Einführung in die Arbeitsweise des Computers

Autor(en): **Einsele, T.**

Objekttyp: **Article**

Zeitschrift: **Bulletin des Schweizerischen Elektrotechnischen Vereins : gemeinsames Publikationsorgan des Schweizerischen Elektrotechnischen Vereins (SEV) und des Verbandes Schweizerischer Elektrizitätswerke (VSE)**

Band (Jahr): **61 (1970)**

Heft 9

PDF erstellt am: **29.05.2024**

Persistenter Link: https://doi.org/10.5169/seals-915939

Nutzungsbedingungen

Die ETH-Bibliothek ist Anbieterin der digitalisierten Zeitschriften. Sie besitzt keine Urheberrechte an den Inhalten der Zeitschriften. Die Rechte liegen in der Regel bei den Herausgebern.
Die auf der Plattform e-periodica veröffentlichten Dokumente stehen für nicht-kommerzielle Zwecke in Lehre und Forschung sowie für die private Nutzung frei zur Verfügung. Einzelne Dateien oder Ausdrucke aus diesem Angebot können zusammen mit diesen Nutzungsbedingungen und den korrekten Herkunftsbezeichnungen weitergegeben werden.
Das Veröffentlichen von Bildern in Print- und Online-Publikationen ist nur mit vorheriger Genehmigung der Rechteinhaber erlaubt. Die systematische Speicherung von Teilen des elektronischen Angebots auf anderen Servern bedarf ebenfalls des schriftlichen Einverständnisses der Rechteinhaber.

Haftungsausschluss

Alle Angaben erfolgen ohne Gewähr für Vollständigkeit oder Richtigkeit. Es wird keine Haftung übernommen für Schäden durch die Verwendung von Informationen aus diesem Online-Angebot oder durch das Fehlen von Informationen. Dies gilt auch für Inhalte Dritter, die über dieses Angebot zugänglich sind.

Ein Dienst der *ETH-Bibliothek*
ETH Zürich, Rämistrasse 101, 8092 Zürich, Schweiz, www.library.ethz.ch

http://www.e-periodica.ch

Einführung in die Arbeitsweise des Computers [1])

Von *T. Einsele,* München

921-935

681.31

Nach einem kurzen historischen Rückblick werden die Zahlendarstellung sowie Aufbau und Wirkungsweise einer stark vereinfachten 1-Adressmaschine behandelt. Am Beispiel der Wurzelberechnung wird die Programmierung in der Maschinensprache und deren Nachteile aufgezeigt; diese konnten in den vergangenen 10 Jahren durch Entwicklung höherer, problemorientierter Programmiersprachen weitgehend behoben werden. Abschliessend wird versucht, einen Überblick über den heutigen Stand und die wesentlichsten Entwicklungstendenzen der Maschinenseite («Hard ware») zu geben. Hierbei stehen die Fragen der Miniaturisierung und Zuverlässigkeit im Vordergrund des Interesses.

Après un rapide aperçu rétrospectif, l'exposé traite de la représentation des chiffres, ainsi que de la structure et du fonctionnement d'une machine d'adresses-1 fortement simplifiée. A l'exemple d'un calcul de racine, on démontre la programmation dans le code de la machine et ses inconvénients; ces derniers ont pu être éliminés au cours des dernières 10 années par le développement de codes de programmation davantage orientés sur les problèmes à traiter. Pour terminer, on tente de fournir un aperçu de l'état actuel et des tendances essentielles d'évolution du point de vue mécanique («Hand ware»). L'intérêt se reporte à ce point de vue principalement aux questions touchant la miniaturisation et la fiabilité.

1. Einleitung

Computer, oder besser Datenverarbeitungs (DV)-Anlagen, verarbeiten Informationen und kommen daher von allen Maschinen und Geräten, die je von Menschen erdacht und gebaut wurden, seinen eigenen Intelligenzleistungen am nächsten. Es kann daher nicht ausbleiben, dass sie unsere Zivilisations- und Kulturentwicklung in steigendem Masse beeinflussen. Die spektakulären Fortschritte in jüngster Zeit stimulieren die menschliche Phantasie in einem Masse, dass in naher Zukunft wohl alle Gebiete unseres Denkens und Schaffens vom Einsatz dieser Maschinen profitieren werden.

Trotz ihrer scheinbar recht jungen Geschichte zeigt ein aufmerksamer Blick in die Vergangenheit [1][2]), dass die Erfolge in der Gegenwart auf Erkenntnissen und Leistungen früherer Generationen aufbauen. Hier ist u. a. *Wilhelm Schickard,* Professor für alte Sprachen, Theologie, Astronomie und Mathematik in Tübingen zu nennen, der 1623 mit seiner Rechenuhr die erste Vierspeziesrechenmaschine gebaut und in einem Brief an den Astronomen *Johannes Kepler* beschrieben hat. Sie beruht auf dem noch heute verwendeten Prinzip des mechanischen Zählrades, das in 10 diskreten Stellungen die 10 Ziffern des Dezimalsystems darstellt.

Das duale Zahlensystem mit den beiden Ziffern 0 und 1 wurde bereits 1703 von dem Philosophen und Mathematiker *G. W. Leibniz* vorgeschlagen und kommt den technischen Möglichkeiten elektronischer Bauelemente sehr entgegen: man braucht nicht die relativ fein quantisierten Stufen von 0...9 darzustellen, sondern kann den Binärziffern 1/0 z. B. den EIN/AUS-Zustand eines Relaiskontakts oder den Stromfluss/kein Stromfluss in einem Transistor zuordnen. Eine solche Realisation ist nicht nur einfach, sondern sie erhöht auch die Sicherheit der einzelnen Bausteine und damit des ganzen Systems.

Die grundlegende Idee des gespeicherten Programms, d. h. des automatischen Ablaufs einer ganzen Serie von Rechenoperationen verdanken wir *Ch. Babbage,* Professor für Mathematik an der Universität Cambridge («Analytical Engine», 1833). Daraus entwickelten sich die heute für alle speicherprogrammierten DV-Anlagen gültigen Prinzipien:

a) Das Programm, eine Folge von Befehlen, wird wie zu verarbeitende Daten gespeichert;
b) Die Befehle können genauso wie Daten verarbeitet und dadurch modifiziert werden;
c) Der unbedingte Sprungbefehl ermöglicht es, Programmteile mehrfach und an beliebigen Stellen zu benutzen;
d) Der bedingte Sprungbefehl erlaubt es der Maschine, selbständig logische Entscheidungen aufgrund vielfältiger Kriterien zu treffen und damit das Programm zu verzweigen.

