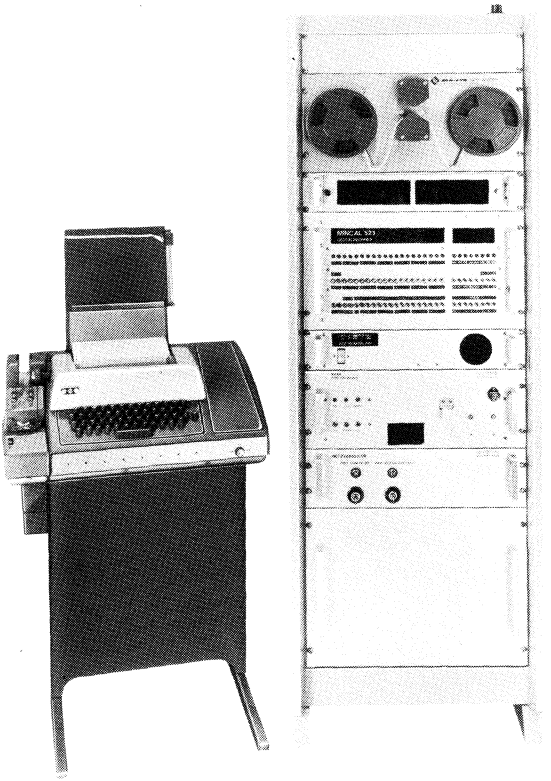
mincal 513/523

Handbuch

DIETZ Computer SYSTEME



mincal 513/523

DIETZ Computer SYSTEME

Heinrich Dietz
Industrie-Elektronik
433 Mülheim-Ruhr
Kölner Straße 115
Tel. (0 21 33) 48 85 41
Telex 08 567 70

HANDBUCH 4/72

Dietz bietet immer etwas mehr

Ausgabe: April 1972

Herausgeber: Heinrich Dietz
Industrie-Elektronik
433 Mülheim a d Ruhr 13, Kölner Straße 115
Telefon (02133) 48 85 41 . Telex 0856770
W-Germany

Druck: Hoppe + Werry KG, Mülheim a d Ruhr

DIETZ in Kurzform

Die Firma HEINRICH DIETZ INDUSTRIE-ELEKTRONIK besteht seit 1951. Das Programm war und ist die industrielle Automation mit elektronischen Mitteln. Diese Mittel sind heute Computer.

Der Weg führte von elektronisch geregelten Getrieben über Kompensationsmeßgeräte und Analogrechner zur Digitaltechnik, über den DIGIVERTER (den ersten deutschen Digitalumsetzer) zu den ZDE-Anlagen und mit dem Aufkommen der Halbleiter als industrielle Baukomponenten zum COMBIDAT-System.

1965 wird das COMBIDAT-System durch die ersten technischen Kleincomputer aus Deutschland, die MINCAL-Digitalrechner, erweitert. In dieser Zeit entsteht eine Rechner-Familie von festprogrammierten Kleincomputern mit der Bezeichnung MINCAL 0, MINCAL E, MINCAL Q und MINCAL 1 und einem speicherprogrammierten Computer, dem MINCAL 3.

Auf die Computer der ersten Generation folgt 1968 der MINCAL 4, der erste in Deutschland entwickelte Prozeßrechner in integrierter Technik. Mit dem MINCAL 4 wurde die Multiprogramming-Struktur und die 19-bit-Wortlänge eingeführt.

Diese Struktur findet sich auch bei den Computern MINCAL 513 und MINCAL 523, erweitert um neue Eigenschaften, wie parallele Verarbeitung, Mikroprogrammierbarkeit und modularen Aufbau.

Heute entwickeln und fertigen in Mülheim 200 Mitarbeiter nicht nur Computer, sondern auch Computer-Peripherie und Standard-Software. Außerdem liefert DIETZ schlüsselfertige Computer-Anlagen einschließlich Planung, Systemanalyse, Ausarbeitung der Anwenderprogramme und der Prozeßperipherie.

DIETZ COMPUTER SYSTEME ist ein Begriff geworden für ein eigenständiges Entwicklungskonzept. Die Prozeßrechner der MINCAL 500-Baureihe sind ein Teil dieser Gesamtkonzeption.

Über die MINCAL 500-Computer

Die Computer MINCAL 513 und MINCAL 523 sind für den Einsatz als Prozeßrechner - in sehr kleinen bis zu relativ großen Systemen - und für die Lösung technisch-wissenschaftlicher Rechenprobleme konzipiert. In Struktur, Peripherie und Software weisen sie zum Teil neuartige Merkmale auf, die sie für diese Aufgaben besonders geeignet machen.

Wortlänge 19 bit: 3 bit mehr (als die für Prozeßrechner wenigstens erforderlichen 16 bit) bedeuten umfangreicheren Befehlsvorrat, bessere und vielfältigere Adressierung, Zugriff zu größeren Speicherbereichen, ausreichende Genauigkeit auch bei extremen Anforderungen, keine Probleme bei Speicher- und Peripher-Erweiterungen.

Großer Befehlsvorrat: 10 speicherbezogene Befehle (einschließlich Multiplikation und Division); 2 Konstantenbefehle; 6 Registerbefehle; 7 Manipulationsbefehle für 512 Speicheradressen; 14 Schiebebefehle; 4 Konversionsbefehle; 6 Verzweigungs- und Aufrufbefehle; 12 Ein/Ausgabebefehle und 13 weitere Instruktionstypen -, das sind 74 verschiedene Maschinenbefehle, die zum großen Teil vielfältig modifizierbar und sehr leistungsfähig sind.

Flexible Adressierung bei speicherbezogenen Befehlen: Absolut sind 0.5 k, relativ sind 1 k Worte zugänglich. Durch indirekte Adressierung können 32 k Kernspeicher-Worte und der gesamte Festspeicher erreicht werden. Hinzu kommen 3 Indexregister je Ebene. Durch Ebenen-Bindung von Speicheradressen kann sich der Benutzer weitere Register aufbauen.

Mikroprogrammierung: Für den Aufbau von Sonderbefehlen und besonders schnell ablaufenden Routinen kann der Benutzer auf die Mikrobefehle zurückgreifen.

Multiprogramming-Struktur: Der MINCAL 523 bietet dem Benutzer bis zu 64 unabhängige Unterrechner mit eigenem Instruktionszähler, eigenen Akkumulatoren, Indexregistern, Datenspeichern und eigener Peripherie. Diese Unterrechner benutzen abwechselnd die eigentliche Recheneinheit, gesteuert von ihren Prioritäten. Bei entsprechender Priorität kann am Ende jedes Befehls ein anderer Unterrechner oder, besser ausgedrückt, eine andere Programmebene die Recheneinheit benutzen. Dabei müssen weder von der Hardware noch durch ein Organisationsprogramm Speicher- oder Register-Inhalte gerettet werden. Jede Ebene verfügt über 8 eigene Register. Multiprogramming ist extrem einfach, denn für jede Aufgabe gibt es eine völlig unabhängige Ebene.

3 Datenkanäle: Für programm- und fremdgesteuerten Datentransfer sowie für die Kernspeichererweiterung sind 3 getrennte Datenkanäle vorgesehen, - ein bedeutender Faktor für die Arbeitsgeschwindigkeit und den Umfang der anschließbaren Peripherie.

Festspeicher: Mikroprogramme und Programmierhilfen sind unzerstörbar in speziellen Festspeichern enthalten. Sie sind voll kompatibel mit Kernspeichern und können daher wichtige Unterprogramm-Pakete und auch Anwenderprogramme enthalten. Die Festspeicher haben 750 ns Zykluszeit.

Bedienungskomfort: Alle Register und Speicherplätze sind über die Frontplatte zugänglich, alle wichtigen Zustände sichtbar und beeinflußbar. Adreßstop und andere Funktionen erleichtern den Testbetrieb. Eine als Festspeicher eingebaute Programmierhilfe erlaubt sofortiges Einlesen sowie bequemes Ändern und Ausgeben von Programmen.

Modularer Aufbau: Zentraleinheit und Peripherie sind modular konzipiert, so daß für jede Aufgabenstellung eine optimale Konfiguration gewählt werden kann.

Externspeicher: Schnelle Trommel- und Plattenspeicher erhöhen - einschließlich der zugehörigen Betriebssysteme - die Leistungsfähigkeit von MINCAL 523-Anlagen. Außerdem sind Magnetband-Systeme verfügbar.

Peripherie: Eine vielseitige Bedienungsperipherie - Fernschreiber, Drucker, Streifenleser und -locher, Datensichtgeräte, Kartenleser, Plotter - ist verfügbar. Hinzu kommen Interfaces für Datenfernübertragung und ein umfangreicher Katalog von Prozeßperipherie: Ein/Ausgänge für digitale und analoge Signale sowie spezielle Analog-Meßsysteme. Echtzeituhren. Speicherschutz. Schnelle Speicher-Zählkanäle.

Basis-Software: ASSEMBLER/EDITOR. Bibliothek mit Doppelwort-Paket, Einwort- und Doppelwort-Gleitkomma-Paket sowie mathematischen Funktionen. MONITOR-Testhilfe. CALCULATOR-Tischrechnerprogramm. Graphische Programme. FORTRAN- und ALGOL-Compiler. BASIC-Interpreter. Betriebssystem für das Arbeiten mit Formalsprachen.

Prozeßsysteme mit Rechenplatz: Als Folge der Multiprogramming-Struktur ist es möglich, auf einem Prozeßsystem gleichzeitig technisch-wissenschaftliche Datenverarbeitung zu betreiben, ohne den on-line-Betrieb zu stören und ohne aufwendige Betriebssysteme.

Computer-Fibel

WIE ES HINTER DEN KULISSEN AUSSIEHT

Jeder Computer, der etwas auf sich hält, besitzt einen SPEICHER. Das ist ein Regal mit vielen Fächern, SPEICHERPLÄTZE genannt, und da Computern die Ordnung im eigenen Hause über alles geht, gibt es für jeden Platz eine Nummer, die ADRESSE, und zwar immer hübsch in aufsteigender Reihenfolge.

In jedem Platz des Speichers ist etwas enthalten, was wir nicht umhin können, als seinen INHALT zu bezeichnen. Alle Plätze sind gleich groß, und was da hineinpaßt, nennen wir ein WORT. Und wie sich die Worte unserer Sprache aus Buchstaben zusammensetzen, so bestehen Computer-Worte aus binären Elementen, BITs; binär deshalb, weil sie nur die beiden Formen 0 oder 1 annehmen können (dieses Schwarz-Weiß-Denken haben alle Computer an sich, bemühen sich aber, es durch differenzierten Umgang mit den Bits wettzumachen.)

Die Anzahl der Bits in einem Wort nennt man die WORTLÄNGE; je größer diese und die Zahl der Speicherplätze (KAPAZITÄT) ist, desto höher der Preis des Computers, seine Leistungsfähigkeit und der Stolz seines Besitzers.

Nehmen wir an, unsere Worte seien 3 Bit lang; dann gibt es, wie man sich leicht überzeugt, folgende Kombinationen von 0 und 1:

000
001
010
011
100
101
110
111

Das sind 8 verschiedene Worte; allgemein gilt die Regel, daß ein n-bit-Wort 2^n verschiedene Inhalte haben kann (in unserem Beispiel $2^3 = 8$).

Hat ein Computer-Wort die Länge von 18 bit (wie die MINCAL 500-Computer, worauf wir wiederum besonders stolz sind), so ergeben sich $2^{18} = 262144$ verschiedene mögliche Worte. Schreibt man allerdings so ein langes BINÄRES Wort hin:

001010000100111011

so verliert man vor lauter Nullen und Einsen schnell die Übersicht, und man kann sich so ein Wort weder merken noch einem anderen schnell zurufen, der sich möglicherweise auch dafür interessiert.

Deshalb führen wir schnell einen Trick ein, nämlich den mit der OKTALEN Schreibweise. Zuerst ordnen wir die 18 Bit in 6 Dreierpäckchen:

001 010 000 100 111 011

(zum Glück ist 18 durch 3 teilbar!) und ersetzen jedes Päckchen, je nach seinem Inhalt, durch eine der Ziffern 0 bis 7, wobei folgende Zuordnung gilt:

000 = 0
001 = 1
010 = 2
011 = 3
100 = 4
101 = 5
110 = 6
111 = 7

(Der geschulte Betrachter erkennt, daß es sich dabei um die BINÄRZAHLEN von 0 bis 7 handelt; die zugehörige OKTALZIFFER ergibt sich, wenn man den Binärstellen - den Bits - von links nach rechts die Wertigkeit 4, 2 und 1 gibt und sie zusammenzählt. Bitte selbst nachrechnen!).

So wird also aus unserem binären Wort

$$\underbrace{001}_{1}\ \underbrace{010}_{2}\ \underbrace{000}_{0}\ \underbrace{100}_{4}\ \underbrace{111}_{7}\ \underbrace{011}_{3}$$

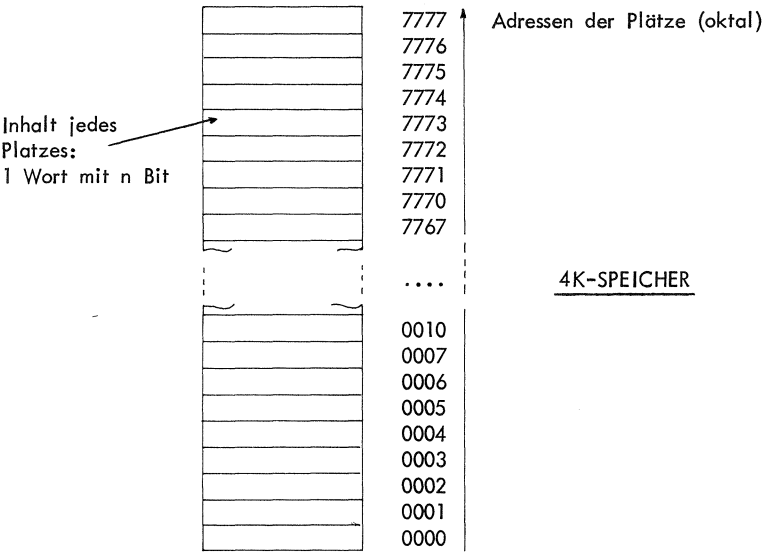
die OKTALZAHL 120473, die man gut aussprechen (je nach Geschmack zwölf null-vier dreiundsiebzig oder eins-zwanzig vier-dreiundsiebzig oder eins-zwei-null-vier-sieben-drei) und sich notfalls auch noch merken kann. Wenn Gefahr besteht, sie mit unseren üblichen Dezimalzahlen zu verwechseln, geben wir Oktalzahlen eine 8 als Anhänger mit: 120473_8; dann weiß jeder Bescheid.

Kehren wir zum Speicher und seinen Adressen zurück. Computer sind ein bißchen einseitig, aber darin konsequent: Gehen sie schon nur mit binären Informationen um, so bezeichnen sie ihre Speicherplätze auch mit Binärzahlen. Weil wir die nicht so sehr mögen, wenden wir auch hier unsere List mit den Oktalzahlen an.

Nehmen wir an, der Speicher habe eine Kapazität von 4096 Worten (eine bei Computern übliche Größe; $4096 = 2^{12}$ = "4K", wobei $1K = 1024 = 2^{10}$ ist). Dann sind alle Adressen mit 12-stelligen Binärzahlen - also 12 Bit - darstellbar, die wir gleich in 4-stellige Oktalzahlen umschreiben:

1. Adresse:	$000\ 000\ 000\ 000 = 0000_8$
2. " :	$000\ 000\ 000\ 001 = 0001_8$
...	
usw.	
...	
vorletzte Adresse:	$111\ 111\ 111\ 110 = 7776_8$
letzte Adresse:	$111\ 111\ 111\ 111 = 7777_8$

Stellen wir uns diesen Speicher vor:



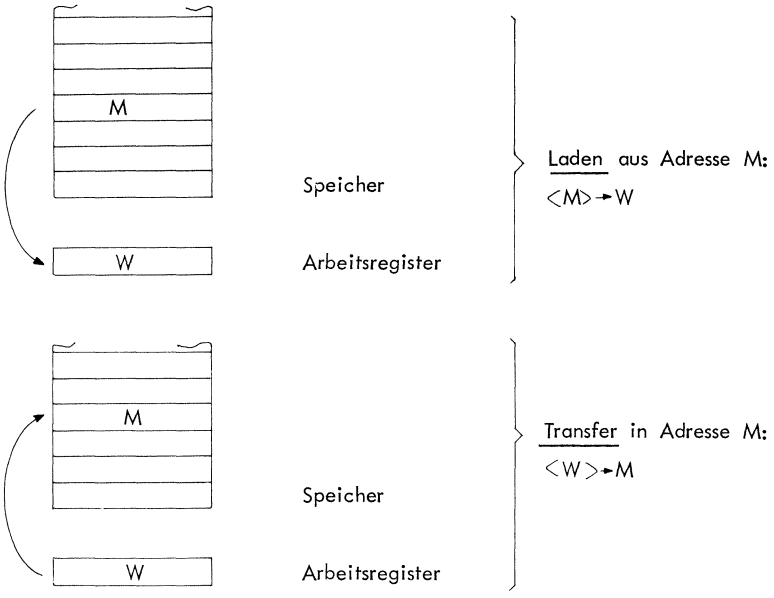
Womit, wie es sich für Computer-Leute gehört, das Thema Speicher schön geordnet und eingeteilt wäre.

WIE ES IM SPEICHER LEBENDIG WIRD

Ein Speicher macht noch keinen Computer, und kein Regal, dessen Fächer nicht von Zeit zu Zeit einen neuen Inhalt bekommen.

Jeder Computer besitzt (mindestens) ein außerhalb des Speichers liegendes Fach, in das gerade ein Wort hineinpaßt und welches man mit dem Inhalt eines Speicherplatzes laden kann. Oder dessen Inhalt man in jede Speicheradresse übertragen kann. Wir nennen es das ARBEITSREGISTER (oder W-Register oder im üblichen Sprachgebrauch auch Akkumulator).

Das geht so vor sich:



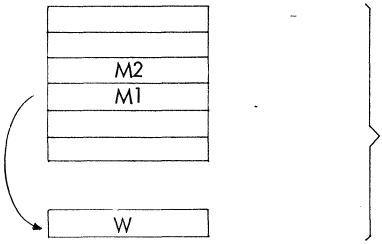
W ist das Symbol für das W-Register, M das für einen beliebigen Speicherplatz, und <...> bedeutet "Inhalt von ...".

Bemerkenswert an diesen Transportvorgängen ist, daß der frühere Inhalt von W beziehungsweise M keine Rolle spielt, und daß im ersten Falle M, im anderen W seinen Inhalt beibehält (womit unser Bild vom Lagerregal heftig zu hinken anfängt und wir, mit freundlicher Genehmigung des Lesers, es endgültig über Bord werfen).

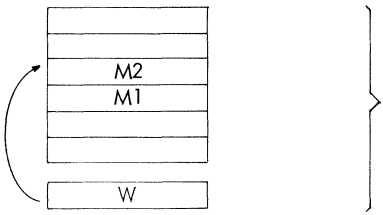
Mit LADEN und TRANSFER haben wir gleich zwei fundamentale Operationen eines Computers kennengelernt. Eine OPERATION ist ein abgeschlossener Vorgang einfacher Art, und viele aufeinanderfolgende Operationen machen einen PROGRAMMABLAUF.

Noch zwei Eigenschaften machen unsere Operationen deutlich: Der Inhalt jedes Speicherplatzes kann verändert werden, indem man den Inhalt des W-Registers dorthin überträgt. Und dieses spielt die Rolle einer Daten-Drehscheibe, was auch an der folgenden Aufgabe klar wird.

Sie lautet: Übertrage den Inhalt der Speicheradresse M1 in den Platz M2. Es geht nur über das W-Register, nämlich so:



1. Operation:
Laden aus Adresse M1
<M1>→W



2. Operation:
Transfer in Adresse M2
<W>→M2

Wir können das auch in SYMBOLISCHER Form notieren:

```
LD      M1
TR      M2
```

womit wir schon unser erstes (Mini-)Programm geschrieben hätten, und zwar in symbolischer Schreibweise, weil die BEFEHLE (Laden, Transfer) durch Abkürzungen (LD, TR) und die Adressen durch MARKEN (M1, M2) - anstelle der echten oktalen Nummern - angegeben sind.

Unser Mini-Programm besteht aus zwei INSTRUKTIONEN (oder Anweisungen an den Computer, dies oder jenes zu tun), und wie wir sehen, beantwortet eine solche Instruktionen dem Computer Fragen:

Was soll ich tun? (Es antwortet der Befehl.)
Womit, woher, wohin? (Es antwortet die Adresse).

Damit ist klar, daß zur Ausführung einer bestimmten Operation eine dementsprechende Instruktion gehört. Und die Gesamtheit aller Instruktionen, die einen von uns gewünschten Ablauf vorschreiben, - eben diese nennt man ein PROGRAMM.

Womit fast nichts mehr im Wege stünde, lustig mit dem Programmieren zu beginnen.

NOCH EIN STÜCK COMPUTER-INTIMSPHÄRE

Zuvor wollen wir noch einen Blick ins Innenleben unseres geliebten Spielzeugs werfen, diskret zwar, aber doch so eindringlich, daß wir herausfinden, wo es eigentlich die Programminstruktionen aufhebt und wie es sie versteht.

Die Antwort auf die erste Frage ist ganz einfach: Im Speicher natürlich. Dort stehen, säuberlich in der Reihenfolge steigender Adressen angeordnet, alle Instruktionen. Und jede hat die Länge eines Wortes. Denken wir an unser Mini-Programm und stellen wir uns vor, die erste Anweisung stehe im Platz 0200, dann hat die zweite die folgende Adresse:

```
0200    LD  M1
0201    TR  M2
```

Nun muß es einen dienstbaren Geist geben, der aufpaßt, daß die jeweils benötigte Instruktion auch aus dem richtigen Kästchen geholt wird und vor Holen der nächsten die Wahl auf die folgende Adresse fällt. Das tut für uns der INSTRUKTIONSZÄHLER (oder Programmstand oder das N-Register, wie wir's nennen): Er gibt die Speicheradresse an, in der die Instruktion für die Operation enthalten ist, die daraufhin ablaufen soll.

Der Mechanismus geht so:

```
Schritt 1:  <N> + 1→N          N-Register um 1 erhöhen
Schritt 2:  ≪N≫  →INSTR        Instruktion holen
Schritt 3:  .................  Ausführen der Instruktion
```

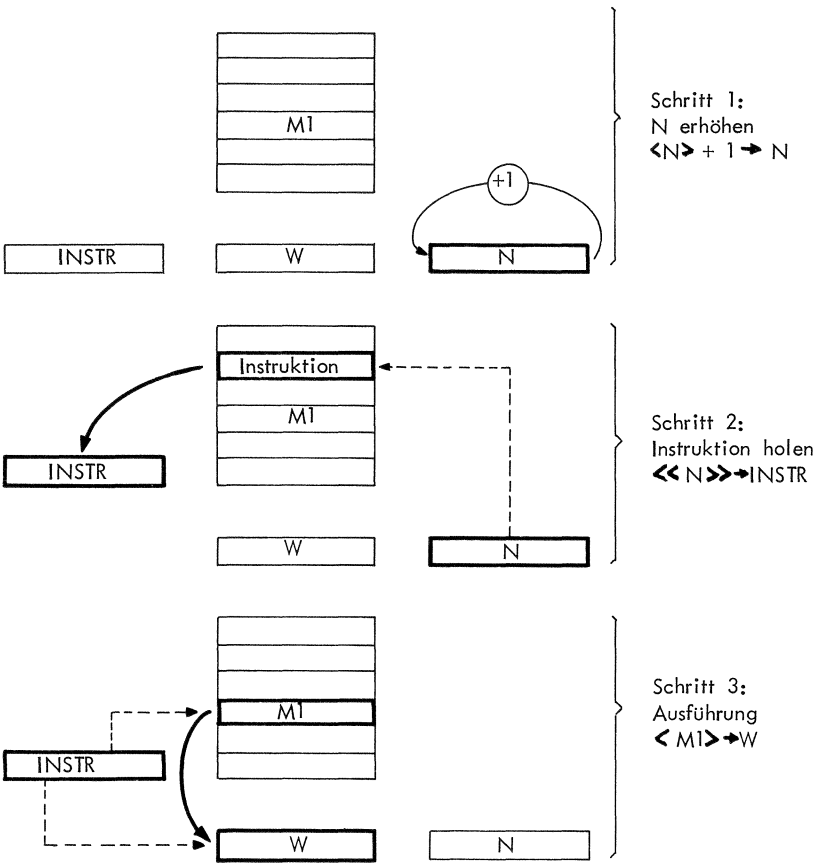
Dabei bedeutet ≪N≫ den "Inhalt vom Inhalt von N", oder, in Nicht-Computer-Deutsch, den Inhalt des Speicherplatzes, dessen Adresse im N-Register steht. Und INSTR ist für uns einstweilen ein weiterer dienstbarer Geist, der die Instruktion aufhebt und den Ablauf der Operation - die Ausführung - steuert.

Wohlgemerkt: Die Schritte 1 und 2 laufen ab, ohne daß Sie dies dem Computer ausdrücklich sagen - ein eingebauter Reflex sozusagen. Erst auf Schritt 3 haben Sie durch Programmieren der geeigneten Instruktion Einfluß.

Zeichnen wir den vollen Ablauf der Instruktion

LD M1

noch einmal auf:

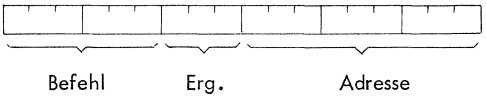


Wir sehen, daß erst im dritten Schritt das für die Instruktion Spezifische passiert. Was da vor sich gehen soll, sagt das Wort aus, das in INSTR steht, und jedes Bit darin (oder jede Oktalzahl, wie wir sehen werden) hat eine bestimmte Funktion.

Zum Beispiel verstehen die MINCAL-Rechner ihre 18 bit-Instruktionen so:

1. Oktalstelle 2. Oktalstelle	enthalten den Befehl OP
3. Oktalstelle	enthält die Ergänzung Q
4. Oktalstelle 5. Oktalstelle 6. Oktalstelle	enthalten die Adresse AAA

Oder aufgezeichnet:



(OPQAAA)

Was Befehl und Adresse ist, wissen wir schon; die Geheimnisse der Ergänzung heben wir uns für später auf.

Nehmen wir die Oktalzahl 120437 als Instruktion:

O	P	Q	A	A	A
1	2	0	4	3	7

und setzen wir voraus, daß 12 der BEFEHLSCODE für "Laden" sei, so bedeutet sie: "Lade den Inhalt von Adresse 437". Und hätte der Platz M1 unseres Mini-Programms gerade diese Adresse, so wäre diese identisch mit

```
LD    M1
```

- aber eben nicht in einer symbolischen SPRACHE, sondern in der, die der Computer einzig und allein versteht: in MASCHINENSPRACHE (oder auch Maschinencode genannt).

Wenn wir noch wissen, daß 16 der Befehlscode für "Transfer" ist, und annehmen, 440 sei die Adresse M2, dann kann unser Mini-Programm sogar in zwei Zungen sprechen:

	symbolisch		Maschinencode
0200	LD	M1	120437
0201	TR	M2	160440

(Zwischenfrage: Wieviele Plätze liegen zwischen M1 und M2?)

Damit genug der Computer-Anatomie.

WIE EIN RECHNER AUCH RECHNEN KANN

Füttern wir das Raubtier mit einem nützlichen Programm und lassen wir es mal rechnen:

$$z = x + y$$

Damit die Reihenfolge stimmt, schreiben wir die Aufgabe um:

$$x + y \longrightarrow z$$

und denken uns, die Werte x und y stünden in den Plätzen M1 und M2 des Speichers, und die Summe z solle nach M3. Wenn wir noch wissen, daß AD... das Symbol für "Addiere... zum Inhalt des W-Registers" ist, so lautet das Programm:

LD	M1	$x \longrightarrow W$
AD	M2	$\langle W \rangle + y \longrightarrow W$
TR	M3	$z = x + y \longrightarrow M3$

Hätten wir stattdessen y gerne von x abgezogen:

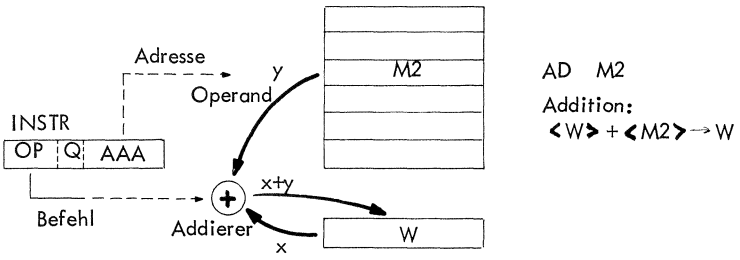
$$x - y \longrightarrow z$$

- mit im übrigen gleichen Plätzen für x, y und z -, so hätte es heißen müssen:

```
LD  M1      x ⟶ W
SB  M2      <W>  - y ⟶ W
TR  M3      z = x - y ⟶ M3
```

wobei SB... bedeutet: "Subtrahiere ... vom Inhalt des W-Registers".

Betrachten wir im Bild, was bei der Instruktion "AD M2" passiert:



AD M2

Addition:
<W> + <M2> → W

Wie man sieht, steuert der Befehl AD die additive VERKNÜPFUNG der Inhalte von W und M2 in einem ADDIERER, wobei die Summe wieder ins W-Register gebracht wird. Übrigens heißt y, als Inhalt der angesprochenen Adresse, der OPERAND.

Vielleicht ist es ganz nützlich, die Addition zweier Binärzahlen - nur solche kann der Rechner zueinander addieren - näher zu betrachten. (Man könnte sie dem Computer selbst überlassen, aber manchmal möchte man bs bestimmt - trau schau wem - auch mal selbst nachvollziehen).

Nehmen wir 3-stellige Binärzahlen und rechnen 3 + 1 = 4:

	2^2	2^1	2^0	
x	0	1	1	(= 3)
+y	0	0	1	(= 1)
	1	1		
=x+y	1	0	0	(= 4)

wobei 1 + 1 = 0 + ÜBERTRAG in die nächsthöhere Stelle!

Noch ein Beispiel für 18 bit-Zahlen, die wir aber gleich als 6-stellige Oktalzahlen schreiben (mit entsprechender Wertigkeit 8^n der Oktalstellen):

	8^5	8^4	8^3	8^2	8^1	8^0	
x =	0	0	1	7	5	0	(= 1000)
y =	0	0	0	0	3	0	(= 24)
			1	1			
x + y =	0	0	2	0	0	0	(= 1024)

wobei z.B. 3 + 5 = 0 + Übertrag 1 in die nächsthöhere Stelle (Ziffern > 7 sind bei Oktalzahlen nicht erlaubt!).

Man erkennt, daß Binär- und Oktalzahlen im Prinzip wie unsere üblichen Dezimalzahlen behandelt werden; jede Stelle kann allerdings die Ziffern 0 oder 1 (bei Binärzahlen) beziehungsweise 0..7 (bei Oktalzahlen) enthalten, im Gegensatz zu den Dezimalzahlen, wo 0...9 erlaubt ist.

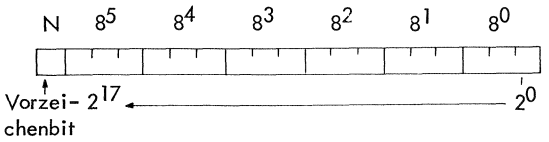
Subtrahieren wir noch schnell zwei Oktalzahlen voneinander, damit wir das auch geübt haben:

	8^5	8^4	8^3	8^2	8^1	8^0	
x	0	0	2	0	0	0	(= 1024)
- y	0	0	0	0	3	0	(= 24)
			1	1			
= x -y	0	0	1	7	5	0	(= 1000)

(hier muß man sich etwas von der höheren Stelle "borgen", wenn die x-Ziffer größer als die y-Ziffer ist, - wie gehabt!).

Alle Zahlen, mit denen wir bisher zu tun hatten, hatten eines gemeinsam: sie waren positiv. Doch ein Computer, der nicht auch negative Zahlen akzeptiert, wäre nicht positiv zu beurteilen. Aber wie unterscheidet er sie?

Da muß nun eingestanden werden, daß wir dem geneigten Leser bisher etwas Kleines, wenn auch Wichtiges unterschlagen haben: das Vorzeichen-Bit, das jedes Computer-Wort hat. Das neunzehnte Bit in jedem MINCAL-Wort:



Je nachdem, ob das N-Bit eine 0 oder 1 enthält, ist die Zahl positiv oder negativ. Und wir vereinbaren, vor jeder negativen Oktalzahl das N-Bit durch ein Minuszeichen darzustellen (das Pluszeichen vor positiven sparen wir uns).

Nun gibt es - leider - zwei Arten, negative Zahlen darzustellen. Die erste: Im N-Bit steht das Vorzeichen, in den übrigen der Betrag der Zahl. Zum Beispiel:

-001750 (= -1000)

So war's früher beim MINCAL 4.

Die MINCAL 500-Computer haben sich bei ihrer Geburt eine andere Darstellung ausgesucht: Negative Zahlen werden als ZWEIERKOMPLEMENT der entsprechenden positiven Zahl ausgedrückt.

Zunächst: Was ist ein Komplement, besser gesagt, ein EINERKOMPLEMENT? Nun, das ist die Zahl, die eine andere zur größten darstellbaren Zahl (in unserem Falle -777777_8) ergänzt. Beispiel:

```
 001750      (= 1000)
-776027      (= Einerkomplement von 1000)
--------
-777777
```

(wir addieren einfach die Zahlen und denken uns das Vorzeichen-Bit als Ziffer).

Wir bilden das Zweierkomplement einer Zahl, indem wir zu ihrem Einerkomplement eine 1 addieren:

```
-776027      (= Einerkomplement von 1000)
+      1     (= 1)
--------
-776030      (= Zweierkomplement von 1000 = -1000)
```

Und das wäre nun unsere negative Zahl.

Rechnen wir ein Beispiel:

```
 002000      (= 1024)
+(-776030)   (= -1000)
--------
 000030      (=  24 )
```

wobei Sie bitte das Vorzeichen wie ein Bit behandeln, einen Übertrag dazu addieren und den im N-Bit entstehenden Übertrag schlicht vergessen sollten.

Viele Computer begnügen sich, was das Rechnen angeht, mit Addition und Subtraktion (einige sogar mit dem Addieren allein, wobei das Subtrahieren durch Hinzuzählen einer Zahl erfolgt, die man vorher in ihr Zweierkomplement verwandelt hat). Will man mehr, so muß man das durch eine Folge von Additionen und Subtraktionen programmieren, denn alle höheren Rechenarten bauen auf diesen auf.

Andere Rechner hingegen, darunter die MINCALs, können noch ein bißchen mehr; sie kennen nämlich Befehle für Multiplikation (MP) und Division (DV).

Schauen wir uns das einmal an. Wenn man zwei 18-bit-Zahlen miteinander multipliziert, dann entsteht daraus eine Zahl (das Produkt), die doppelt so groß sein kann, nämlich 36 bit lang. Weil die in unser W-Register nicht mehr hineinpaßt, erfinden wir noch ein weiteres, das HILFSREGISTER (oder X-Register), in dem die zweite Hälfte des Ergebnisses (die UNBEDEUTENDEN Stellen) Platz hat, während die erste Hälfte (die BEDEUTENDEN Stellen) im W-Register steht.

