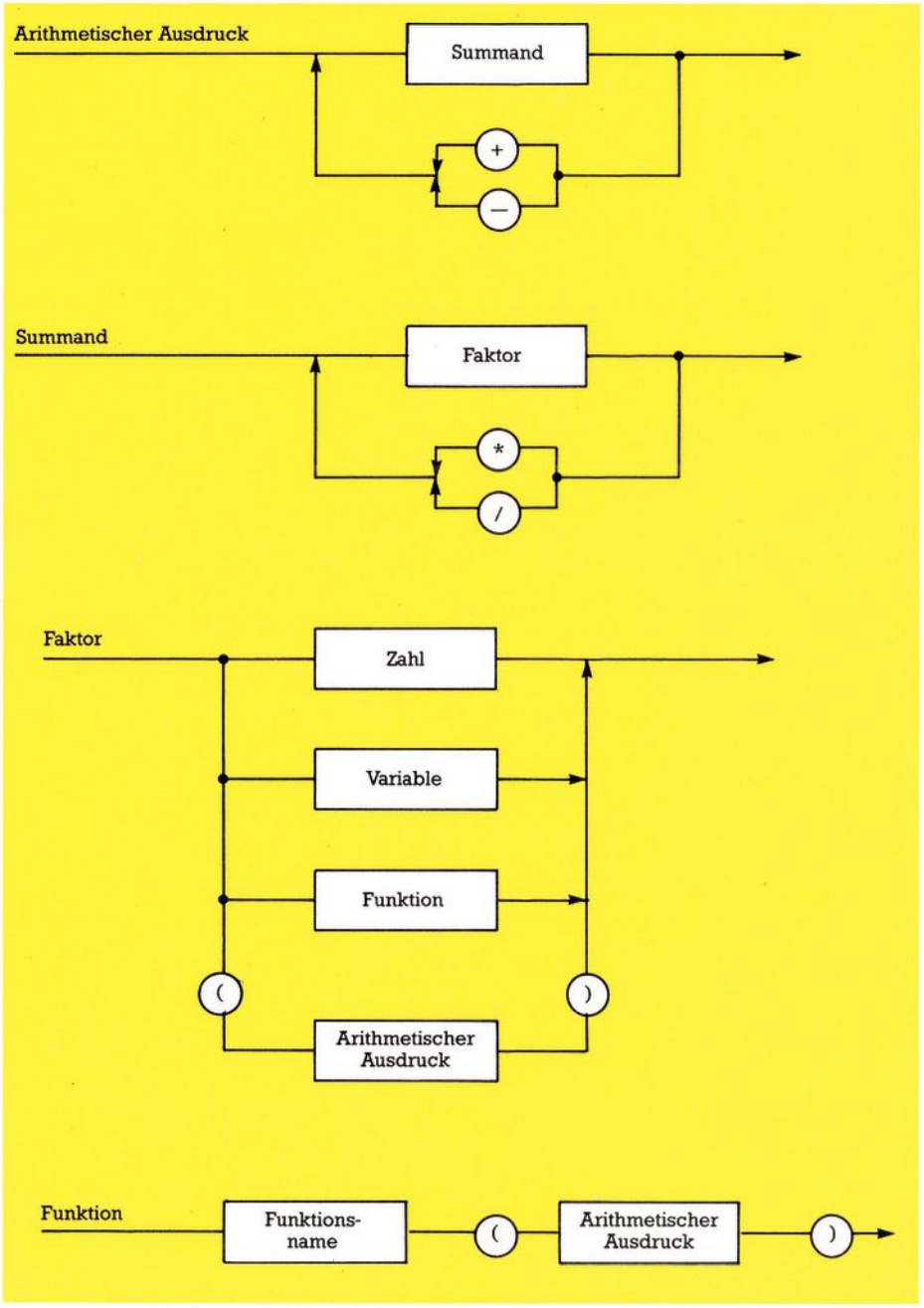

Bild 2. Syntax-Diagramme zur Auswertung arithmetischer Ausdrücke. Die »Punkt-vor-Strich«-Regel ergibt sich aus der Darstellung von selbst.

Tabelle 2 zeigt einige Beispiele für die Übersetzung von Basic-Anweisungen in Zwischencode. Natürlich ist der hier vorgestellte Zwischencode nur ein (unvollständiges) Beispiel. Sie sollten daher nicht erwarten, daß Ihr Basic-Compiler genau diesen speziellen Code erzeugt.

Ist Ihnen bis hierhin etwas aufgefallen ? In der bisherigen Beschreibung der Funktionsweise eines Compilers wurde weder darauf eingegangen, in welcher Sprache der Compiler selbst geschrieben ist, und vor allem nicht, für welchen Prozessor die Übersetzung durchgeführt wird. Der Grund dafür ist einfach: Ein Compiler ist ein Programm, das einem bestimmten Algorithmus realisiert. Es ist also völlig gleichgültig, in welcher Programmiersprache der Compiler geschrieben wurde. So kann ein Pascal-Compiler durchaus in Basic geschrieben werden, wenn das aus verschiedenen Gründen auch nicht besonders sinnvoll wäre. Außerdem haben wir bisher auch noch keinen wirklichen Maschinencode erzeugt, es ist also egal, für welchen Prozessor die Übersetzung stattfinden soll.

Einige Compiler belassen es sogar bei diesem Zwischencode. Es wird dann ein spezielles Interpreterprogramm benötigt, das diesen Zwischencode interpretiert (»Run-Time-Modul«). Derartige Compiler sind besonders bei 8-Bit-Computern verbreitet, da der Zwischencode sehr kompakt ist (ein Byte pro Befehl). Nachteilig ist natürlich, daß dieser Code immer noch interpretiert werden muß, wenngleich das wegen der sehr einfachen Struktur recht schnell geht. Derartige Zwischencode-Programme sind in der Ausführungszeit etwa zwei- bis zehnmal so schnell wie Basic, bei vermindertem Platzbedarf. Der sogenannte »P-Code«, den Pascal-Compiler erzeugen, ist übrigens ebenfalls ein solcher Zwischencode.