Diese vornehmlich logischen Prinzipien haben höchst erwünschte praktische Nebeneffekte zur Folge gehabt: Im Vergleich zu einem menschlichen Rechner sind heute die Verarbeitungsgeschwindigkeit und Funktionssicherheit etwa um den Faktor $10^6 ... 10^7$ höher und die Kosten etwa um einen Faktor 10^4 geringer.

2. Aufbau und Wirkungsweise einer DV-Anlage [2]

In Fig. 1 sind die 5 wesentlichen Teile einer DV-Anlage gezeigt:

Speicher
Rechenwerk
Eingabe
Ausgabe
Steuerung

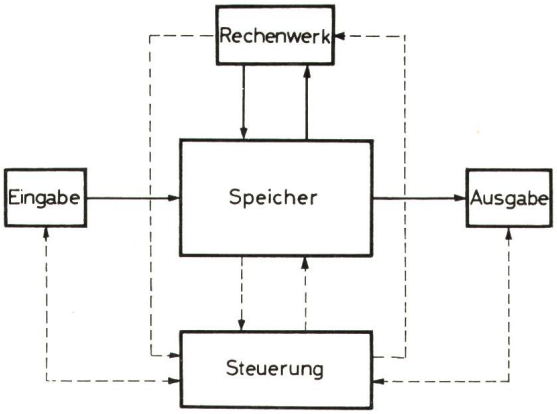
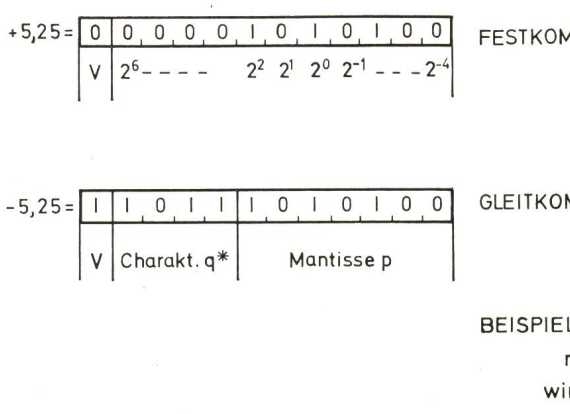


Fig. 1
Vereinfachtes Blockschaltbild einer DV-Anlage

[1]) Gekürzter Vortrag, gehalten im Rahmen der Vortragsreihe über den Stand in wichtigen Bereichen der Elektronik des Eidg. Personalamtes in Bern.

[2]) Siehe Literatur am Schluss des Aufsatzes.

Fig. 2
Duale Zahlendarstellung in Festkomma und Gleitkomma



Numerische Daten, also Zahlen, werden vielfach in dem bereits erwähnten dualen Zahlensystem mit sogenannter fester Wortlänge dargestellt: jede Zahl hat eine feste und gleichbleibende Stellenzahl, z. B. 40 Dualstellen, was etwa 12 Dezimalstellen entspricht. Ausserdem gibt es noch in jedem beliebigen Zahlensystem die beiden Möglichkeiten der Festkomma- und Gleitkomma-Darstellung (Fig. 2):

Festkomma: $Z = \pm \sum_{i=-m}^{+n} b_i B^i$ mit B Basis des Zahlensystems

b_i Ziffern, ganze Zahlen

$0 \leqq b_i \leqq B - 1$

Gleitkomma: $Z = \pm p B^q$ mit p Mantisse, z. B. $p < 1$, d. h. Komma ganz links

q Exponent, ganze Zahl

bzw: $Z = \pm p B^{q*}$ mit q^* Charakteristik, $q^* = q + q_0 \geqq 0$

Positive Zahlen werden durch 0, negative durch 1 in der Vorzeichenstelle V markiert. Die Festkommadarstellung wird in erster Linie für kaufmännische Problemstellungen mit im voraus bekannter Stellenzahl, die Gleitkommadarstellung für technisch-wissenschaftliche Aufgaben mit sehr grossem und oft nicht vorhersehbarem Wertebereich angewandt.

Vor Ausführung einer Arbeit wird der Rechenspeicher mit dem Programm geladen, einer Folge von Befehlen. Dieses veranlasst die Verarbeitung der Daten in der gewünschten Weise und steuert die Ein- und Ausgabe. Als Datenträger wie auch für das Programm werden z. B. Lochkarten, Lochstreifen und Magnetbänder verwendet, aber auch die manuelle Direkteingabe über eine Konsolschreibmaschine oder entfernte Datenstationen bei grossen Teilnehmer-Rechensystemen ist möglich.

Das Programm setzt sich aus verschiedenen Arten von Befehlen zusammen:

a) Arithmetische Befehle, z. B. für die 4 Grundrechnungsarten, Vergleiche von Zahlen, Ausführen logischer Verknüpfungen;
b) Transportbefehle, z. B. für das Umspeichern von Daten auf andere Speicherplätze oder in andere Speicher (Zusatzspeicher);
c) Befehle für Sprünge, Verzweigungen und Adressmodifikationen;
d) Ein-Ausgabebefehle für das Einlesen und Ausgeben von Daten durch periphere Geräte;
e) Start-Stopbefehle für Start und Beendigung eines Programms.

Befehle werden wie Daten codiert und bestehen aus dem Operationsteil, mindestens einem Adressteil und einem Zusatzteil. Der Operationsteil gibt an, was mit dem im Adressteil definierten Operanden geschehen soll. Adressen dienen zur Kennzeichnung von Speicherplätzen oder grösserer Speicherbereiche, da jede eingegebene Information nur über ihre Adresse zugänglich ist. Fig. 3 zeigt den Befehls- oder Operationsschlüssel einer vereinfachten 1-Adressmaschine mit fester Wortlänge von 5 Zeichen, d. h. ein Speicherplatz kann entweder eine 5-stellige Dezimalzahl, 5 alphabetische Zeichen oder einen Befehl aufnehmen. Der Operationsteil eines Befehls benötigt 2 Dezimalstellen, beispielsweise 02 für ADDIERE, verbleiben somit für die Adresse 3 Stellen; damit können 1000 verschiedene Operanden in 1000 Speicherplätzen mit den Adressen 000...999 angesprochen werden. Die Tatsache, dass in all diesen Befehlen nur eine Adresse vorkommt und demzufolge auch nur ein Operand definiert ist, gibt dem Systemkonzept seinen Namen «1-Adressmaschine», Fig. 4. Bei vielen Operationen, z. B. der Addition, sind jedoch zwei Operanden beteiligt. Einer der beiden muss daher an einer fest vereinbarten Stelle gespeichert sein; dies ist im allgemeinen der Akkumulator AK, in dem auch nach Ausführung der Operation das Ergebnis steht. Die Addition von 2 Zahlen, die unter den Adressen a_1 bzw. a_2 gespeichert sind, erfordert daher 2 Befehle. Sie lauten in der abgekürzten Schreibweise nach Fig. 3:

$\langle a_1 \rangle \rightarrow \langle AK \rangle$ $\langle a_1 \rangle$, $\langle AK \rangle$ bedeutet Inhalt der Speicherzelle a_1 bzw. AK

$\langle AK \rangle + \langle a_1 \rangle \rightarrow \langle AK \rangle$ $\rightarrow$ ersetzt Inhalt (Zeitfolge beachten)

Das Resultat, in diesem Fall die Summe der beiden Operanden, befindet sich im Akkumulator und kann durch einen Befehl:

$$\langle AK \rangle \rightarrow \langle a_3 \rangle$$

unter der Adresse a_3 abgespeichert werden.