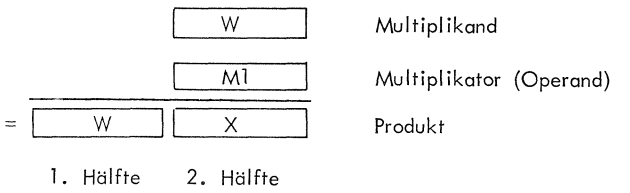
Die Instruktion

MP M1

bedeutet also:

$$<W> \cdot <M1> \rightarrow W, X$$

Oder im Bild:



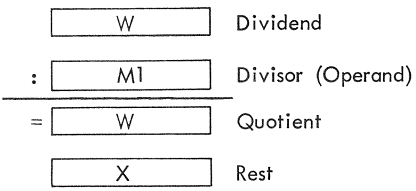
Der Vollständigkeit halber noch kurz eine Division

DV M1

bei der der W-Register-Inhalt durch den Operanden < M1 > geteilt wird; der Quotient steht wieder im W- und der Rest im X-Register:

$$< W > : < M1 > \rightarrow W$$
$$\text{Rest} \rightarrow X$$

Oder so:



Zum Schluß wollen wir ein kurzes arithmetisches Programm schreiben. Die Aufgabe laute:

$$u \cdot (v + w) + x \cdot y \rightarrow z$$

oder, weil die Zahlen in entsprechenden Speicherplätzen untergebracht sind:

$$< M5 > \cdot (< M3 > + < M4 >) + < M1 > \cdot < M2 > \rightarrow M6$$

Wir werden sehen, daß wir die Formel in zwei Abschnitten rechnen und ein Zwischen - ergebnis in einem Speicherplatz aufheben müssen; dazu mißbrauchen wir den Platz M6, wo später das Ergebnis stehen soll. Wir fangen mit dem rechten Term an, nehmen uns dann den linken vor und programmieren:

```
LD      M1
MP      M2
TR      M6      \1. TEILERGEBNIS X*Y
LD      M3
AD      M4
MP      M5      \2. TEILERGEBNIS U*(V+W)
AD      M6
TR      M6      \RESULTAT Z
```

Man verfolge das Programm Schritt für Schritt! Und beachte dabei, daß wir uns bei MP nur für die "bedeutende" Hälfte des Produkts interessieren, die im W-Register steht. Wäre die andere Hälfte interessant gewesen, so hätte nach der Multiplikation der X-Register-Inhalt ins W-Register transferiert werden müssen. Die Instruktion hierfür heißt TRRX W, - womit wir auch gleich einen typischen REGISTERBEFEHL kennengelernt hätten.

Rechts stehen Anmerkungen - KOMMENTARE -, die uns die Übersicht erleichtern sollen; ein guter Programmierer kommentiert sein Programm, damit andere es verstehen (und er selbst sich später darin zurecht findet).

Womit in diesem Kapitel bewiesen wäre, daß Rechner auch rechnen können.

WIE COMPUTER ZEICHEN SCHLUCKEN, ZERSCHNEIDEN, VERSCHIEBEN, ZUSAMMENSETZEN UND WIEDER AUSSPUCKEN

Bis jetzt war unser Computer nur mit sich und seinem Innenleben beschäftigt. Wir wollen ihm nun die Augen öffnen, damit er sieht, was um ihn herum vorgeht. Irgendwie muß er ja die Daten hereinbekommen, mit denen er umgeht, und irgendwann einmal soll er seine Ergebnisse ausspucken.

Die Umgebung des Rechners, mit der er Informationen austauscht, die PERIPHERIE, kann man als eine Art Speicher auffassen, jedenfalls vom Standpunkt des Programmierers aus. Jedes an den Computer angeschlossene Gerät, mag es nun eine Schreibmaschine, ein Lochstreifenleser oder -stanzer, eine Meßstelle, eine Anzeigeeinheit oder sonst etwas sein, bekommt eine EXTERNE ADRESSE (oder GERÄTE-ADRESSE) zugeteilt, und Informationen (DATEN) werden in Form von Worten ausgetauscht, - ganz wie beim Speicher. Und wieder ist das W-Register der Tauschpartner.

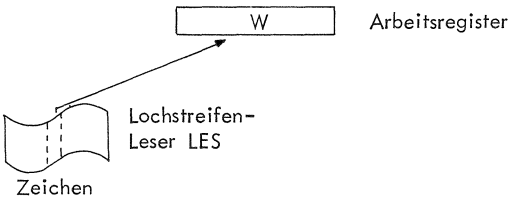
Nehmen wir als Beispiel einen Lochstreifenleser als externes Eingabegerät. Seine Geräteadresse sei LES (hinter dieser symbolischen Bezeichnung verbirgt sich natürlich wieder eine 3-stellige oktale Adresse), und der EINGABE-Befehl laute IBG; dann bewirkt die Instruktion

IBG LES

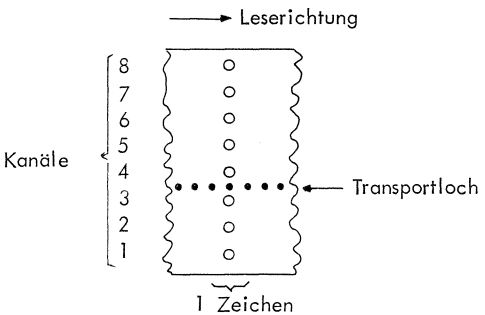
das Lesen eines ZEICHENS aus dem Lochstreifen ins W-Register:

Zeichen → W

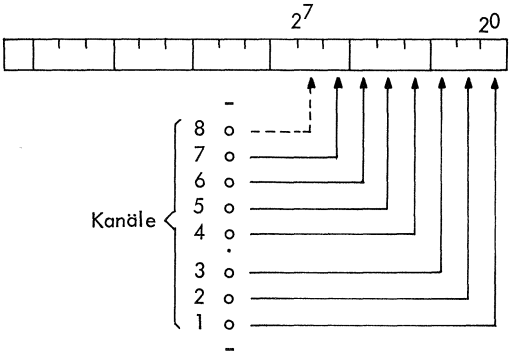
Zeichnen wir es auf:



Wie sieht nun ein solches Zeichen aus? MINCAL-Computer mögen 8-Kanal-Lochstreifen besonders gern; sie besitzen - außer dem Transportloch - acht Lochreihen:



Ein Zeichen besteht aus 8 Löchern (oder Nicht-Löchern) quer zur Transportrichtung, die mit einem Mal gelesen und in die rechten 8 Bit des W-Registers übertragen werden (wobei Loch = 1 und Nicht-Loch = 0 bedeutet):



Genau genommen, interessiert uns Kanal 8 nicht -, wie wir gleich sehen werden. Mit LES sei gemeint, daß nur die Kanäle 1 bis 7 übertragen werden; das Zeichen wird MASKIERT. Die zu LES entsprechende Adresse enthält also eine FORMAT-Angabe (7-bit-Format in unserem Falle).

Was können nun die 8-Kanal-Zeichen bedeuten? Darüber ist eine Vereinbarung zu treffen, ein CODE. Nehmen wir einen gebräuchlichen, den ASCII- (oder ISO-7-)-Code; jeder Buchstabe und jede Ziffer haben ein bestimmtes Code-Zeichen, zum Beispiel:

Kanal: 7654321 Bit : $2^6 \leftarrow 2^0$	oktale Darstellg.	ASCII-Code- Bedeutung
0110000	060	Ziffer 0
* 0110001	061	" 1
* 0110010	062	" 2
0110011	063	" 3
* 0110100	064	" 4
0110101	065	" 5
0110110	066	" 6
* 0110111	067	" 7
* 0111000	070	" 8
0111001	071	" 9
1000001	101	Buchstabe A
1000010	102	" B
	...	usw.

Und Kanal 8? Der ist eigentlich überflüssig - und doch sehr wichtig: Im Streifen ist er immer dann gelocht, wenn die Summe der übrigen Löcher im Zeichen ungerade ist. Man kann dann prüfen, ob die Gesamtsumme immer gerade ist (0, 2, 4, 6 oder 8), auf daß kein Loch zu viel oder zu wenig gelesen werde. Kanal 8 ist also ein Prüf- oder PARITY-Bit. Wo es auftaucht, haben wir oben einen Stern hingeschrieben.

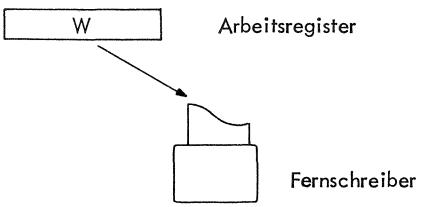
Nun zur AUSGABE, die wir ganz schnell hinter uns bringen. Das Ausgabegerät sei eine mit FSS bezeichnete Fernschreibmaschine; sie arbeite mit unserem 8-Kanal-Code, und der Ausgabebefehl heiße OBH . Dann ist

OBH FSS

die Ausgabe-Instruktion, die

< W > → Fernschreiber

bewirkt:



Wieder werden die rechten 8 (oder 7) Bit des W-Registers herangezogen.

Jetzt hätten wir das leidige (und zugegeben etwas langweilige) Kapitel der Ein/Ausgabe beinahe hinter uns gebracht. Ganz fertig sind wir noch nicht damit, denn so, wie wir die Codes lesen, kann sie unser Computer noch nicht verdauen. Benötigt er doch keine ASCII-Zeichen, sondern bekanntlich Binärzahlen, um damit zu rechnen. Also machen wir uns welche!

Wir stellen uns eine Aufgabe: 2 ASCII-Zeichen, die eine 2-stellige Dezimalzahl enthalten, sollen eingelesen, mit einer Konstanten CON multipliziert und als Produkt 2-stellig ausgeschrieben werden.

Fangen wir an. Das erste Zeichen wird eingelesen:

```
IBG   LES
```

Im W-Register steht ein 7-bit-Zeichen. Davon interessieren uns nur die letzten 4 (denn die stellen die Dezimal-Ziffer in binärer Form dar: BCD-Darstellung genannt: bitte in der ASCII-Tabelle selbst nachsehen!). Also schneiden wir die anderen ab:

```
FA    MA4
```

FA ist ein LOGISCHER Befehl, die sogenannte UND-Operation (oder konjunktive Verknüpfung):

$$<W> \;\&\; <MA4> \rightarrow W$$

die bewirkt, daß nur die Bits eine 1 behalten, die im W-Register und im Operanden schon eine 1 hatten. MA4 ist eine entsprechende MASKE. Wir vollziehen das für die Dezimalziffer 5 einmal nach:

```
          0110101        (= ASCII-Zeichen "5")
000000000000001111       (= Maske MA4)
000000000000000101       (= BCD-Ziffer "5")
```

Dann verschieben wir den Inhalt des W-Registers um 4 Bit nach links und transferieren den Inhalt in einen Zwischenspeicher ZWS:

```
SLLW 4.
TR    ZWS
```

SLLW bedeutet "SCHIEBE links logisch W-Register", in unserem Falle um 4 Bit, mit folgendem Ergebnis:

```
0000000000101 0000
```

(und somit hätten wir schon einen der SHIFT-Befehle kennengelernt).

Wir lesen das zweite Zeichen und maskieren es auf die gleiche Weise:

```
IBG   LES
FA    MA4
```

Und dann fügen wir das erste, in ZWS aufgehobene Zeichen dazu

```
FO    ZWS
```

FO ist auch ein logischer Befehl, ODER genannt (genauer: inklusives Oder, auch disjunktive Verknüpfung genannt). Hier erhält jedes Bit eine 1, in dem im W-Register oder im Operanden (oder in beiden) eine 1 schon stand:

$$< W > \ v \ < ZWS' > \rightarrow W$$

Jetzt stehen unsere beiden Ziffern schön 4-bit-weise (in BCD-Darstellung) nebeneinander in W:

```
000000000000000101 1001
                   5    9
```

(Beispiel: die zweite Ziffer war eine 9).

So weit, so gut. Aber noch immer ist keine Binärzahl daraus geworden! Zum Glück hat unser Computer (weil's ein MINCAL ist) einen Umwandlungsbefehl VBR, der die KONVERSION der Dezimal- in eine Binärzahl durchführt. Wir schreiben also

```
VBR
```

was im W-Register

$$bin < W > \rightarrow W$$

bewirkt, und endlich haben wir unsere Binärzahl.

Wenn wir jetzt noch unsere Multiplikation mit CON durchführen, das Produkt wieder in eine Dezimalzahl zurückverwandeln und als zwei ASCII ausgeben, ist unser Programm fertig. Wir schreiben es einfach hin und überlassen es dem beflissenen Leser, Instruktion für Instruktion zu verfolgen und zu verstehen (sofern er dies auf sich nimmt):

```
LMD   IBG   LES      \ 1. ZEICHEN LESEN
      FA    MA4      \ MASKE
      SLLW  4.       \ 4 MAL LINKS
      TR    ZWS
      IBG   LES      \ 2. ZEICHEN LESEN
      FA    MA4      \ MASKE
      FO    ZWS      \ VEREINIGEN
      VBR            \ BINAERUMWANDLUNG
      MP    CON      \ MULTIPLIKATION
      VDR            \ DEZIMALUMWANDLUNG
      TR    ZWS
      SRLW  4.       \ 4 MAL RECHTS
      FO    ASC      \ ASCII-ZEICHEN
      OBH   FSS      \ 1. ZEICHEN DRUCKEN
      LD
      FA    MA4      \ MASKE
      FO    ASC      \ ASCII-ZEICHEN
      OBH   FSS      \ 2. ZEICHEN DRUCKEN
      ...
ZWS   V
CON   B     .4711
MA4   O     ØØØØ17
ASC   O     ØØØØ6Ø
```

Noch ein paar kleine Hinweise dazu: Unser Programm soll einen Namen haben: LMD (Lesen/Multiplizieren/Drucken), den wir als LINKSMARKE vor die erste Instruktion schreiben. SRLW ist der zu SLLW analoge Befehl für Rechtsverschiebung, und VDR konvertiert eine Binär- in eine Dezimalzahl.

Da wir im Programm einige Zwischenspeicher, Konstanten und Masken benutzen, müssen wir sie auch irgendwo im Programm definieren, jeweils mit einer Linksmarke davor (damit sie das Programm auch findet). Das tun wir mit den Symbolen V (Variablenspeicher, hierfür ein Wort freihalten!), B (Binärzahl, im Beispiel 0.4711) und O (Oktalzahl, im Beispiel die erwähnte Maske MA4 und ein Bit-Muster ASC, das mit der BCD-Ziffer geodert ein ASCII-Zeichen ergibt), - welches schon die wichtigsten DEFINITIONEN unserer symbolischen Programmsprache sind.

Beachte, lieber Leser: In symbolischen Programmen durchstreichen wir die Ø (Null), um sie besser vom O (Oh!) unterscheiden zu können. Man weiß ja: Computer nehmen's genau, und wir sollten da nicht zurückstehen.

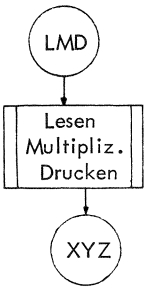
WIE EIN PROGRAMM SPRÜNGE MACHT UND SICH IM KREISE DREHT

Betrachten wir noch einmal das Programm LMD im letzten Kapitel. Es läuft von oben nach unten und führt - wie sich der fleißige Leser inzwischen überzeugt haben wird - unsere Aufgabe richtig durch. Aber dann?

Wenn wir es einfach weiterlaufen lassen, wird es das Wort ZWS als Instruktion auffassen und je nach dessen Inhalt irgendeinen Unsinn machen, was wir als zielbewußte und seriöse Programmierer doch nun wirklich nicht wollen.

Es muß also dort, wo wir in weiser Voraussicht drei Pünktchen Abstand gelassen haben, etwas eingebaut werden, was das Programm veranlaßt, woandershin zu gehen: Eine VERZWEIGUNG.

Nehmen wir an, wir wollten danach zu einem Programmteil XYZ:



dann setzen wir statt der 3 Pünktchen die Instruktion

```
BR   XYZ
```

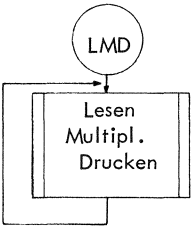
ein, was heißt: Verzweige nach XYZ. Dahinter verbirgt sich natürlich die Adresse des Speicherplatzes, in dem die nächste auszuführende Instruktion steht; XYZ muß im symbolischen Programm als Linksmarke vor der betreffenden Stelle erscheinen.

Verzweigungen werden übrigens dadurch ausgeführt, daß der Instruktionszähler (das N-Register) auf die betreffende Adresse gesetzt wird.

Stattdessen hätten wir auch nach LMD zurückverzweigen können:

```
BR   LMD
```

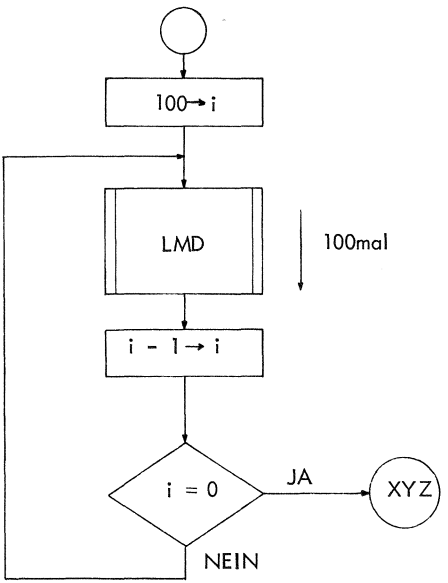
wodurch sich der Vorgang, wie er im vorigen Kapitel beschrieben ist, immer und immer wiederholt. Ein Zeichen-Paar nach dem anderen wird gelesen, verarbeitet und gedruckt:



Aus dieser PROGRAMMSCHLEIFE kommen wir allerdings nie wieder heraus; der Streifenleser liest und liest und liest, der Fernschreiber druckt und - und so weiter.

Erlösen wir das Programm und sagen: Es sollen genau 100 Zeichen-Paare (100 2-stellige Dezimalzahlen) gelesen und verarbeitet und dementsprechend 100 Werte gedruckt werden. Dazu brauchen wir einen Zähler.

Wir entdecken dabei einen neuen Typ von Registern: das INDEXREGISTER. Jeder Computer, der diesen stolzen Namen verdient, hat wenigstens ein solches (die MINCALs haben 3 davon). Wir benutzen es als Zähler, setzen es zunächst auf 100, ziehen nach jedem Durchlauf eins ab und fragen, ob es Null geworden ist:



Wenn nein, geht das Programm nach LMD zurück; wenn ja, und das ist nach dem hundertsten Durchlauf der Fall, nach XYZ.

Das zugehörige Programm sieht so aus:

```
      LDC1  1ØØ.
LMD   ...          }
      ...          }  Programm LMD (siehe voriges Kapitel)
      ...          }
      ADC1-1.
      BZ  1 XYZ
      BR    LMD
```

Jetzt haben wir allerdings einiges zu erklären!

```
LDC1  1ØØ.
```

bedeutet: Lade die Konstante 100 in das Indexregister Nr.1. Und

```
ADC1-1.
```

heißt: Addiere zum Inhalt desselben die Konstante (-1); es wird also eins abgezogen. LDC und ADC sind FESTWERT-Befehle; im Adreßteil steht ausnahmsweise keine Adresse, sondern der Operand selbst.

Die "1" danach gibt die Nummer des Indexregisters an (weil es davon mehrere gibt); sie ist nicht Bestandteil des Befehls, sondern bildet seine ERGÄNZUNG, - womit gleich eine Erklärung für diesen mysteriösen Bestandteil der Instruktion nachgeliefert wäre. Übrigens bezeichnen wir der Einfachheit halber den Inhalt des Indexregisters 1 mit i, als LAUFINDEX.

Fassen wir noch die Abfrage-Instruktion

```
BZ 1 XYZ
```

ins Auge. Sie lautet: Verzweige nach XYZ, wenn Indexregister 1 Null ist, und gehört zur Gruppe der BEDINGTEN VERZWEIGUNGEN. Der Sprung nach XYZ wird nur ausgeführt, wenn der Inhalt des in der Ergänzung erwähnten Registers Null ist; andernfalls wird die nächstfolgende Instruktion ausgeführt.

Schleifen sind des Computers Lieblingsspeise (besonders mit einem Indexregister - wobei wir dessen Nutzen bisher nur zur Hälfte entdeckt haben). Aber deshalb sollten wir ihn nicht einen Wiederkäuer schimpfen. Denn Programmschleifen sind praktisch und platzsparend. Oder würde jemand vorschlagen, in unserem Beispiel 100mal hintereinander das Programm LMD zu schreiben?

Was aber tut man, wenn der Programmteil (die ROUTINE) LMD mal hier, mal da im Programm gebraucht wird? Schleifenbinden ist unmöglich, und die Routine jedesmal neu schreiben ist Platzverschwendung und langweilig dazu.

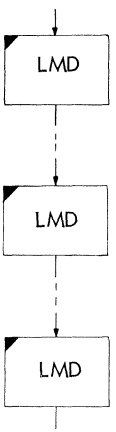
Zum Glück gibt es die Möglichkeit, LMD als UNTERPROGRAMM (oder Subroutine) aufzufassen und mit einem Befehl CS aufzurufen:

```
CS U LMD
```

Dabei geschieht zweierlei: Erstens springt das Programm zur Stelle LMD, und insofern verhält es sich wie bei einem Verzweigungsbefehl. Vorher aber, und das ist das Besondere daran, wird der Programmstand N (um 1 vermindert) als RÜCKKEHRADRESSE in ein Register übertragen, das als Ergänzung angegeben ist; in unserem Beispiel das U-Register, welches wir bei dieser Gelegenheit gleich vorstellen können. Und wenn das Unterprogramm zu Ende ist, programmieren wir den RÜCKSPRUNG ins HAUPTPROGRAMM (das heißt an die Stelle nach dem Aufruf CS...) mit

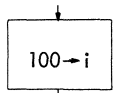
```
TRRU N
```

was heißt: Transferiere den Inhalt des U-Registers (die Rückkehradresse) ins N-Register. Worauf das Spiel an anderer Stelle des Hauptprogramms wiederholt werden kann:

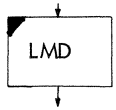


Es ist wohl an der Zeit, ein klärendes Wort über die Kästchen zu sagen, die in diesem Kapitel aufgetaucht sind. Sie sind Bestandteile von Programmablaufplänen (kürzer: BLOCKDIAGRAMMEN) und sollen Programmverläufe anschaulich darstellen. Die Kästchen werden durch Pfeile so miteinander verbunden, wie sie im Programm aufeinanderfolgen, und zwar immer schön von oben nach unten und wenn möglich von links nach rechts.

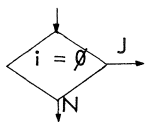
Die wichtigsten grafischen Symbole seien kurz erwähnt:



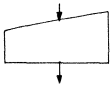
Allgemeine Verarbeitung:
Ein Kästchen für alles, wofür es kein spezielles Kästchen gibt.



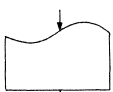
Unterprogramm-Aufruf



Bedingte Verzweigung mit Ausgängen für JA und NEIN

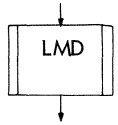


Manuelle Eingabe aus der Peripherie

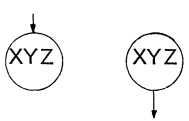


Ausgabe auf Registriergerät

Lesen oder Stanzen eines Lochstreifens
(im Zweifelsfall hineinschreiben)



Längerer, definierter Programmteil (ROUTINE, PROZEDUR, ALGORITHMUS - und was der Namen mehr sind)



Verknüpfungspunkt (CONNECTOR) bzw. Beginn eines Programmteils bzw. markanter Punkt im Programm

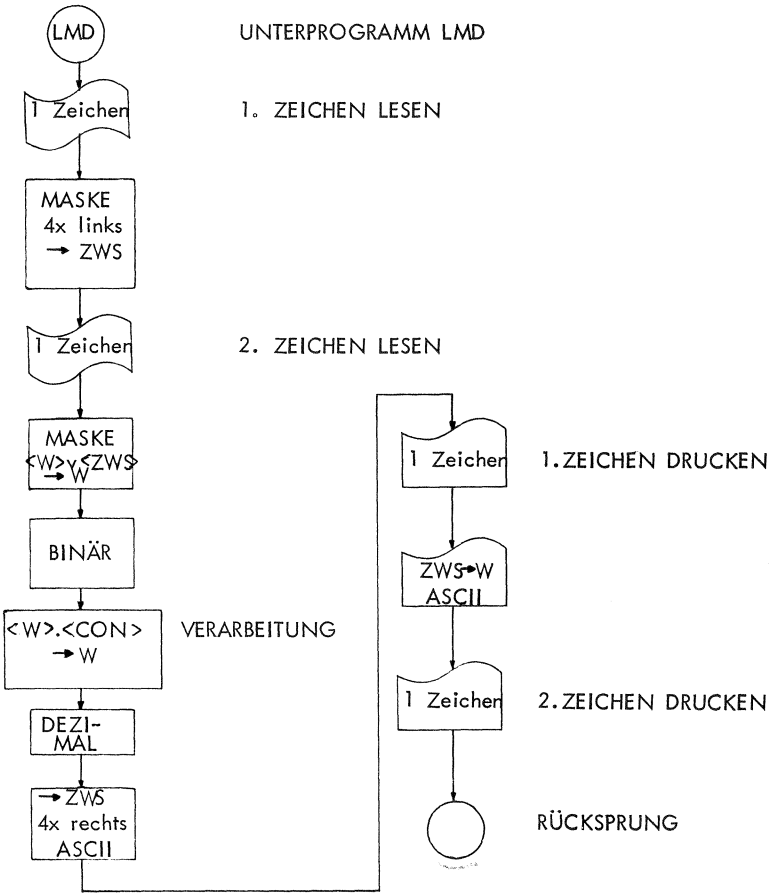
Anhalten des Programms; es muß einen Anstoß von außen bekommen, damit es weitergeht

Noch ein Wort zum letzten Symbol:

HLT

heißt "Anhalten Programm" und ist ein STEUERBEFEHL.

Zeichnen wir zum Abschluß noch das Blockdiagramm unserer Routine LMD auf, und zwar als Unterprogramm:



Womit wir am Ende unseres Exkurses wären, hätten wir nicht noch einiges unterschlagen, was zu wissen wichtig ist.

WIE DIE ENTFERNTESTEN WINKEL DES SPEICHERS ERREICHT WERDEN

Erinnern wir uns an die Darstellung der Adresse im Maschinencode. Die Instruktion OPQAAA hält 9 Bit (= 3 Oktalstellen AAA) für sie bereit. Ein bißchen wenig eigentlich, denn damit kann man nur $2^9 = 1000_8 = 512$ Speicherplätze erreichen. Was tun, wenn der Speicher größer ist (und das wird im allgemeinen der Fall sein)?

Zunächst kann man die ersten 512 Worte mit dem Adreßteil AAA bedienen, also die Adressen 0000_8 bis 0777_8. Diesen Bereich nennen wir die SEITE NULL des Speichers, und Instruktionen, die dorthin zugreifen, heißen ABSOLUT adressiert:

Absolute Adressierung: Effektive Adresse = Adreßteil

wobei wir, wie auch im folgenden, unter EFFEKTIVER Adresse diejenige verstehen wollen, welche den Operanden enthält, mit dem die Instruktion arbeiten soll.

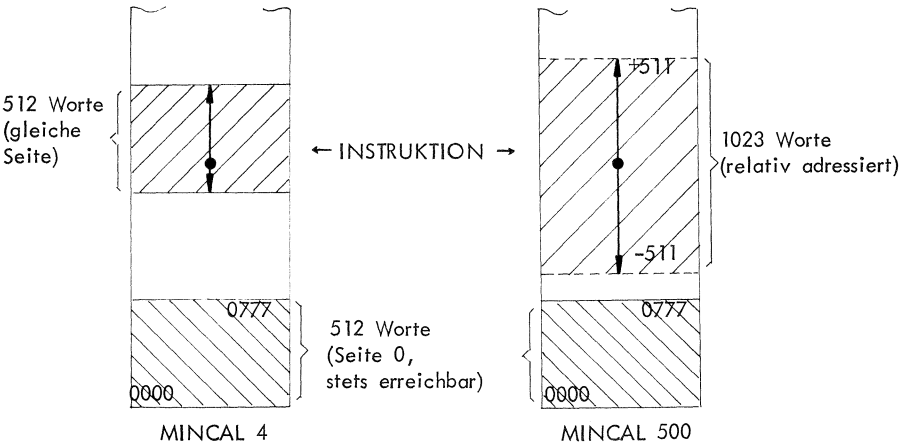
Daneben gibt es zwei Methoden, Adressen in der Nachbarschaft der jeweiligen Programminstruktion zu erreichen. Bei der ersten denkt man sich den Speicher in weitere Seiten von je 512 Worten eingeteilt, und dann kann man alle die Plätze der Seite erreichen, in der die Instruktion selbst steht. Diese Art heißt SEITENWEISE Adressierung (und würde beim MINCAL 4 verwendet):

Seitenweise Adressierung: Effektive Adresse = Seiten-Anfangsadresse
+ Adreßteil AAA

Die zweite Methode läßt es zu, jeweils maximal 511 (= 777_8) Plätze vor oder hinter der Instruktion zu bedienen (je nach Vorzeichen); man nennt sie RELATIVE Adressierung (und findet sie beim MINCAL 500):

Relative Adressierung: Effektive Adresse = Adresse der Instruktion (N)
$\pm$ Adreßteil AAA

Zeichnen wir uns diese Möglichkeiten einmal auf:



(Übrigens braucht man bei symbolischer Programmierung auf diese Adressierungsart nur insoweit zu achten, als man sich über die Erreichbarkeit im klaren ist. Der Unterschied steckt in Vorzeichen und Ergänzung der Maschineninstruktion).

Wie kommt man aber an Speicheradressen heran, die außerhalb der absolut, seitenweise oder relativ erreichbaren Grenzen liegen?

Hier hilft die INDIREKTE Adressierung, und das geht so: Wir reservieren im erreichbaren Gebiet einen Platz, dessen Adresse wir in der Instruktion programmieren, und legen dort die effektive Adresse des weit weg liegenden Platzes ab:

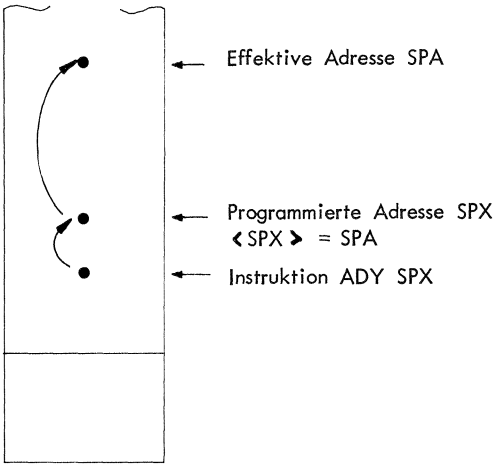
Indirekte Adressierung: Effektive Adresse = Inhalt der programmierten Adresse.

Beispiel: Der Inhalt des (weit entfernten) Platzes SP A soll zum W-Register addiert werden. Wir machen ein Speicherwort SPX in der Nähe unserer Instruktion (oder auch in Seite 0) auf und programmieren:

```
      ADY  SPX
      ...
      ...
SPX   Y    SPA
```

wobei ADY "Addiere indirekt" bedeutet und Y... die Definition für "Adresse ..." ist.

Im Bild sieht das so aus:



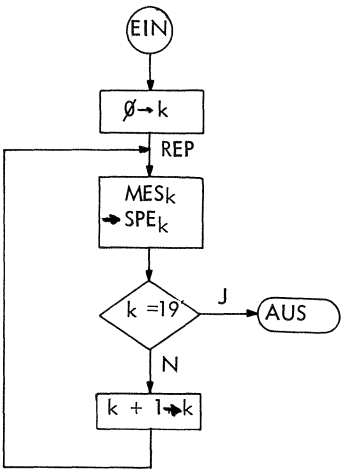
(Für Interessenten: Enthält bei SPEICHERBEZOGENEN Befehlen das letzte Bit des Befehlscodes OP eine 1, findet indirekte Adressierung statt).

Schließlich gibt es noch eine dritte (und, geduldiger Leser, letzte) Kategorie der Adressierungsarten: die INDIZIERUNG. Die Regel hierfür lautet:

Indizierte Adressierung: Effektive Adresse = direkt oder indirekt gefundene Adresse + Inhalt des angesprochenen Indexregisters

Und diese Eigenschaft macht die Indexregister erst richtig wertvoll. Folgendes Beispiel mag ihre Funktion erklären: Die Daten von 20 Meßstellen (mit aufeinanderfolgenden externen Adressen, beginnend bei MES) sollen abgefragt und in ein FELD von 20 Speicherplätzen übertragen werden, dessen erste Adresse SPE sein möge. Wir benutzen das Indexregister 3 und nennen seinen Inhalt den Laufindex k.

Zunächst das Blockdiagramm (wir bauen wieder eine Schleife):



Und dazu das Programm:

```
EIN   LDC3 Ø.
REP   IBG 3 MES       \ MESSEN
      TR  3 SPE       \ SPEICHERN
      ADC3-19.
      BZ  3 AUS       \ K = 19?
      ADC 3 2Ø.
      BR    REP
      ...
SPE   F     2Ø.       \ SPEICHERFELD
MES   X     7ØØ       \ ZUORDNUNG
```

Woraus wir noch dreierlei lernen: Auch Ein- und Ausgabebefehle - wie IBG in unserem Beispiel - sind indizierbar. Speicherfelder darf man nicht vergessen zu definieren, und zwar mit dem Symbol F unter Angabe der Feldgröße in Worten. Und die Marken für externe Adressen bedürfen der Zuordnung einer Oktalzahl (Definition X) - sonst weiß der Computer nichts damit anzufangen.

Und jetzt wären wir wirklich am Ende. Nur eine kleine Sprachregelung steht aus.

WIE MAN SICH MIT SEINEM COMPUTER UNTERHÄLT

Lassen wir einmal alle graue Theorie beiseite, lieber Leser, und vergessen wir das Computer-ABC (Adressen, Bits, Codes, ...). Reden wir davon, wie Sie mit Ihrem Rechner reden sollen.

Da gibt es eine kleine Schwierigkeit: Ihr Maschinchen spricht seine eigene Sprache, Maschinensprache. Diese müssen Sie wohl oder übel lernen, wenigstens so weit, daß eine gewisse Verständigungsbasis erreicht ist. Dann können Sie den Speicher Instruktion für Instruktion, Wort für Wort damit füllen (entsprechende Knöpfe und Schalter hält Ihr Computer für Sie bereit, und in vielen Fällen auch einen Blattschreiber mit Tastatur und einen Lochstreifenleser).

Aber ganze Programme in dieser Weise schreiben? Befehlscodes memorieren, Adressen rechnen, Zahlen aus Tabellen zusammenholen? Sie weisen das mit Recht entrüstet von sich! Kann man das Ding denn nicht in symbolischer Sprache füttern, an die wir uns so sehr gewöhnt haben?!

Gewiß - eben daran hat Ihr Computer-Bauer schon gedacht. Denn neben der HARDWARE - den Gerätschaften Rechner samt Peripherie - liefert er Ihnen (unter anderem) ein bißchen SOFTWARE, ein Umwandlungsprogramm vor allem für die Übersetzung symbolischer in Maschinensprache, ASSEMBLER genannt (weshalb die symbolische Programmiersprache auch Assembler-Sprache heißt).

Sie füttern Ihren Rechner mit dem Assembler (einem langen Stück Lochstreifen) und stellen erfreut fest, daß seine Verständigungsbereitschaft erheblich gestiegen ist. Er versteht auf einmal alle Ihre Anweisungen, übersieht milde alle Kommentare, reserviert Speicherplätze, merkt sich Marken, wandelt Zahlen um und vieles mehr. Er hilft Ihnen sogar, indem er ungültige oder überflüssige Eingaben höflich aber bestimmt zurückweist und fehlende anmahnt. Und zum Schluß spuckt er einen Lochstreifen aus, der Ihr Programm in Maschinencode enthält, mit dem Sie ihn dann wieder füttern.