Die Erzeugung von Maschinencode

Die Erzeugung von reiner Maschinensprache aus dem Zwischencode ist nun relativ einfach und funktioniert ähnlich wie bei einem Assembler. Statt des Zwischencodebefehls ADD wird zum Beispiel bei einem 6502-System die Befehlsfolge für eine 16-Bit-Addition oder auch einfach nur der Code für »JSR ADD« erzeugt. Dadurch, daß der so erzeugte Code vom Prozessor direkt ausgeführt werden kann, ergeben sich sehr günstige Ausführungszeiten. Solcherart compilierte Programme laufen zwischen zehn- und hundertmal schneller als über einen Interpreter. Allerdings ist der Speicherbedarf gegenüber Zwischencodeprogrammen in der Regel um etwa das Doppelte erhöht.

Interessant ist, daß die unterschiedlichen Eigenschaften der verschiedenen Prozessoren erst bei der Erzeugung von Maschinencode aus dem Zwischencode zum Tragen kommen. Außerdem hatten wir festgestellt, daß es egal ist, in welcher Sprache ein Compiler geschrieben wird. Diese beiden Erkenntnisse haben weitreichende Konsequenzen bei der Entwicklung von Compilern.

Bootstrapping

Eine in der Praxis fast immer angewandte Methode der Compilerentwicklung beruht darauf, den Compiler in der Programmiersprache zu schreiben, die er übersetzen soll. Ein Basic-Compiler würde daher selbst in Basic geschrieben werden.

Der Grund dafür ist einfach: Hat man einmal einen (wenn auch noch sehr einfachen) in Basic geschriebenen Basic-Compiler zur Verfügung, dann kann dieser Compiler sich

Fortsetzung auf Seite 163

Fortsetzung von Seite 41

selbst übersetzen. Denn schließlich läuft der Compiler, da in Basic geschrieben, auf jedem Basic-Computer. Als Ergebnis erhält man den Compiler selbst, aber in Maschinensprache. Nun kann der ursprüngliche Compiler erweitert und verbessert werden. Die Menge der Basic-Befehle, die übersetzt werden können, kann ausgeweitet werden. Mit dem frisch übersetzten »alten« Compiler übersetzt man den erweiterten Compiler. Nun hat man schon einen Compiler zur Verfügung, der einiges mehr kann als die erste Version. Aber auch diesen Compiler kann man verbessern, erweitern und schließlich wieder übersetzen und so fort.

So kann man aus einem primitiven, sogenannten Tiny-Basic-Compiler einen sehr komplexen Basic-Compiler »züchten«. Der große Vorteil, einen Compiler in seiner eigenen Quellsprache zu schreiben, liegt also darin, daß der Compiler selbst als erstes Programm von jeder Erweiterung des Sprachumfanges profitiert. Dieser Vorgang, der auf den ersten Blick ein wenig an Münchhausen erinnert, der sich am eigenen Schopfe aus dem Sumpf zog, wird als »Bootstrapping« bezeichnet.

Wie bei allen rekursiven Verfahren bleibt noch die Frage nach dem Anfangsschritt offen. Denn man braucht mindestens einen ganz einfachen und primitiven ersten Compiler für das Bootstrapping.

Nun, im Falle eines Basic-Compilers ist die Lösung einfach. Da der Compiler selbst in Basic geschrieben ist, kann er auch vom Basic-Interpreter ausgeführt werden. Bei anderen Sprachen muß man in den sauren Apfel beißen und die erste Version »von Hand« in Basic übersetzen, damit das Ding überhaupt erst mal läuft. (ev)

Inserentenverzeichnis

Der Schweizer Ausgabe liegen Prospekte der Firma Techn. Lehrinstitut Onken 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, ev = Volker Everts, gk = Georg Klinge, hm = Harald Meyer, rg = Christian Rogge
Redaktionsassistenz: Gerda Siegl (202)
Fotografie: Janos Feitser, Titelfoto: Alex Kempkens

Layout: Leo Eder (Ltg.), Dagmar Berninger, Willi Gründl

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. (415) 424-0600; Telex 752351

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)

Anzeigenverkauf: Brigitta Fiebig (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. 2 vom 1. Januar 1985.
Anzeigengrundpreise: 1/1 Seite sw: DM 8500,-. Farbzuschlag: erste und zweite Zusatzfarbe aus Europaskala je DM 1400,-. Vierfarbzuschlag DM 3800,-. Plazierung innerhalb der redaktionellen Beiträge: Mindestgröße 1/2-Seite
Anzeigen im Computer-Markt: Die ermäßigten Preise im Computer-Markt gelten nur innerhalb des geschlossenen Anzeigenteils, der ohne redaktionelle Beiträge ist. 1/1-Seite sw: DM 6400,-. 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 11,- 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-Vertriebsgesellschaft mbH, Hauptstätterstraße 96, 7000 Stuttgart 1, Telefon (07 11) 64 83-0

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 6,50. Der Abonnementspreis beträgt im Inland DM 78,- 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: E. Schwend GmbH, Schmollerstr. 31, 7170 Schwäbisch Hall

Verantwortlich: Für redaktionellen Teil: Michael M. Pauly.
Für Anzeigen: Hannelore Schmidt.

Redaktions-Direktor: Michael Pauly

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 522052

Mitglied der Informationsgemeinschaft zur Feststellung der Verbreitung von Werbeträgern e.V. (IVW), Bad Godesberg.
ISSN 0344-8843

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.