Der in Fig. 3 gezeigte Operationsschlüssel besitzt keinen Zusatzteil, auf den man in der Praxis aus verschiedenen Grün-

Op. Schlüssel	Bedeutung	Kurzschrift
0 0	Stop, Adresse belanglos	
0 1	Löschen, addieren	$\langle a \rangle \Rightarrow \langle AK \rangle$
0 2	Addieren	$\langle AK \rangle + \langle a \rangle \Rightarrow \langle AK \rangle$
0 3	Subtrahieren	$\langle AK \rangle - \langle a \rangle \Rightarrow \langle AK \rangle$
0 4	Multiplizieren	$\langle AK \rangle \times \langle a \rangle \Rightarrow \langle AK \rangle$
0 5	Dividieren	$\langle AK \rangle / \langle a \rangle \Rightarrow \langle AK \rangle$
0 6	Stellenversetzen rechts	$\langle AK \rangle \times 10^{-a} \Rightarrow \langle AK \rangle$
0 7	Stellenversetzen links	$\langle AK \rangle \times 10^{+a} \Rightarrow \langle AK \rangle$
0 8	Speichern	$\langle AK \rangle \Rightarrow \langle a \rangle$
0 9	Sprung	$a \Rightarrow p_{neu}$
1 0	Sprung bei ⟨AK⟩ NEGATIV	$a \Rightarrow p_{neu}$
1 1	Einlesen	$\langle EP \rangle \Rightarrow \langle a \rangle$
1 2	Ausliefern	$\langle a \rangle \Rightarrow \langle AP \rangle$
⋮	⋮	⋮

Operat.	Adresse a

Fig. 3
Operationsschlüssel einer vereinfachten 1-Adressmaschine

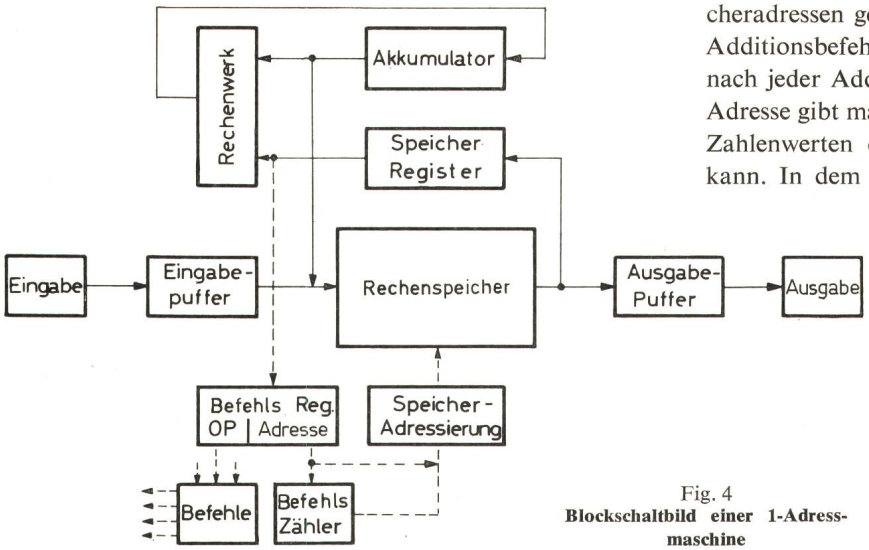


Fig. 4
Blockschaltbild einer 1-Adress-maschine

den nicht verzichten kann. So sind hier ergänzende Funktionen wie z. B. Befehlserweiterungen, Operandenlängen, Speichersperren u. a. untergebracht.

Die einzelnen Befehle eines Programms werden in aufeinanderfolgenden Speicherplätzen gespeichert (lineares Programm) und ohne besondere Anweisung durch die Steuerung mit Hilfe eines sich stetig fortschaltenden Befehlszählers zur Ausführung aus dem Rechenspeicher abgerufen. Soll im Verlauf eines Programms ein Sprung zu einem anderswo gespeicherten Programmteil erfolgen, so bewirkt dies ein unbedingter Sprungbefehl — nach Fig. 3 ist dies der Operationsschlüssel 09 —, dessen Adressteil den Speicherplatz des nächsten Befehls angibt; der Befehlszähler wird auf diesen Wert P_{neu} eingestellt und setzt durch die automatische Fortschaltung (z. B. Erhöhung der Adresse um +1) den weiteren Programmablauf fort. Ein Beispiel dafür zeigt Fig. 5a, wo nach dem Leseprogramm für Lochkarten in einem unbedingten Sprung das Leseprogramm für Lochstreifen ausgelassen und zum Verarbeitungsprogramm übergegangen wird. Bei bedingten Sprungbefehlen entscheidet der Computer selbst aufgrund der Bewertung von z. B. Zwischenresultaten, ob ein Sprung zu einem anderen Programmteil stattfindet oder ob der nächstfolgende Befehl ausgeführt wird. Programme, deren Teile mehrmals durchlaufen werden, nennt man iterative Programme. Sie ermöglichen den Ablauf einer grossen Anzahl von Zyklen, die immer aus denselben Befehlen, aber mit verschiedenen Daten bestehen. Der Iterationsprozess wird automatisch beendet, wenn ein vom Programmierer definiertes Resultat erreicht ist, z. B. nach Fig. 5b, sich das Kapital verdoppelt hat.

Fig. 6 zeigt einen Iterationsprozess zur Berechnung der Wurzel $r = \sqrt{x^2 + y^2}$, Fig. 7 die Iterationsschleife mit dem bedingten Sprungbefehl ⟨AK⟩ NEGATIV? auf Speicherplatz 024. Als erster Näherungswert der Wurzel ist im einfachsten Fall von ganzen Zahlen der grösstmögliche Wert $r_0 = 317$ zugrunde gelegt.