Worauf Sie dann aber selbst probieren müssen, ob es auch richtig läuft, - denn mitdenken können die MINCAL-Computer (noch) nicht.

Spezifikationen

TECHNISCHE DATEN

MINCAL 513 und MINCAL 523 sind zwei verschiedene Zentraleinheiten des gleichen Computer-Systems. Sie unterscheiden sich hinsichtlich ihrer Konfiguration und ihrer Ausbaufähigkeit; im übrigen sind sie strukturell völlig gleich.

	MINCAL 513	MINCAL 523
Typ:	Universal-Computer für Prozeßanwendungen und technisch-wissenschaftliche Datenverarbeitung	
Programmierung:	Speicher-, fest- oder mischprogrammiert	
Wortlänge:	19 bit (Vorzeichen + 18 Datenbits)	
Verarbeitung:	Parallel (wortweise)	
Kernspeicher:	Ferritkernspeicher 20 bit (19 bit + Paritybit), 1.5 us Vollzyklus in Zentraleinheit enthalten:	
	0.25 kWorte, oder 1 kWorte, oder 4 kWorte	4 kWorte, oder 8 kWorte
	extern erweiterbar:	
	-	bis 32 k in Einheiten von 4 k
Parity-Logik:	(Option)	(Option)
Festspeicher:	ROM-Einheiten (folien-programmiert) 0.5 k/19 bit; 0.75 us Zykluszeit in Zentraleinheit enthalten:	
	max. 4 kWorte (= 8 Einheiten)	max. 2 kWorte (= 4 Einheiten)
	davon stets 1 Einheit für Mikroprogramm Grundbefehle; ferner je 1 Einheit für erweiterten Befehlsvorrat und 1 Einheit für Programmierhilfe XØØ (Optionen)	
Technologie:	Integrierte Schaltkreise (TTL)	
Instruktionen:	74 Maschinenbefehle sowie 5 Typen von Mikrobefehlen	
Instruktionslänge:	1 Wort	
Arbeitsregister:	1 Haupt-Arbeitsregister + 3 Zusatzregister je Ebene	
Indexregister:	3 Indexregister je Ebene	

	MINCAL 513	MINCAL 523
Ebenen:	2 Programmebenen	8 Programmebenen
	mit getrennten Registern und hierarchischer Priorität	
	erweiterbar auf:	
	-	bis zu 64 Programmebenen in Stufen von 8
Interrupt:	Durch Wechsel der Programmebene bei Ende jeder Operation möglich	
X-Kanal:	Schneller Datenkanal für programmgesteuerten wortweisen Datentransfer zur Peripherie. Je 19 Daten-Ein/Ausgänge, 15 Adreßeingänge, Ebenen-Ausgänge, Ebenen-Starteingänge, Steuersignale. TTL-Schnittstelle. Bis ca. 20 kWorte/s.	
PX-Kanal:	Gepufferter Datenkanal für Datentransfer mit reduzierter Geschwindigkeit. Transistor-Schnittstelle (30 V). (Option)	-
DMA-Kanal:	-	Datenkanal für fremdgesteuerten wortweisen Datentransfer zur Peripherie (direkter Speicherzugriff). TTL-Schnittstelle. Bis 667 kWorte/s. (Option)
DMI-Zusatz:	-	8 Speicher-Zählkanäle, extern auf bis zu 64 erweiterbar. (Option)
Interfaces:	8- oder 5-Kanal-Fernschreiber (Option)	8-Kanal-Fernschreiber
	Schnelle Lochstreifenausrüstung (Option) oder Sonder-Interfaces	Schnelle Lochstreifenausrüstung (Option)
Frontplatte:	Enthält Bedienungskonsole mit 20-bit-Datenanzeige, 19 Datentasten, 16 Adreß-Vorwahlschaltern, 5 Zustands- und 3 Prüfanzeigen, 7 Sensorschaltern sowie 14 weiteren Tasten und Schaltern.	
Stromversorgung:	Liefert alle Versorgungsspannungen für die Computer-Zentraleinheit. Netzausfallschutz mit und ohne Wiederstart (Option).	

	MINCAL 513	MINCAL 523
Netzanschluß:	220 V ±10 % 50 Hz einphasig Leistungsaufnahme 0.6 kVA	
Größe:	19"-Einschub 10 Einheiten hoch (ca. 445 mm, einschließlich Stromversorgung) ca. 580 mm tief	
Gewicht:	ca. 65 kg	

Operationszeiten:

Befehlsgruppe	Operationsdauer (us)
Steuerbefehle	7.5... 9.25 us
Register-Befehle	6.75...13.5 us
Speicher-Befehle	6.0 ...35.0 us
Ein/Ausgabe-Befehle	10.5 ...40.0 us + Arbeitszeit des externen Gerätes
Komplexe Befehle (erweiterte Arithmetik)	20.25...390 us

Struktur

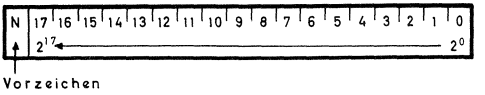
WORTSTRUKTUREN

Die MINCAL 513/523 sind binäre Parallelrechner; Daten und Befehlsworte liegen in binärer Form vor und werden bit-parallel, d.h. wortweise verarbeitet. Das gilt sowohl für den Austausch von Daten zwischen der Recheneinheit und den Speichern sowie der Peripherie als auch für die Verarbeitung innerhalb der Recheneinheit.

Ein Wort hat beim MINCAL 513/523 insgesamt 19 bit (1 Vorzeichen-, 18 Datenbits). Jeder Kernspeicherplatz besitzt darüberhinaus ein weiteres Bit, das als Prüfbit (Parity) benutzt werden kann.

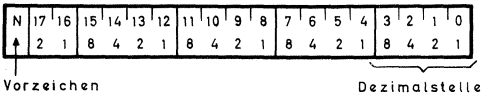
Typische Wortstrukturen sind:

Binärzahl mit Vorzeichen und 18 bit (6 Oktalstellen):

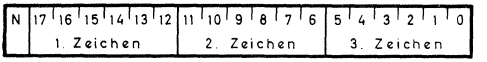


Negative Zahlen werden in Form des Zweierkomplements dargestellt.

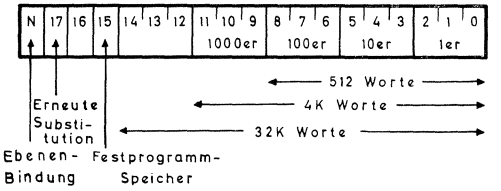
Dezimalzahl (Ganzzahl) mit Vorzeichen und 4 1/2 Dezimalstellen:



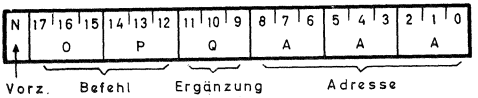
Alphanumerisches Wort mit 3 Zeichen:



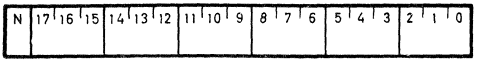
Adreßwort:



Maschinenbefehl:

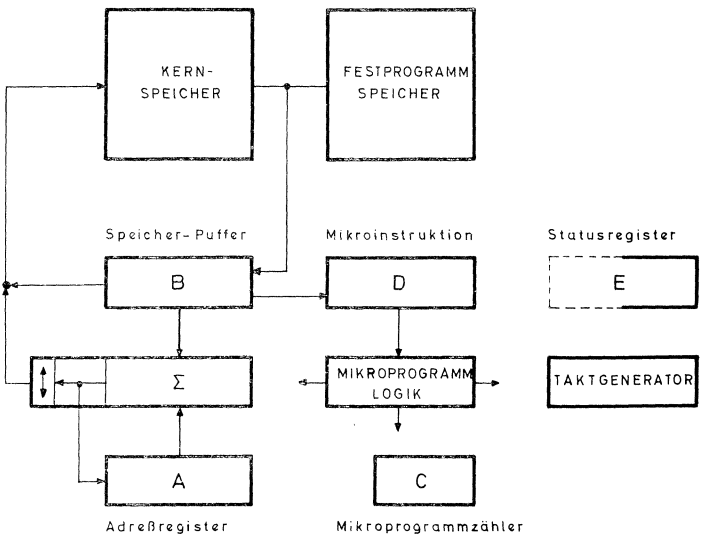


Mikroinstruktion (Bedeutung der Stellen je nach Typ unterschiedlich):



MIKROSTRUKTUR

Zur Verarbeitung der Informationen während eines Operationsverlaufs dient eine Reihe von Komponenten in der Recheneinheit, die mit dem Kernspeicher, dem Festprogrammspeicher und den Datenkanälen in Beziehung stehen:



A, B und D sind 3 Flipflopregister von Wortlänge. Register B dient als Speicherpuffer und zugleich als erstes Datenregister; Register A als Adreß- und zweites Datenregister; beide sind mit einem Addier- und Verschiebeelement Σ verbunden. Register D enthält vorwiegend die Mikroinstruktion, die von der Mikroprogramm-Logik interpretiert und in Steuerbefehle umgesetzt wird. Das Statusregister E mit 8 bzw. 20 bit speichert wichtige Zustände bzw. zusätzlich die jeweils anstehende Programmebene; Register C ist ein 12-bit-Zähler, der die Mikroprogramm-Schritte bestimmt. Der Taktgenerator steuert den Ablauf der Zyklen.

Der Zustand der Recheneinheit ist nur während eines Maschinenbefehls bzw. eines Mikroprogramms von Bedeutung. Eine genaue Beschreibung ihrer Funktion und ihrer Verknüfpungen findet sich im Abschnitt "Mikroprogrammierung".

Für den Benutzer, der nur Maschinen- (ASSEMBLER-) Befehle verwendet - d.h. für den Normalfall - ist die Kenntnis der Mikrostruktur nicht erforderlich.

REGISTER

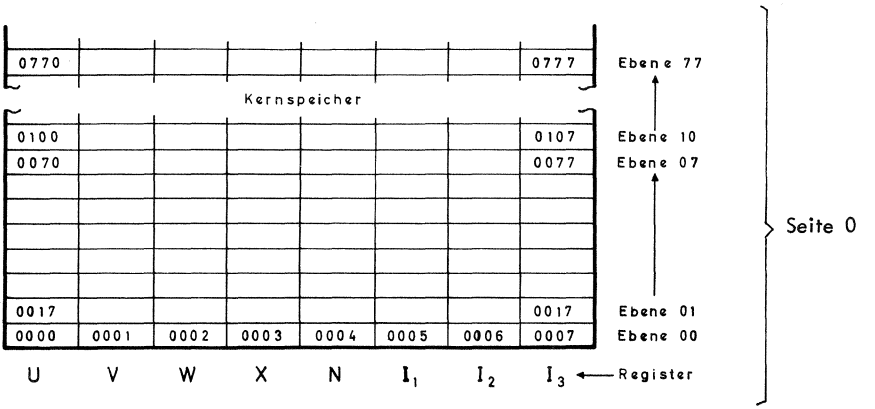
Der Kernspeicher des MINCAL 513/523 hat je nach Ausbauzustand 256, 1024, 4096 oder 8192 Plätze von Wortlänge (extern kann er bis auf 32768 Worte erweitert werden); jeder Platz kann einen beliebigen Dateninhalt haben, der gelesen und gelöscht, zurückgeschrieben oder verändert werden kann. Die ersten 8 Plätze des Kernspeichers mit den oktalen Adressen 0000...0007 haben die Funktion von besonderen Registerplätzen mit der Bedeutung:

Adresse	Bezeichnung	Funktion
0000	U	Sonderregister/Rückkehradresse
0001	V	linkes Hilfsregister/Rückkehradresse
0002	W	Arbeitsregister
0003	X	rechtes Hilfsregister
0004	N	Instruktionszähler
0005	I1	Indexregister 1
0006	I2	Indexregister 2
0007	I3	Indexregister 3

Diese Register sind für den Benutzer von Maschinenbefehlen allein interessant. Ihr Inhalt wird beim Ablauf eines Maschinenbefehls in einer durch den Befehl gekennzeichneten Weise abgefragt oder verändert; wenn der Befehl ausgeführt ist, sind alle für den Programmverlauf wichtigen Informationen in diesen Registerplätzen enthalten.

Der MINCAL 513/523 hat mehrere Programmebenen; beim MINCAL 513 sind 2 Ebenen vorgesehen; beim MINCAL 523 8 (die extern bis auf 64 erweitert werden können). In jeder Ebene kann der Rechner ein selbständiges und von den anderen Ebenen unabhängiges Programm ausführen (Multiprogramming). Um dies zu gewährleisten, besitzt jede

Ebene einen eigenen Satz von 8 Registern im Kernspeicher; deren Adressen sind nach der Rangfolge der Ebenen so geordnet, daß die (oktale) 100er- und 10er-Adresse der (oktalen) Nummer der Programmebene entspricht:



Die Programmebenen haben eine feste Rangordnung in der Weise, daß jede Ebene Vorrang vor allen anderen mit niedrigeren Nummern hat. Wird eine Ebene mit höherem Rang gestartet, so wird am Ende der laufenden Operation das Programm der niedrigeren Ebene unterbrochen, und der Rechner setzt das Programm der höheren Ebene fort, indem er deren Register benutzt.

KERNSPEICHER

Der Kernspeicher des MINCAL 513/523 enthält Programm und Daten in beliebiger Weise. Die oktalen Adressen der Kernspeicherplätze lauten:

0.25	k	000000...000377	MINCAL 513
1	k	000000...001777	
4	k	000000...007777	
8	k	000000...017777	MINCAL 523
...			
32	k	000000...077777	

Die ersten 512 Speicherplätze (Seite 0; Adressen 000000...000777) haben insofern eine besondere Bedeutung, als sie durch absolute Adressierung von jedem anderen Platz aus erreichbar sind und bestimmte Befehlsgruppen (Manipulations- und Registerbefehle) sich auf sie beziehen.

Am Anfang der Seite 0 liegen die Registersätze der einzelnen Programmebenen; bei 64 Ebenen ist die gesamte Seite 0 von ihnen belegt.

Jeder Kernspeicherplatz enthält 20 bit (19 bit + Parity). Das Paritätsbit wird automatisch erzeugt und auf ungerade Parität überprüft. Bei Parity-Fehler erfolgt Rechner-Stop (auf Anzeige CK3) oder Start einer besonderen Programmebene. Die Parity-Logik ist eine Option.

FESTSPEICHER

Die Computer MINCAL 513/523 enthalten bis zu 4 bzw. 8 Festspeicher-Einheiten mit je 512 Worten zu je 19 bit. Die oktalen Adressen der Einheiten lauten:

0	100000...100777	Mikroprogramm Grundbefehle	oder Benutzer-programm
1	101000...101777	" erweiterter Befehlsvorrat	
2	102000...102777	Programmierhilfe X00	
3	103000...103777	Benutzerprogramm	
4	104000...104777	"	nur bei MINCAL 513
5	105000...105777	"	
6	106000...106777	"	
7	107000...107777	"	

Die Einheiten 0 und 1 enthalten Standard-Mikroprogramme, die der Computer zur Interpretation des Grund- und erweiterten Vorrats der Maschinenbefehle benötigt.

In Platz 2 kann eine festverdrahtete Programmierhilfe (XØØ) eingesetzt werden, die bequemes Einlesen, Ändern und Ausgeben von Programmen erlaubt. Das zugehörige Programm ist 256 Worte lang und belegt damit nur die erste Hälfte des Festspeichers; die andere Hälfte kann auf Wunsch ein weiteres Programmpaket erhalten.

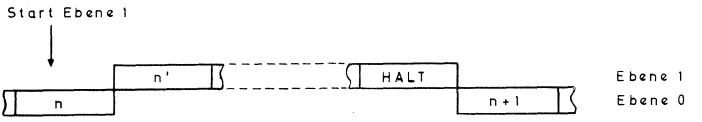
Sowohl in Platz 1 und 2 als auch insbesondere in den übrigen Plätzen können Festspeicher mit Benutzerprogrammen (Maschinenprogramme, Mikroprogramme, Festwerte) vorgesehen werden. Sie werden vom Hersteller aufgrund von Maschinencode-Lochstreifen im Oktalformat hergestellt.

In der Grundstellung (nach Betätigen der Taste END oder bei Netzeinschaltung, wenn Netzausfallschutz mit Wiederstart vorgesehen ist) steht das N-Register der Ebene 0 auf 102000; die erste dann ausgeführte Instruktion hat die Adresse 102001 (zweites Wort im Festspeicher 2). Ist dort die Programmierhilfe X00 enthalten, so wird (wenn kein Sensor erregt ist) auf Platz 001000 im Kernspeicher verzweigt; dort muß die erste Programminstruktion stehen, die in diesem Falle auszuführen ist.

PROGRAMMEBENEN

Wie bereits erwähnt, besitzt der Rechner MINCAL 513 zwei Programmebenen. Die Ebene 0 ist für das normale Arbeitsprogramm, die vorrangige Ebene 1 für Programmunterbrechungen vorgesehen. Zu jeder Ebene gehört ein Satz von 8 Registern im Kernspeicher; der Rechner bedient sich automatisch der Register der jeweiligen Programmebene. Außerdem können Teile der Peripherie bestimmten Ebenen zugeordnet werden.

Eine Programmunterbrechung geht z.B. in folgender Weise vor sich: Während der Rechner den Maschinenbefehl (n) in Ebene 0 ausführt, wird von außen Ebene 1 gestartet. Am Ende der Operation n wird Ebene 1 wirksam und dadurch Ebene 0 unterbrochen; der Rechner setzt unverzüglich die Arbeit fort, jedoch in der Ebene 1, d.h. unter Benutzung der betreffenden Register. Die erste Operation wird durch den Stand (n') des Instruktionszählers der Ebene 1 bestimmt. Das Programm läuft bis zu einem Halt-Befehl in Ebene 1, wodurch das Programmniveau wieder ausgeschaltet wird; am Ende dieser Operation wird das unterbrochene Programm in Ebene 0 mit der Instruktion (n+1) unverzögert fortgesetzt.



Der Vorzug getrennter Programmebenen mit getrennten Registern liegt einmal darin, daß Programmunterbrechungen ohne Programmieraufwand beherrscht werden können. Außerdem besteht, insbesondere wenn die Anzahl der Ebenen größer wird und jeder Ebene außerdem ein eigener Datenkanal von und zu spezifischen Peripheriegeräten zugeordnet wird, die Möglichkeit, eine Mehrfachausnutzung des Rechners zu erreichen (Multiprogrammierung). Dies geschieht ohne jeden Programmaufwand, da jede Ebene eigene Register (Instruktionszähler, Arbeits-, Hilfs- und Indexregister) besitzt und damit völlig unabhängig von den anderen ist.

Die Anzahl der Programmebenen ist beim MINCAL 523 durch Einbau eines Prioritätssystems (und Erweiterung des Statusregisters E) auf 8 erhöht, und extern kann die Ebenenzahl bis auf 64 erweitert werden. Die Ebenen haben untereinander eine feste Rangordnung.

Eine Ebene wird entweder von einem äußeren Signal, durch die Fertigmeldung eines ihr zugeordneten Peripheriegeräts oder mittels eines speziellen Programmbefehls in einer niedrigeren Ebene, die damit unterbrochen wird. Beendet wird das Programm einer Ebene durch einen Halt-Befehl.

Durch speziellen Programmbefehl kann erreicht werden, daß der Rechner das Programm in einer Ebene auch dann fortsetzt, wenn eine Ebene mit höherem Rang gestartet wird. Diese mit DISABLE bezeichnete Funktion ist nur beim MINCAL 523 vorgesehen.

NETZAUSFALLSCHUTZ

Diese Option bewirkt, daß bei Netzausfall der Rechner mit dem Ende des gerade ablaufenden Maschinenbefehls bzw. Mikroprogramms angehalten wird; alle Daten sind im Kernspeicher enthalten und damit gesichert. Die Anzeige CK1 leuchtet dabei auf.

In der Ausführung "mit Wiederstart" geschieht bei Wiederkehr des Netzes (oder auch bei erstmaliger Netzeinschaltung) folgendes: Alle speichernden Halbleiterschaltungen werden nullgestellt, das N-Register der Ebene 0 wird auf 102000 gesetzt und die Ebene 0 wird gestartet (entspricht dem Betätigen der Tasten RES-END-STA in dieser Reihenfolge.

Ab Adresse 102001 (bzw. bei eingebender Programmierhilfe X00 ab Adresse 001000) ist ein Programm vorzusehen, das die notwendigen Funktionen für den Fall der Netzwiederkehr vorsieht.

STEUERANSCHLÜSSE

An einen Adapterbaustein sind beim MINCAL 513/523 wichtige Steuersignale anschließbar:

je ein Eingang für den Start der Programmebenen 0 und 1 ($\overline{LST0}$, $\overline{LST1}$)
bzw. 0 bis 7 ($\overline{LST0}$...$\overline{LST7}$)

je ein Ausgang für "Programm läuft" (PROGR) und Programm läuft in höherer Ebene" (INTERRUPT)

7 Sensoreingänge ($\overline{SR1}$...$\overline{SR7}$)

Die beiden Ausgänge werden zur Meldung des erfolgten Starts bzw. des laufenden Programms benutzt. Die 7 Sensoreingänge wirken auf Verzweigungsbefehle, die Sensor-bedingt sind, und dienen zur Modifikation des Programmverlaufs; sie entsprechen den Schaltern auf der Frontplatte.

X-KANAL

Der programmgesteuerte Datenkanal (X-Kanal) des MINCAL 513/523 dient zum schnellen Austausch von Informationen mit der Peripherie. Der X-Kanal wird benutzt:

a) zur wortweisen, bit-parallelen Abfrage von Informationen aus Datenträgern bzw. zur Ausgabe in Datenträger, die mit integrierten Schaltkreisen bestückt und in räumlicher Nachbarschaft des Rechners angeordnet sind, wobei der Datenträger unabhängig von der Programmebene nur durch seine Adresse bestimmt wird,

b) zum bit-parallelen Transfer von Worten oder Teilworten zwischen dem Rechner und räumlich benachbarten Puffern, die den Programmebenen zugeordnet sind und die z.B. Eingabe- und Registriergeräte, Multiplexer und andere Einrichtungen steuern, deren Arbeitsgeschwindigkeit vergleichsweise gering ist.

Der X-Kanal hat folgende Schnittstellen:

19 Dateneingänge ($\overline{SXN}$, $\overline{SX17}$...$\overline{SX0}$)
19 Datenausgänge (ZXN, ZX17...ZX0)
15 binäre Adreßausgänge (MX14...MX0)
1 Ausgang für nicht-niveaugebundene Ein/Ausgabe ($\overline{NLX}$)
16 Ausgänge für niveaugebundene Ein/Ausgabe ($\overline{LX00}$...$\overline{LX17}$) von und zu Puffern, die den 8 Programmebenen zugeordnet sind (bei externer Erweiterung der Ebenenzahl sind weitere Ausgänge dort verfügbar)
16 Eingänge für Programmstarts (RX00...RX17) bei Fertigmeldung der Puffer, die den 8 Programmebenen zugeordnet sind (bei externer Erweiterung der Ebenenzahl sind weitere Eingänge dort vorgesehen)

1 Steuerpotential bei Dateneingabe ($\overline{SX}$)
1 Taktimpuls für Datentransfer nach außen ($\overline{ZTX}$)
1 Taktimpuls für Adreßtransfer nach außen ($\overline{MTX}$)
5 Steuerpotentiale Eingabegerät Ein (IBUSY ON)
Ausgabegerät Ein (OBUSY ON)
Ein/Ausgabegerät Aus (BUSY OFF)
Fertigmeldung Aus (READY OFF)
Startverriegelung (LOCK)
2 Abfrageeingänge Gerät in Aktion ($\overline{BUSY}$)
Gerät fertig ($\overline{READY}$)
2 Steuerpotentiale für Blocktransfer Anfang (BXB)
Blocktransfer Ende (BXE)
1 allgemeiner Taktimpuls für jeden Ein/Ausgabezyklus ($\overline{TX}$)

Jeder Ein/Ausgabevorgang über den X-Kanal hat eine Dauer von 1.5 us; während dieser Zeit stehen die Potentiale an den Schnittstellen an. Taktimpulse liegen am Ende des Vorgangs und dauern 0.375 us.

Im nicht-niveaugebundenen Datentransfer (Fall a) wird der externe Informationsträger durch die Adresse MX... sowie zusätzlich durch den Ausgang $\overline{NLX}$ identifiziert. Bei Dateneingabe wird $\overline{SX}$ erregt; der Inhalt des Informationsträgers wird am Dateneingang bereitgestellt und am Ende des Zyklus' vom Rechner übernommen. Bei Datenausgabe stellt der Rechner die Information am Datenausgang bereit und überschreibt sie mit dem Impuls $\overline{ZTX}$ in den externen Datenempfänger. Wenn erforderlich, kann auch die Adresse MX... ganz oder teilweise nach außen überschrieben werden; hierzu wird der Taktimpuls $\overline{MTX}$ ausgegeben.

Beim niveaugebundenen Datentransfer (Fall b) wird der Puffer durch den Ausgang $\overline{LX}$... identifiziert, der zur jeweiligen Programmebene gehört. Sind je Ebene mehrere Puffer vorgesehen entsprechend mehreren Ein/Ausgabegeräten, die gleichzeitig und unabhän-

gig voneinander arbeiten, so wird der Puffer durch die Ausgänge MX8...MX3 (entsprechend der aktalen 100er- und 10er-Adreßstelle) zusätzlich bestimmt. Zu einem externen Datenpuffer gehört u.U. ein Adreßpuffer (der zusätzliche Angaben wie Ein/Ausgabeformat, Meßstelle usw. speichert) sowie ein Satz von 4 Flipflops für die Zustände "Eingabegerät" (IBUSY), "Ausgabegerät in Aktion" (OBUSY), "Gerät fertig" (READY) und "Startverriegelung" (LOCK). Die BUSY-Flipflops werden vom Rechner gesetzt und lösen einen Eingabe- bzw. Ausgabevorgang zwischen Puffer und Periphergerät aus; ist dieser beendet, so wird das betreffende BUSY-Flipflop selbsttätig zurückgesetzt und das READY-Flipflop eingeschaltet, wodurch ein Programmstart RX... der zugehörigen Ebene zugeordnet wird. Der Programmstart kann dadurch verhindert werden, daß das LOCK-Flipflop vom Rechner gesetzt wurde; er wird wieder wirksam, wenn LOCK zurückgestellt wird. READY wird in jedem Falle, BUSY kann vom Rechner zurückgestellt werden. Solange READY eingeschaltet (und LOCK nicht gesetzt) ist, steht der Programmstart $\overline{RX}$... an.

Der Verkehr zwischen Rechner und Puffern kennt verschiedene Arten von Ein/Ausgabe-Zyklen über den X-Kanal:

Übernahme:	$\overline{SX}$ ist erregt; der Inhalt des Datenpuffers wird dem Dateneingang $\overline{SX}$... angeboten und vom Rechner übernommen. READY wird ausgeschaltet.
Eingabe-Start:	Der Datenausgang ZX... ist nicht durchgeschaltet; mit $\overline{ZTX}$ wird Nullinhalt in den Datenpuffer überschrieben. Die am Ausgang MX... bereitstehende Adresse wird mit $\overline{MTX}$ in den Adreßpuffer überschrieben (falls vorhanden; auch teilweise). IBUSY wird gesetzt.
Ausgabe:	Der Datenausgang ZX... ist durchgeschaltet; mit $\overline{ZTX}$ wird der Datenpuffer geladen. Die am Ausgang MX... bereitstehende Adresse wird mit $\overline{MTX}$ in den Adreßpuffer überschrieben (falls vorhanden; auch teilweise). OBUSY wird gesetzt.
Zustandsänderung:	Die BUSY-Flipflops oder READY werden zurückgestellt; oder LOCK wird gesetzt oder zurückgestellt.
Zustandsabfrage:	Der Rechner fragt ab, ob BUSY oder READY gesetzt sind.

Die Schnittstellen des X-Kanals sind mit integrierten TTL-Schaltkreisen ausgeführt. Sie befinden sich auf Adapterkarten und sind über spezielle Flachkabel mit den Puffern usw. zu verbinden, die in unmittelbarer Nähe des Rechners angeordnet sein müssen. Die Interface-Schaltungen können mit Bausteinen der MINCAL 500-Baureihe aufgebaut werden.

Ein oder zwei Puffer mit Interface-Schaltungen für spezielle Ein/Ausgabegeräte können sich im Rechner befinden.

KONSOL-PERIPHERIE

Die dem MINCAL 523 zugeordnete (und beim MINCAL 513 auf Wunsch vorgesehene) Konsol-Peripherie besteht aus

einem 8-Kanal-Fernschreiber (Teletype ASR 33) mit angebautem, durch Codes ein/ausschaltbarem Locher und Leser

einem 8-Kanal-Streifenleser und einem 8-Kanal-Streifenlocher (beides Optionen)

Die Interfaces hierzu befinden sich in der Zentraleinheit; die Geräteadressen sind 00X (Teletype) und 06X (Locher/Leser). Die Konsol-Peripherie arbeitet auf Programmebene 0.

PX-KANAL

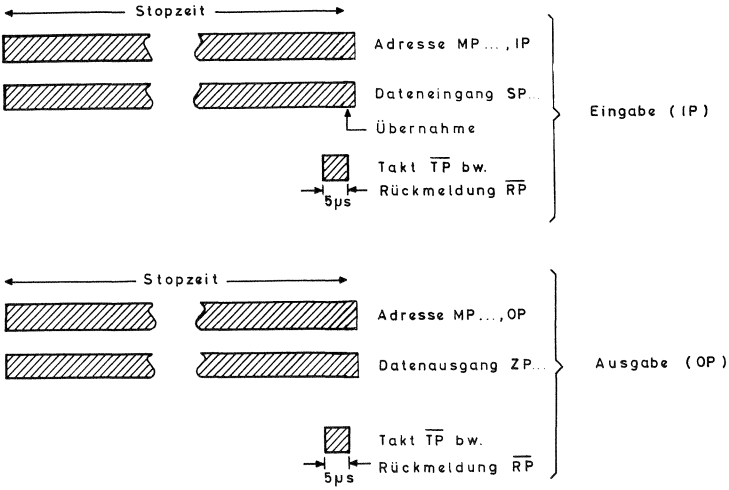
Der PX-Kanal (eine Option des MINCAL 513) bietet eine einfache Kommunikationsmöglichkeit mit externen Datenträgern. Er ermöglicht bit-parallele, wortweise Ein- und Ausgabe von Informationen von und zu maximal 256 externen Datenträgern mit geringer Übertragungsgeschwindigkeit.

Der PX-Kanal hat folgende Schnittstellen (Normalausführung):

19 Dateneingänge ($\overline{SPN}$, $\overline{SP17}$...$\overline{SP0}$)
19 Datenausgänge (ZPN, ZP17...ZP0)
3x8 oktal entschlüsselte Adreßausgänge (MP00...MP07; MP10...MP17; MP20...MP24)
1 Ausgang "Eingabe" (IP)
1 Ausgang "Ausgabe" (OP)
1 Taktimpuls-Ausgang ($\overline{TP}$)
1 Rückmeldeeingang ($\overline{RP}$)

Der Adreßausgang identifiziert den peripheren Datenträger durch eine der 256 oktalen Adressen 000..377, wobei durch die Ausgänge IP und OP zusätzlich nach Gebern und Empfängern von Informationen unterschieden werden kann. Am Ende des Eingabevorgangs übernimmt der Rechner das am Dateneingang anstehende Wort, während er für die Dauer eines Ausgabevorgangs ein Wort am Datenausgang bereitstellt.

Der PX-Kanal hat 2 Register. Die Adresse ist im A-Register enthalten, das auszugebende Wort im D-Register. Der Vorgang dauert, je nach 100er-Adresse, entweder eine im Bereich 0.1...100 ms fest eingestellte Zeit oder unbegrenzt lange. Im ersten Fall wird am Ende der Ein/Ausgabe-Zeit ein Taktimpuls $\overline{TP}$ ausgegeben, im anderen der Vorgang durch ein Rückmeldesignal $\overline{RP}$ beendet, das von außen kommt (Unterscheidung durch Adreßbit 8, das im ersten Falle 1, im zweiten 0 sein muß).



Der PX-Kanal wird in Ebene 0 über normale X-Kanal-E/A-Befehle programmiert. Die zugehörigen externen Adressen sind 000...777. Adressen über 400 bewirken Transfer mit fest eingestellter Zeit ohne Rückmeldung. Die Adressen evtl. vorhandener Konsol-Peripherie dürfen nicht für den PX-Kanal benutzt werden.

Die Schnittstellen des PX-Kanals sind in diskreten Transistorschaltungen ausgeführt, die auf zwei Adapterbausteinen (Adreß- und Datenadapter) angeordnet sind.

DMA-KANAL

Auf Wunsch kann der Rechner MINCAL 523 mit einem Adapter-Baustein für direkten Speicherzugriff ausgerüstet werden. Die Schnittstellen (integrierte TTL-Schaltkreise) umfassen einen 19-bit-Dateneingang, einen 19-bit-Datenausgang, einen 15-bit-Adreßeingang, zwei Steuereingänge für Lese- und Schreib-Betrieb sowie einen Meldeausgang.

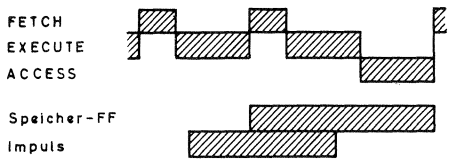
Der Speicherzugriff dient dazu, unabhängig vom Programm Daten von Wortlänge mit sehr hoher Transferrate in den Kernspeicher einzugeben oder aus dem Speicher zu lesen. Er arbeitet mit höchster Prioritätsstufe, indem je Wort ein ACCESS-Zyklus erregt und das Programm für dessen Dauer unterbrochen wird (cycle stealing).

Die maximale Transferrate des Speicherzugriffs-Kanals beträgt ca. 667000 Worte je Sekunde.

Der DMA-Kanal ist in Verbindung mit Trommel-, Platten- und Magnetbandspeichern sowie mit schnellen Analog-Digital-Umsetzern erforderlich.

DMI-ZUSATZ

Durch Einbau eines Zusatzes ist es beim MINCAL 523 möglich, programm-unabhängige Zählkanäle unter Benutzung des Kernspeichers aufzubauen. Jeder Kanal ist einem Speicherplatz zugeordnet; bei jedem Impuls wird der Inhalt des betreffenden Speicherplatzes um 1 erhöht. Die Zählimpulse werden in Flipflops gespeichert; unmittelbar darauf bzw. am Ende der nächsten Mikroinstruktion wird ein ACCESS-Zyklus von 1.5 us Dauer ausgelöst, während dessen der Inhalt des Speicherplatzes gelesen und um 1 erhöht zurückgeschrieben wird; das Speicher-Flipflop wird wieder ausgeschaltet. Für die Dauer des ACCESS-Zyklus' wird das laufende Programm unterbrochen.



Im Rechner kann eine Baugruppe für 8 Zählkanäle vorgesehen werden; extern kann das System auf maximal 64 Zählkanäle erweitert werden. Die Zähleingänge haben untereinander eine feste Rangordnung, so daß auf mehreren Kanälen gleichzeitig eintreffende Zählimpulse nacheinander und in geregelter Weise verarbeitet werden.

Die maximale Zählrate bei 1...5 Kanälen beträgt ca. 130 kHz; bei größerer Kanalzahl ist sie entsprechend niedriger.

Die Zuordnung der Zählkanäle zu den Speicheradressen ist fest programmiert; zu jeder Gruppe von 8 Kanälen gehören 8 aufeinanderfolgende Plätze auf den 1er-Adressen 0...7. Jeder Kanal hat eine Kapazität von $2^{18} - 1 = 262143$ Impulsen. Jeder Kanal kann so geschaltet werden, daß bei Überlauf der benachbarte Speicherplatz inkrementiert wird (unter Fortfall des entsprechenden Zähleingangs).