Von besonderer Bedeutung ist auch die Möglichkeit, den Adressteil von Befehlen durch arithmetische Operationen zu verändern. Zur Berechnung der Summe $Y = \sum_{i=1}^{100} x_i$, deren Summanden x_i in 100 aufeinanderfolgenden Speicherplätzen untergebracht sind, müssen ohne die Möglichkeit der Adressenänderung 100 Additionsbefehle mit 100 verschiedenen Speicheradressen gegeben werden. Stattdessen genügt ein einziger Additionsbefehl mit Adressmodifikation, dessen Adressteil nach jeder Addition um 1 erhöht wird. Anstelle der direkten Adresse gibt man nur an, wie sich die Maschine aus gegebenen Zahlenwerten die effektive Speicheradresse selbst errechnen kann. In dem erwähnten Beispiel durchläuft das Programm den Additionszyklus so oft, bis die vorgegebene Anzahl von 100 Additionen erreicht ist.

3. Programmiersprachen

Eine DV-Anlage kann nur solche Befehle ausführen, die in der vom Konstrukteur festgelegten «Maschinensprache», z. B. nach Fig. 3, vorliegen. Diese Maschinensprache ist wie die Informationsdarstellung auf den beiden Binärwerten 0 und 1 aufgebaut und hat deshalb zu unserer natürlichen Sprache keine Verwandtschaft. Die Programmierer der ersten Computergeneration mussten diese primitive Maschinensprache erlernen, die erhebliche Nachteile aufweist:

a) Operationsschlüssel, ob dual oder dezimal, ist wenig aussagefähig;
b) Adressen werden als überflüssig empfunden, desgleichen eine Speicherübersicht, die bei umfangreicheren Problemen unbedingt notwendig ist;
c) Viele Befehle, z. B. Transportbefehle, haben mit der Problemlösung nichts zu tun;
d) Programme in Maschinensprache sind lang und unübersichtlich, daher
e) Hohe Fehlermöglichkeit, korrekturfeindlich;
f) Programmierer muss über Maschinenkenntnisse verfügen.

Es war nur natürlich, dass man sehr bald höhere Programmiersprachen entwickelte, die unserer natürlichen Sprache näher stehen und leichter zu erlernen sind. Die symbolischen Programmiersprachen verwenden für den Operationscode leicht zu merkende mnemonische Abkürzungen (z. B. «ADD» für «Addiere») und auch die nichts aussagenden Adressen der Speicherplätze werden durch symbolische Adressen (echte

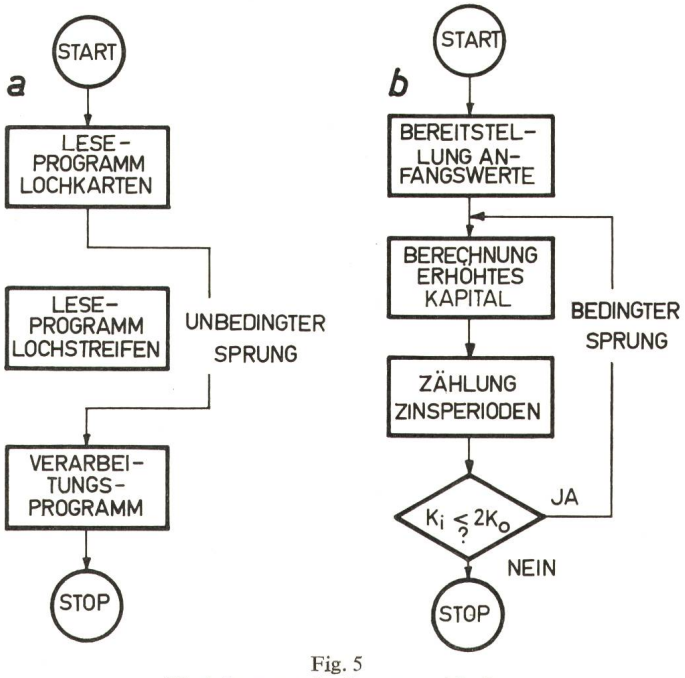


Fig. 5
Blockdiagramm des Programmablaufes
a unbedingter Sprung; *b* bedingter Sprung

a) PROGRAMM

Speicherpl.	Inhalt	Bemerkungen	Akku.
⋮	⋮	⋮	⋮
010	01 100	Akku löschen, Add. x	x
011	04 100	Mult. mit x	x^2
012	08 102	$x^2 \Rightarrow$ (102)	
013	01 101	Akku löschen, Add. y	y
014	04 101	Mult. mit y	y^2
015	02 102	Add. von x^2	$x^2 + y^2 = z$
016	08 102	$z \Rightarrow$ (102)	
017	01 103	Akku löschen, Add. r_0	r_0 = 00317 (0-te Näherung für $r = \sqrt{z}$)
018	08 104	$r_n \Rightarrow$ (104)	
019	01 102	Akku löschen, Add. z	z
020	05 104	Division mit r_n	z/r_n
021	02 104	Add. von r_n	$r_n + z/r_n$
022	05 105	Division mit 2	$(r_n + z/r_n)/2 = r_{n+1}$
023	03 104	Subtraktion von r_n	$r_{n+1} - r_n$
024	10 026	Sprung bei (Ak) NEGATIV	
025	09 ...	Unbedingter Sprung ...	
026	02 104	Add. von r_n	r_{n+1}
027	09 018	Sprung nach 018	

b) DATEN

Speicherpl.	Inhalt
100	— x —
101	— y —
102	— z —
103	— r_0 —
104	— r —
105	00 002

Fig. 6
Programmierbeispiel

$$r = \sqrt{x^2 + y^2} + \sqrt{2}$$

mit Iterationsformel

$$r_{n+1} = \frac{r_n + \frac{z}{r_n}}{2}$$

Adressen oder beliebige Namen) ersetzt, welche die zu verarbeitende Information eindeutig kennzeichnen und vom Programmierer frei wählbar sind, z. B.

SUB RABATT

d. h. subtrahiere Rabatt (von einem zuvor errechneten Rechnungsbetrag).

Bei den noch weiter entwickelten «problemorientierten Programmiersprachen», die zur Lösung einer ganzen Klasse von Problemstellungen besonders geeignet sind, wird die Programmierarbeit noch weiter vereinfacht, indem durch eine einzige Anweisung gleich mehrere Befehle in der Maschinensprache ausgelöst werden; z. B. für die Ermittlung des Rechnungsbetrags einer Stromrechnung:

RECHNBETR = (NEU-ALT) * PREIS + GRUNDPREIS

Die Umwandlung eines Programms, das in einer höheren Programmiersprache abgefasst ist (Quellenprogramm) wird von der Maschine in einem ersten Durchlauf mit Hilfe spezieller Übersetzerprogramme (Assembler bzw. Compiler) in die Maschinensprache (Objektprogramm) übersetzt (Fig. 8). Höhere Programmiersprachen wie FORTRAN (**For**mula **Tran**slation), ALGOL (**Algo**rithmic **L**anguage), COBOL (**C**ommon **B**usiness **O**riented **L**anguage), PL I (**P**rogramming **L**anguage **I**) u. a. sind leicht verständliche Problemsprachen, erfordern keine Maschinenkenntnisse und haben den Vorteil,

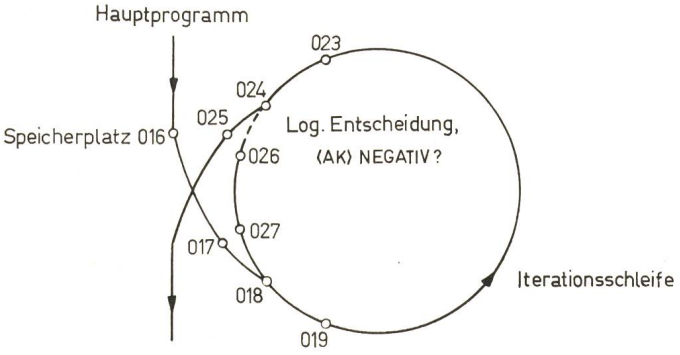


Fig. 7
Flussdiagramm des Programmablaufes

dass Programmierverwaltungsarbeiten, wie Ausrechnen der Speicheradressen und Führen einer Speicherbelegung, durch das Übersetzungprogramm automatisch erledigt werden. Dadurch werden die Fehlermöglichkeiten erheblich vermindert und die Programmierzeiten entscheidend verkürzt.

In FORTRAN oder PL I lautet die Anweisung zur Berechnung der Wurzel des obigen Beispiels:

R = SQRT (X * X + Y * Y)

SQRT Operator «Square Root»

4. Entwicklungstendenzen [3]

Die zukünftige Entwicklung vollzieht sich in 3 wesentlichen Richtungen:

a) Geräteentwicklung («Hardware»);
b) Programmierungsentwicklung («Software»);
c) Erschliessung neuer Anwendungsmöglichkeiten.

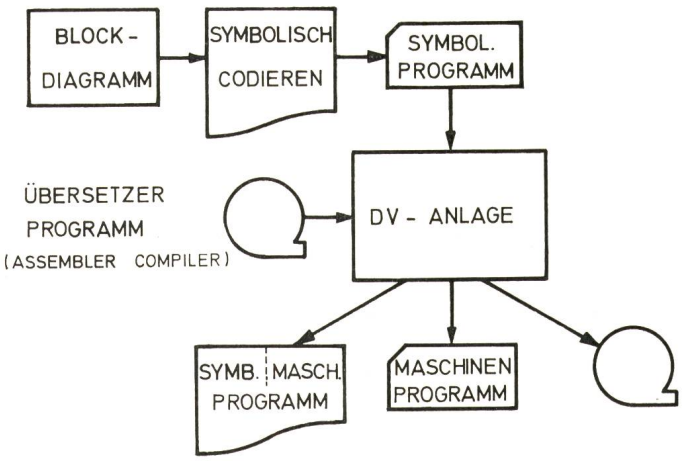


Fig. 8
Programmerstellung bei höheren Programmiersprachen

An die Geräteentwicklung, die hier kurz aufgezeigt werden soll, sind folgende Forderungen zu stellen:

a) Hohe Verarbeitungsgeschwindigkeit, d. h. logische Verknüpfungs- und Speicherelemente mit möglichst kurzer Schaltzeit;
b) Hohe Speicherkapazität bei kurzer Zugriffszeit;
c) Geringer Leistungsverbrauch der logischen Bausteine und Speicherelemente;
d) Miniaturisierung, damit die Signallaufzeiten verkürzt und die höchstmögliche Verarbeitungsgeschwindigkeit erreicht wird;
e) Hohe Zuverlässigkeit, da sehr viele Bauelemente in einem System erforderlich sind; die Computer sollen möglichst wartungsfrei arbeiten;
f) Niedrige Herstellungskosten, d. h. günstiges Leistungs-Preisverhältnis.

Viele dieser Forderungen widersprechen sich, wie z. B. kurze Schaltzeit und geringer Leistungsverbrauch oder hohe Speicherkapazität bei kurzer Zugriffszeit. Es ist daher Aufgabe des Ingenieurs einen optimalen Kompromiss zu finden.

Die enorme Steigerung der Verarbeitungsgeschwindigkeit — beispielsweise die Erhöhung der Additionsgeschwindigkeit seit 1950 alle 10 Jahre etwa um den Faktor 100 (Fig. 9) — beruht auf Fortschritten der Technologie und Systemplanung, d. h.

a) Hohe Schaltgeschwindigkeit bzw. kurze Schaltzeit elektronischer Bauelemente;
b) Sinnvolle Organisation des zeitlichen Ablaufs der Maschinenoperationen.

4.1 Stand und Entwicklung der Speichertechnik

Die Speicher von DV-Anlagen haben sich in eine ausgesprochen hierarchische Struktur entwickelt. Grund dafür ist die Tatsache, dass man bis heute keinen wirtschaftlichen Speicher bauen kann, der die beiden Forderungen nach hoher Speicherkapazität und kurzer Zugriffszeit in sich vereinigt. So finden wir neben den Registerspeichern sehr geringer Kapazität von ca. 100 bit den Rechenspeicher als integralen Bestandteil der Zentraleinheit einer DV-Anlage. Hinzu treten schon bei kleinen bis mittleren Anlagen die externen Zusatzspeicher sowie in jüngster Zeit die Archiv/Festspeicher sehr grosser Kapaziät.

Die Entwicklung der Speicherkapazität (Fig. 10), insbesondere der Rechen- und Zusatzspeicher, zeigt über die Jahre hinweg einen ähnlichen Verlauf wie die Additionsgeschwindigkeit; bei beiden Arten erhöhte sie sich alle 10 Jahre etwa um den Faktor 100. Ausserdem findet man, dass die Zusatzspeicher ganz grob um den Faktor 100 über den Rechenspeichern liegen, was auch auf das Verhältnis der Archiv/Festspeicher zu den Zusatzspeichern zuzutreffen scheint.

Soweit sich die Entwicklung übersehen lässt, wird sich der Trend zu immer grösseren Speicherkapazitäten fortsetzen. Dabei werden die optischen Speicher mit ihrer sehr hohen Speicherdichte sehr gute Chancen haben. Vielleicht entwickeln sie sich zu einem Archivspeicher, dessen Speicherstellen nur einmal beschrieben und dann ausschliesslich nichtlöschend gelesen werden (Prinzip des Festspeichers); das Löschen und ein Austausch der Information könnte damit entfallen. Voraussetzung für einen solchen Archivspeicher ist allerdings eine so grosse Speicherkapazität, dass sie für die gesamte Lebensdauer des Systems ausreicht und dass die Adressierung vermutlich nach assoziativen Begriffen, ähnlich dem menschlichen Gedächtnis, erfolgen müsste.