Die Schnittstellen der Zählkanäle (1 Eingang je Kanal) sind mit integrierten TTL-Schaltkreisen ausgerüstet und auf einem Adapter-Baustein angeordnet.

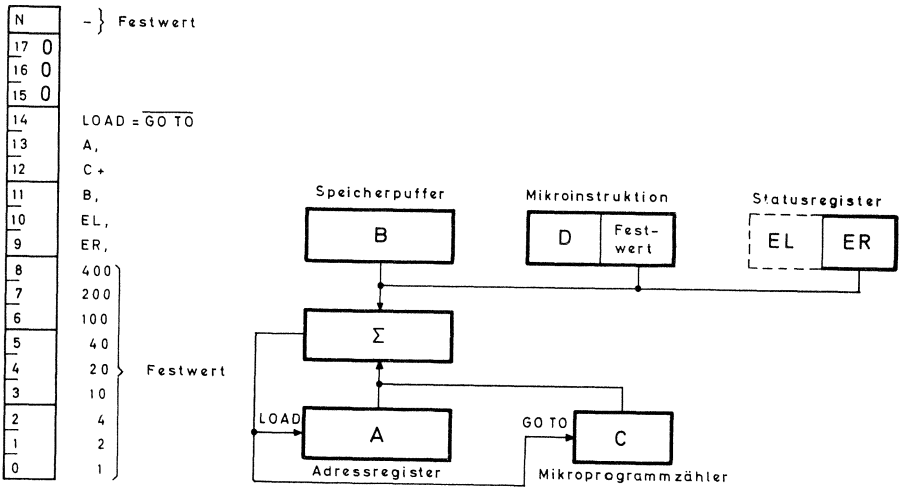
Mikroinstruktionen

Der MINCAL 513/523 besitzt einen festen Vorrat an Mikroinstruktionen, die für die Darstellung der Makrobefehle verwendet werden. Jedem Benutzer stehen die Mikroinstruktionen darüberhinaus zum Aufbau von speziellen Befehlen und schnellaufenden Standard-Routinen zur Verfügung. Mikroprogramme werden durch die Befehle COD, CM oder CMY aufgerufen.

Jede Mikrooperation besteht aus FETCH- und EXECUTE- Zyklus; der erstere dauert 0.75 µs (wenn im Kernspeicher programmiert: 1.5 µs).

Jede Mikroinstruktion hat die Länge eines 19 bit-Wortes; Mikroprogramme können jeweils in den ersten 4 k des Festprogrammspeichers oder des Kernspeichers stehen.

5 Typen von Mikroinstruktionen sind vorhanden:



GOTO/LOAD-Typ (EXECUTE-Dauer: 0.75 µs).

Diese Mikroinstruktion hat die Form GOTO/LOAD (A, C + B, EL, ER, Festwert) und bewirkt, daß der Mikroprogrammzähler C (bei GOTO) bzw. das Adressregister A (bei LOAD) auf einen neuen Wert gesetzt werden, der sich durch ODER-Verknüpfung bzw. durch Addition aus dem Inhalt vom A-, C-, B-Register, linker und rechter Hälfte des E-Registers und einem positiven oder negativen Festwert von 3 Oktalstellen ergibt.

N	RESET = $\overline{SET}$
17	0
16	0
15	1
14	0
13	
12	
11	DISABLE
10	BN
9	FLAG (ADR)
8	FLAG 2
7	FLAG
6	LINK
5	CARRY
4	BORRO
3	LEVEL
2	OP
1	IP
0	END

SET/RESET-Typ (EXECUTE-Dauer: 0.75 µs).

In der Form SET/RESET ..., ..., ...; END bewirkt diese Mikroinstruktion Setzen oder Rückstellen bestimmter Flipflops im B-Register (BN) oder im Statusregister E, Ein- oder Ausschalten einer Programmebene (LEVEL) oder Einleiten eines Ein/Ausgabevorgangs über den P-Kanal (IP, OP; stets mit SET). Mit SET-FLAG (ADR) wird das FLAG-Bit im Statusregister gesetzt, wenn die Stellung der Adresschalter in der Frontplatte mit dem Inhalt des A-Registers übereinstimmt.

END bedeutet Ende des Mikroprogramms mit diesem Schritt (darf nicht mit Set Level oder Reset Level zusammen programmiert werden).

N	
17	0
16	0
15	1
14	1
13	LOCK
12	ADR TR
11	DAT OUT
10	DAT IN
9	DAT TR
8	OBUSY ON
7	IBUSY ON
6	BUSY OFF
5	READY OFF
4	FLAG (BUSY)
3	FLAG (READY)
2	BLOCK BEGIN
1	BLOCK END
0	END

INPUT/OUTPUT-Typ (EXECUTE-Dauer 1.5 µs).

Diese Mikroinstruktion steuert Ein/Ausgabevorgänge über den X-Kanal. In jedem Falle ist der Inhalt des A-Registers auf den Adreßausgang MX... geschaltet; bei positivem Vorzeichen von A ist NLX, bei negativem Vorzeichen der zur jeweiligen Ebene gehörige Ausgang $\overline{LX}$... erregt.

LOCK, OBUSY ON, IBUSY OFF und READY OFF steuern die gleichnamigen Ausgänge, BLOCK-BEGIN/END die Steuerpotentiale BXB/BXE. Mit FLAG (BUSY)/(READY) wird das FLAG-bit des Statusregisters gesetzt , wenn der betreffende Eingang erregt ist.

Mit ADR TR wird der Impuls $\overline{MTX}$, mit DAT TR der Impuls $\overline{ZTX}$ ausgegeben.
DAT OUT schaltet den Inhalt des B-Registers auf den Datenausgang ZX...; DAT IN erregt das Potential $\overline{SX}$ und schaltet die am Dateneingang $\overline{SX}$... anstehende Information durch zwecks Übernahme ins B-Register.

Der Impuls $\overline{TX}$ wird bei jeder I/O-Operation ausgegeben.

END bedeutet Ende des Mikroprogramms mit diesem Schritt.

N		
17	0	DONT
16	1	
15		END = $\overline{\text{SKIP}}$
14		SENSOR
13		AN
12		A17
11		
10		BN
9		B17
8		FLAG 2
7		FLAG
6		LINK
5		CARRY
4		BORRO
3		NEG
2		Q 4
1		Q 2
0		Q 1

IF...

SKIP/END-Typ (EXECUTE-Dauer: 0.75 µs).

In der Form (DONT) SKIP/END, IF..., ..., ... bewirkt diese Mikroinstruktion, daß (bei SKIP) der folgende Mikrobefehl (nicht) übersprungen oder (bei END) das Mikroprogramm (nicht) beendet wird, wenn einer oder mehrere der angegebenen Zustände vorhanden sind. SENSOR bedeutet, daß der von den 3 letzten Stellen Q... bezeichnete Sensoreingang erregt ist; AN, A17, BN, B17 sind die entsprechenden bits im A- und B-Register; die übrigen Stellen betreffen den Zustand des Statusregisters E.

N		
17	1	CLEAR = $\overline{\text{READ}}$
16		(M)
15		(R)
14		4
13		2
12		1
11		RESTORE
10		LOAD
9		LOAD
8		STORE
7		STORE
6		ARITHM
5		A,
4		$\overline{A}$ +
3		1 +
2		B,
1		$\overline{B}$
0		END

Adresse

U	V	W	X	N	I
0	0	0	0	0	0
0	0	0	0	0	1
0	0	0	0	1	1
0	0	1	1	0	0
0	1	0	1	0	0

&MARK
&MODIFY

LEFT
RIGHT

(Summe)

READ/CLEAR-Typ (EXECUTE-Dauer: 1.5 µs).

Diese Mikroinstruktion hat die Form READ/CLEAR (Adresse); RESTORE; LOAD&..., STORE... (Summe); END und steuert den Datentransfer zwischen den Speichern und den Flipflop-Registern A und B.

Durch CLEAR wird der Speicherplatz gelöscht; mit READ wird der Inhalt des Speicherplatzes ins Register B übertragen und der Speicherplatz gelöscht.

RESTORE schreibt den Inhalt von B in den Speicherplatz zurück. Die Summe (der Ausgang des Addierers Σ) wird mit LOAD ins A-Register, mit STORE in den Speicherplatz übertragen.

Die Speicheradresse wird entweder vom A-Register (M), von den drei letzten Stellen des E-Registers (R) oder in der Mikroinstruktion selbst vorgegeben. Sie ist nur bei (M) und positivem Vorzeichen des A-Registers nicht niveaugebunden.

LOAD&MARK dient zur Separierung der einzelnen Oktalstellen einer Makroinstruktion $\pm$OPQAAA und ihrer Übertragung in die bestimmten Register:

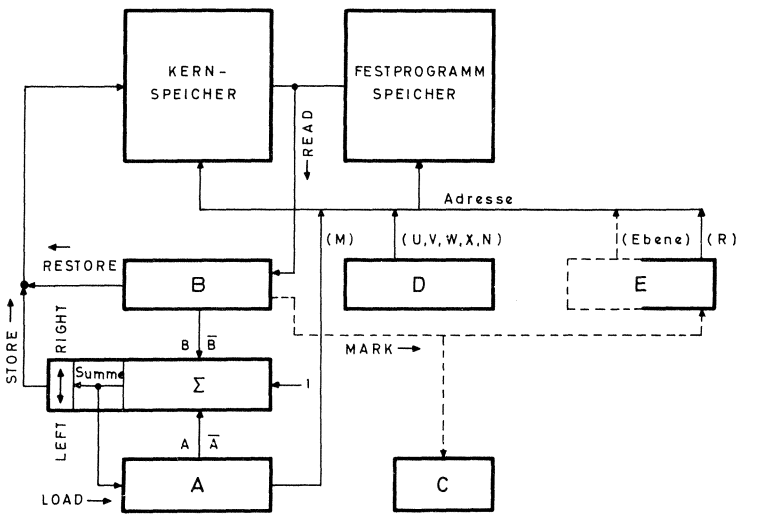
Befehlscode → Mikroprogrammzähler C:	00P0
Ergänzung → Statusregister E:	00Q
Adresse + Vorzeichen → Adressregister A:	$\pm$000AAA

LOAD&MODIFY bewirkt das Laden des A-Registers mit dem Summenausgang, wobei das A-Register als ein Summand auftritt. Ist das Vorzeichen des A-Registers negativ, wird das Zweier-Komplement des A-Inhalts gebildet.

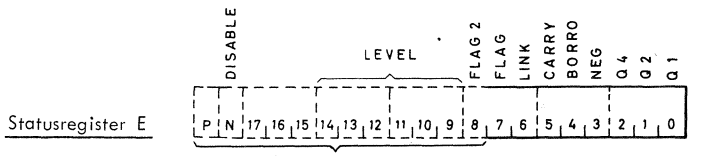
Das Abspeichern der Summe kann um 1 bit links verschoben (STORE LEFT) oder rechts verschoben (STORE RIGHT) geschehen, wobei zwischen logischer und arithmetischer (ARITHM) Verschiebung unterschieden wird.

Die Summe wird aus dem Inhalt des A-Registers und des B-Registers sowie einer rechtsbündigen 1 (2^0) gebildet. Es kann der Inhalt von A und/oder B oder ein einfaches Komplement ($\overline{A}$ und/oder $\overline{B}$) als Summand benutzt werden. Mit A, $\overline{A}$ oder B, $\overline{B}$ wird die Zahl (-1) als Summand eingeführt.

END bedeutet Ende des Mikroprogramms mit diesem Schritt.



Das Statusregister E hat 8 oder 20 bit und ist in folgender Weise organisiert:



Q1, Q2, Q4 speichern die Ergänzung Q einer Makrooperation. In NEG, BORRO, CARRY steht eine 1, wenn die Summe der vorhergehenden READ/CLEAR-Operation negativ war bzw. negativen oder positiven Übertrag ergeben hatte. LINK nimmt das beim Verschieben rechts oder links herauslaufende Bit auf, und sein Zustand geht selbst beim Verschieben links- oder rechts-ergänzend ein. FLAG und FLAG2 sind Merkspeicher. Die Stellen 0...8 behalten nur für den Verlauf eines zusammenhängenden Mikroprogramms ihren Inhalt.

Die Stellen 9...14 beinhalten die "Adresse" der jeweiligen Programmebene (LEVEL); sie werden u.U. bei Ende eines Mikroprogramms (bzw. einer Makrooperation) neu gesetzt, was jedoch verhindert wird, solange DISABLE eine 1 enthält.

Mikroprogramme werden nur in Sonderfällen benutzt. Bei Verwendung steht ein MIKRO-ASSEMBLER zur Übersetzung der symbolischen Befehle in die Mikroprogrammierung zur Verfügung.

Im Normalfall werden nur die Maschinenbefehle (s. nächstes Kapitel) benutzt, für die ebenfalls ein Übersetzungsprogramm (ASSEMBLER) zur Verfügung steht.

Maschinenbefehle

VORBEMERKUNG

Die Maschineninstruktionen der MINCAL 513/523 Computer sind jeweils ein 19-bit-Wort lang; sie haben im allgemeinen die oktal gegliederte Form

* OPQAAA

mit der Bedeutung

*	N-Bit	Vorzeichen (- = Inhalt 1; sonst 0)
O	Bit 17, 16, 15	Befehlscode
P	Bit 14, 13, 12	
Q	Bit 11, 10, 9	Ergänzung
A	Bit 8, 7, 6	Adreßteil
A	Bit 5, 4, 3	
A	Bit 2, 1, 0	

Die Befehle sind im folgenden beschrieben; der Inhalt von je 3 bit eines Befehlswortes ist dabei zu einer Oktalziffer (0...7) zusammengefaßt.

Erläuterung: Vorzeichen ist von Bedeutung
(e) Befehl gehört zum erweiterten Befehlsvorrat
(2) Befehl nur bei MINCAL 523 möglich

NULL-, KONVERSIONS-, CODEBEFEHLE

N	17					0
-	0	0	A	A	A	A

Befehle:								
	NOP	Nulloperation	0	0	0	0	0	0
(e)	VBL	Binärumwandlung Linkskomma	0	0	1	4	0	0
(e)	VBR	" Rechtskomma	0	0	1	4	4	0
(e)	VDL	Dezimalumwandlung Linkskomma	0	0	1	5	0	0
(e)	VDR	" Rechtskomma	0	0	1	5	4	0
	COD	Code-Operation allgemein	0	0	.	.	.	.
Adresse:		kleinste (100000)	0	0	0	0	0	0
		größte (107777)	0	0	7	7	7	7

Diese Befehle rufen ein im Festspeicher liegendes Mikroprogramm auf. Für 4 Konversionsbefehle ist das Mikroprogramm vorgegeben. Ein Befehlswort mit Nullinhalt wird übersprungen.

NOP Nulloperation 0 0 0 0 0 0

Dieser Befehl wird übersprungen.

VBL Binärumwandlung Linkskomma 0 0 1 4 0 0

Der im W-Register stehende dezimale Bruch wird in einen binären Bruch umgewandelt.

VBR Binärumwandlung Rechtskomma 0 0 1 4 4 0

Die im W-Register stehende dezimale Ganzzahl wird in eine binäre Ganzzahl umgewandelt.

VDL Dezimalumwandlung Linkskomma 0 0 1 5 0 0

Der im W-Register stehende binäre Bruch wird in einen dezimalen Bruch umgewandelt.

VDR Dezimalumwandlung Rechtskomma 0 0 1 5 4 0

Die im W-Register stehende binäre Ganzzahl wird in eine dezimale Ganzzahl umgewandelt.

Bemerkung: Bei binären Brüchen hat Bit 17 den Wert 2^{-1}, Bit 0 den Wert 2^{-18}.
Bei binären Ganzzahlen hat Bit 17 den Wert 2^{17}, Bit 0 den Wert 2^{0}.
Negative binäre Brüche und Ganzzahlen werden als Zweierkomplement dargestellt.

Dezimalzahlen sind in BCD-Form dargestellt (je Dezimalstelle 4 bit mit dem Wert 8-4-2-1).

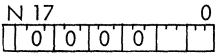
Bei dezimalen Brüchen hat Bit 17 den Wert $8 \cdot 10^{-1}$, Bit 0 den Wert $4 \cdot 10^{-5}$.
Bei dezimalen Ganzzahlen hat Bit 17 den Wert $2 \cdot 10^{4}$, Bit 0 den Wert $1 \cdot 10^{0}$.
Negative dezimale Brüche und Ganzzahlen werden wie positive dargestellt; das N-Bit hat jedoch den Inhalt 1.

Die maximal konvertierbare Ganzzahl hat den Betrag 39999.

COD Code-Operation 0 0 A A A A

Es wird ein (vom Benutzer zu erstellendes und vom Hersteller zu implementierendes Mikroprogramm aufgerufen, das bei Adresse (10AAAA+1) des Festspeichers beginnt. Nach Ausführung des Mikroprogramms wird das Programm mit der auf den COD-Befehl folgenden Instruktion fortgesetzt.

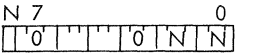
EINFACH-SCHIEBEBEFEHLE



Befehle:	SRLW	Schieben	Rechts	logisch	W		0	0	0	0	1	0
	SRLD	"	"	"	Doppelt		0	0	0	0	1	1
	SRLX	"	"	"	X		0	0	0	0	1	2
	SRAW	"	"	Arithmetisch	W		0	0	0	0	1	4
	SRAD	"	"	"		Doppelt	0	0	0	0	1	5
	SRAX	"	"	"		X	0	0	0	0	1	6
	SLLX	"	Links	Logisch	X		0	0	0	0	2	0
	SLLD	"	"	"	Doppelt		0	0	0	0	2	1
	SLLW	"	"	"	W		0	0	0	0	2	2
	SLAX	"	"	Arithmetisch	X		0	0	0	0	2	4
	SLAD	"	"	"		Doppelt	0	0	0	0	2	5
	SLAW	"	"	"		W	0	0	0	0	2	6

Diese Befehle dienen zum logischen und arithmetischen Rechts- und Linksverschieben des Inhalts von W- und X-Register bzw. beider Register gemeinsam um 1 bit. Ihre Funktion entspricht den im folgenden Abschnitt beschriebenen Mehrfach-Schiebebefehlen mit der gleichen symbolischen Bezeichnung.

ɜ) MEHRFACH-SCHIEBEBEFEHLE



efehle:											
	SRLW	Schieben Rechts Logisch W				0	1	0	0	.	.
	SRLD	" " "	Doppelt			0	1	1	0	.	.
	SRLX	" " "	X			0	1	2	0	.	.
	SRAW	" "	Arithmetisch	W		0	1	4	0	.	.
	SRAD	" " "	"	D oppelt		0	1	5	0	.	.
	SRAX	" " "	"	X		0	1	6	0	.	.
	SLLX	" Links	Logisch	X		0	2	0	0	.	.
	SLLD	" " "	"	Doppelt		0	2	1	0	.	.
	SLLW	" " "	"	W		0	2	2	0	.	.
	SLAX	" "	Arithmetisch	X		0	2	4	0	.	.
	SLAD	" " "	"	Doppelt		0	2	5	0	.	.
	SLAW	" " "	"	W		0	2	6	0	.	.
	SRR	Schiften Rechts mit Runden				0	1	3	0	.	.
	SLN	Normalisieren				0	2	3	0	0	0
Anzahl der Schiebestellen:		kleinste (0)				.	.	.	.	0	0
		größte sinnvolle (37)				.	.	.	.	4	5

Diese Befehle dienen zum logischen und arithmetischen Rechts- und Linksverschieben des Inhalts von W- und X-Register bzw. beider Register gemeinsam um eine beliebige Zahl von Binärstellen. Je ein weiterer Befehl bewirkt Rechtsschieben mit Runden sowie Normalisieren des W-Register-Inhalts.

Beim logischen Schieben werden alle 19 bit einbezogen. Herauslaufende Bits gehen verloren; frei werdende Bits bekommen Nullinhalt. Beim doppelten Schieben werden beide Register (W links, X rechts) zu einem Doppelwort verbunden; bit 0 des W-Registers und das N-Bit des X-Registers grenzen aneinander.

Beim arithmetischen Schieben bleibt der Inhalt des N-Bits erhalten; sein Inhalt setzt sich beim Rechtsschieben nach rechts fort; beim Linksschieben gehen aus Bit 17 herausgeschobene Inhalte verloren. Beim doppelten Schieben wird angenommen, daß in beiden Registern eine Doppelwort-Zweierkomplementzahl steht (Vorzeichen + 18 bit in W, restliche 18 bit in Bit 17...0 von X); Bit 0 des W-Registers und Bit 17 des X-Registers grenzen aneinander, während das N-Bit des X-Registers Nullinhalt haben soll und völlig außer Betracht bleibt.

SRLW Schieben rechts logisch W 0 1 0 0 N N

Der Inhalt des W-Registers wird um NN bit logisch nach rechts verschoben.

SRLD Schieben rechts logisch Doppelt 0 1 1 0 N N

Der Inhalt von W- und X-Register wird gemeinsam um NN bit nach rechts verschoben.

SRLX Schieben Rechts logisch X 0 1 2 0 N N

Der Inhalt des X-Registers wird um NN bit logisch nach rechts verschoben.

SRAW Schieben rechts arithmetisch W 0 1 4 0 N N

Der Inhalt des W-Registers wird um NN bit arithmetisch nach rechts verschoben.

SRAD Schieben rechts arithmetisch doppelt 0 1 5 0 N N

Der Inhalt des W- und X-Registers wird um NN bit arithmetisch nach rechts verschoben.

SRAX Schieben rechts arithmetisch X 0 1 6 0 N N

Der Inhalt des X-Registers wird um NN bit arithmetisch nach rechts verschoben.

SLLX Schieben links logisch X 0 2 0 0 N N

Der Inhalt des X-Registers wird um NN bit logisch nach links verschoben.

SLLD Schieben links logisch doppelt 0 2 1 0 N N

Der Inhalt des W- und X-Registers wird um NN bit logisch nach links verschoben.

SLLW Schieben links logisch W 0 2 2 0 N N

Der Inhalt des W-Registers wird um NN bit logisch nach links verschoben.

SLAX Schieben links arithmetisch X 0 2 4 0 N N

Der Inhalt des X-Registers wird um NN bit arithmetisch nach links verschoben.

SLAD Schieben links arithmetisch doppelt 0 2 5 0 N N

Der Inhalt des W- und X-Registers wird um NN bit arithmetisch nach links verschoben.

SLAW Schieben links arithmetisch W 0 2 6 0 N N

Der Inhalt des W-Registers wird um NN bit arithmetisch nach links verschoben.

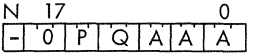
SRR Schieben Rechts mit Runden 0 1 3 0 N N

Der Inhalt des W-Registers wird um NN bit nach rechts verschoben. War das letzte rechts herausgeschobene Bit gleich 1, so wird anschließend der W-Register-Inhalt um 1 erhöht. Dieser Befehl darf nur auf positive Zahlen angewendet werden.

SLN Normalisieren 0 2 3 0 0 0

Der Inhalt des W-Registers wird normalisiert, d.h. so weit nach links verschoben, daß sie dem Betrage nach gleich oder größer als 2^{17} ist. Die Anzahl der hierfür benötigten Schiebevorgänge steht anschließend binär rechtsbündig im X-Register. War der Inhalt bereits normalisiert, so bleibt er unverändert, und in X steht eine 0. War der Inhalt gleich Null, so bleibt er es, und in X steht 18. Auf die Zahl -2^{18} (-000000) darf der Befehl nicht angewendet werden.

MANIPULATIONSBEFEHLE



Befehle:	MZR	Null setzen	. 0 3 0 . . .
	MPO	Plus Eins setzen	. 0 3 1 . . .
	MMO	Minus Eins setzen	. 0 3 2 . . .
	MIC	Inkrementieren	. 0 3 3 . . .
	MDC	Dekrementieren	. 0 3 4 . . .
	MCO	Komplementieren	. 0 3 5 . . .
	MCI	Zweierkomplement bilden	. 0 3 6 . . .
Adresse:		kleinste (000000)	. . . . 0 0 0
		größte (000777)	. . . . 7 7 7
Ebenenbindung der Adresse:			- . . . . .

Diese Befehlsgruppe verändert den Inhalt eines beliebigen Speicherplatzes in Seite 0. Der Speicherplatz kann ebenengebunden adressiert werden (zur Adresse 000AAA wird die Ebenen-Adresse 000LL0 geodert); dadurch sind die Befehle insbesondere auch auf Register anwendbar.

MZR Null setzen * 0 3 0 A A A

Der Speicherplatz erhält Nullinhalt (000000).

MPO Plus Eins setzen * 0 3 1 A A A

Der Speicherplatz erhält den Inhalt 1 (000001).

MMO Minus Eins setzen * 0 3 2 A A A

Der Speicherplatz erhält den Inhalt -1 (-777777).

MIC Inkrementieren * 0 3 3 A A A

Der Inhalt des Speicherplatzes wird um 1 erhöht. Ein eventueller Überlauf wird nicht berücksichtigt.

MDC Dekrementieren * 0 3 4 A A A

Der Inhalt des Speicherplatzes wird um 1 erniedrigt. Ein eventueller Überlauf wird nicht berücksichtigt.

MCO Komplementieren * 0 3 5 A A A

Der Inhalt des Speicherplatzes wird komplementiert (Einerkomplement).

MCI Zweierkomplement bilden * 0 3 6 A A A

Der Inhalt des Speicherplatzes wird komplementiert und dann um 1 erhöht.

REGISTERBEFEHLE

			N 17 ... 0
			0 P R A A A
Befehle:	LDR	Laden in Register	. 1 1 . . . .
	TRR	Transfer aus Register	. 1 5 . . . .
	ADR	Addieren zu Register	. 0 4 . . . .
	SBR	Subtrahieren von Register	. 0 5 . . . .
	FOR	ODER mit Register	. 0 6 . . . .
	FAR	UND mit Register	. 0 7 . . . .
Register:		U	. . . 0 . . .
		V	. . . 1 . . .
		W	. . . 2 . . .
		X	. . . 3 . . .
		N	. . . 4 . . .
		IXR1	. . . 5 . . .
		IXR2	. . . 6 . . .
		IXR3	. . . 7 . . .
Adresse:		kleinste (000000)	. . . . 0 0 0
		größte (000777)	. . . . 7 7 7
Ebenenbindung der Adresse:			- . . . . . .

LDR Laden in Register * 1 1 R A A A

Das Register R wird mit dem Inhalt der Adresse geladen.

TRR Transfer an Register * 1 5 R A A A

Der Inhalt des Registers R wird in der Adresse abgespeichert.

ADR Addieren zu Register * 0 4 R A A A

Zum Inhalt des Registers R wird der Inhalt der Adresse addiert. Ein eventueller Überlauf wird nicht berücksichtigt.

SBR Subtrahieren von Register * 0 5 R A A A

Vom Inhalt des Registers R wird der Inhalt der Adresse subtrahiert. Ein eventueller Überlauf wird nicht berücksichtigt.

FOR ODER mit Register * 0 6 R A A A

Der Inhalt des Registers R wird mit dem der Adresse R in inklusive ODER-Verknüpfung gebracht; das Ergebnis steht im Register R.

FAR UND mit Register * 0 7 R A A A

Der Inhalt des Registers R wird mit dem der Adresse R in inklusive UND-Verknüpfung gebracht; das Ergebnis steht im Register R.

KONSTANTENBEFEHLE

			N 17 ... 0
			- O P R C C C
Befehle:	LDC	Laden Konstante	. 1 0 . . . .
	ADC	Addieren Konstante	. 1 4 . . . .
Register:		U	. . . 0 . . .
		V	. . . 1 . . .
		W	. . . 2 . . .
		X	. . . 3 . . .
		N	. . . 4 . . .
		IXR1	. . . 5 . . .
		IXR2	. . . 6 . . .
		IXR3	. . . 7 . . .
Konstanten:		größte negative (-511)	- . . . 7 7 7
		Null	. . . 0 0 0
		größte positive (+511)	. . . 7 7 7

Diese Befehlsgruppe verändert den Inhalt eines der 8 Register durch eine im Befehl definierte Konstante.

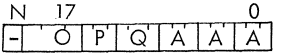
LDC Laden Konstante * 1 0 R C C C

Das angegebene Register wird mit der Konstanten geladen. Bei negativen Konstanten wird das Zweierkomplement geladen.

ADC Addieren Konstante * 1 4 R C C C

Zum Inhalt des angegebenen Registers wird die Konstante addiert. Bei negativen Konstanten wird deren Zweierkomplement addiert. Ein eventueller Überlauf wird nicht berücksichtigt.

SPEICHERBEZOGENE BEFEHLE



Befehle:	LD	Laden	.	1	2	.	.	.	.
	TR	Transfer	.	1	6	.	.	.	.
	AD	Addieren	.	2	0	.	.	.	.
	SB	Subtrahieren	.	2	2	.	.	.	.
(e)	MP	Multiplizieren	.	3	0	.	.	.	.
(e)	DV	Dividieren	.	3	2	.	.	.	.
	FO	Inklusives ODER	.	2	4	.	.	.	.
	FA	UND	.	2	6	.	.	.	.
	FE	Exklusives ODER	.	3	4	.	.	.	.
	CP	Vergleichen	.	3	6	.	.	.	.
Adressierung:	..	direkt	.	.	.	.	.	.	.
	.Y	indirekt	.	.	1	.	.	.	.
		absolut	.	.	.	0	.	.	.
		absolut ebenengebunden	-	.	.	0	.	.	.
		relativ vorwärts		.	.	4	.	.	.
		relativ rückwärts	-	.	.	4	.	.	.
		nicht indiziert	.	.	.	0	.	.	.
		indiziert über IXR1	.	.	.	1	.	.	.
		2	.	.	.	2	.	.	.
		3	.	.	.	3	.	.	.
Adreßteil:		kleinster Wert (0)	.	.	.	.	0	0	0
		größter Wert (511)	.	.	.	.	7	7	7

Diese Befehlsgruppe setzt das W-Register mit einer beliebigen Speicheradresse in Beziehung. Bei einigen Befehlen wird außer dem W-Register auch das V- oder X-Register verwendet.

Adressierungsmöglichkeiten sind:

a) absolut:	Die Adresse ist 000AAA (in Seite 0 des Speichers)
absolut ebenengebunden:	Die Adresse ist 000AAA geodert mit der Ebenenadresse 000LL0
relativ vorwärts:	Die Adresse ist n+AAA
relativ rückwärts:	Die Adresse ist n-AAA
b) indirekt:	Der Inhalt der nach a) berechneten Adresse wird als neue Adresse genommen. Ist Bit 17 des Adreßinhalts gleich 1, wird erneut substituiert.
c) indiziert:	Zu der nach a) bzw. b) berechneten Adresse wird der Inhalt des angegebenen Indexregisters addiert.

Beliebige Kombinationen a)-b)-c) sind erlaubt; sie laufen in dieser Reihenfolge zur Berechnung der effektiven Adresse ab.

LL = Nummer der laufenden Ebene; n = Programmstand (Instruktionsadresse).

LD Laden * 1 2 Q A A A

Das W-Register wird mit dem Inhalt der effektiven Adresse (Operand) geladen.

TR Transfer * 1 6 Q A A A

Der Inhalt des W-Registers wird in der effektiven Adresse abgespeichert.

AD Addieren * 2 0 Q A A A

Zum W-Register-Inhalt wird der Inhalt der effektiven Adresse (Operand) addiert. Bei positivem Überlauf wird der Inhalt des V-Registers um 1 erhöht, bei negativem Überlauf um 1 erniedrigt.

SB Subtrahieren * 2 2 Q A A A

Vom W-Register-Inhalt wird der Inhalt der effektiven Adresse (Operand) subtrahiert. Bei positivem Überlauf wird der Inhalt des V-Registers um 1 erhöht, bei negativem Überlauf um 1 erniedrigt.

MP Multiplizieren * 3 0 Q A A A

Der W-Register-Inhalt wird mit dem Inhalt der effektiven Adresse (Operand) multipliziert. Das Doppelwort-Produkt steht mit den unbedeutenden Stellen im X-Register und mit den bedeutenden Stellen im W-Register. Die Multiplikation läuft vorzeichenrichtig ab; das N-Bit des W-Registers enthält das Vorzeichen des Produkts. Das N-Bit des X-Registers bekommt stets Nullinhalt.

DV Dividieren * 3 2 Q A A A

Der W-Register-Inhalt wird durch den Inhalt der effektiven Adresse (Operand) dividiert. Der Einwort-Quotient steht im W-Register, der Rest im X-Register. Die Division läuft vorzeichenrichtig ab; das N-Bit des W-Registers enthält das Vorzeichen des Quotienten. Der Rest ist stets positiv. Bit 17 des Quotienten hat die Bedeutung 2^{-1}; daher sind nur Quotienten erlaubt, die dem Betrage nach kleiner als 1 sind; andernfalls wird das Ergebnis unrichtig.

Bemerkung: Die Befehle MP und DV verändern den Inhalt des V-Registers.

FO Inklusives ODER * 2 4 Q A A A

Der W-Register-Inhalt wird mit dem Inhalt der effektiven Adresse (Operand) in inklusive ODER-Verknüpfung gebracht; das Ergebnis steht im W-Register.

FA UND * 2 6 Q A A A

Der W-Register-Inhalt wird mit dem Inhalt der effektiven Adresse (Operand) in UND-Verknüpfung gebracht; das Ergebnis steht im W-Register.

FE Exklusives ODER * 3 4 Q Ø Ø Ø

Der W-Register-Inhalt wird mit dem Inhalt der effektiven Adresse (Operand) in exklusive ODER-Verknüpfung gebracht; das Ergebnis steht im W-Register.

CP Vergleichen * 3 6 Q Ø Ø Ø

Der Inhalt des W-Registers wird mit dem der effektiven Adresse (Operand) verglichen. Stimmen beide überein, wird die folgende Instruktion übersprungen und die übernächste ausgeführt; andernfalls die auf den CP-Befehl folgende.

VERZWEIGUNGSBEFEHLE

	N	17					0
	-	O	P	Q	A	A	A

Befehle:	BR	Verzweigen	.	4	0	.	.	.	.
	BZ	Verzweigen wenn Null	.	4	2	.	.	.	.
	BP	Verzweigen wenn Plus	.	4	4	.	.	.	.
	BM	Verzweigen wenn Minus	.	4	6	.	.	.	.
Ergänzung:	bei BR: Sensor; sonst Register	U	.	.	.	0	.	.	.
		V	.	.	.	1	.	.	.
		W	.	.	.	2	.	.	.
		X	.	.	.	3	.	.	.
		N	.	.	.	4	.	.	.
		IXR1	.	.	.	5	.	.	.
		IXR2	.	.	.	6	.	.	.
		IXR3	.	.	.	7	.	.	.
Adressierung:	..	direkt	.	.	.	.	.	.	.
	..Y	indirekt	.	.	.	1	.	.	.
		relativ vorwärts		.	.	.	.	.	.
		relativ rückwärts	-	.	.	.	.	.	.
Adreßteil:		kleinster Wert (0)	.	.	.	.	0	0	0
		größter Wert (512)	.	.	.	.	7	7	7

Diese Befehle bewirken einen Sprung im Programmablauf. Die Ausführung des Sprungs kann an eine Bedingung geknüpft sein; ist sie nicht erfüllt, so wird das Programm mit der folgenden Instruktion fortgesetzt.

Effektive Sprungadressen werden relativ berechnet (n±AAA). Bei indirekter Adressierung wird der dort gefundene Wortinhalt als Sprungadresse aufgefaßt und auf diese verzweigt (s. auch vorigen Abschnitt).

BR Verzweigen * 4 0 S A A A

Das Programm verzweigt zur effektiven Adresse. Hat S den Wert 1, 2, ...7, so wird nur dann verzweigt, wenn der entsprechende Sensor 1, 2, ... 7 erregt ist.