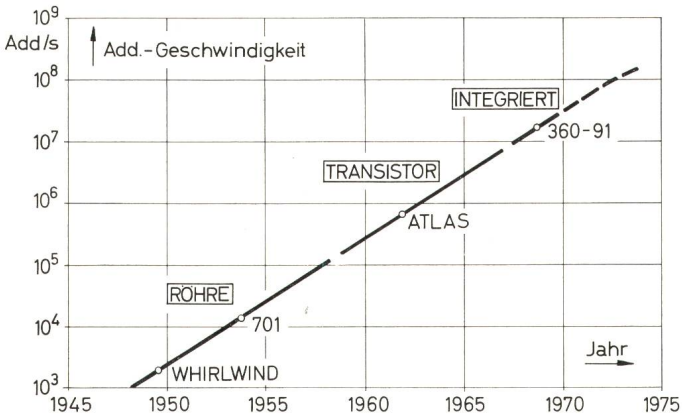


Fig. 9
Entwicklung der Additionsgeschwindigkeit

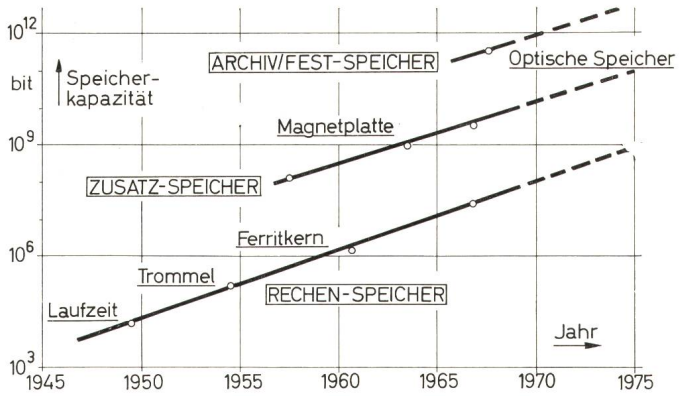


Fig. 10
Entwicklung der Speicherkapazität

Den Zusammenhang zwischen Speicherkapazität und Zugriffszeit zeigt Fig. 11. Für die technischen Speicher finden wir eine Kurve, bei der mit wachsender Speicherkapazität die Zugriffszeit ebenfalls ansteigt. Die Speicherplatzadressierung mit den z. Zt. üblichen Zugriffsarten — seriell bei Magnetbändern, zyklisch bei Magnettrommeln und wahlfrei bei Ferritkernspeichern — unterscheidet sich grundsätzlich vom assoziativen Zugriff des menschlichen Gedächtnisses, das eine Information nach ihrem Inhalt und nicht nach Adressen aufsucht. Das bewusste Gedächtnis dürfte bei einer Kapazität von etwa $10^7 ... 10^8$ bit beginnen, was man relativ leicht durch Lernversuche ermitteln kann; neurophysiologische Untersuchungen führen dagegen auf sehr viel höhere Werte ($10^{12} ... 10^{15}$ bit).

Den z. Zt. möglichen Stand der verschiedenen Speichertechnologien zeigt Fig. 12. Ausgehend von der Magnettrommel mit Zugriffszeiten von 500...0,5 ms ging die Entwicklung in zwei Richtungen:

a) Massenspeicher mit relativ langer Zugriffszeit. Vertreter sind der Magnetplatten- und die Kassettenspeicher mit einer Speicherkapazität grösser als 10^9 bit bei Zugriffszeiten von 0,01...1 s. Hinzu kommt der Magnetbandspeicher mit seiner relativ langen Zugriffszeit im Bereich von ca. 1 min, die jedoch durch geschickte Organisation der Daten auf dem Magnetband nicht in diesem Ausmass in Erscheinung tritt. In nicht allzu ferner Zukunft sind optische Festspeicher zu erwarten, die neben hoher Speicherkapazität auch sehr kurze Zugriffszeiten bis herunter zu einigen µs aufweisen (in Fig. 12 nicht eingezeichnet).

b) Schneller Rechenspeicher mit möglichst kurzer Zugriffszeit. Typischer Vertreter ist der Ferritkernspeicher mit etwa 1 µs Zugriffszeit, bei einer Speicherkapazität bis etwa 10^8bit.

In naher Zukunft werden aktive Halbleiterspeicher in grösserem Umfang zum Einsatz kommen (in Fig. 12 mit gestrichelter Gebietsumrandung) und mit dem klassischen Ferritkernspeicher konkurrieren. In Form von monolithischen Flip-Flop-Speicherzellen haben sie den Vorteil, dass sie mit den integrierten monolithischen Schaltkreisen kompatibel sind und eine Integration von Schaltkreisen und Speicher ermöglichen. Die Entwicklung ist hier noch in vollem Fluss, und es lassen sich daher über mögliche Speicherkapazitäten nur ungenaue Angaben machen.

4.2 Stand und Entwicklung der Schaltkreistechnik

Digitale Schaltkreise führen die Verarbeitung und Steuerung einer DV-Anlage aus. Sie sollen, wie oben gefordert, eine möglichst kurze Schaltzeit (Zeitspanne für das Umschalten

von 0 nach 1 bzw. umgekehrt) erreichen, und dies bei einem möglichst geringen Leistungsbedarf; Fig. 13 zeigt den Gang der Entwicklung. Der Leistungsbedarf ist deshalb von so entscheidender Bedeutung, weil datenverarbeitende Systeme aus einer sehr grossen Anzahl stromverbrauchender Bauelemente (ca. 10000 Transistoren und mehr) aufgebaut sind und man den Leistungsbedarf der gesamten Anlage schon auch mit Rücksicht auf die Wärmeabfuhr möglichst gering halten will. Das erwünschte Ziel ist ja Datenverarbeitung und nicht Erzeugung von Wärme! Die erste in den USA im Jahre 1946 in Betrieb genommene elektronische Rechenmaschine ENIAC mit ihren über 17000 Elektronenröhren benötigte eine elektrische Leistung von 174 kW. Heutige Großsysteme dagegen mit einem Vielfachen an Verarbeitungsleistung der ENIAC brauchen weniger als ein Drittel.

Die in Fig. 13 schraffierten Gebiete charakterisieren den möglichen Arbeitsbereich; die Jahreszahlen ohne Klammern geben den Beginn, die in Klammern stehenden den ungefähren Höhepunkt der betreffenden Technologie an.

4.2.1 Elektromechanik. Ausgehend von der rein mechanischen Lösung der «Schickardschen Rechenuhr» im Jahr 1623, mit einer Schaltzeit von ca. 1 s bei einer Leistung von ca. 10 W,

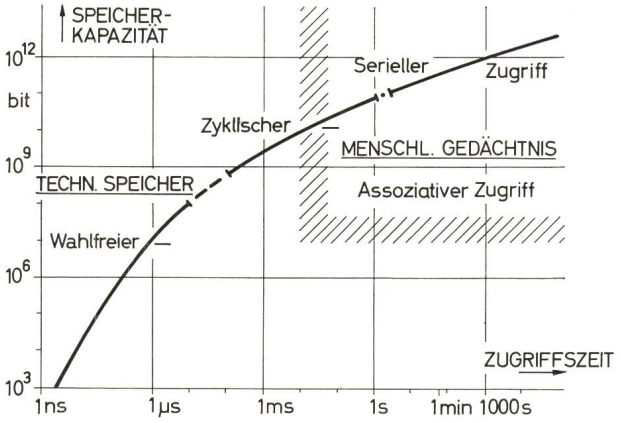


Fig. 11
Speicherkapazität und Zugriffszeit

erfolgte der Übergang zur Elektromechanik erst im Jahre 1890, als *Hollerith* seine Zählmaschine bei der 11. amerikanischen Volkszählung zum Einsatz brachte. Grenzwerte für Schaltzeit und Leistung 1 ms und 10 µW.

4.2.2 Röhre. Eine ganz wesentliche Reduktion der Schaltzeit bis unter 0,1 µs wurde durch die Einführung der Elektronenröhre bei etwa gleichem Leistungsbedarf erzielt. Begrenzung der Geschwindigkeit in erster Linie durch Verdrahtung, bedingt durch die Grösse der Röhre.

4.2.3 Transistor. Mit der Erfindung des Transistors und seiner raschen Einführung in datenverarbeitenden Systemen, etwa ab 1955, konnte der Leistungsbedarf ganz erheblich bis herunter zu etwa 1 mW reduziert werden. Auch die Schaltzeit wurde weiter bis in den Bereich um 1 ns verbessert, was einem Lichtweg von nur 30 cm entspricht. Die Kleinheit des Transistors und seine hohe Lebensdauer waren weiter Gründe für seinen erfolgreichen Einzug nicht nur in der Rechenmaschinentechnik, sondern auch in fast allen Bereichen der elektronischen Gerätetechnik.

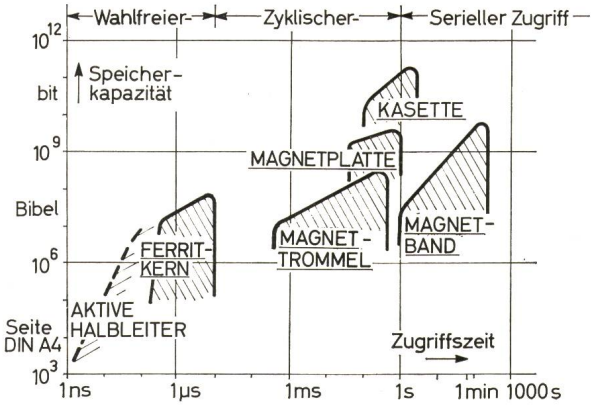


Fig. 12
Speicherkapazität und Zugriffszeit digitaler Speicher

4.2.4 Integrierte Schaltkreise. Eine weitere Reduktion der Schaltzeit des Transistors ist bei den bereits erreichten Werten nur dann sinnvoll, wenn es gelingt, die Signallaufzeiten auf den Verbindungsleitungen entsprechend zu reduzieren. Dies zwingt zu einer weiteren Miniaturisierung, die z. B. in der sog. integrierten monolithischen Halbleitertechnologie (senkrecht schraffiertes Gebiet in Fig. 13) am weitesten fortgeschritten ist. Aktives Transistorelement, Widerstände und die immer problematischer werdenden Verbindungen der Elemente eines Schaltkreises werden alle zusammen in einem Halbleiterplättchen hergestellt. Beispielsweise können z. Zt. auf einer Fläche von 1×1 mm eines Siliziumplättchens etwa 100 Komponenten untergebracht werden.

Grundbausteine der digitalen Schaltkreistechnik bestehen aus etwa 5...15 Komponenten und werden nicht einzeln, sondern in Massenfertigung bis zu mehreren hundert bis tausend Stück auf grösseren Siliziumplatten von 25...30 mm Durchmesser hergestellt, geprüft und danach in die einzelnen Grundbausteine zersägt. Das Zerschneiden und Wiederverbinden von Grundbausteinen zu Baugruppen ist nicht nur kostspielig, sondern birgt weitere Fehlerquellen in sich. Dem eigentlichen Ziel, der Integration ganzer Baugruppen, steht noch die unbefriedigende Ausbeute bei der Herstellung grösserer Baugruppen entgegen. Man versucht daher, sozusagen als Zwischenlösung, die Gut-Prüfung der Schaltkreise auf den grösseren Siliziumplatten mit Hilfe einer Rechenmaschine zu automatisieren und die Verbindungen zwischen den guten Schaltkreisen nach der gewünschten und vorher der Rechenmaschine eingegebenen Funktion nach optimalen Gesichtspunkten zu ermitteln. Dies ist ein Beispiel dafür, wie datenverarbeitende Systeme zur

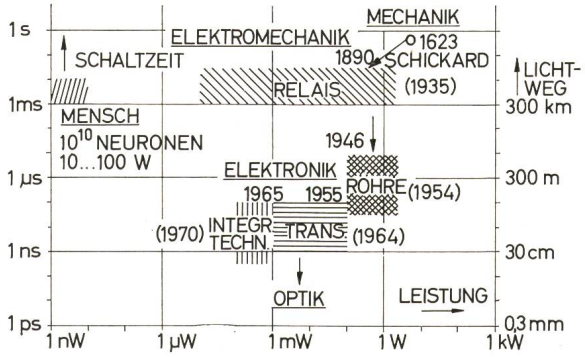


Fig. 13
Schaltzeit und Leistungsbedarf digitaler Schaltkreise

Konstruktion neuer Systeme angewandt werden können nach dem Schlagwort «Computer bauen Computer».

Diese Technologie verspricht eine untere Schaltzeit von weniger als 1 ns bei einem minimalen Leistungsbedarf von etwa 100 µW. Sie eignet sich, wie bereits oben angedeutet, in hervorragender Weise zur Herstellung aktiver Halbleiterspeicher (Fig. 12). Wegen ihrer wesentlich höheren Packungsdichte und ständig abnehmender Kosten werden sie wohl in wenigen Jahren die Ferritkernspeicher ablösen.

4.2.5 Optische Technologie. Schaltzeiten wesentlich unter 1 ns sind vermutlich nur mit optischer Technologie zu erzielen. Es ist zu bedenken, dass in 1 ns das Licht nur einen Weg von 30 cm zurücklegt. Da die Lichtgeschwindigkeit die höchstmögliche Signalgeschwindigkeit darstellt, gelangt man an eine prinzipielle Grenze, die äusserst kurze Signalwege fordert. Opto-elektrische Schaltelemente und die als Lichtverstärker geeigneten Laser sind mögliche Bauelemente.