BZ Verzweigen wenn Null * 4 2 R A A A

Das Programm verzweigt zur effektiven Adresse, wenn das Register R Nullinhalt hat.

BP Verzweigen wenn Plus * 4 4 R A A A

Das Programm verzweigt zur effektiven Adresse, wenn das Register R positiven oder Nullinhalt hat.

BM Verzweigen wenn Minus * 4 6 R A A A

Das Programm verzweigt zur effektiven Adresse, wenn das Register R negativen Inhalt hat.

Bemerkung: Sprünge werden dadurch ausgeführt, daß das N-Register verändert wird. Jede durch irgendeinen anderen Befehl bewirkte Veränderung des N-Registers führt ebenfalls zu einem Programmsprung.

UNTER-, MIKROPROGRAMM-SPRUNGBEFEHLE

-	O	P	Q	A	A	A

Befehle:	CS	Aufruf Unterprogramm		.	5	0	.	.	.	.
	CM	Aufruf Mikroprogramm		.	5	2	.	.	.	.
Ergänzung:		Rückkehr-Adresse bei CS	in U-Register	.	.	.	0	.	.	.
			in V-Register	.	.	.	1	.	.	.
			vor Unterprogramm	.	.	.	2	.	.	.
Adressierung:	..	direkt		.	.	.	.	.	.	.
	..Y	indirekt		.	.	1	.	.	.	.
		absolut			.	.	0	.	.	.
		absolut ebenengebunden		-	.	.	0	.	.	.
		relativ vorwärts			.	.	4	.	.	.
		relativ rückwärts		-	.	.	4	.	.	.
Adreßteil:		kleinster Wert (0)		.	.	.	.	0	0	0
		größter Wert (511)		.	.	.	.	7	7	7

Diese Befehlsgruppe dient zum Aufruf eines Unterprogramms oder Mikroprogramms. Die Adressierungsarten entsprechen denen der speicherbezogenen Befehle; Indizierung ist jedoch nicht möglich.

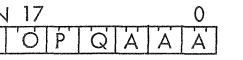
CS Aufruf Unterprogramm * 5 0 Q A A A

Das Programm springt auf die der effektiven Adresse folgende Instruktion. Die Instruktionsadresse des CS-Befehls (Rückkehradresse) wird - je nach Angabe in der Ergänzung Q - entweder in einem der Register U oder V oder in der effektiven Adresse (vor Beginn des Unterprogramms) abgelegt. Der Rücksprung ins Hauptprogramm geschieht im ersten Falle durch Übertragung des Register-Inhalts nach N (TRRU N bzw. TRRV N), im zweiten Falle durch einen indirekten Sprung über den Unterprogramm-Anfang (BRY...).

CM Aufruf Mikroprogramm * 5 2 Q A A A

Ein bei der effektiven Adresse beginnendes Mikroprogramm wird aufgerufen. Nach Ausführung des Mikroprogramms wird das Programm mit der auf den CM-Befehl folgenden Instruktion fortgesetzt.
Mikroprogramme können jeweils in den ersten 4 k Worten des Kernspeichers (Adressen 000000...000777) oder des Festspeichers (100000...107777) liegen. Der Inhalt bei den rechten Bits in Q ist ohne Bedeutung, kann dem Aufruf jedoch als Parameter mitgegeben werden.

STEUERBEFEHLE



Befehle:	HLT	Halt	5	4	0	0	0	0
	HSL	Halt, Start Ebene	5	4	2	.	.	0
	HBR	Halt mit Verzweigen	5	4	4	.	.	.
	STL	Start Ebene	5	5	0	.	.	0
	ECL	Unterbrechung zulassen	5	5	3	0	0	0
	DCL	Unterbrechung verhindern	5	5	4	0	0	0
Ebene:	(bei HSL, STL):	niedrigste (00)	.	.	.	0	0	.
		höchste (77)	.	.	.	7	7	.
Adressierung:	(bei HBR):	relativ vorwärts	.	.	.	.	.	.
		relativ rückwärts	-	.	.	.	.	.
Adreßteil:	(bei HBR):	kleinster Wert (0)	.	.	.	0	0	0
		größter Wert (511)	.	.	.	7	7	7

Diese Befehle bewirken Anhalten des Programms (ohne und mit Verzweigung danach), internen Start einer beliebigen Programmebene sowie die Steuerung der Unterbrechbarkeit.

HLT Halt 5 4 0 0 0 0

Das Programm hält an, indem die laufende Programmebene ausgeschaltet wird. Ein Start dieser Ebene setzt den Programmablauf mit der folgenden Instruktion fort.

HSL Halt, Start Ebene 5 4 2 L L 0

Das Programm hält an, indem die laufende Programmebene ausgeschaltet wird; gleichzeitig wird die Programmebene LL gestartet.
Die letzte Oktalstelle der Instruktion ist ohne Bedeutung; sie kann statt 0 auch eine der Ziffern 1...7 enthalten.

HBR Halt mit Verzweigen * 5 4 4 A A A

Das Programm hält an, indem die laufende Programmebene ausgeschaltet wird. Ein Start dieser Ebene setzt den Programmablauf an einer Stelle fort, die durch (relativ berechnete) effektive Adresse bestimmt ist.

STL Start Ebene 5 5 0 L L 0

Die Programmebene LL wird gestartet.
Die letzte Oktalstelle der Instruktion ist ohne Bedeutung; sie kann statt 0 auch eine der Ziffern 1...7 enthalten.

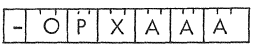
(2) ECL Unterbrechung zulassen 5 5 3 0 0 0

Durch diesen Befehl wird der DISABLE-Zustand wieder aufgehoben; die laufende Programmebene kann durch jede höhere unterbrochen werden.

(2) DCL Unterbrechung verhindern 5 5 4 0 0 0

Dieser Befehl stellt den DISABLE-Zustand her, in dem die laufende Programmebene nicht durch den Start einer anderen Ebene unterbrochen werden kann.
Spätestens vor einem Halt (HLT, HSL, HBR) muß der DISABLE-Zustand durch ECL wieder aufgehoben werden, damit der Halt wirksam wird.

EIN/AUSGABE-BEFEHLE



Befehle:	GB,GX	Übernahme	. 6 0 . . . .
	FB, FX	Übernahme mit ODER	. 6 1 . . . .
	IBG	Eingabe, Übernahme	. 6 2 . . . .
	IBF	Eingabe, Übernahme mit ODER	. 6 3 . . . .
	IB	Eingabe	. 6 4 . . . .
	OB,OX	Ausgabe	. 6 5 . . . .
	IBH	Eingabe, Halt	. 6 6 . . . .
	OBH	Ausgabe, Halt	. 6 7 . . . .
Ebenen-Bindung:	.X	nein	. . . . .
	.B.	ja	- . . . . .
Indizierung:	keine		. . . 0 . . .
	über IXR1		. . . 1 . . .
	2		. . . 2 . . .
	3		. . . 3 . . .
Adreßteil:	kleinste (000) }	externe	. . . . 0 0 0
	größte (777) }	Adresse	. . . . 7 7 7

Diese Befehlsgruppe steuert wortweise die Ein- und Ausgabe über das W-Register den programmgesteuerten Datankanal (X-Kanal) des Computers. Die externe Adresse 00AAA ist gleich dem Adreßteil AAA der Instruktion; gegebenenfalls wird dazu der Inhalt des angegebenen Indexregisters addiert. Die externe Adresse kann in der peripheren Hardware mit der laufenden Programmebene verknüpft sein; dann sind die entsprechenden Befehle zu benutzen (Minuszeichen gesetzt). Dies empfiehlt sich insbesondere bei Verwendung externer Register (Interfaces), die den Datentransfer zur eigentlichen Peripherie übernehmen; nach Ende des - beliebig langen - Transfervorgangs erhält die jeweilige Programmebene ein Startsignal.

GB Übernahme - 60XAAA
GX Übernahme (nicht ebenengebunden) 60XAAA

Der Inhalt der externen Adresse wird in das W-Register übernommen.

FB Übernahme mit ODER - 61XAAA
FX Übernahme mit ODER (nicht ebenengebunden) 61XAAA

Der Inhalt der externen Adresse wird mit dem des W-Registers geodert und das Ergebnis in das W-Register übernommen.

IBG Eingabe, Übernahme - 62XAAA

Das durch die externe Adresse bestimmte Interface wird in Eingabebereitschaft versetzt und das Programm durch Ausschalten der laufenden Ebene angehalten. Nach Eintreffen des Datenworts von außen startet das Interface die Ebene wieder; das Datenwort wird in das W-Register übernommen, und das Programm läuft weiter.

IBF Eingabe, Übernahme mit ODER - 63XAAA

Ablauf wie bei IBG; jedoch werden W-Register-Inhalt und externes Datenwort geodert und das Ergebnis ins W-Register übernommen.

IB Eingabe - 64XAAA

Das durch die externe Adresse bestimmte Interface wird in Eingabebereitschaft versetzt. Das Programm läuft weiter.

OB Ausgabe - 65XAAA
OX Ausgabe (nicht ebenengebunden)

Der Inhalt des W-Registers wird in die externe Adresse übertragen. Bei OB handelt es sich um ein Interface, welches das Datenwort übernimmt und mit dem Befehl einen Ausgabevorgang beginnt.
Das Programm läuft weiter.

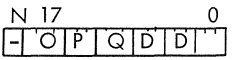
IBH Eingabe, Halt - 6 6 X A A A

Ablauf wie bei IB; jedoch wird das Programm angehalten. Mit dem nächsten Start der Ebene - z.B. durch das Interface nach Beendigung des Eingabevorgangs - läuft das Programm weiter.

OBH Ausgabe, Halt - 6 7 X A A A

Ablauf wie bei OB; jedoch wird das Programm angehalten. Mit dem nächsten Start der Ebene - z.B. durch das Interface nach Beendigung des Ausgabevorgangs - läuft das Programm weiter.

INTERFACE-STEUERBEFEHLE



Befehle:	RBL	Rücksetzen LOCK	- 7 0 0 . . 0
	SBL	Setzen LOCK	- 7 0 0 . . 1
	RBR	Rücksetzen READY	- 7 1 0 . . 0
	SKB	Sprung wenn BUSY	- 7 2 0 . . 0
	SKR	Sprung wenn READY	- 7 3 0 . . 0
Gerätenummer:		kleinste (000)	. . . . 0 0 .
		größte (770)	. . . . 7 7 .

Diese Befehle steuern Status-Flipflops in Ein/Ausgabe-Interfaces oder fragen deren Zustand ab. Die Flipflops haben folgende Bedeutung:

BUSY Wird durch die Befehle IB... bzw. OB... gesetzt und bei beendetem Ein- oder Ausgabevorgang selbsttätig rückgesetzt. Zeigt an, daß ein Ein- oder Ausgabevorgang läuft.

READY Wird bei beendetem Ein- oder Ausgabevorgang selbsttätig gesetzt und startet dadurch die zugehörige Programmebene. Es wird zu Beginn der IB...- und OB...-Befehl rückgesetzt. Wichtig: Nach IB, OB IBH und OBH (da im Interface, sobald der Ein/Ausgabevorgang zu Ende ist, READY gesetzt wird) das Flipflop mit RBR rücksetzen, bevor das Programm auf HLT, HBR oder HSL läuft; sonst ist der Halt unwirksam.

LOCK Verhindert, wenn gesetzt, den Start der Ebene durch READY. Dieses bleibt jedoch unbeeinflußt; nach Rücksetzen von LOCK wird der Start wirksam, wenn READY gesetzt ist.

Interfaces dieser Art haben eine oktale Gerätenummer DD, die der externen Adresse 00DD0 entspricht.

BL Rücksetzen LOCK - 7 0 0 D D 0

Das LOCK-Flipflop im Interface DD wird ausgeschaltet.

BL Setzen LOCK - 7 0 0 D D 1

Das LOCK-Flipflop im Interface DD wird eingeschaltet.

BR Rücksetzen READY - 7 1 0 D D 0

Das READY-Flipflop im Interface DD wird ausgeschaltet.

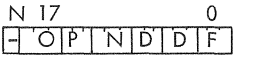
KB Sprung wenn BUSY - 7 2 0 D D 0

Die folgende Instruktion wird übersprungen und die übernächste ausgeführt, wenn BUSY im Interface DD gesetzt ist.

KR Sprung wenn READY - 7 3 0 D D 0

Die folgende Instruktion wird übersprungen und die übernächste ausgeführt, wenn READY im Interface DD gesetzt ist.

ORMATISIERTE EIN/AUSGABE-BEFEHLE



efehle:	IBS	Eingabe Seriell	- 7 4 . . . .
	OBS	Ausgabe Seriell	- 7 5 . . . .
nzahl:	1 Zeichen		. . . 1 . . .
	2 "		. . . 2 . . .
	3 "		. . . 3 . . .
	4 "		. . . 4 . . .
	5 "		. . . 5 . . .
	6 "		. . . 6 . . .
erätenummer:		kleinste (00)	. . . . 0 0 .
		größte (77)	. . . . 7 7 .
ormat:	O	(3 bit/Zeichen)	 2
	D	(4 " ")	 3
	F	(5 " ")	 4
	A	(6 " ")	 5
	S	(7 " ")	 6
	E	(8 " ")	 7

Diese Befehle steuern die serielle (zeichenweise) Ein/Ausgabe von Teilen eines Wortes. Die Länge der Wortteile kann von 3 bis 8 bit gewählt werden; ebenso die Anzahl, die ein- oder ausgegeben wird (1 bis 6 Zeichen).

Nach einer Eingabe stehen die Teilworte von links nach rechts entsprechend ihrer zeitlichen Reihenfolge im W-Register (das zuletzt eingegebene Zeichen rechtsbündig); die restlichen Bits haben Nullinhalt. Vor einer Ausgabe muß das erste Zeichen linksbündig (einschließlich Bit 17!) im W-Register stehen; die danach auszugebenden schließen sich nach rechts an.

Das N-Bit des W-Registers bleibt außer Betracht.

Die zugehörigen Ein/Ausgabe-Interfaces haben eine Gerätenummer DD, die der externen Adresse 00DD0 entspricht. In ihnen wird jedes Zeichen u.U. besonders verschlüsselt (z.B. im ASCII-Code).

Die Befehle IBS und OBS entsprechen einer Folge von IBG- bzw. OBH-Befehlen; während der Zeichen-Ein/Ausgabe über das Interface wird die Programmebene ausgeschaltet.

IBS Eingabe Seriell - 7 4 N D D F

N Zeichen werden im Format F über das Interface DD in das W-Register eingegeben.

OBS Ausgabe Seriell - 7 5 N D D F

N Zeichen werden im Format F über das Interface DD aus dem W-Register ausgegeben.

Achtung: Durch diese Befehle wird der Inhalt des V- und X-Registers verändert.

(e) BLOCK-EIN/AUSGABE-BEFEHLE

N 17

-	O	P	Ø	D	D	0

Befehle:	IBB	Eingabe Block	- 7 6 0 . . 0
	OBB	Ausgabe Block	- 7 7 0 . . 0
Gerätenummer:		kleinste (00)	. . . . 0 0 .
		größte (77)	. . . . 7 7 .

Diese Befehle dienen zur Eingabe eines Blocks von Daten in ein Speicherfeld bzw. zur Ausgabe des Speicherfeld-Inhalts. Die beiden auf den Befehl folgenden Worte müssen folgende Angaben enthalten:

0AAAAA Speicherfeld-Basisadresse
0BBBBB Blocklänge (binär, in Worten)

Je Ein/Ausgabevorgang wird ein Wort transferiert; das erste Wort gehört zur Basisadresse des Speicherworts, die folgenden zu aufsteigenden Adressen des Feldes.

Zugehörige Ein/Ausgabe-Interfaces haben eine Geräte-Nummer DD, die der externen Adresse 00DD0 entspricht. Die letzte Oktalstelle der Instruktion kann statt 0 auch eine der Ziffern 1...7 enthalten (zu einer genaueren Adreß- oder Formatspezifikation).

Während des Ein/Ausgabevorgangs über das Interface wird die Programmebene ausgeschaltet.

Achtung: Durch diese Befehle wird der Inhalt des V- und X-Registers verändert. Das W-Register bleibt unberührt.

IBB Eingabe Block

```
- 7 6 0 D D 0
  0 A A A A A
  0 B B B B B
```

Ein Block der Länge BBBBB wird über das Interface DD in ein Speicherfeld ab Adresse AAAAA eingegeben.

OBB Ausgabe Block

```
- 7 7 0 D D 0
  0 A A A A A
  0 B B B B B
```

Ein Block der Länge BBBBB wird über das Interface DD aus einem Speicherfeld ab Adresse AAAAA ausgegeben.

Bedienung

Die Frontplatte der MINCAL 500-Computer ist als Bedienungsfeld mit Anzeige- und Bedienungselementen ausgerüstet. Das Bedienungsfeld wird benutzt, wenn Programme getestet, Funktionen und Abläufe schnell überprüft oder Daten manuell in den Kernspeicher eingegeben werden sollen.

Beim festprogrammierten Rechner MINCAL 513 kann u.U. die Bedienungs-Frontplatte durch eine Blindplatte ersetzt werden.

Das Bedienungsfeld enthält im einzelnen folgende Elemente:

- ein 20-bit-Lampenfeld als zentrale Datenanzeige (18 bit + Vorzeichen + Parity)
- ein 20-bit-Tastenfeld zur Informationseingabe in angewählte Kernspeicherplätze. CLR dient zum Löschen von Kernspeicherinhalten, NP zum Löschen einer Parity-Fehlermeldung
- ein 16-bit-Adreß-Schalterfeld zur Vorwahl einer beliebigen Festprogramm- oder Kernspeicheradresse.
  Das linke, 15. bit entscheidet zwischen Kernspeicher- (Schalter nicht eingelegt) und Festprogrammadressen (Schalter eingelegt).
- je eine Taste
  RES zur Nullstellung aller Hardware-Register in der Recheneinheit
  GO zur Fortschaltung des Programms im Einzelschrittbetrieb (in Verbindung mit CYC, INS NOP oder ADR).
- die Stopschalter
  CYC stoppt nach jeder Mikrooperation
  INS stoppt nach jeder Makroinstruktion
  ADR stoppt bei der vorgewählten Instruktionsadresse
- 7 SENSOR-Schalter; diese liegen parallel zu den externen Sensoreingängen.
  INT-Schalter; er macht - wenn eingelegt - die externen Sensoreingänge unwirksam.
- Die Taste STA startet das Programm auf Ebene 0.
  HLT schaltet die laufende Programmebene aus.
  END setzt den Instruktionszähler (N-Register) auf die Kernspeicher-Anfangsadresse 10200_8.
  MEM bewirkt Lesen und Anzeigen des Speicherinhaltes der angewählten Adresse bzw. Neueinschreiben der eingestellten Information in die angewählte Adresse.
  REG entspricht MEM für die angewählte Registeradresse der laufenden Ebene (Zu der angewählten Adresse wird die jeweils laufende Programmebene hinzugeodert.

- Der Taster A bewirkt die Anzeige des Adreßregister-Inhalts
  B " " " der letzten ausgeführten Mikroinstruktion
  C " " " des Mikroprogrammzähler-Inhalts (nächster Mikroschritt)
  D " " " " Speicherpuffer-Inhalts
  E " " " " Statusregister-Inhaltes
  F " " " " Startspeicher-Inhalts der Ebene 0...7

- Die Lampe PRG zeigt an: Programm läuft
  IPT " ": Interrupt; eine höhere Programmebene als 0 läuft
  PSE " ": Pause; bei Stop oder langsamer Ein/Ausgabe
  DPL " ": Display; Anzeigezyklus läuft
  ACC " ": Access; ein direkter Speicherzugriff (der Peripherie) läuft.
  CK1 " ": Parityfehler vom Kernspeicher
  CK2 " ": Parityfehler von der Peripherie
  CK3 " ": Netzausfall nach Netzwiederkehr

Die Frontplatte der Stromversorgung enthält den Netzschalter (Wippschalter), der aufleuchtet, wenn der Rechner eingeschaltet ist.

Programmtest

Zum Überprüfen des Programmverlaufs sind verschiedene Testmöglichkeiten eingebaut, die unter Zuhilfenahme der Stopschalter benutzt werden können.

Ist der gelbe Schalter INS nach unten geschaltet, so führt das Programm bei jeder Betätigung der Taste GO einen Befehl (Makrobefehl) aus und hält bei Beginn der nächsten Operation an. Die neue Instruktion ist bereits gelesen und auf C-, E- und A-Register verteilt. Diese Registerinhalte kann man so lange im 20-bit-Lampenfeld sichtbar machen, wie man die entsprechenden Tasten betätigt.

Der Operationsteil steht dann in den Bits 3...8 des C-Registers, die Befehlsergänzung in den Bits 0...2 des E-Registers und der Adreßteil in den Bits 0...8 des A-Registers. Im D-Register ist noch die komplette Instruktion zu sehen. Im B-Register steht die gerade ausgeführte Mikroinstruktion (Bit 17, 16, 11 und 10).

Außer den Registerinhalten können beliebige Speicheradressen angezeigt werden. Hierzu ist es erforderlich, die gewünschte Adresse mit dem 16-stelligen Adreßschalter vorzuwählen (Schalter nach unten - Adreßbit vorgegeben). Außerdem muß die Taste MEM betätigt werden. Für diese Zeit ist dann der Inhalt dieser Adresse im Lampenfeld links oben sichtbar. Außerdem leuchtet die gelbe Lampe DPL auf.

Will man den Inhalt der niveaugebundenen Kernspeicher-Register sehen, so genügt es, mit den Adreßschaltern die oktale "Einer-Adresse" (3 Adreßschalter am rechten Ende des Adreßschaltersatzes) vorzuwählen und die Taste REG zu betätigen. In diesem Falle werden die Registerinhalte des gerade in Ausführung stehenden Niveaus angezeigt.

Ist der Schalter CYC nach unten geschaltet, so führt das Programm nach jeder Betätigung der Taste GO einen Mikroprogrammschritt durch und hält nach Beendigung dieses Schrittes an.

Ebenso wie die Stops durch den Schalter INS können alle Register (A, B, C, D, E, F) und alle Speicher-Zellen auf die gleiche Weise angezeigt werden.

In den Registern A, D und E ist dann der jeweils vom Mikroprogramm bestimmte Inhalt anzeigbar. Die gerade ausgeführte Mikroinstruktion ist im B-Register, die Adresse der folgenden Mikroinstruktion im C-Register enthalten.

Durch Betätigen der Taste ADR (Schalterstellung nach oben) ist es möglich, den Rechner dann anzuhalten, wenn die Instruktionsadresse und die im Adreßschaltersatz eingestellte Adresse übereinstimmen. Bei der eingestellten Instruktion hält der Rechner an, als sei der Schalter INS betätigt.

Alle Register und alle Speicher-Zellen können auch nach diesem Stop angezeigt werden.

Wenn der Rechner durch einen der Stop-Schalter angehalten worden ist, leuchtet die gelbe Lampe PSE im Lampenfeld oben rechts auf. Werden die Stop-Schalter wieder in die Mittelstellung gebracht, läuft das Programm nach Betätigung der Taste GO normal weiter.

Ändern von Kernspeicherplätzen

Will man den Inhalt von Kernspeicherplätzen ändern, so muß man den Inhalt der entsprechenden Adressen anzeigen (Vorwahl der Adresse und Betätigen von MEM oder REG). Das ist möglich, nachdem der Rechner durch einen Stop angehalten worden ist oder wenn alle Niveaus ausgeschaltet sind.

Während die Tasten MEM oder REG betätigt sind, muß man nun die Taste CLR (mittleres Tastenfeld links) nach oben schalten. Dadurch wird der Inhalt dieser Kernspeicher-Zelle Null gesetzt. Nun kann man neue Bits in diese Zelle übergeben, indem man die entsprechenden Tasten (mittleres Tastenfeld links) nach unten schaltet. Das eingegebene Bit ist sofort im Lampenfeld sichtbar. Während die Tasten MEM oder REG betätigt sind, können so beliebige Inhalte in alle Kernspeicher-Zellen eingegeben werden.

Will man die Kernspeicheradressen ändern, muß man für die Zeit der Adreßumschaltung die Tasten MEM oder REG loslassen. Erst wenn die neue Adresse eingestellt ist, kann der Inhalt auf die oben beschriebene Weise geändert werden.

Umschalten der Adressen bei betätigten Tasten MEM oder REG zerstört mehrere Kernspeicher-Zellen!

Durch eine solche Eingabe kann man z.B. auch kleine Testprogramme in den Kernspeicher geben. Dann ist es allerdings erforderlich, dem Instruktionszähler des entsprechenden Niveaus einen Inhalt zu geben, der um 1 kleiner ist als die erste Instruktionsadresse (Eingabe auf die oben beschriebene Weise).

Programmierbetrieb

Beim MINCAL 523 läßt sich über die SENSOR-Schalter der Frontplatte ein Testprogramm zum Eingeben, Ändern und Ausgeben von Speicherzellen aufrufen. Als Option kann dieses Testprogramm auch beim MINCAL 513 eingebaut werden.

Bedienung des Programmierbetriebs

Zu Beginn und bei Wechsel der Betriebsart stets Taste RESET und END betätigen.
Wenn externe Sensoreingänge angeschlossen, Schalter INT einlegen.

Streifenvorlauf, wenn erwünscht, so herstellen: Fernschreiber auf LOCAL schalten (sonst stets auf LINE); Locher einschalten, mehrfach NUL lochen (= ASCII-Code 000_8), dann mehrfach DEL lochen (= RUBOUT = ASCII-Code 377_8).

EINZEL-AUSGABE auf Fernschreiber

SENSOR 1 ein. START betätigen.
Adresse 6-stellig oktal eingeben. Inhalt wird im OKTAL-Format ausgedruckt und CR, LF wird ausgegeben. Danach wieder eine Adresse eingeben, usw.

EINZEL-EINGABE über Fernschreiber

SENSOR 1 + 3 ein. START betätigen.
Adresse 6-stellig oktal sowie unmittelbar danach Inhalt (im OKTAL-Format) eingeben. CR, LF werden ausgegeben. Danach wieder Eingabe von Adresse und Inhalt, usw.

GESAMT-AUSGABE (OKTAL-Format) auf Fernschreiber

SENSOR 1 + 2 ein. START betätigen.
Erste und letzte Adresse des auszugebenden Speicherbereichs je 6-stellig oktal eingeben. CR, LF werden ausgegeben.

Locher einschalten, wenn Streifen gestanzt werden soll. START betätigen. Speicherinhalt wird Wort für Wort im OKTAL-Format gedruckt und ggfs. gelocht; nach jedem achten Wort (1er-Adresse = 7) werden CR, LF ausgegeben. Vorgang wird nach Ausgabe der letzten Adresse beendet.

GESAMT-EINGABE (OKTAL-Format) über Fernschreiber

SENSOR 1 + 2 + 3 ein. Wenn Streifen-Eingabe: Im OKTAL-Format gelochten Streifen einlegen (im Zufuhrbereich oder auf 1. Zeichen), START betätigen.
Erste und letzte Adresse des einzulesenden Speicherbereichs je 6-stellig oktal eingeben. CR, LF werden ausgegeben.

START betätigen. Wenn Streifen-Eingabe: Leser einschalten; wenn manuelle Eingabe: Wort für Wort im OKTAL-Format sowie nach jedem achten Wort (1er-Adresse = 7) CR, LF eingeben. Lochkombination 021_8 (entsprechend X-ON) wird überlesen. Vorgang wird nach Eingabe in die letzte Adresse beendet. Leser jetzt ggfs. ausschalten.

Sollen die eingegebenen bzw. eingelesenen Werte gleichzeitig mitgedruckt werden, muß zusätzlich SENSOR 6 betätigt werden.

GESAMT-AUSGABE (ALPHA-Format) auf Fernschreiber

SENSOR 1 + 2 + 4 ein. START betätigen.

Erste und letzte Adresse des auszugebenden Speicherbereichs je 6-stellig oktal eingeben. Locher einschalten. START betätigen. Speicherinhalt wird Wort für Wort im ALPHA-Format gestanzt. (Gleichzeitig werden die Werte ausgedruckt. Sie sind aber nicht als Protokoll zu gebrauchen, da 1. unleserliches ALPHA-Format gedruckt wird und 2. CR, LF fehlen, so daß nach einer Zeile der Druckvorgang beendet wird). Vorgang wird nach Ausgabe der letzten Adresse beendet.

GESAMT-EINGABE (ALPHA-Format) über Fernschreiber

SENSOR 1 + 2 + 3 + 4 ein. Im ALPHA-Format gelochten Streifen einlegen (im Zufuhrbereich oder auf 1. Zeichen), START betätigen.

Erste und letzte Adresse des einzulesenden Speicherbereichs je 6-stellig oktal eingeben. CR, LF werden ausgegeben. START betätigen. Leser einschalten. Vorgang wird nach Eingabe in die letzte Adresse beendet. Leser ausschalten.

GESAMT-EIN/AUSGABE über schnelle Lochstreifeneinheit

Zusätzlich SENSOR 5 ein.

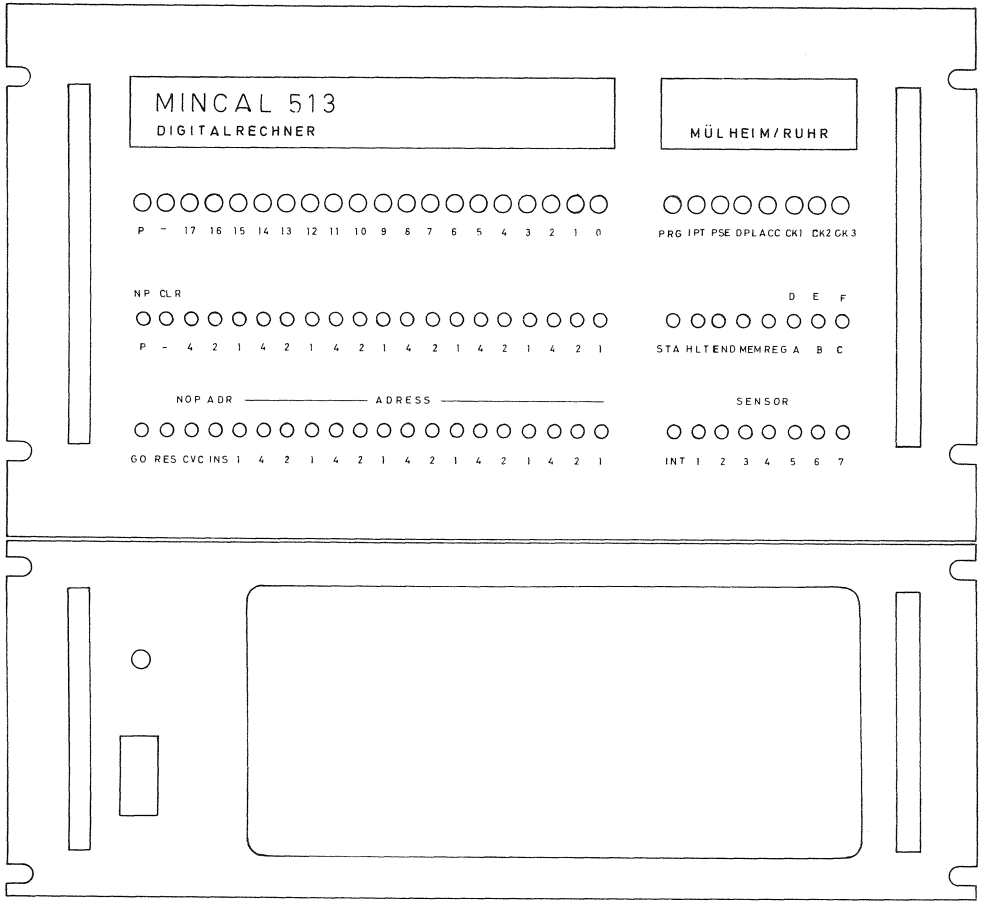
Bedienung wie bei GESAMTEINGABE (OKTAL- oder ALPHA-Format) oder bei GESAMT-AUSGABE (OKTAL- oder ALPHA-Format). START betätigen. Erste und letzte Adresse werden über Fernschreiber je 6-stellig oktal eingegeben. CR, LF werden ausgegeben. Lochstreifen einlegen und Leser einschalten bzw. Locher einschalten und Streifenvorlauf (erst TRANS und dann ZUF betätigen) herstellen. START betätigen. Einlese- bzw. Ausgabevorgang wird nach Erreichen der Endadresse beendet. Leser ausschalten bzw. Streifenende (Taste TRANS betätigen) herstellen und Locher ausschalten.

Bemerkung: Gilt für normalen Anschluß des 8-Kanal-Fernschreibers über X-Kanal und Standard-ASCII-Interface (und normalen Anschluß von schnellem Leser und Locher).

OKTAL-Format bedeutet: Vorzeichen (Leerschritt oder Minuszeichen) und 6 Oktalziffern.

ALPHA-Format " : Vorzeichen (@ entsprechend + oder A entsprechend -) und 3 Alpha-Stellen

Sensor	1	2	3	4	5	6	7	
	x							Einzel-Ausgabe oktal Fernschreiber
	x		x					Einzel-Eingabe oktal Fernschreiber
	x	x						Gesamt-Ausgabe oktal Fernschreiber
	x	x	x					Gesamt-Eingabe oktal Fernschreiber
	x	x	x			x		Gesamt-Eingabe oktal Fernschreiber mit Protokoll
	x	x		x				Gesamt-Ausgabe alpha Fernschreiber
	x	x	x	x				Gesamt-Eingabe alpha Fernschreiber
	x	x			x			Gesamt-Ausgabe oktal schneller Locher
	x	x	x		x			Gesamt-Eingabe oktal schneller Leser
	x	x		x	x			Gesamt-Ausgabe alpha schneller Locher
	x	x	x	x	x			Gesamt-Eingabe alpha schneller Leser


MINCAL 513
DIGITALRECHNER
MÜLHEIM/RUHR
P - 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
PRG IPT PSE DPL ACC CK1 CK2 CK3
NP CLR
D E F
P - 4 2 1 4 2 1 4 2 1 4 2 1 4 2 1 4 2 1
STA HLT END MEM REG A B C
NOP ADR
ADRESS
SENSOR
GO RES CYC INS 1 4 2 1 4 2 1 4 2 1 4 2 1 4 2 1
INT 1 2 3 4 5 6 7

Aufbau

Die Standardausrüstung der MINCAL 500-Rechner besteht aus zwei 19"-Einschüben:

Stromversorgung	19"-Einschub D
Rechnerrahmen	
Speicherrahmen	19"-Einschub F
Frontplatte	

Beide Einschübe sind miteinander verschraubt und an zwei Teleskopschienen befestigt. Somit können beide bequem aus dem Schrank herausgezogen werden.

Rechnerrahmen, Speicherrahmen und Frontplatte des oberen Einschubs sind durch Scharniere verbunden und können auseinandergeklappt werden.

Wie die Zeichnung zeigt, können die einzelnen Bauteile leicht erreicht werden.

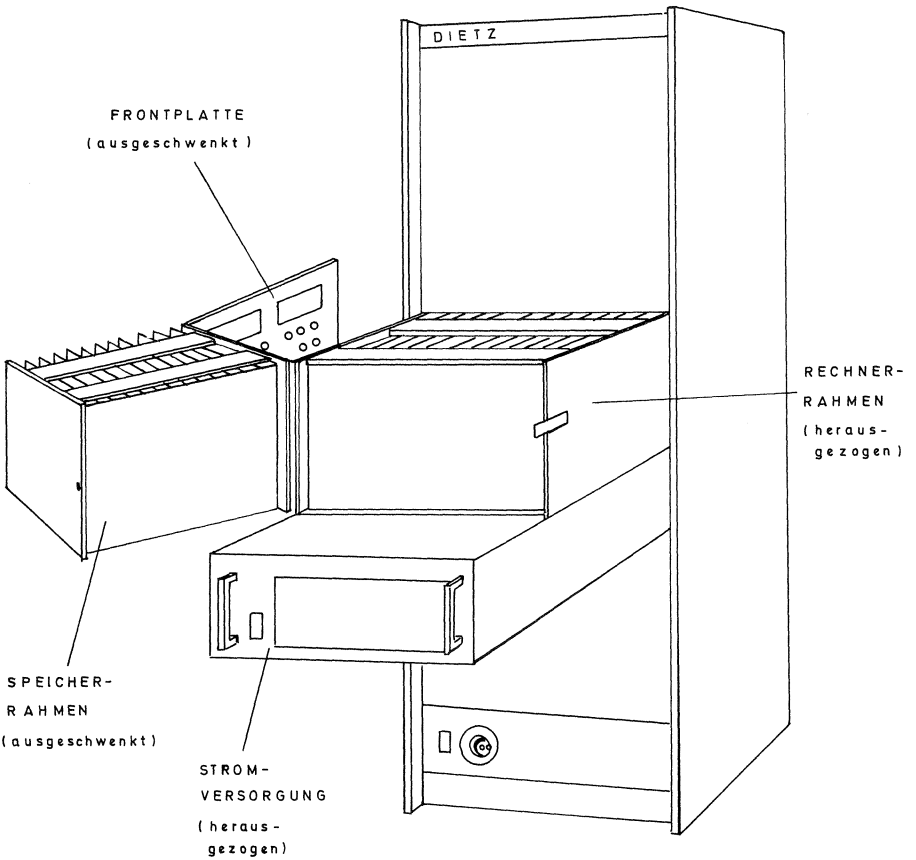
Die Frontplatte der Stromversorgung enthält den Netzschalter (Wippschalter), der aufleuchtet, wenn der Rechner eingeschaltet ist. Auf der Rückseite ist der Anschluß für die Netzspannung.

Durch Entriegeln des Schnellverschlusses unter dem rechten Griff der Rechner-Frontplatte kann diese aufgeklappt werden. Somit sind die Bausteine der Speichereinheit (Festwert- und Kernspeicher) von vorne zugänglich.

Ist der gesamte Einschub aus dem Schrank herausgezogen, so kann mit einem Schraubenzieher ein weiterer Schnellverschluß, der sich rechts an der Seitenwand des Speicherrahmens befindet, gelöst werden. Danach läßt sich der gesamte Speicherrahmen nach links schwenken, so daß die Verdrahtung von Speicher- und Rechnerrahmen zugänglich ist.

Der Rechnerrahmen enthält alle Baugruppen des Rechners außer Fest- und Kernspeicher. Auch ist im Rechnerrahmen noch Platz für einige Erweiterungen bzw. Interfaces (Optionen).

Die Adapter können erst nach Lösen der Befestigungsschrauben entfernt werden.


DIETZ
FRONTPLATTE
(ausgeschwenkt)
RECHNER-
RAHMEN
(heraus-
gezogen)
SPEICHER-
RAHMEN
(ausgeschwenkt)
STROM-
VERSORGUNG
(heraus-
gezogen)

Assembler

VORBEMERKUNG

Der Assembler/Editor MINCASS 500 ist ein Programm zur Übersetzung von symbolischen Programmen in die Maschinensprache der Rechner-Baureihe MINCAL 500.

Der Assembler wird dem Benutzer in Form eines Lochstreifens zur Verfügung gestellt. Der Lochstreifen wird in den Kernspeicher eingelesen; danach ist die Anlage zur Programmumwandlung bereit. Die Umwandlung erfolgt in 2 Läufen:

A-Lauf: Eingabe des symbolischen Programms (manuell, über Lochstreifen oder aus Editor-Bereich im Kernspeicher) mit Prüfung auf formale Fehler, Aufbau der Markenliste sowie ggfs. manueller Korrektur und ggfs. Ausgabe des symbolischen Programms als Lochstreifen oder in Editor-Bereich

C-Lauf: Erneute Eingabe des symbolischen Programms (wie oben) und Ausgabe in umgewandelter Form (Maschinencode) als Lochstreifen bzw. Drucken des Programms in Form einer Liste.

Weiterhin kann der Assembler den C-Lauf ohne Ausgabe durchführen, wobei nur auf Fehler geprüft wird. Außerdem läßt sich die Markenliste löschen, prüfen, verändern und ausdrucken.

Rechner-Ausstattung:	MINCAL 513/523 Grund + erweiterter Befehlsvorrat Programmierhilfe X00 (empfohlen) Kernspeicher 4k oder größer Konsol-Fernschreiber (8-Kanal) an Ebene 0 Schnelle Lochstreifenausrüstung (8-Kanal-Leser/Locher) an Ebene 0 (empfohlen)
Speicher-Belegung:	0000...6041
Startadresse:	Platz 1000
Anweisungsliste:	Enthält für 105 Anweisungstypen Raum, davon 99 fest installiert und 6 zur Verfügung des Benutzers
Merkmalsliste:	Enthält Raum für 500 Merkmale
Editor-Bereich:	ca. 1000 Worte lang (bei 4k Kernspeicher); entspricht ca. 200 symbolischen Anweisungen.

Der Lochstreifen MINCASS 500 wird im Zufuhrbereich in den schnellen Leser eingelegt.

An der Rechner-Frontplatte werden die Tasten RES und END betätigt. Sensor-Schalter INT, 1, 2, 3, 4 und 5 einlegen (für eingebaute Programmierhilfe X00: Gesamt-Eingabe in Alpha-Format über schnellen Leser). Taste STA betätigen und über Fernschreiber die Adressen 000100 und 004033 eingeben.

Taste STA erneut betätigen. Der Assembler wird eingelesen. Danach Tasten RES und END drücken. Sensor-Schalter 1 bis 5 wieder zurücksetzen. Die Umwandlung kann beginnen.

Betriebsarten

Mit Betätigen der Taste STA wird der Assembler gestartet. Von diesem Augenblick an erfolgt die Bedienung im Dialogverkehr über den Konsol-Fernschreiber: Nach dem Ausdruck der Programmbezeichnung (MINCASS 500) fragt das Programm, welche Betriebsart infrage kommt (Ausdruck auf Fernschreiber):

R-A	(A-Lauf)
R-C	(C-Lauf)
PRI	(Drucken Programm)
ADR	(Markenliste untersuchen)
COM	(Kommentar zum Programm)
END	(Ende)

Die gewünschte Betriebsart ist durch Eingabe von Y zu wählen; nicht erwünschte Betriebsarten sind durch N (oder ein beliebiges Zeichen außer Y) zu verwerfen.

Nach Ablauf einer Betriebsart fordert der Rechner wieder zur Vorwahl auf.

Mit END wird das Programm beendet. Es kann durch Betätigen der Taste STA wieder begonnen werden.

A-Lauf

Nach Betätigen von R-A durch Y beginnt ein A-Lauf mit dem Ausdruck INP (Eingabe) und der Frage nach der Herkunft des symbolischen Programms:

MAN	(wird manuell über Konsol-Fernschreiber eingegeben)
RDR	(wird als Lochstreifen eingegeben)
MEM	(befindet sich im Editor-Bereich des Kernspeichers).

Darauf folgt Drucken von OTP (Ausgabe) und die Frage, wohin das symbolische Programm während des A-Laufs (ggfs. in korrigierter Form) wieder ausgegeben werden soll:

PCH	(über Locher ausgeben)
MEM	(in Editor-Bereich ablegen)

Es schließen sich folgende Fragen an:

STP	(Anhalten bei ausgewählten Anweisungen zwecks Korrektur)
ERR	(Anhalten bei formalen Fehlern)
CLR	(Löschen Markenliste).

Erwünschte Funktionen sind mit Y zu bestätigen, nicht benötigte mit N (oder beliebigen Zeichen außer Y) zu verwerfen.

Dabei ist folgendes zu beachten:

Nach INP ist eine und nur eine Eingabeart zu wählen. MAN ist für die Ersteingabe eines symbolischen Programms zweckmäßig. Bei RDR liegt bereits ein Lochstreifen mit dem symbolischen Programm vor (vor Start des Assembler-Programms in schnellen Leser einlegen!). MEM ist nur bei einem wiederholten A-Lauf sinnvoll; dazu ist vorher das symbolische Programm in einem A-Lauf (mit OTP MEM) in den Editor-Bereich des Kernspeichers abzulegen.

Nach OTP kann eine der beiden Ausgabearten gewählt werden. PCH ist zweckmäßig bei Korrekturen, wenn der Editor-Bereich nicht benutzt wird, aber auch zur Ausgabe des im Editor-Bereich korrigierten, endgültigen symbolischen Programms MEM dagegen empfiehlt sich stets bei Editor-Betrieb (sowohl für Ersteingabe als auch für Korrektur-A-Läufe).

STP bewirkt, daß das Einlesen des symbolischen Programms beim A-Lauf an bestimmten Stellen angehalten wird, um Korrekturen durchzuführen. Zur Vorwahl der Stelle, wo das Programm angehalten werden soll, ist einzugeben:

Die laufende Anweisungsnummer (4-stellig dezimal), oder
das Instruktionsmerkmal (falls vorhanden)

der Anweisung, bei der angehalten werden soll. Der A-Lauf wird an der betreffenden Stelle angehalten, die Anweisung wird ausgedruckt, und nach Ausdruck von

SEL	(Vorwahl)

ist eine der folgenden Korrekturarten einzugeben:

D	(Anweisung löschen)
A	(Anweisung ändern, d.h. neu eingeben)
I	(Anweisung ist in Ordnung, jedoch danach neue Anweisung einfügen)
␣	(= Leertaste; Anweisung ist in Ordnung; nächste Anweisung holen)
G	(Anweisung ist in Ordnung; Aufheben des Stops, verbunden mit neuer Stop-Vorwahl).

ERR bedeutet, daß formal fehlerhafte Anweisungen während des A-Laufs mit Fehlerart ausgedruckt werden sollen (siehe Fehlerliste).

CLR hat, wenn mit Y bestätigt, zur Folge, daß die Markenliste vor Beginn des A-Laufs gelöscht wird. Im Regelfall ist dies vor jedem A-Lauf erforderlich; jedoch kann es erwünscht sein, die Merkmalsliste zu erhalten, wenn mehrere Programme, deren Adressen aufeinander Bezug nehmen, nacheinander assembliert werden.

Die Anweisungen werden vom Assembler gezählt und mit einer laufenden Nummer versehen, die zu Beginn der Anweisung gedruckt wird. Die gedruckten Nummern beziehen sich auf den aktuellen Lauf; zwecks Anhaltens eingegebene Nummern beziehen sich auf das zu korrigierende Programm, d.h. sind unabhängig von Löschungen oder Einfügungen im aktuellen Lauf.

Es ist mindestens ein A-Lauf für jede Umwandlung erforderlich. A-Läufe können beliebig oft wiederholt werden, z.B. zwecks Korrektur des symbolischen Programms; auf vorheriges Löschen der Merkmalsliste mit CLR ist dabei zu achten.

C-Lauf

Hierfür wird R-C mit Y bestätigt; es folgt mit INP die Frage nach dem Eingabemedium für das symbolische Programm:

RDR	(wird als Lochstreifen eingegeben)
MEM	(befindet sich im Editor-Bereich des Kernspeichers)

wovon eine und nur eine mit Y zu bestätigen ist, und mit OTP die Vorwahl der Umwandlungsart:

LIS	(Ausdrucken der Programmliste)
MAC	(Lochen Maschinencode-Streifen).

Vorgewählte Ein/Ausgabearten sind mit Y zu bestätigen.

Hierbei ist zu beachten:

Nach INP ist entweder RDR zu wählen (symbolischen Lochstreifen vorher mit seinem Anfang in Leser einlegen!) oder MEM (setzt A-Lauf mit Editor-Betrieb voraus, d.h. symbolisches Programm muß im Editor-Bereich des Kernspeichers stehen).

Nach OTP kann LIS oder MAC gewählt werden:

LIS führt zum Ausdruck des gesamten Programms. Je Anweisung wird eine Zeile gedruckt; sie enthält:

Laufende Anweisungsnummer (4 Dezimalstellen),
Fehlerart (bei Fehler),
Instruktionsadresse (6 Oktalstellen),
Anweisung (Instruktionsmerkmal, Befehl, Ergänzung, Vorzeichen, Adreßmerkmal),
Maschinencode (Vorzeichen + 6 Oktalstellen),
Kommentar.

Beim Listen werden jeweils 64 Zeilen zusammenhängend gedruckt mit Abstand zu den nächsten 64 Zeilen, so daß trennbare Seiten der Größe DIN A4 entstehen; die Seiten werden mit 001, 002, ... numeriert. Dahinter steht die Programmbezeichnung.

MAC erzeugt einen Maschinencode-Lochstreifen im Oktal-Format für das umgewandelte Programm. Der notwendige Streifenvorlauf (Transport- und Zufuhrlochung) wird automatisch erzeugt.

Wählt man beim C-Lauf weder LIS noch MAC vor, werden nur fehlerhafte Anweisungen auf dem Fernschreiber ausgedruckt (wichtig für Feststellung von Adressierungsfehlern durch fehlende Adreßmarken oder Überschreitung des relativ adressierbaren Bereichs).

Drucken Protokoll

Diese besondere, durch Y nach PRI zu wählende Betriebsart hat den Zweck, das gesamte symbolische Programm oder Teile davon auszudrucken.

Der zu druckende Bereich ist einzugeben mit

lfd.Nr. der ersten und
lfd.Nr. der letzten Anweisung des Bereichs

(jeweils 4-stellig dezimal); danach folgt als Bestätigung eine Leertaste (Eingabe eines anderen Zeichens bewirkt, daß die lfd.Nummern erneut eingetastet werden müssen).

Danach folgt mit INP die Frage nach der Eingabeart für das symbolische Programm:

RDR	(wird als Lochstreifen eingelesen)
MEM	(befindet sich im Editor-Bereich des Kernspeichers).

Zutreffendes ist mit Y zu bestätigen.

Es folgt der Ausdruck des symbolischen Programms im ausgewählten Bereich.

Untersuchen Markenliste

Beim A-Lauf wird ein Keller aufgebaut, der die im Programm verwendeten symbolischen Marken und die ihnen zugewisenen Adressen bzw. sonstigen Werte enthält (Merkmalsliste); der C-Lauf bedient sich dieses Kellers beim Einsetzen der Werte anstelle der Marken.

Es kann nützlich sein, die Markenliste vor dem C-Lauf zu untersuchen bzw. auch zu ändern, z.B. um fehlende Adressen für anschließende Programmteile einzusetzen oder unvollständige Marken zu suchen.

Hierzu bestätigt man die Betriebsart ADR mit Y, worauf FCT ausgedruckt wird. Die erwünschte Funktion wählt man vor durch Eintasten eines Zeichens:

L	(Listen aller Marken)
E	(Listen aller fehlerhaften Marken)
F	(Suchen/Ändern einer Marke).

Mit L wird die Gesamtheit der Marken einschließlich der zugehörigen Adressen bzw. der zugewiesenen Werte ausgedruckt, und zwar je Zeile

die symbolische Marke (3 druckbare Zeichen), und
die Adresse bzw. der zugewiesene Wert (6 Oktalstellen).

Hierbei ist zu beachten, daß letztere in der höchsten Oktalstelle normalerweise eine 6 enthalten müssen. Ist nur eine 2 vorhanden, so ist die betreffende Marke nur als Adreß-(Rechts-)Merkmal im Programm vorgekommen; eine 4 an dieser Stelle bedeutet, daß die Marke nur als Instruktions-(Links-)Merkmal gefunden worden ist.

Mit E werden alle die Marken (wie oben beschrieben) ausgedruckt, die keine 6 in der ersten Oktalstelle enthalten, also unvollständig sind. Dies gibt Hinweise auf Programmfehler (insbesondere bei fehlenden Linksmarken).

Mit F wird eine bestimmte Marke in der Liste gesucht. Die Marke ist 3-stellig einzugeben. Ist die Marke vorhanden, wird die zugehörige Adresse 6-stellig oktal gedruckt. Dann ist A einzutasten, wenn eine Änderung gewünscht wird; danach müssen Marke (3-stellig) und Adresse (6 Oktalstellen) neu eingegeben werden. Jedes andere Zeichen statt A bestätigt die Marke und ändert sie nicht. Ist die Marke nicht vorhanden, wird die Betriebsart beendet (Ausdruck R-A). Neue Marken können dadurch eingefügt werden, daß man @@@ eingibt (und danach die zugehörige Adresse). Ebenso werden Marken dadurch gelöscht, daß man @@@ eingibt sowie für die Adresse sechs Nullen.

Programmbezeichnung eingeben

Hierfür ist COM mit Y zu bestätigen und anschließend eine aus 18 Zeichen bestehende Programmbezeichnung einzugeben. Diese wird dann beim Protokoll am Anfang jeder Seite gedruckt.

FEHLERLISTE

Die beim Assembler-Betrieb ausgedruckten Fehler-Schlüssel haben folgende Bedeutung:

Schlüsselzahl	Fehlerart
01	Adreßmerkmal nicht dezimal
02	" " oktal
03	Instruktionsmerkmal mehrfach vorhanden
04	Verbotenes Zeichen in Ergänzung oder Adreßmerkmal
05	Überlauf Markenliste
06	Anweisungstyp nicht vorgesehen
10	Adreßmerkmal fehlerhaft
11	Sonstiger Fehler
12	Adresse relativ nicht erreichbar

ALLGEMEINE REGELN

Programmaufbau

Ein symbolisches Programm, das vom Assembler MINCASS 500 verarbeitet wird, besteht aus einer Folge von "Anweisungen". Die Anweisungen sind in einzelne Gruppen von Zeichen gegliedert (siehe Syntaxregeln).

Kennzeichnend für eine Anweisung ist ihr "Typ"; es sind nur die gültigen Anweisungstypen zulässig (siehe später). Es gibt vier Arten davon:

Befehle
Definitionen
Zuweisungen
Steueranweisungen.

Die ersten beiden belegen im Maschinenprogramm jeweils ein Wort (im Falle der Felddefinition F sogar mehrere); symbolische Befehle erzeugen entsprechende Maschinenbefehle, während Definitionen Plätze reservieren oder mit Festwerten belegen. Zuweisungen ordnen symbolischen Marken gewisse Werte zu; sie belegen im Maschinenprogramm keinen Platz. Steueranweisungen stehen am Anfang und am Ende des Programms oder halten reine Textzeilen frei; auch sie belegen keinen Platz.

Jedes Programm beginnt mit der Steueranweisung U (mit Angabe der Anfangsadresse) und endet mit der Steueranweisung Z. Die dazwischen liegenden Befehle und Definitionen werden in ihrer Reihenfolge ins Maschinenprogramm eingesetzt. Zuweisungen können an beliebiger Stelle im Programm stehen; jedoch muß jede Marke, die einer anderen zugewiesen wird, vorher definiert sein.

Bei der Niederschrift des symbolischen Programms benutze man die MINCAL-Instruktionslisten; jede Zeile mit einem gültigen Anweisungstyp entspricht darin einer Anweisung.

Zeichenvorrat

Es sind alle 64 druckbaren Zeichen des ASCII-(ISO-7-) Codes zulässig, einschließlich Leerschritt, soweit sie nicht durch die Gestalt der einzelnen Anweisungen eingeschränkt sind.

Zu beachten ist jedoch:

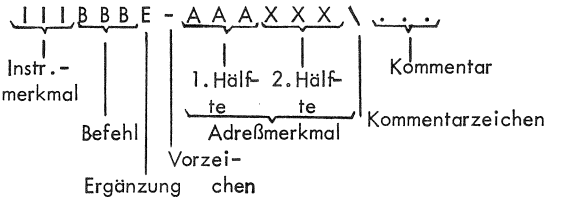
Das "Fehlerzeichen" ← (Code 137_8) löscht alle davor stehenden Zeichen der Anweisung; sie beginnt danach von neuem. Das "Kommentarzeichen" \ (= Code 134_8) leitet einen Kommentar ein und darf nur hierzu benutzt werden.

Alle nicht-druckbaren Zeichen werden überlesen. Das Zeichen "Leser aus" (X-OFF = Code 023_8) ist das Zeichen für Anweisungs-Ende. Bei manueller Eingabe kann das Semikolon (;) für "Ende Anweisung" benutzt werden; es erzeugt automatisch die notwendigen Steuerzeichen.

Mit "echten Zeichen" sind im folgenden die 64 druckbaren Zeichen außer Leerschritt, Fehler- und Kommentarzeichen gemeint.

Syntaxregeln

Eine Anweisung ist (in dieser Reihenfolge) aus Instruktionsmerkmal, Befehl, Ergänzung, Vorzeichen, Adreßmerkmal und eventuellem Kommentar aufgebaut:



Das Instruktionsmerkmal ist eine symbolische Marke (siehe später); sie umfaßt 2 oder 3 echte Zeichen und muß unmittelbar nach dem Ende der vorigen Anweisung stehen. Sie kann entfallen.

Der "Befehl" enthält den Anweisungstyp; er besteht aus 1, 2 oder 3 echten Zeichen. Er folgt auf Anweisungs-Ende + Leerschritt oder auf das Instruktionsmerkmal mit beliebig vielen Leerschritten dazwischen (mindestens 1 Leerschritt bei Instruktionsmerkmalen mit 2 echten Zeichen). Jede Anweisung muß einen gültigen Anweisungstyp in der Befehlsspalte enthalten. Für den Befehl müssen (ggfs. durch nachfolgende Leerschritte) insgesamt 3 Zeichen vorgesehen werden.

Die Ergänzung ist unmittelbar an den Befehl anzuschließen; sie besteht aus einem echten Zeichen oder einem Leerschritt (falls nicht vorgesehen).

Folgt (sofort oder mit beliebig vielen Leerschritten) auf die Ergänzung ein Minuszeichen (-), wird es als negatives Vorzeichen aufgefaßt.

Das Adreßmerkmal hat bis zu 6 echte Zeichen (falls vorgesehen); es folgt unmittelbar auf das Minuszeichen bzw. beginnt mit dem ersten echten Zeichen nach der Ergänzung.

Das Kommentarzeichen beendet die eigentliche Anweisung; alle danach stehenden Zeichen bis zum Anweisungsende werden als Kommentar aufgefaßt. Der Kommentar sollte nicht mehr als 35 Zeichen umfassen. Er kann unmittelbar hinter dem letzten Zeichen der eigentlichen Anweisung beginnen und wird durch das Anweisungsende begrenzt.

Das Anweisungsende kann unmittelbar nach dem letzten Zeichen der Anweisung stehen.

Regeln für Marken und Adreßmerkmale

Symbolische Marken

Instruktionsmerkmale bestehen stets aus symbolischen Marken; Adreßmerkmale können eine oder zwei symbolische Marken enthalten.

Eine symbolische Marke repräsentiert im symbolischen Programm eine Adresse oder einen zugewiesenen Wert.

Marken bestehen aus 2 oder 3 echten Zeichen; das erste Zeichen darf keine Ziffer sein.

Adreßmerkmale

Als Adreßmerkmale können vorkommen:

Marken (siehe oben).
Oktalzahlen (3 oder 6 Zeichen aus den Ziffern 0...7),
Dezimalzahlen (1 bis 5 Zeichen aus den Ziffern 0...9, stets mit Dezimalpunkt dahinter).

Ferner sind Kombinationen aus einer Marke (3 echte Zeichen oder 2 echte Zeichen + Leerschritt) und einer weiteren Marke, einer 3-stelligen Oktalzahl oder einer 1- bis 2-stelligen Dezimalzahl (mit Punkt danach) zulässig. In diesen Fällen wird die Summe beider dem Adreßmerkmal zugewiesen (außer bei Produktzuweisung P, wo das Produkt gebildet wird).

Alle obengenannten Adreßmerkmale können verwendet werden, wenn in Abschnitt 5 "universell" angegeben ist.

Ferner ist bei einigen Anweisungen vorgesehen:

Registerangabe (1 Zeichen: U, V, W, X, N, 1, 2 oder 3).

GÜLTIGE ANWEISUNGEN

In der Befehlsliste des Assemblers sind die nachstehenden Anweisungstypen mit den folgenden Vorschriften für die Form der Anweisung vermerkt:

Steueranweisungen

U Ursprung Programm
Definiert die Adresse der nächstfolgenden speicherbelegenden Anweisung und muß am Anfang des Programms stehen.
Adreßmerkmal: 6-stellige Oktalzahl.

C Kommentarzeile
Wird überlesen. Ein dahinterstehender Kommentar wird jedoch im Protokoll an üblicher Stelle gedruckt.

Z Ende Programm
Schließt das symbolische Programm ab.

Zuweisungen

M Speicheradreß-Zuweisung (Seite 0 des Kernspeichers)

X Externe Adreß-Zuweisung

Instruktionsmerkmal: Notwendig
Adreßmerkmal: 3-stellige Oktalzahl.

Q Beliebige Zuweisung (Equal)

Instruktionsmerkmal: Notwendig
Vorzeichen: Minuszeichen erlaubt
Adreßmerkmal: universell

P Produkt-Zuweisung

Instruktionsmerkmal: Notwendig
Adreßmerkmal: Marke * Marke
Marke * Oktalzahl
Marke * Dezimalzahl
(es wird das Produkt zugewiesen).

E Element

Instruktionsmerkmal: Notwendig
Adreßmerkmal: Dezimalzahl
(Weist auf einen Speicherplatz in einem vorher mit F reservierten Feld. Der erste Platz des Feldes hat den Index 0).

Definitionen

V Variable (freier Platz)

F Feld (von Variablen)

Instruktionsmerkmal: Bezieht sich, wenn vorhanden, auf den ersten Platz des Feldes
Adreßmerkmal: Universell

B Binärzahl

Vorzeichen: Minuszeichen bedeutet negativen Wert (Zweier-Komplement)
Adreßmerkmal: Dezimale Ganzzahl (1. bis 39999.), oder dezimaler Bruch (.00004 bis .99998)

O Oktalzahl

Vorzeichen: Minuszeichen setzt N-Bit auf 1
Adreßmerkmal: 6-stellige Oktalzahl

A Alpha-Wort

Vorzeichen: Minuszeichen setzt N-Bit auf 1. Erforderlich bei führenden Leerschritten oder Minuszeichen
Adreßmerkmal: 3 druckbare Zeichen

Y Adresse

Vorzeichen: Minuszeichen bedeutet Niveaubindung
Adreßmerkmal: Marke

YY Adressen, mehrfach indirekt (mit 1 in Bit 17)
Wie Y

Y1 Adresse (mit 1 in Bit 15)
Y2 Adresse (mit 1 in Bit 16)
Y3 Adresse (mit 1 in Bit 15 und 16)
Wie Y

} Indizierte Adressen für MINCDOS 500

MINCAL INSTRUKTIONSLISTE

DIETZ Computer SYSTEME

BEARBEITER Meyer DATUM 10.3.72

ZKZ	SORTIERMERKMAL	INSTR-MERKMAL	BEFEHL	ERGÄNZG.	VORZEICH	ADRESSMERKMAL	ZKZ	BEMERKUNGEN
	0							
	1							
	2							
	3		C				\	UHRZEIT PROGRAMM FUER 24 STUNDEN
	4		C				\	MIT SEKUNDEN-START
	5		C				\	
	6		U			Ø07100	\	URSPRUNGS-ADRESSE
	7		C				\	
	0	ANF	LDC	1		Ø.	\	RESET
	1		LDC	2		Ø.		
	2		LDC	3		Ø.		
	3		C				\	
	4	U1	ADC	1	-	59.	\	SEKUNDEN
	5		BZ	1		U2		
	6		ADC	1		60.		
	7		HBR			U1		
	0	U2	ADC	2	-	59.	\	MINUTEN
	1		BZ	2		U3		
	2		ADC	2		60.		
	3		HBR			U1		
	4	U3	ADC	3	-	23.		
	5		BZ	3		U4		
	6		ADC	3		24.		
	7	U4	HBR			U1		
	0		Z					
	1							
	2							
	3							
	4							
	5							
	6							
	7							

KOMM.NR. / PROGR.NR.

BLATT

NOP Keine Operation
VBL Dezimal-Binär Linkskomma
VBR Dezimal-Binär Rechtskomma
VDL Binär-Dezimal Linkskomma
VDR Binär-Dezimal Rechtskomma
SLN Normalisieren
HLT Halt
ECL Unterbrechung zulassen
DCL Unterbrechung verhindern

Keine weiteren Angaben

COD Code-Operation

Adreßmerkmal: 6-stellige Oktalzahl (4 letzte Stellen geben Beginn des Mikroprogramms im Festspeicher an)

SRL Schiften rechts logisch
SRA Schiften rechts arithmetisch
SLL Schiften links logisch
SLA Schiften links arithmetisch

Ergänzung: W, X oder D (vorgeschrieben)

Adreßmerkmal: entfällt (Einbit-Schiftbefehle), oder
Universell (Mehrbit-Schiftbefehle)

SRR Schiften rechts mit Runden

Adreßmerkmal: Universell

MZR Null Setzen
MPO Plus Eins Setzen
MMO Minus Eins Setzen
MIC Inkrementieren
MDC Dekrementieren
MCO Komplementieren
MCI Komplementieren und Inkrementieren

Adreßmerkmal: 3-stellige Oktalzahl oder
Marke oder
Register
(= Adresse in Seite 0)

LDC Laden Konstante
ADC Addieren Konstante

Ergänzung: Register (vorgeschrieben)
Vorzeichen: Minuszeichen bedeutet negative Konstante
Adreßmerkmal: Universell

LDR In Register Laden
TRR Register Transferieren
ADR Zu Register Addieren
SBR Von Register Subtrahieren
FOR ODER mit Register
FAR UND mit Register

Ergänzung: Register (vorgeschrieben)
Adreßmerkmal: Universell, oder
Register
(= Adresse in Seite 0)

LD Laden
TR Transfer
AD Addieren
SB Subtrahieren
MP Multiplizieren
DV Dividieren
FO Inklusives ODER
FA UND
FE Exklusives ODER
CP Vergleichen

Ergänzung: Fehlt, oder
1, 2 oder 3 (Indexregister)

Vorzeichen: Minuszeichen bedeutet Niveaubindung (vor Marke bzw. 3-stelliger Oktalzahl) oder relative Adressierung in Rückwärtsrichtung (vor Dezimalzahl)

Adreßmerkmal: Marke, oder
3-stellige Oktalzahl (Adresse in Seite 0), oder
Dezimalzahl (1. bis 511. = relative Adressierung), oder
Register

Bei indirekter Adressierung wird an den Befehl ein Y angehängt (z.B. LDY).

BR Verzweigen

Ergänzung: Fehlt, oder
1, 2, 3, 4, 5, 6 oder 7 (Sensor)

Vorzeichen: Minuszeichen vor Dezimalzahl bedeutet relative Adressierung in Rückwärtsrichtung

Adreßmerkmal: Marke, oder
Dezimalzahl (1. bis 511. = relative Adressierung)

Bei indirekter Adressierung wird an den Befehl ein Y angehängt (z.B. BRY).

BZ Verzweigen wenn Null
BP Verzweigen wenn Plus
BM Verzweigen wenn Minus

Ergänzung: Register (vorgeschrieben)
Vorzeichen: wie BR
Adreßmerkmal: wie BR

Bei indirekter Adressierung wird an den Befehl ein Y angehängt (z.B. BZY).

CS Unterprogramm-Aufruf

Ergänzung: Fehlt, oder
U oder V (Rückkehradresse)
Vorzeichen: wie LD
Adreßmerkmal: wie LD

Bei indirekter Adressierung wird an den Befehl ein Y angehängt (z.B. CSY).

CM Mikroprogramm-Aufruf

Vorzeichen: wie LD
Adreßmerkmal: wie LD

Bei indirekter Adressierung wird an den Befehl ein Y angehängt (z.B. CMY).

STL Start
HSL Halt, Start Ebene

Adreßmerkmal: 3-stellige oder 6-stellige Oktalzahl
(die drittletzte und die vorletzte Stelle geben die Zehner- und Einer-Adresse der Ebene an, die gestartet wird)..

HBR Halt mit Verzweigen

Vorzeichen: wie BR
Adreßmerkmal: wie BR

GX	Eingabe (Laden)	X-Kanal
FX	" (ODER)	"
OX	Ausgabe	"
GB	Übernahme (Laden)	X-Kanal, niveaugebunden
FB	" (ODER)	"
IBG	Eingabe, Übernahme (Laden)	"
IBF	" " (ODER)	"
IB	Eingabe-Auslösung	"
OB	Ausgabe	"
IBH	Eingabe-Auslösung, Halt	"
OBH	Ausgabe, Halt	"
RBL	Rückstellen LOCK	"
SBL	Setzen LOCK	"
RBR	Rückstellen READY	"
SKB	Sprung wenn BUSY	"
SKR	" " READY	"
IBS	Serielle Eingabe	"
OBS	" Ausgabe	"
IBB	Blocktransfer-Eingabe	"
OBB	" -Ausgabe	"

Ergänzung: bei IP...OBH (1...3 = Indexregister) zulässig, und
bei IBS, OBS (1...6 = Zeichenzahl) vorgeschrieben

Adreßmerkmal: 3-stellige Oktalzahl (externe Adresse), oder
Marke, oder
2-stellige Oktalzahl (Geräteadresse) mit Format danach,
oder
Format

Bemerkung: Zulässige Formatangaben sind:

O	Oktal
D	Dezimal
F	Fünfkanal
A	Alphanumerisch
S	Sieben-Bit
E	Acht-Bit

Bemerkung: Statt einer Marke kann stets auch ein Doppelmerkmal verwendet werden, bestehend aus einer Marke sowie einer weiteren Marke, einer 3-stelligen Oktalzahl (000 bis 777) oder einer Dezimalzahl (1. bis 99.). Zugewiesen wird die Summe der Werte beider Marken.

Eine ausführliche Beschreibung des MINCASS 500 Assembler/Editor steht auf Wunsch zur Verfügung.

Monitor

VORBEMERKUNG

Der MONITOR 500 ist ein Programm zum Austesten von Programmen, die im Kernspeicher des MINCAL 513/523 abgelegt sind. Das zu testende Programm läuft unter Steuerung des MONITORS ab und bleibt an vereinbarten Stellen stehen, so daß der Benutzer Register- und Speicherplätze auf ihren Inhalt untersuchen oder diesen verändern sowie Befehle ein- oder ausbauen kann. Der Dialog erfolgt über den Konsol-Fernschreiber. Außerdem enthält der MONITOR Routinen zur Ein- und Ausgabe des Speicherinhalts über Konsol-Fernschreiber und schnelle Lochstreifengeräte.

Rechner-Ausstattung:	MINCAL 513/523 Grund + erweiterter Befehlsvorrat Programmierhilfe X00 (ampfohlen) Kernspeicher 4k oder größer Konsol-Fernschreiber (8-Kanal) an Ebene 0 Schnelle Lochstreifenausrüstung an Ebene 0 (empfohlen)
Programmlänge:	$13ØØ_8$ Worte Lage im Kernspeicher beliebig
Benutzte feste Speicherplätze:	8 aufeinanderfolgende Plätze in Seite 0 Adressen 040...047, veränderbar)

TESTBEGINN

Zunächst wird der MONITOR-Lochstreifen in einen freien Kernspeicherbereich eingelesen, von einer Anfangsadresse a bis zu einer Endadresse e = a + 1277_8. Dies geschieht z.B. über die eingebaute Programmierhilfe X00 wie folgt: Schalter INT und SENSOR 1, 2, 3, 4 (und 5 bei schnellem Leser) einlegen. Tasten RES, END, STA in dieser Reihenfolge betätigen. Auf dem Fernschreiber a und e als 6-stellige Oktalzahlen eingeben. STA betätigen; der Streifen wird gelesen. Danach N-Register (Platz 000004) über die Frontplatte auf a-1 setzen (war a = 001000 gewählt worden, genügt Betätigen der Taste END). Schalter INT und SENSOREN in Ausgangsstellung bringen. Tasten RES, STA betätigen.