Über den erforderlichen Leistungsbedarf lassen sich heute noch keine vernünftigen Angaben machen; er liegt sicher erheblich über dem der heutigen Transistortechnologie. Sollte dieser Zustand nicht wesentlich verbessert werden können, so ist wohl kaum mit einem grösseren Einsatz der optischen Technologie in der Verarbeitung und Steuerung, zumindest in kommerziellen Anlagen, zu rechnen.

Vorbild und Ziel sind die biologischen Systeme, insbesondere das menschliche Gehirn, das mit seinen ca. 10 Milliarden Neuronen und einem Leistungsbedarf von ca. 10...100 W in vieler Hinsicht erheblich mehr leistet als die grössten DV-Anlagen.

4.3 Miniaturisierung

Das Bemühen und den Erfolg auf dem Gebiet der Miniaturisierung im Vergleich zu einem biologischen Datenverarbeitungssystem, dem menschlichen Gehirn, zeigt Fig. 14 [4]. Dieses

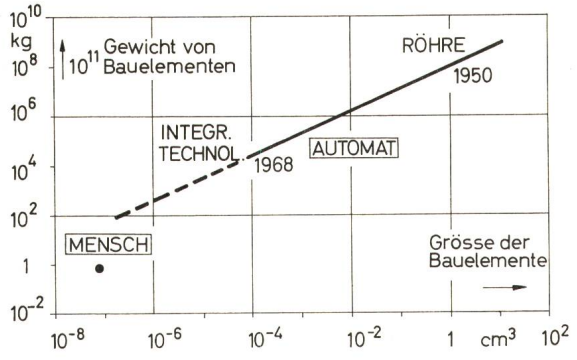


Fig. 14
Grösse und Gewicht von 10^{11} Bauelementen als Äquivalent zum menschlichen Gehirn

wiegt etwa 1 kg, hat ein Volumen von etwa 1 l und besteht aus etwa 10^{10} Neuronen. Die Grösse eines Neurons beträgt somit 10^{-7} cm³.

Man schätzt, dass die logische Verknüpfungs- bzw. Speicherleistung eines Neurons ein technisches Äquivalent von etwa 10 Transistoren erfordert. Wollte man also einen Automaten mit dem Leistungsvermögen des menschlichen Gehirns bauen, wären dazu 10^{11} Transistoren notwendig. Die ausgezogene Kurve in Fig. 14 gibt in Abhängigkeit von der Grösse

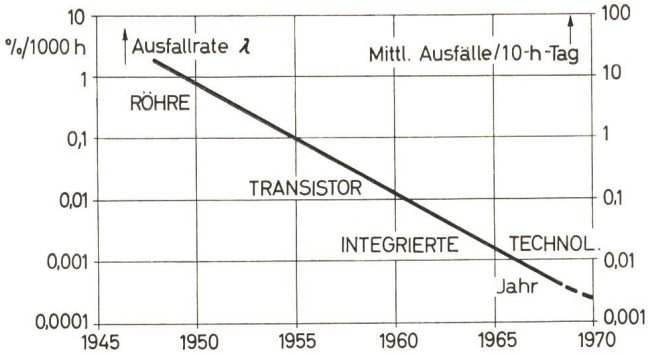


Fig. 15
Ausfallrate λ, mittlere Ausfälle/10-Std.-Tag für 100 000 Bauelemente

der Bauelemente das Gewicht eines solchen Automaten an; mit den z. Z. kleinsten Bauelementen von ca. 0,0001 cm³ würde dieser immer noch über 10 t wiegen.

4.4 Entwicklung der Zuverlässigkeit

Je komplizierter ein technisches Gerät, um so grösser werden die Anforderungen an die Zuverlässigkeit der verwendeten Bauelemente; nur so kann eine zufriedenstellende Zuverlässigkeit und Wartungsfreiheit für den Kunden garantiert werden, wenn man von der Philosophie der Anschaffung eines (Zweit-) Ersatz-Geräts zunächst einmal absieht. Für die hochkomplizierten und mit sehr vielen Bauelementen aufgebauten elektronischen Rechenautomaten treffen diese Überlegungen in ganz besonderem Masse zu.

Die Ausfallrate λ eines Bauelements ist ein Mass für die Zuverlässigkeit und wird angegeben in % je 1000 Betriebsstunden; z. B. besagt $\lambda = 1$ %/1000 h, dass ein Gerät mit 100 gleichen Bauelementen im Mittel alle 1000 Betriebsstunden 1 Bauelement-Ausfall aufweist. Fig. 15 zeigt den grossen Fortschritt der Ausfallrate und die Anzahl der mittleren Ausfälle/10-h-Tag für ein Gerät mit 100000 gleichartigen Bauelementen (rechte Ordinatenskala).

Eine Verbesserung der Ausfallrate der Bauelemente allein genügt jedoch nicht mehr, um die hohen Zuverlässigkeitsansprüche an grosse DV-Anlagen zu erfüllen. Es wird daher schon heute vom Prinzip der Redundanz ausgiebig Gebrauch gemacht. Die zukünftige Entwicklung auf diesem Gebiet wird durch ein gründliches Studium biologischer Systeme, die unseren technischen Gebilden um einiges voraus sind, möglicherweise profitieren können.

Literatur

[1] *K. Ganzhorn* und *W. Walter:* Die geschichtliche Entwicklung der Datenverarbeitung. Jahrbuch des elektrischen Fernmeldewesens 17(1966), S. 9...68.

[2a] *A. P. Speiser:* Digitale Rechenanlagen. Grundlagen, Schaltungstechnik, Arbeitsweise, Betriebssicherheit. 2. Auflage. Berlin/Göttingen/Heidelberg, Springer, 1965.

[2b] *K. Steinbuch:* Taschenbuch der Nachrichtenverarbeitung. 2. Auflage. Berlin/Göttingen/Heidelberg, Springer, 1967.

[2c] *P. Rechenberg:* Grundzüge digitaler Rechenautomaten. München/Wien, Oldenbourg, 1964.

[3] *T. Einsele:* Entwicklungstendenzen der Datenverarbeitung. Zeitschrift für Vermessungswesen 93(1968), S. 507...515.

[4] *K. Steinbuch:* Automat und Mensch, 3. Auflage. Berlin/Göttingen/Heidelberg, Springer, 1965.

Adresse des Autors:
Prof. Dr.-Ing. *T. Einsele,* Taxisstrasse 23, D-8000 München 19.

Weitere Vorträge dieser Vortragsreihe folgen