Jetzt wird durch Eingabe von

LEV l (cr) (cr = Wagenrücklauf)

die Ebene l (= 0...7) vorgewählt, in der das zu testende Programm laufen soll.

Dann wird das zu testende Programm über eine der Einlese-Betriebsarten in den Speicher gelesen (s. EIN/AUSGABE).

STEUERKOMMANDOS

Das zu testende Programm wird mit

RUN s (cr)

gestartet, wobei s die Startadresse angibt. Diese ist - wie alle im folgenden beschriebenen Adressen - als 1- bis 6-stellige Oktalzahl einzugeben.

Soll später nach einem Monitor-Halt das Programm wieder gestartet werden, ist

RUN (cr)

ohne Adresse einzugeben.

Durch das Kommando

END (cr)

wird der Monitor-Betrieb beendet.

MONITOR-HALT

Das zu testende Programm kann an beliebigen Stellen angehalten werden; man bereitet sie durch die Eingabe

STI h (cr)

vor, wobei h die Adresse ist, an der das Programm später einmal anhält. Will man, daß es dort beliebig oft anhält, wird

STI h,Ø (cr)

eingegeben.

Jeder so eingebaute Halt kann durch das Kommando

STO h (cr)

wieder ausgebaut werden, wobei h die Stopadresse ist. Durch

STD (cr)

werden sämtliche Halts wieder eliminiert.

LÖSCHEN, ABFRAGEN UND ÄNDERN

Durch das Kommando

DEL m,n (cr)

wird der Adreßbereich von m bis n gelöscht (alle Plätze bekommen Nullinhalt). Soll nur ein Platz m gelöscht werden, gebe man

DEL m (cr)

ein.

Der jeweilige Inhalt der 7 Register U, V, W, X, 1, 2 und 3 der Ebene 1 wird, durch

IXR (cr)

ausgelöst, in dieser Reihenfolge oktal ausgedruckt.

Ein weiteres Kommando

ITC m (cr)

bemerkt, daß der Inhalt der Adresse m oktal ausgedruckt wird. Jetzt hat der Benutzer folgende Möglichkeiten:

Eingabe C: Der Inhalt der folgenden Adresse wird gedruckt (kann beliebig oft wiederholt werden)

Eingabe xC: Der neue Inhalt x (1- bis 6-stellige Oktalzahl), evtl. Minuszeichen davor) wird in den gerade angewählten Speicherplatz übertragen.

Eingabe (cr): Der Inhalt des gerade angewählten Speicherplatzes wird ausgedruckt.

Eingabe E: Die Betriebsart ITC wird beendet.

EIN/AUSGABE

Für die Ein- und Ausgabe der zu testenden Programme oder von Programmteilen hält der MONITOR folgende Kommandos bereit:

ITO	m,n (cr)	Eingabe	über	Teletype	Oktal-Format
ITA	m,n (cr)	"	"	"	Alpha- "
IRO	m,n (cr)	"	"	Leser	Oktal- "
IRA	m,n (cr)	"	"	"	Alpha- "
OTO	m,n (cr)	Ausgabe	"	Teletype	Oktal- "
OTA	m,n (cr)	"	"	"	Alpha- "
OPO	m,n (cr)	"	"	Locher	Oktal- "
OPA	m,n (cr)	"	"	"	Alpha- "

Mit m ist die erste, mit n die letzte Adresse des Speicherbereichs gemeint.

ZUR BEACHTUNG

Der MONITOR benutzt 8 aufeinanderfolgende Speicherplätze in Seite 0, die für ihn freigehalten werden müssen. Im Normalfall sind dies die Plätze $040_8 \ldots 047_8$. Soll stattdessen der Platz p in Seite 0 (und die 7 folgenden) verwendet werden, so gebe man zu Anfang das Kommando

```
ITC q  (cr)
...pC
```

wobei $q = a + 27_8$ zu wählen ist (a = Anfangsadresse MONITOR).

Bei Systemen mit Externspeicher (Trommel) kann durch das Kommando

```
DOS
```

jederzeit die Kontrolle an den OPERATING-Teil des Trommelbetriebssystems MINCDOS 500 übergeben werden.

FORTRAN

VORBEMERKUNG

FORTRAN ist eine Programmiersprache für mathematische und technisch-wissenschaftliche Aufgaben und, in Form des MINCAL FORTRAN, auch für begrenzten Einsatz in der Prozeßtechnik. Für die Umwandlung von FORTRAN-Programmen in die Maschinensprache des MINCAL 523 steht ein Übersetzungsprogramm (Compiler) zur Verfügung.

Rechner-Ausstattung:	MINCAL 523 Grund- + erweiterter Befehlsvorrat Programmierhilfe XØØ Kernspeicher ab 4 k (ab 8 k empfohlen) Konsol-Fernschreiber (8-Kanal) an Ebene 0 Schnelle Lochstreifenausrüstung an Ebene 0 wahlweise zusätzlich Externspeicher
Compiler:	Phasen-Compiler, ca. 16 k lang, 20 Phasen übersetzt Quellprogramm in einem Arbeitsgang
Speicherbelegung:	Compiler und compiliertes Objektprogramm arbeiten auf Ebene 0. Die 8 Register dieser Ebene (ØØ...Ø7) sowie die Plätze 500...777 in Seite 0 werden benutzt; ferner die Speicheradressen von 1ØØØ bis zur eingegebenen Endadresse.

Soweit nicht nachstehend anders beschrieben, entspricht MINCAL FORTRAN den ASA-Spezifikationen für FORTRAN.

QUELLPROGRAMM

Das FORTRAN-Quellprogramm, das als Lochstreifen im ASCII-Code vorliegen muß, besteht aus einer Folge von Anweisungen; jede umfaßt im Normalfall eine Zeile:

1. Zeichen	Leerschritt
2.-5. Zeichen	Leerschritte oder 1- bis 4-stellige Anweisungsnummer (label)
6. Zeichen	Leerschritt
ab 7. Zeichen	eigentliche Anweisung

Für Folgezeilen gilt:

1.-5. Zeichen	Leerschritte
6. Zeichen	nicht Leerschritt und nicht Ø
ab 7. Zeichen	Fortsetzung der Anweisung

Für Kommentarzeilen gilt:

1. Zeichen	Buchstabe C
ab 2. Zeichen	beliebiger Kommentar

Das physikalische Ende des FORTRAN-Programms ist gekennzeichnet durch:

1. Zeichen	Schrägstrich (/)
2. Zeichen	Stern (*)

Jede Zeile wird mit Semikolon (;) abgeschlossen.

ZEICHENVORRAT

Buchstaben	A B C D E F G H I J K L M N O P Q R S T U V W Z
Ziffern	0 1 2 3 4 5 6 7 8 9
Symbole	= + - * / ( ) , . ' ;

Leerschritte werden, außer in Textstrings und Kommentaren, überlesen.

ELEMENTE

Ganzzahl-Konstanten:	1 bis 5 Ziffern, bei negativen Konstanten Minuszeichen davor maximaler Betrag bei Ein/Ausgaben: 39999
Gleitkomma-Konstanten:	Ziffernfolge im F-Format (z.B.: 2. oder -.ØØØ7) oder im E-Format (z.B.: 1374.9E27 oder -.5896E-1Ø2) maximaler Betrag des Exponenten: 153
Einfache Variable:	Variablen-Namen (names) beginnen mit einem Buchstaben, dem bis zu 5 Buchstaben oder Ziffern folgen können. Namen, die mit I, J, K, L. M oder N anfangen, bezeichnen Ganzzahl-Variablen. Verboten sind FORTRAN-Wörter sowie Namen von Standard-Funktionen und Unterprogrammen.
Indizierte Variable:	Variablen-Name, gefolgt von einem oder zwei Indizes (durch Komma getrennt) in Klammern. Indizes dürfen aus einer einfachen Ganzzahl-Variablen, einer Ganzzahl-Konstanten oder der Summe oder der Differenz beider bestehen. Der Wert des Index muß im Bereich 1...511 bleiben.

Ganzzahlen werden intern durch ein Wort dargestellt (max. Betrag 262143), Gleitkommazahlen durch ein Doppelwort (Genauigkeit der Mantisse $\approx 10^{-8}$, max. Betrag des Exponenten 153).

Arithmetischer Ausdruck: (expression)

Als Operanden sind Konstanten, Variablen-Namen und Namen von Standard-Funktionen zugelassen. Als Operatoren sind erlaubt:

+	Addition
-	Subtraktion
*	Multiplikation
/	Division
**	Potenzierung

Beliebige Klammerung ist zugelassen. In einem Ausdruck dürfen nur Operanden eines Typs (Ganzzahl oder Gleitkomma) vorkommen.

AUSFÜHRBARE ANWEISUNGEN

Ergibtanweisung:

<name> = expression
Weist der links stehenden Variablen den Wert des rechts stehenden Ausdruckes zu. Name und Ausdruck dürfen von verschiedenem Typ sein.

GOTO

GOTO <label>
Unbedingter Sprung

COMPUTED GOTO

GOTO (<label 1>, <label 2>, ...) <name>
Berechneter Sprung

IF

IF (<expression>) <label 1>, <label 2>, <label 3>
Bedingter Sprung

IF SENSOR

IF (SENSOR<n>)<label 1>, <label 2>
Bedingter Sprung in Abhängigkeit von Sensor-Schaltern (wenn EIN, label 1)

DO

DO<label><name> = <first>, <last>, <step>
Laufanweisung.
Die Lauf-Variable (name) muß eine einfache Ganzzahl-Variable, die restlichen dürfen einfache Ganzzahl-Variablen oder positive Ganzzahl-Konstanten sein. Die Schrittweite (step) kann entfallen; sie wird dann zu 1 angenommen.
Als letzte Anweisung einer DO-Schleife (hinter label) darf nicht GOTO, IF, IF SENSOR, DO, RETURN, STOP oder PAUSE stehen. Stattdessen ist CONTINUE zu verwenden.
DO-Schleifen dürfen 5fach geschachtelt werden. Die innere Schleife muß innerhalb der äußeren liegen. Mehrere DO-Schleifen dürfen auf die gleiche Anweisung ziehen.
Aus einer DO-Schleife darf herausgesprungen werden; der augenblickliche Wert der Lauf-Variablen (name) ist verfügbar. Jede DO-Schleife wird mindestens einmal durchlaufen.

CONTINUE	CONTINUE Leeranweisung
STOP	STOP<octal> Ende des Programmablaufs. Ist danach eine 1- bis 4-stellige Oktalzahl (octal) angegeben, wird diese ausgedruckt.
PAUSE	PAUSE<octal> Anhalten des Programmablaufs. Ist danach eine 1- bis 4-stellige Oktalzahl (octal) angegeben, wird diese ausgedruckt.
CALL NIVEAU	CALL NIVEAU <level> Startet eine andere Programmebene, die als 1- oder 2-stellige Oktalzahl (level) anzugeben ist.
CALL SUBROUTINE	CALL <subr>(<par 1>,<par 2>, ...) Es können bis zu 15 aktuelle Parameter (par...) übergeben werden, die aus Konstanten, einfachen Variablen oder Feldnamen bestehen. Im letzteren Falle wird das ganze Feld übergeben. Parameter müssen in Anzahl, Typ und Größe mit den formalen Parametern des Unterprogramms (subr) übereinstimmen. Die Parameter-Ausgabe kann entfallen. Unterprogramm-Aufrufe dürfen im Hauptprogramm und im Unterprogramm in beliebiger Zahl und Schachtelung vorkommen.
RETURN	RETURN Rücksprung aus Unterprogramm
CALL CODE	CALL CODE<number> Aufruf einer Code-Prozedur, deren Nummer (number) durch eine 1- oder 2-stellige Oktalzahl anzugeben ist.
READ	READ (<device>,<format>),<list> Leseanweisung
WRITE	WRITE (<device>,<format>) <list> Schreibanweisung

Diese Anweisungen beziehen sich auf die Konsol-Peripherie des Rechners.
Als Geräteadressen (device) sind anzugeben:

Ø für Teletype
6 für schnellen Leser/Locher

Die Formatangabe ist in einer getrennten FORMAT-Anweisung enthalten, deren Anweisungs-Nummer in (format) anzugeben ist.

Am Ende der Anweisung steht eine Variable oder eine Liste von Variablen (list), die durch Kommata getrennt sind, oder auch eine implizite DO-Schleife der Form (<name>(<index>), <index>=<first>,<last>,<step>).

INPUT

INPUT (<first>,<last>,<step>) <list>
Eingabeanweisung

OUTPUT

OUTPUT (<first>,<last>,<step>) <list>
Ausgabeanweisung

Diese Anweisungen beziehen sich auf binäre Prozeß-Ein/Ausgaben. In der Klammer stehen externe Anschlußnummern oder Ganzzahl-Variablen, denen vorher eine Nummer zugewiesen wurde. Die Anweisung bewirkt den Transfer zwischen einer Reihe von Prozeßanschlüssen und der in der Liste angegebenen Ganzzahl-Variablen (list, s.oben).
Die Angabe der Schrittweite (step) kann entfallen; sie wird dann zu 1 angenommen. Für nur einen transferierten Wert entfällt auch (last).

DREAD

DREAD <mem>,<ext>
Externspeicher-Leseanweisung

DWRITE

DWRITE <mem>,<ext>
Externspeicher-Schreibanweisung

Diese Anweisungen dienen zum Transfer von Datenfeldern zwischen Kernspeicher und Externspeicher (Trommel oder Platte). Der Name der Kernspeicher-Adresse (mem) muß in einer DIMENSION-, der Name der Externspeicher-Adresse in einer DCOMMON-Anweisung vereinbart sein.

NICHTAUSFÜHRBARE ANWEISUNGEN

DIMENSION

DIMENSION <name 1>(<dim 11>,<dim 12>), ...
Feldanweisung.
Reserviert beliebig viele ein- oder zweidimensionale Felder (mit 1 bis 511 Plätzen je Dimension) für jeden Programmteil getrennt.

COMMON

COMMON <name 1>(<dim 11>,<dim 12>), ...
Speicherblockanweisung.
Reserviert beliebig viele Variablen, ein- oder zweidimensionale Felder (mit 1 bis 511 Plätzen je Dimension) für alle Programmteile gemeinsam.

DCOMMON

DCOMMON (<device>)<name>(<dim>), ...
Externspeicher-Belegung.
Reserviert beliebig viele Variablen oder eindimensionale Felder auf dem Externspeicher (Trommel oder Platte). Wenn mehrere Externspeicher angeschlossen sind, ist für jeden eine DCOMMON-Anweisung vorzusehen mit Angabe der Nummer (device, Ø bis 3).

CODE

CODE <number>(<length>)
Code-Prozedur.
Reserviert für eine Code-Prozedur mit einer 1- bis 2-stelligen oktalen Nummer (number) Speicherplatz, dessen Größe (in Worten) als Dezimalzahl anzugeben ist (length).
Code-Prozeduren sind getrennt erstellte Maschinenprogramme, die nach der Compilierung eingelesen werden.

SUBROUTINE

SUBROUTINE <subr>(<par 1>,<par 2>, ...)
Unterprogramm.
Gekennzeichnet durch Namen (subr) mit bis zu 6 alphanumerischen Zeichen (1. Zeichen = Buchstabe). Parameter siehe CALL SUBROUTINE.

FORMAT

FORMAT (<spec 1>,<spec 2>, ...)
Formatanweisung.
Die Spezifikationen (spec...) können folgende Formate enthalten:

rIW	Ganzzahl	r = Wiederholungszahl
rFw.d	Gleitkommazahl im F-Format	w = Zeichenzahl
rEw.d	Gleitkommazahl im E-Format	d = Stellen hinter Dezimalpunkt
rAw	Alphanumerischer Wert	
'<text>'	Literal (Textstring)	
nX	n Leerschritte bzw. n Zeichen überlesen	
/	Zeilenende	

Gruppen von Formatspezifikationen können in Klammern eingeschlossen und mit einer davorstehenden positiven Ganzzahl als Wiederholungsfaktor versehen werden.

END

END
Bezeichnet das physikalische Ende eines Programmteils (Haupt- oder Unterprogramm).

STANDARDFUNKTIONEN

Folgende Standardfunktionen sind vorgesehen:

Name	Bedeutung	Argument	Ergebnis
ABS	Absolutwert	R	R
IABS'	"	I	I
FLOAT	Typumwandlung	I	R
IFIX	"	R	I
EXP	Exponentialfunktion	R	R
ALOG	Natürlicher Logarithmus	R	R
SIN	Sinus	R	R
COS	Cosinus	R	R
TANH	Tangens hyperbolicus	R	R
SQRT	Quadratwurzel	R	R
ATAN	Arcus tangens	R	R

I = Typ Ganzzahl; R = Typ Gleitkommazahl

CODE-PROZEDUREN

Jede Code-Prozedur kann als einzelner Maschinencode-Lochstreifen vorliegen. Die Streifen müssen in Alpha-Format gestanzt sein und folgenden Aufbau haben:

1. Wort: Name der Code-Prozedur (1...77 oktal)
2. Wort: Länge der Prozedur (binär)
3. letztes Wort: Befehle

Die Code-Prozeduren brauchen nicht in der Reihenfolge ihrer Definition eingelesen zu werden.

Die Code-Prozeduren liegen später in der Reihenfolge ihrer Nummern im Speicher vor dem COMMON-Bereich, der am Ende des verfügbaren Speicherbereichs liegt (festgelegt durch Eingabe der Compiler-Endadresse). Sie sind möglichst relativ adressiert zu schreiben. Die Übergabe von Daten erfolgt zweckmäßig über den COMMON-Bereich; in diesem sind Variable und Felder von hinten nach vorn in der Reihenfolge ihrer Definition enthalten.

FORTRAN-WÖRTER

Außer den Namen der Standard-Funktionen dürfen folgende Wörter nicht als Variablen-Namen verwendet werden:

GOTO	CODE	DIMENSION
IF	NIVEAU	COMMON
SENSOR	RETURN	SUBROUTINE
DO;	WRITE	FORMAT
CONTINUE	READ	DREAD
STOP	INPUT	DWRITE
PAUSE	OUTPUT	DCOMMON
CALL	END	

Ferner die Namen der Standard-Funktionen.

HANDHABUNG DES COMPILERS

Compiler-Lochstreifen vor Phase 1 in den schnellen Leser einlegen. INT, SENSOR 1, 2, 3, 4, 5 nach unten; RES, END, STA betätigen; Anfangs- und Endadresse (auf Lochstreifen vermerkt) 6-stellig oktal über Fernschreiber eingeben; STA erneut betätigen. Phase 1 wird eingelesen; danach INT und SENSOREN nach oben; RES, END, STA betätigen.

Auf dem Fernschreiber werden nun Fragen ausgedruckt, die der Benutzer beantworten muß. Zuvor ist das Quellprogramm mit dem Zufuhrbereich in den Leser einzulegen.

LISTE: Y, wenn ein Protokoll des eingelesenen FORTRAN-Programms gewünscht wird, sonst N.

DOS-NIVEAU: Ebene der DOS-EXECUTIVE 2-stellig oktal eingeben*).

TROMMEL-ANZ.: Anzahl der Externspeicher eingeben*). Die Basisadresse für den DCOMMON-Bereich auf den Trommeln ist 6-stellig oktal einzugeben, z.B.:

Anfangsadresse auf Trommel 1: 1ØØ ØØØ
" " " 2: 2ØØ ØØØ

MAXIMUM KERNSP.: 6-stellige Oktalzahl als Endadresse für den Compiler eingeben.

Nach diesen Angaben wird das Quellprogramm eingelesen und im Kernspeicher abgelegt. Danach muß der Compiler-Lochstreifen vor Phase 2 in den Leser gelegt und nach Ausgabe der Nachricht START: ein Zeichen getippt werden. Nach Eingabe dieses Zeichens wird dann sukzessiv compiliert.

Nach dem Compilieren wird der noch freie Kernspeicherbereich ausgedruckt, z.B. FREI: Ø1Ø334-Ø17776. Anschließend wird nach einfügenden Code-Prozeduren (CP?) gefragt. Der Compiler-Streifen kann jetzt herausgenommen werden.

*) bei Systemen mit Externspeicher. Falls nicht vorhanden, Nullen eingeben.

Sind Code-Prozeduren vorgesehen, Antwort auf CP?: "Y", nachdem der Code-Lochstreifen eingelegt ist. Der Lochstreifen wird eingelesen. Diese Manipulation ist für jede Code-Prozedur zu wiederholen.

Auf die Eingabe eines beliebigen Zeichens, außer "Y" erfolgt die Ausgabe der Nachricht: START? xxxxxx (Startadresse). Der Dialog wird mit "Y" fortgesetzt, falls das generierte Programm sofort ausgeführt werden soll. Werden in diesem Programm Informationen über den Leser angefordert, ist der zugehörige Datenlochstreifen vor dem Programmaufruf in den Leser einzulegen.

Die Startaufforderung wird mit jedem anderen Zeichen außer "Y" negiert. In diesem Fall läuft der Rechner auf HALT; das Objektprogramm kann danach zu einem beliebigen Zeitpunkt über die Taste "STA" (an der Rechner-Frontplatte) gestartet werden.

Fehler im Quellprogramm werden durch eine Nachricht registriert; führen diese Fehler zum Abbruch der Compilierung, kann eine neue Umwandlung erst nach der Korrektur des Quellprogramms wiederholt werden. Es werden auch Fehler gemeldet, die zur Objektzeit festgestellt werden. In diesen Fällen stoppt der Rechner nach Ausgabe der Fehlernachricht; erneutes Starten ist möglich und zulässig.

FEHLERMELDUNGEN

Warnungen ***

001	ungültiges Zeichen in Spalte 1, Zeile wird als Kommentar aufgefaßt
002	kein gültiges Programmende-Zeichen
003	ungültiges Zeichen in Spalte 2...5
004	Folgestatement darf keine Statement-Nr. besitzen
005	erstes Statement kann kein Folgestatement sein
007	Statement-Nr. vor END, COMMON, SUBROUTINE, CODE oder DIMENSION
008	END als 1. Statement
009	END fehlt vor SUBROUTINE; wird vom Compiler generiert; nachfolgende Statement-Nrn. sind dann um 1 erhöht
010	mehrere ENDs hintereinander

Fehler<<<zur Compilezeit

100	Kernspeicher-Überlauf
101	kein imperatives Statement zwischen SUBROUTINE und END
102	nach END kein SUBROUTINE
103	öffnende Klammer nach DIMENSION fehlt. Variable nicht indiziert.
104	Variable nicht durch Komma getrennt (auch indizierte)
105	keine schließende Klammer bei DIMENSION
106	mehr als 2 Indizes (mehr als 1 Komma vorhanden)
107	keine öffnende Klammer vor Parametern bei SUBROUTINE
108	keine schließende Klammer nach Parameter bei SUBROUTINE

109	mehr als 15 Parameter
110	kein Statementende nach schließender Klammer bei SUBROUTINE oder CODE
111	keine oktale Nummer bei CODE
112	keine öffnende Klammer vor Längenangabe in CODE
113	keine schließende Klammer nach Längenangabe in CODE
114	keine Längenangabe in CODE (nach öffnender Klammer folgt sofort schließende)
115	keine Längenangabe bei CODE
116	Variable beginnt nicht mit Buchstaben
117	Index > 511
118	frei
119	Variablenname = Name einer Standardfunktion
120	Variablenname doppelt definiert
121	Parametername in SUBROUTINE doppelt definiert
122	Einfache Variable indiziert gebraucht
123	indizierte Variable einfach gebraucht
124	unzulässiges Zeichen nach UP-Name in CALL-Statement
125	Parametername = Standardfunktions-Name
126	gesamtes Feld in READ, WRITE, INPUT, OUTPUT darf nicht in D COMMON definiert sein
127	kein gültiger UP-Name
128	RETURN im Hauptprogramm nicht erlaubt
129	falsches Zeichen vor LITERAL
130	mehr als ein "E" in REAL-Konstante
131	mehr als ein Punkt in REAL-Konstante
132	INTEGER-Konstante > 39999
133	FORMAT-Spezifikation beginnt nicht mit öffnender Klammer
134	keine Statement-Nummer bei FORMAT vorhanden
135	zwei FORMAT-Spezifikationen teilende Zeichen hintereinander
136	mehr als ein Buchstabe pro FORMAT-Spezifikation
137	unzulässiges Zeichen im FORMAT-Statement
138	Anzahl der öffnenden Klammern ungleich der schließenden
139	kein Punkt in E- bzw. F-FORMAT
140	D ezimalstellen > 80
141	Feldlänge - Dezimalstellen < 7 bei E-FORMAT
142	Feldlänge - Dezimalstellen < 2 bei F-FORMAT
143	mehr als ein Punkt in E- oder F-FORMAT
144	Feldlänge - bei I-, E- bzw. F-FORMAT > 80
145	Feldlänge bei I-, E- bzw. F-FORMAT = 0
146	Feldlänge bei A-Spezifikation = 0
147	Feldlänge bei A-Spezifikation > 3
148	Wiederholungsfaktor vor I-, E-, F-, A- bzw. X-Spezifikation = 0
149	Wiederholungsfaktor vor I-, E-, F-, A- bzw. X-Spezifikation >130
150	die FORMAT-Nr. einer READ- oder WRITE-Anweisung ist nicht definiert
151	FORMAT-Nr. doppelt definiert
152	frei
153	Kernspeicher-Basisadresse darf kein Feld = COMMON-Bereich sein
154	trotz Angabe Trommelzahl = 0 werden Trommelbefehle benutzt. Abbruch.

155 Index = Standardfunktion bzw. 156
156 Index ist indiziert
157 Index ist nicht INTEGER
158 Reihenfolge: "Variable ± Konstante" nicht eingehalten
159 unzulässiger Index
160 Index 511
161 Index 0
162 Index = 0
163 bei doppelter Indizierung Indizes nicht durch Komma getrennt
164 schließende Klammer nach Index fehlt
165 keine Variable auf linker Seite bei arithmetischem Statement
166 "=" Zeichen im arithmetischen Statement fehlt
168 kein " (" bei Standardfunktionen
169 ungültiger Operand
170 Klammerung unpaarig im arithmetischen Statement
171 ungültiges Zeichen im arithmetischen Statement
172 ungültiges Zeichen in IF (arithmetisch)
173 Klammerung unpaarig in IF (arithmetisch)
174 unzulässiges Zeichen im Statement
175 frei
176 Ziffer nicht oktal in CALL NIVEAU
177 ungültiges Zeichen in CALL NIVEAU
178 keine öffnende Klammer bei CALL name
179 unzulässiges Zeichen in CALL name
180 nach Vorzeichen (±) folgt keine Konstante in Parameterliste von CALL name
181 mehr als 15 Parameter in CALL name
182 Parameter in CALL name indiziert
183 keine schließende Klammer nach Parameter in CALL name
184 unzulässiges Zeichen in CALL CODE
186 keine Oktalzahl in CALL CODE
187 ungültiges Zeichenin PAUSE oder STOP
188 Kernspeicher-Basisadresse bei DRED/DWRITE muß gesamtes Feld in E/A sein
189 keine Oktalzahl bei STOP/PAUSE
190 nach schließender Klammer bei CALL NIVEAU kein Statementende
191 ungültiges Zeichenin DREAD/DWRITE
192 keine oder falsche Geräte-Nr. bei READ/WRITE
193 kein Komma als Trennzeichen zwischen Geräte-Nr. und Format-Nr. bei READ/WRITE
194 frei
195 keine schließende Klammer nach Geräte/Format-Nr. in READ/WRITE
196 mehr als 5fache Schachtelung der Impl.-DO-Schleife
197 frei
198 ungültiges Zeichen in E/A-Liste
199 unpaarige Klammerung in E/A-Liste
200 mehr als 64 Zeichen pro Statement-Zeile
201 Variable in E/A-Liste nicht durch Komma getrennt
202 Lauf-Variable in impl.-DO-Schleife ist nicht vom Typ INTEGER
203 Externspeicher-Basisadresse nicht durch DCOMMON definiert
204 Anfangswert der impl. DO-Schleife nicht INTEGER
205 Parameter der impl. DO-Schleife nicht durch Komma getrennt

206 Endwert der impl. DO-Schleife nicht INTEGER
207 auf den Endwert der impl. DO-Schleife folgt keine ")" oder Komma
208 die Schrittweite der impl. DO-Schleife ist nicht INTEGER
209 "(" bei INPUT/OUTPUT fehlt
210 externe Adresse bei INPUT/OUTPUT nicht INTEGER
211 Anfangs- und Endwert der externen Adressen bei INPUT/OUTPUT nicht durch Komma getrennt
212 Endwert der externen Adresse bei INPUT/OUTPUT nicht INTEGER
213 auf den Endwert der externen Adresse folgt kein Komma oder ")"
214 Schrittweite der externen Adresse bei INPUT/OUTPUT nicht INTEGER
215 ")" bei INPUT/OUTPUT fehlt
216 mehr als 5fache Schachtelung der DO-Statement
217 DO-Schleife rückwärts nicht erlaubt
218 falsche DO-Schachtelung
219 keine gültige DO-Ende-Nr.
220 keine Lauf-Variable vorhanden
221 nach Lauf-Variablen fehlt "="-Zeichen
222 kein Anfangswert für Lauf-Variable vorhanden
223 kein Komma nach Anfangswert
224 kein Endwert für Lauf-Variable vorhanden
225 keine gültige Schrittweite für Lauf-Variable
226 kein gültiges Statementende
227 GOTO darf nicht als DO-Ende-Statement benutzt werden
228 Comp. GOTO "
229 IF "
230 IF SENSOR "
231 DO "
232 STOP "
233 RETURN "
234 im vorangegangenen Programmteil wurden folgende DO-Ende-Statements nicht gefunden
235 ungültiges Zeichen in GOTO
236 Marke enthält mehr als 4 Ziffern bei comp. GOTO
237 Marken nicht durch Komma getrennt oder keine schließende Klammer bei comp. GOTO
238 keine Marke innerhalb des Klammerausdruckes bei comp. GOTO vorhanden
239 nach öffnender Klammer keine Marke vorhanden
240 Sprungverteiler-Variable im comp. GOTO-Statement nicht INTEGER
241 kein Statementende nach Sprungverteiler-Variablen
242 ungültiges Zeichen zwischen Marken bei arithmetischem IF
243 keine drei Marken bei arithmetischem IF vorhanden
244 ungültige Marke bei arithmetischem IF
245 Marke in arithmetischem IF besteht aus mehr als 4 Ziffern
246 mehr als 3 Marken bei arithmetischem IF vorhanden
247 Sensorangaben in IF SENSOR falsch
248 keine schließende Klammer bei IF SENSOR vorhanden
249 Marken bei IF SENSOR nicht durch Komma getrennt
250 ungültige Marke
251 Marke nicht definiert
252 Statement-Nr. doppelt definiert
253 Mixed Mode

254	IFIX/FLOAT mehr als 5fach geschachtelt
255	GP läuft in Markentabelle Abbruch
300	keine Oktalziffern für Trommel vorhanden
301	Fehler kein ")" nach Trommel-Nr.-Angabe
304	kein Punkt nach E erlaubt
305	Exponent fehlt
333	Statement enthält Fehler

Fehler <<< zur Objektzeit

400	keine Liste bei INPUT/OUTPUT
401	ungültiger Satzabschluß
402	Typ-Unterschied zur FORMAT-Spezifikation und Liste
403	INTEGER-Wert zu groß
404	FORMAT-Spezifikation zu klein (bei F E, I, A)
405	Liste bei INPUT/OUTPUT zu kurz
406	Ergebnis bei ADD, SUB, MUL oder POT zu groß
407	IFIX: Zahl zu groß
408	SQRT: Radikand negativ
409	LN für negative Werte nicht definiert
410	LN (0) = unendlich
411	EXP: Ergebnis zu groß
412	UP-Name nicht definiert
418	NEU STARTEN! (Tabellenüberlauf)

MINCAL FORTRAN

PROGRAM QUADRATZAHLEN-BERECH. DATE 6.2.72 NAME Heyer

DIETZ Computer SYSTEME

COMMT	LABEL	CONTIN	STATEMENT
C			BEISPIEL: BERECHNUNG VON QUADRATZAHLEN
			DIMENSION N(50,40)
			READ (0,5) K,M
			DO 100 I=1,K
			DO 200 J=1,M
			N(I,J)=I*I
	200		WRITE (0,2) I,J,N(I,J)
	100		WRITE (0,3)
			WRITE (0,4)
			DO 300 I=1,M
			DO 300 J=1,K
			IF (I-J) 300,301,300
	301		WRITE (0,1) N(I,I)
	300		CONTINUE
			STOP
	1		FORMAT (I3)
	2		FORMAT (I3,' HAL ',I3,' IST ',I4)
	3		FORMAT (/)
	4		FORMAT ('QUADRATZAHLEN:')
	5		FORMAT (2I3,I1)
			END
/*			

ALGOL

VORBEMERKUNG

ALGOL ist eine Programmiersprache für mathematische und technisch-wissenschaftliche Aufgaben. Für die Umwandlung von ALGOL-Programmen in die Maschinensprache des MINCAL 523 steht ein Übersetzungsprogramm (Compiler) zur Verfügung.

Rechner-Ausstattung:	MINCAL 523 Grund- + erweiterter Befehlsvorrat Programmierhilfe X00 Kernspeicher ab 4k (ab 8k empfohlen) Konsol-Fernschreiber (8-Kanal) an Ebene 0 Schnelle Lochstreifenausrüstung an Ebene 0
Compiler:	Phasen-Compiler, ca. 7 k lang, 6 Phasen übersetzt Quellprogramm in einem Arbeitsgang
Speicherbelegung:	Compiler und compilierte Objektprogramme arbeiten auf Ebene 0. Die 8 Register dieser Ebene (00...07) sowie die Plätze 400...777 in Seite 0 werden benutzt; ferner die Speicheradressen von 1000 bis zur Endadresse.

Soweit nicht nachstehend anders beschrieben, entspricht MINCAL ALGOL dem IFIP-Subset von ALGOL 60.

QUELLPROGRAMM

Das ALGOL-Quellprogramm muß als Lochstreifen im ASCII-Code vorliegen. Das physikalische Ende wird durch ein Doppelkreuz (#) gekennzeichnet. NULL- oder RUBOUT-Zeichen innerhalb des ALGOL-Programms sind verboten.

ZEICHENVORRAT

Buchstaben A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Ziffern 0 1 2 3 4 5 6 7 8 9

Symbole = : + - * / ( ) ⊏ ⊐ , . ' ;

Leerschritte werden, außer in Textstrings, überlesen.

Bemerkung: Die bei ALGOL sonst übliche tiefgestellte $_{10}$ ist durch den Buchstaben E zu ersetzen.

ELEMENTE, WORTSYMBOLE, ANWEISUNGEN

Zahlkonstante:	Ziffernfolge ohne Dezimalpunkt (z.B. 99 oder -12345). Ziffernfolge mit Dezimalpunkt (z.B. 3.1415 oder -.Ø6789). Vollständige Form (z.B. 8.88E12 oder -.54321E-135).
Name:	Ein Buchstabe, dem beliebig viele Buchstaben oder Ziffern folgen können. Bei Marken-Namen sind nur die ersten 3 Zeichen, bei den übrigen nur die ersten 5 Zeichen von Bedeutung. Verboten sind ALGOL-Namen.
Variable:	Wird durch einen Namen vereinbart. Bei indizierten Variablen stehen ein, zwei oder drei Indizes, durch Komma getrennt, in eckige Klammern [ ] eingeschlossen hinter dem Namen. Indizes können beliebige arithmetische Ausdrücke sein.

Zahlen werden intern mit 2 Worten dargestellt (Genauigkeit $\approx 10^{-8}$; Betrag des Exponenten bei Gleitkommazahlen max.150).

Arithmetischer Ausdruck: Als Operanden sind Zahlkonstanten, Variablen-Namen und Prozedur-Namen zugelassen.

Operatoren:	+	Addition
	-	Subtraktion
	*	Multiplikation
	/	Division
	'POWER'	Potenzierung
Boole'sche Konstanten:	'TRUE'	wahr
	'FALSE'	falsch
Boole'sche Operatoren:	'NOT'	Negation
	'AND'	Konjunktion
	'OR'	Disjunktion
	'IMPL'	Implikation
	'EQUIV'	Äquivalenz
Vergleichsoperatoren:	'LESS'	kleiner als
	'NOT LESS'	nicht kleiner als
	'EQUAL'	gleich
	'NOT EQUAL'	ungleich
	'GREATER'	größer als
	'NOT GREATER'	nicht größer als
Klammerung:	(	öffnende Klammer
	)	schließende Klammer
Wertzuweisung:	:=	ist gleich

Kommentar:	'COMMENT'	folgt Kommentar
Blockvereinbarung:	'BEGIN'	Anfang eines Blocks
	'END'	Ende eines Blocks
Vereinbarungen:	'INTEGER'	Ganzzahl-Variable
	'REAL'	Gleitkomma-Variable
	'BOOLEAN'	Aussage-Variable
	'INTEGER ARRAY'	Feld von Ganzzahl-Variablen
	'ARRAY'	Feld von Gleitkomma-Variablen
	'BOOLEAN ARRAY'	Feld von Aussage-Variablen
	'INTEGER PROCEDURE'	Prozedur mit Ganzzahlen
	'REAL PROCEDURE'	Prozedur mit Gleitkommazahlen
	'LABEL'	Marke
Prozeduren:	'PROCEDURE'	eigentliche Prozedur
	'VALUE'	Wertaufruf
Unbedingter Sprung:	'GOTO'	Sprunganweisung
Bedingung:	'IF'	Vergleichsanweisung
	'THEN'	Folgeanweisung wenn wahr
	'ELSE'	Folgeanweisung wenn falsch
Schleife:	'FOR'	Anfangswert
	'STEP'	Schrittweite
	'UNTIL'	Endwert
	'WHILE'	Bedingung
	'DO'	Laufanweisung

STANDARD-PROZEDUREN

Funktionen:	ABS	Absolutwert
	ENTIER	Ganzzahlwert
	SIGN	Vorzeichen
	LN	Natürlicher Logarithmus
	EXP	Exponentialfunktion
	SIN	Sinus
	COS	Cosinus
	ARCTAN	Arcus tangens
	SQRT	Quadratwurzel
Ein/Ausgaben:	ININTEGER	Eingabe Ganzzahl
	OUTINTEGER	Ausgabe Ganzzahl
	INREAL	Eingabe Gleitkommazahl
	OUTREAL	Ausgabe Gleitkommazahl
	READ	Eingabe beliebige Zahl
	PRINT	Drucken beliebige Zahl in Standard-Format
	TYPE	Drucken Ganzzahl
	WRITE	Drucken Zeichenfolge

CODE-PROZEDUREN

Code-Prozeduren sind im ALGOL-Programm enthaltene Programme in Maschinensprache. Sie werden als Folge von Maschinenbefehlen in oktaler Schreibweise (evtl. Minuszeichen, 6 Oktalstellen) formuliert.

Vereinbarung einer Code-Prozedur:

'PROCEDURE'	<Name>(<Parameter>,...)
'OUTCODE'	<Oktalliste>
'INCODE'	<Oktalliste>
'CODE'	<Oktalliste>

'OUTCODE' bzw. 'INCODE' dienen zur Übergabe von Parametern vom ALGOL-Programm in die Code-Prozedur bzw. in umgekehrter Richtung. Das erste Wort der Oktalliste gibt die Anzahl der folgenden an; diese enthalten die Adressen der zu übergebenden Parameter. Hinter 'CODE' steht das Maschinenprogramm (1.Wort: Anzahl der folgenden; dann erster Befehl, usw.; zuletzt BRY über 1. Wort). Es darf alle Register sowie Plätze in Seite 0 bis Adresse 399 verändern.

Die Code-Prozedur selbst wird im ALGOL-Programm durch ihren Namen, gefolgt von einer Liste der aktuellen Parameter, aufgerufen.

ZUR BEACHTUNG

Jedes ARRAY muß einzeln mit seinen Indexgrenzen vereinbart sein. Dynamische ARRAYs sind nicht erlaubt.

Bei Prozeduren sind keine ARRAYs, Marken oder Prozeduren als formale Parameter erlaubt. Formale Parameter gelten als zu dem Block gehörig, in dem die Prozedur definiert ist; sie dürfen daher nicht mit Variablennamen dieses Blocks übereinstimmen.

Bedingte arithmetische Ausdrücke sind in runde Klammern einzuschließen.

Die Schachteltiefe bei Laufanweisungen ist unbeschränkt.

Bedingte Marken (berechneter Sprung) sind nicht erlaubt. SWITCH ist nicht vorgesehen; stattdessen verwende man (eventuell geschachteltes) IF...THEN...ELSE...GOTO...

Die Laufvariable einer Laufanweisung darf nur eine nichtindizierte Variable sein.

Für formale Parameter von Prozeduren, die nicht in der VALUE-Liste stehen, dürfen keine Konstanten, indizierten Variablen oder arithmetischen Ausdrucke als aktuelle Parameter verwendet werden, sondern nur einfache Variable.

HANDHABUNG

Compiler-Lochstreifen (Anfangsteil) in den (schnellen) Leser einlegen. INT, SENSOR 1, 2, 3, 4, (5) nach unten; RES, END, STA betätigen. Anfangs- und Endadresse (auf dem Lochstreifen vermerkt) über den Fernschreiber 6-stellig oktal eingeben; STA erneut betätigen. Phase 1 wird eingelesen; danach INT und SENSOREN nach oben. RES, END, STA betätigen.

Auf dem Fernschreiber werden nun Fragen ausgedruckt, die der Benutzer beantworten muß. Zuvor ist das Quellprogramm in den benutzten Leser einzulegen.

CORE: 6-stellige Oktalzahl als Endadresse eingeben (größer als ØØ64ØØ).

LISTING: Eingabe YES, wenn das Quellprogramm nach dem Einlesen protokolliert werden soll, sonst NO.

OP.PUNCH: Eingabe YES, wenn nach der Compilierung ein Maschinencode-Streifen des Objektprogramms ausgegeben werden soll (einschließlich Versorgungsprogrammen); sonst NO.

LIMIT: Eingabe YES bewirkt Einschränkung der Variablenliste (Normalfall). Bei großen ARRAYs NO eingeben.

FASTREAD: Eingabe YES, wenn Quellprogramm über schnellen Leser eingelesen werden soll; das Quellprogramm wird eingelesen bis# ; dann Compiler-Lochstreifen (Hauptteil) in Leser einlegen und STA drücken. Das Quellprogramm wird - nach eventueller Protokollierung - durch die sukzessive eingelesenen Compiler-Phasen compiliert. Bei Eingabe NO wird das Quellprogramm über den Teletype-Leser eingelesen; danach Wagenrücklauf eingeben, und die Compilierung erfolgt wie oben beschrieben.

Bemerkung: Falsch beantwortete Fragen können durch Eingabe von DELET ungültig gemacht werden.

Nach der Umwandlung werden zwei Zahlen ausgegeben: Erste freie Adresse nach den Versorgungsprogrammen und erste freie Adresse vor dem Objektprogramm.

Jetzt evtl. benötigte Datenstreifen in den Leser einlegen. Durch Betätigen von STA wird das umgewandelte Programm gestartet.

BEISPIEL EINES ALGOL-PROGRAMMS

```
001   'BEGIN''COMMENT'TEST DER SIMPSON-INTEGRATION MIT SIN;
002   'REAL'ERALT,ERNEU;
003   'INTEGER'M;
004   'REAL''PROCEDURE'SIM(UG,OG,N);
005   'VALUE'UG,OG,N;
006   'REAL'UG,OG;
007   'INTEGER'N;
008   'BEGIN''REAL'H,S1,S2;
009   'INTEGER'K,N1,N2;
010   'REAL''PROCEDURE'I(R);
011   'VALUE'R;
012   'REAL'R;
013   'BEGIN'I:=SIN(R);
014   'END'INTEGRAND;
015   H:=(OG-UG)/(2*N);
016   S1:=S2:=0;
017   N1:=N+N-1;
018   N2:=N+N-2;
019   'FOR'K:=1'STEP'2'UNTIL'N1'DO'S1:=S1+I(UG+K*H);
020   'FOR'K:=2'STEP'2'UNTIL'N2'DO'S2:=S2+I(UG+K*H);
021   SIM:=((OG-UG)/(6*N))*(I(UG)+4*S1+2*S2+I(OG));
022   'END'SIM;
023   ERALT:=0;
024   'FOR'M:=10'STEP'10'UNTIL'M+10'DO''BEGIN'ERNEU:=SIM(0,1.57079
      633,M);
025   PRINT(M,ERNEU);
026   'IF'ABS(ERALT-ERNEU)'LESS'1E-3'THEN''GOTO'ENDE'ELSE'ERALT:=E
      RNEU;
027   'END';
028   ENDE:'END';
029
```

Fehler zur Compilezeit
(mit Angabe der Zeilennummer des Fehlers. Nummer 000: Zeile läßt sich nicht lokalisieren).
Bei geradzahligen Fehlercodes wird die Übersetzung abgebrochen.

100	Speicherüberlauf bei Einlesen der nächsten Phase
102	Speicherüberlauf, da ALGOL-Programm zu lang
104	# fehlt am Ende
106	Interner Überlauf
101	falsches Zeichen
103	nach Wortsymbol fehlt Apostroph
105	unbekanntes Wortsymbol
107	unerlaubtes Wortsymbol nach 'END'
109	falscher String
111	falsche Oktalzahl
113	falsche ARRAY-Vereinbarung
117	zwei Punkte in einer Zahl
119	zwei E in einer Zahl
121	Zahl endet mit Punkt
123	Zahl endet mit E
125	zu großer Exponent
202	'DO' fehlt oder steht an falschem Platz
206	'END' zuviel; auch: ";" oder # fehlt am Ende
208	'UNTIL' fehlt oder steht an falschem Platz
210	String-Überlauf: ALGOL-Programm zu lang
212	'THEN' fehlt oder steht an falschem Platz
214	Klammer- oder 'BEGIN'/'END'-Anzahl stimmt nicht
216	Programm mehr als ca. 30fach geschachtelt
301	Variable nicht in gültigem Block vereinbart
305	Prozedurname nicht vereinbart
309	ARRAY-Name nicht vereinbart
313	Variable aus Spezifikationsliste fehlt in Parameterklammer
315	Komma oder Klammer fehlt in Parameterteil
317	nach Komma folgt keine Variable
321	Variablenname im selben Block mehrfach vereinbart
323	Prozedurname im selben Block mehrfach vereinbart
325	Strichpunkt fehlt
327	Variable aus VALUE-Liste fehlt in Parameterklammer
329	Komma oder Strichpunkt fehlt in Vereinbarungsliste
331	Zahl der aktuellen Parameter bei Prozeduraufruf falsch
335	desgleichen
340	Überlauf: ALGOL-Programm zu lang
341	Markenname schon im selben Block vereinbart
350	Überlauf der Konstantenliste (> 64 Zahlkonstanten)
352	Überlauf der Variablenliste (> 512 Variable)
354	Überlauf der Markenliste (> 128 Marken)
356	desgleichen
358	Überlauf der Prozedurliste (> 18 Prozeduren)

360 Überlauf der ARRAY-Liste ($>$ 12 ARRAYs)
362 Überlauf ($>$ 32 ineinandergeschachtelte Blöcke)
364 desgleichen
366 Markenname nicht in gültigem Block vereinbart
400 Kellerüberlauf: zu großer arithmetischer Ausdruck
401 Klammer auf zu wenig
403 Klammer auf zu viel
500 nicht definierbarer Fehler im String
502 nicht mögliche Wertzuweisung an eigentliche Prozedur
510 Stringüberlauf: Programm zu lang

Fehler zur Objektzeit

0 Argument von LN $\leqslant 0$
2 Argument von EXP zu groß (größer als $512 \cdot \ln 2 = 355$)
18 Argument von SQRT < 0
20, 22, 24 bei A 'POWER' B
- A = 0, B < 0
- A = 0, B = 0
- A < 0, B real oder integer > 31

26 Zahl zu groß für TYPE (wie bei 53)
52 Division durch Null
53 Zahl zu groß für interne real-integer-Wandlung
54 Zahlbereichsüberschreitung bei Rechenoperation (Betrag größer als ca. 10^{150} bzw. kleiner als ca. 10^{-150})

BASIC

VORBEMERKUNG

BASIC ist eine Programmiersprache für mathematische und technisch-wissenschaftliche Aufgaben. Sie ist besonders leicht lernbar und bietet - wenn, wie beim MINCAL BASIC, die Übersetzung durch einen Interpreter mit Editor-Teil geschieht - dem Benutzer die Möglichkeit, im Dialogbetrieb zu arbeiten. Programme können bequem eingegeben, getestet, korrigiert und schnell ausgeführt werden.

Man unterscheidet bei BASIC die Programmsprache (mit Anweisungen als Sprachelemente) und die Kommandosprache (für den Benutzer-Rechner-Dialog).

Rechner-Ausstattung:	MINCAL 523 Grund- + erweiterter Befehlsvorrat Programmierhilfe X00 Kernspeicher ab 4 k (8 k empfohlen) Konsol-Fernschreiber (oder Datensichtgerät) an Ebene 0 als Dialoggerät Schnelle Lochstreifenausrüstung an Ebene 0 (empfohlen)
Interpreter:	Interpreter (mit Editor) übersetzt jeweils 1 aktuelle Anweisung des als komprimiertes BASIC im Speicher enthaltenen Programms Der Interpreter arbeitet auf Ebene 0.

Soweit nicht nachstehend anders beschrieben, entspricht MINCAL BASIC der Dartmouth-Konvention.

EINLESEN DES INTERPRETERS

Interpreter-Lochstreifen in den (schnellen) Leser einlegen. INT, SENSOR 1, 2, 3, 4, (5) nach unten; RES, END, STA betätigen. Anfangs- und Endadresse (auf dem Lochstreifen vermerkt) über den Fernschreiber 6-stellig oktal eingeben; STA erneut betätigen. Der Lochstreifen wird eingelesen. Danach INT und SENSOREN nach oben. RES END, STA betätigen.

Das System ist betriebsbereit für die Eingabe von Anweisungen oder Kommandos.

KOMMANDOS

Kommandos sind über die Tastatur einzugebende Aufforderungen des Benutzers an das System, bestimmte Arbeiten auszuführen; die Ausführung geschieht unmittelbar, nachdem das Kommando durch die Eingabe von "Wagenrücklauf" abgeschlossen wurde. Kommandos bestehen aus einer Buchstabenfolge (wovon nur die ersten 3 von Bedeutung

sind); danach können eine oder zwei 1- bis 4-stellige Zahlen (m, n) folgen, die sich auf die Anweisungsnummern eines vorher eingegebenen, gespeicherten BASIC-Programms beziehen.

LIST	Gesamtes Programm listen	
LIST m,n	Programm von Anweisung m bis n listen	
LIST m	Anweisung m listen	
PUNCH	Gesamtes Programm lochen	} (schneller Leser)
PUNCH m,n	Programm von Anweisung m bis n lochen	} (schneller Leser)
PUNCH m	Anweisung m lochen	} (schneller Leser)
READ	BASIC-Programm einlesen und dem vorhandenen hinzufügen	} (schneller Leser)
SCRATCH	Gesamtes Programm löschen	
DELETE m,n	Programm von Anweisung m bis n löschen	
DELETE m	Anweisung m löschen	
RUN	Programm starten (bei der niedrigsten Anweisungsnummer)	
RUN m	Programm starten bei Anweisung m	

Fehlerhafte Kommandozeichen können durch Eingabe eines oder mehrerer Linkspfeile (←) rückwirkend gelöscht, das ganze Kommando durch Eingabe von DEL ungültig gemacht werden (vor Eingabe "Wagenrücklauf"). Erkennbare Fehler werden gemeldet (s. Fehlerliste).

Kommandos für Einfügen oder Ändern von Anweisungen gibt es nicht. Zu ändernde Anweisungen werden durch Neueingabe mit der gleichen Nummer ersetzt; für Einfügungen benutze man Nummern, die zwischen den Nummern der vorangehenden und der nachfolgenden Anweisung liegen.

Kommandos können in beliebiger Folge gegeben werden oder sich mit der Eingabe von Programmzeilen (Anweisungen) abwechseln.

PROGRAMMZEILEN

Programmzeilen bestehen aus einer 1- bis 4-stelligen Zahl (der Anweisungs- oder Zeilen-Nummer), gefolgt von der eigentlichen Anweisung in BASIC-Programmsprache. Jede Zeile enthält eine in sich abgeschlossene Anweisung, die mit einem Statement-Code beginnt:

<Nummer> <Statement-Code> <Spezifikation>

Das Programm besteht aus der Gesamtheit dieser Anweisungen, die in aufsteigender Reihenfolge der Nummern argearbeitet werden (soweit keine Sprünge in Betracht kommen).

Programmzeilen werden einzeln eingegeben und durch "Wagenrücklauf" (cr) abgeschlossen, z.B.:

ZØ LET C = A + B (cr)

worauf diese Anweisung gespeichert ist. Danach kann eine neue Programmzeile eingegeben oder auch ein Kommando erteilt werden.

Fehlerhafte Zeichen können durch Linkspfeil (←), ganze Zeilen durch Eingabe von DEL eliminiert werden.

STATEMENT-CODES

Diese Elemente von BASIC kennzeichnen die Art einer Anweisung. Sie stehen (hinter der Nummer) am Anfang einer Programmzeile. Sie bestehen aus einer Buchstabenfolge.

REM

m REM Kommentar

Dient zur Einfügung von Bemerkungen (beliebige ASCII-Zeichen) in das Programm. Bei der Programmausführung wird diese Zeile übergangen.

Beispiel:

```
1ØØ REM DIES IST EIN KOMMENTAR
```

LET

m LET Variable = Ausdruck
m LET Variable 1 = Ausdruck 1, Variable 2 = Ausdruck 2
m LET Variable 1 = Variable 2 = ... = Ausdruck

Einer Variablen wird der Wert des rechts vom Gleichheitszeichen stehenden Ausdruckes zugewiesen. Derselbe Wert kann mehreren Variablen zugewiesen werden; außerdem ist es möglich, verschiedene (durch Komma getrennte) Zuweisungen mit einem LET-Statement vorzunehmen.

Beispiele:

```
  1Ø LET A = 5.Ø2
 2ØØ LET D = (3/C2  N)/(A+SIN(1))
14ØØ LET C = A=B1=D3=Ø, D = E=F7=1-SIN(x)
  5Ø LET B = B+1, A = A+1
```

INPUT

m INPUT Variable 1, Variable 2, ...
m INPUT "Nachricht" Variable 1, Variable 2, ...

Mit INPUT werden einer oder mehreren Variablen Werte zugewiesen, die vom Benutzer über die Tastatur einzugeben sind. Bei Ausführung dieser Anweisung wird ein Fragezeichen (?) ausgegeben. Daraufhin

gibt der Benutzer die Zahlen in beliebigem Format, getrennt durch Kommata, ein; am Ende ist "Wagenrücklauf" einzugeben. Waren es zu wenig Werte, wird ?? gemeldet, und der Benutzer kann die Eingabe vervollständigen. Waren es zu viel, wird eine Fehlermeldung ausgegeben, das Programm jedoch fortgesetzt. Die eingegebenen Werte werden der Reihe nach den Variablen zugeordnet. Fehlerhafte Eingaben können durch Linkspfeil (←) je Zeichen oder durch DEL je Wert eliminiert werden.

Eine in Anführungsstrichen hinter INPUT stehende Nachricht (aus beliebigen ASCII-Zeichen) wird vor der Eingabebereitschaft ausgegeben. Gibt der Benutzer STOP anstelle einer Zahl ein, wird das Programm beendet.

Beispiele:

```
1Ø INPUT  A,B
2Ø INPUT "GIB A1, B1, B2 EIN" A,B C
```

PRINT

m PRINT
m PRINT "Text"
m PRINT Ausdruck 1, Ausdruck 2, ...
m PRINT "Text" Ausdruck 1, Ausdruck 2, ...

Mit PRINT werden Texte oder Zahlenwerte ausgegeben. PRINT ohne weitere Angaben bedeutet "neue Zeile". Im übrigen können Texte (in Anführungszeichen gesetzte, beliebige ASCII-Zeichen) und Ausdrucke in beliebiger Reihenfolge vorkommen; sie werden in der geschriebenen Reihenfolge verarbeitet und ausgegeben.

Für jeden Ausdruck wird der Wert ermittelt und ausgegeben; falls nicht anders spezifiziert, in folgendem Format: -Ø.xxxxxxE-xxx (normales E-Format). Kommata (,) zwischen den Ausdrucken bewirken, daß die Werte nacheinander in 4 festen Spalten ausgegeben werden (entsprechend Zeichenspalte 1, 17, 33 und 49). Ein Semikolon (;) statt eines Kommas unterbindet diese Funktion und läßt den Schreibkopf stehen. Beide Zeichen können auch am Ende der PRINT-Anweisung stehen. Bei Erreichen der Zeichenspalte 65 wird selbsttätig auf eine neue Zeile geschaltet.

Beispiele:

```
1Ø PRINT
2Ø PRINT A,B(5),C+D+1 E*X
3Ø PRINT "ERGEBNIS=";A3+B4
```

m PRINT TAB (Ausdruck) Liste

Steht hinter PRINT das Wort TAB mit einem Ausdruck in Klammern danach, so wird der Schreibkopf in die Zeilenspalte tabuliert, deren Nummer sich aus dem Wert des Ausdruckes ergibt. Von da an wird die Liste gedruckt, die sich wiederum in beliebiger Weise aus Texten und Ausdrucken zusammensetzen kann.

Beispiel:

```
2Ø PRINT TAB (3Ø*SIN(x)+30.5)"+"
```

m PRINT FOR (Format 1) Liste 1, FOR (Format 2) Liste 2, ...

Steht hinter PRINT das Wort FOR mit einer in Klammern eingeschlossenen Formatspezifikation danach, so werden die Ausdrucke der folgenden Liste in besonderen Formaten gedruckt. FOR kann in einer Anweisung beliebig oft angegeben werden.

Die Formatspezifikationen lauten (in Analogie zu FORTRAN) Fw.d (F-Format) oder Ew.d (E-Format mit Exponent), wobei für w (Feldweite ein schließlich aller Zeichen) und d (Stellen hinter Dezimalpunkt) je eine Ziffer vorzusehen ist. Eine Formatspezifikation gilt so lange, bis eine neue erfolgt. Mit FOR (EØ.6) kehrt man zum normalen E-Format zurück.

Beispiele:

```
1Ø PRINT FOR(F5.2) 7,FOR(F6.1)-4+7
2Ø PRINT FOR(F5.0)5ØØ.1
5Ø PRINT FOR(E5.2)1Ø1E+3
```

Folgende Werte werden ausgegeben:

```
   7.ØØ        11.Ø
    5ØØ
Ø.1ØE+ØØ6
```

GOTO

m GOTO n

Das Programm wird durch GOTO mit der Anweisung fortgesetzt, vor der die angegebene Nummer n steht.

Beispiel:

```
1Ø GOTO 1ØØ
```

m GOTO Ausdruck OF n1, n2, ...

Das berechnete GOTO führt zu einem Spring auf die Anweisung mit der Nummer n1 oder n2 oder ..., je nachdem, ob der Wert des hinter GOTO stehenden Ausdrucks gleich 1 oder 2 oder ... ist. Wenn der Wert des Ausdrucks kleiner als 1 oder größer als die Anzahl der Nummern in der Liste ist, so wird die folgende Anweisung ausgeführt. Nicht ganzzahlige Werte des Ausdrucks werden abgerundet.

Beispiel:

```
2Ø GOTO A+B+1 OF 1ØØ,2Ø,5Ø
```

IF

m IF Bedingung THEN n

Das Programm verzweigt auf die Anweisung mit der hinter THEN stehenden Nummer n, wenn die hinter IF stehende Bedingung erfüllt ist; andernfalls wird die folgende Anweisung ausgeführt.

Die Bedingung kann aus einem Ausdruck bestehen oder aus einem Vergleichsausdruck (mit den Vergleichsoperatoren < , <=, =, => , > ,#) oder einer Boole'schen Verknüpfung aus diesen. Sie gilt als erfüllt, wenn der arithmetische Wert ungleich Null bzw. das Boole'sche Ergebnis gleich Eins ist.

Beispiele:

```
2Ø IF X/5>=N THEN 6Ø
3Ø IF SIN(A)<=COS(A) AND SIN(B)# .5 THEN 1Ø
5Ø IF A+B+1<C6 THEN 7Ø
8Ø IF A AND B OR C# Ø THEN 35 }
85 IF A AND B OR C#THEN 35    } identisch!
9Ø IF A=B AND C#D OR B>=Ø THEN 55
```

FOR

m FOR Variable = Ausdruck 1 TO Ausdruck 2
m FOR Variable = Ausdruck 1 TO Ausdruck 2 STEP Ausdruck 3

Mit FOR wird eine Programmschleife eröffnet. Die (Lauf-) Variable wird auf einen Anfangswert gesetzt, der sich aus dem Ausdruck 1 ergibt, und später durch NEXT um jeweils den Wert von Ausdruck 3 erhöht (entfällt dieser, so wird sie um 1 erhöht). Die Schleife wird so oft durchlaufen, bis die Variable den Wert des Ausdrucks 2 erreicht hat, mindestens jedoch einmal. Auf die FOR-Anweisung folgen die Anweisungen der Schleife; sie gehen bis zur zugehörigen NEXT-Anweisung.

Zu beachten ist:
Die Anfangswerte der Lauf-Variablen können beliebig sein, ebenso die Schrittweiten (jedoch ≠ 0).
Schleifen-Schachtelungen sind bis zu einer Tiefe von 10 erlaubt; sie werden von innen nach außen abgearbeitet. Überkreuzte Schleifenschachtelungen sind verboten.
Die Lauf-Variable ist nur für die Schleife selbst definiert.

Beispiele: siehe NEXT.

NEXT

m NEXT Variable

NEXT ist die letzte Anweisung einer Schleife. Die Variable muß mit der Lauf-Variablen der zugehörigen FOR-Anweisung übereinstimmen.
Durch NEXT wird die Schrittweite vor Lauf-Variablen addiert und geprüft, ob der Endwert der Lauf-Variablen erreicht ist. Ist dies der

Fall, so wird die folgende Anweisung ausgeführt; andernfalls wird zu der Anweisung hinter der zugehörigen FOR-Anweisung zurückverzweigt.

Beispiele:

```
1Ø FOR I=1 TO 1Ø STEP 2
...
3Ø NEXT I

1Ø FOR J=1Ø TO -2 STEP -1.2
2Ø FOR K=A/B+C * SIN(x) TO A+1Ø
...
3Ø NEXT K
4Ø NEXT J
```

```
1Ø FOR I=1 TO 1Ø
2Ø FOR K=1 TO 1Ø
...
NEXT I
NEXT K
```

falsch! Überkreuzte Schachtelung

GOSUB

m GOSUB n

Durch GOSUB wird ein Unterprogramm aufgerufen, das mit der Anweisung n beginnt. Nach Ausführung des Unterprogramms wird das Hauptprogramm mit der auf GOSUB folgenden Nummer fortgesetzt. GOSUB-Anweisungen können bis zu einer Tiefe von 6 geschachtelt werden.

Beispiel:

```
2Ø GOSUB 1ØØ
```

m GOSUB Ausdruck OF n1, n2, ...

Das berechnete GOSUB ruft je nach dem Wert des Ausdrucks eines der bei n1, n2, ... beginnenden Unterprogramme auf (s. auch berechnetes GOTO).

Beispiel:

```
2Ø GOSUB A OF 1ØØ,11Ø,12Ø
```

RETURN

m RETURN

Mit RETURN wird aus einem Unterprogramm in das Hauptprogramm zurückgesprungen. Jedes Unterprogramm muß wenigstens ein RETURN enthalten.

Beispiel:

```
13Ø RETURN
```

CALL

m CALL name (par1, par2, ...)

Mit CALL wird ein in Maschinensprache geschriebenes Programm (Code-Prozedur) als Unterprogramm aufgerufen. Der Name der Code-Prozedur besteht aus einem Buchstaben, gefolgt von 2 Buchstaben oder Ziffern. Als Argument können bis zu 7 Parameter übergeben werden; sie werden mit dem Wert der Parameterausdrucke übergeben.

Beispiele:

```
1Ø CALL ADU (A+B,1Ø * 2,5 * SIN(x))
2Ø CALL INT (1,1,1ØØ)
```

DIM

m DIM dim Variable 1, dim Variable 2, ...

Mit DIM werden ein- oder zweidimensionale Variablen-Felder definiert und im Speicher reserviert. Die Dimension wird hinter den Feldnamen angegeben; Dimensionsangaben sind ganzzahlige Konstanten und dürfen nicht größer als 511 sein.
Die DIM-Anweisung muß der erstmaligen Verwendung einer indizierten Variablen vorausgehen; die Zahl der Indizes muß übereinstimmen.
Variablen-Felder werden durch DIM auf Nullinhalt gesetzt.

Beispiel:

```
1Ø DIM A (3,1Ø),B(5),C3(1Ø,1Ø)
```

DEF

m DEF FN Buchstabe (Variable) = Ausdruck

Mit DEF kann der Benutzer eigene Funktionen definieren, die im Programm als Funktionsnamen (ähnlich Standardfunktionen) auftauchen.
Die Funktion muß FN, gefolgt von einem Buchstaben, sein. Außerdem ist eine Variable als Argument sowie ein Ausdruck anzugeben, dessen Wert der Funktion im Augenblick des Aufrufs zugewiesen wird.
In einer DEF-Anweisung darf nur eine Funktion definiert werden.

Beispiele:

```
 6Ø DEF FNA(B)=A+2/B+C
1ØØ DEF FNF(X)=SIN(A* X+B)
```

DATA

m DATA Zahl 1, Zahl 2, ...

DATA eröffnet eine Liste von Konstanten. Diese Daten können durch eine READ-Anweisung dessen Argumenten (Variablen) zugewiesen werden. Sind mehrere DATA-Listen vorhanden, so werden alle Konstanten zu einem Block gehörig betrachtet (in der Reihenfolge der Auflistung bzw. der Zeilen-Nummern).

Die Konstanten sind Dezimalzahlen in beliebigem Format.

Beispiel:

```
1Ø DATA 1ØØ,5.24,-3.ØE+5, 1
```

READ — m READ Variable 1, Variable 2, ...

READ weist Konstanten aus dem DATA-Block den Variablen seiner Argumentliste zu. Dabei wird den Variablen in der Reihenfolge ihrer Auflistung bzw. der Folge der READ-Anweisungen ein Wert des DATA-Blocks nach dem anderen zugeordnet.

Beispiel:

```
 1Ø DATA 4,2,3
 2Ø DATA 9,8,1
1ØØ READ A,B,C,D
 9Ø READ E,F
```

Es wird zugewiesen:
4 → E, 2 → F, 3 → A,
9 → B, 8 → C, 1 → D

STOP — m STOP
END — m END

STOP und END beenden den Programmablauf. END wird üblicherweise am physikalischen Ende des Programms benutzt, während STOP innerhalb des Programms benutzt wird.
Bei Erreichen von STOP oder END wird DONE ausgegeben. Das System wartet auf ein neues Kommando.

SPRACHELEMENTE

Zahlen: Alle Werte werden intern durch Gleitkommazahlen dargestellt. Die Genauigkeit der Mantisse ist 10^{-8}; der (Zehner-) Exponent darf den Betrag 154 nicht übersteigen.

Konstanten: Ganzzahlen, z.B. +123, -267894, 5
Gleitkommazahlen im F-Format, z.B. 12.45, -Ø.ØØØØ12, -.5
Gleitkommazahlen im E-Format, z.B. -12 E-2, 12.34E5Ø, .8E+3

Variablen: Buchstabe oder Buchstabe, gefolgt von Ziffern, z.B.: A, Z, B3, X9
Bei indizierten Variablen stehen dahinter, in Klammern eingeschlossen, ein oder zwei (durch Komma getrennte) Ausdrucke, welche die Lage im Variablen-Feld angeben; z.B. A(5),B6(2.3+Y,Z/3), D8(X+7)

Operatoren: arithmetische:

+	Addition	
-	Subtraktion	
*	Multiplikation	
/	Division	
↑	Potenzierung	(X ↑ 2 bedeutet "X hoch 2")

logische:	NOT	Negation	
	AND	Konjunktion	
	OR	Disjunktion	
	MIN	Kleinstwert	(A MIN B ergibt A wenn A $\leqslant$ B und B wenn B < A)
	MAX	Größtwert	(A MAX B ergibt A wenn A $\geqslant$ B und B wenn B > A)
Vergleich:	<	kleiner	
	<=	kleiner oder gleich	
	=	gleich	
	#	ungleich	
	>=	größer oder gleich	
	>	größer	

Funktionen: mathematisch:

ABS	Absolutwert	
INT	Ganzzahl-Teil	
SGN	Vorzeichen	(=1 wenn Argument positiv; sonst = -1)
SQR	Quadratwurzel	
SIN	Sinus	
COS	Cosinus	
TAN	Tangens	
ATN	Arcus tangens	
LOG	Logarithmus	
EXP	Exponentialfunktion	

sonstige:

RND Zufallsfunktion

(Es ist ein Pseudo-Randomgenerator vorgesehen. Er erzeugt:
RND(Ø) gleichverteilte Zufallszahl zwischen Ø und 1
RND(-1) zufälliges Setzen zweier Startwerte
RND(1) normalverteilte Zufallszahl

Klammerung: Mit runden Klammern ( ) beliebig zulässig.

FEHLERMELDUNGEN

Vom Interpreter erkannte Fehler werden (bei Anweisungen erst während der Durchführung des Programms) durch die Nachricht

m ERR xx

gemeldet, wobei m die Zeilen-Nummer der fehlerhaften Anweisung, xx den nachstehenden Fehlercode bedeutet.

Fehler, die nicht unmittelbar zur Programmunterbrechung führen:

1 Bei INPUT wurden zu viele Daten eingegeben.
2 Argument von SQR oder LOG ist negativ. Es wird der Absolutwert genommen.
3 Argument von SIN, COS ist größer als 5×10^5, von EXP größer als 128 oder LOG gleich 0. Es wird mit den Werten 0 bzw. 10^{154} bzw. -10^{154} weitergerechnet.
4 Berechneter Wert ist absolut größer als 10^{154}. Es wird mit dem Maximalwert weitergerechnet.

Fehler bei Ausdrucken:

5 Auf Operator folgt).
6 Ausdruck beginnt mit *, /, ↑ , Vergleichsoperator, AND, OR, MIN oder MAX.
7 Funktionsargument steht nicht in Klammern.
8 Auf eine Konstante folgt Konstante, Variable oder Funktion.
9 Auf eine Konstante folgt (.
10 Auf eine Variable folgt NOT.
11 Auf eine Variable folgt Variable, Konstante oder Funktion.
12 2 Operatoren in unerlaubter Reihenfolge.
13 Vor dem Zuordnungsoperator (=) steht keine Variable bzw. keine Funktion.
14 Es fehlt eine rechte Klammer.
15 Es fehlt eine linke Klammer.

Fehler bei BASIC-Anweisungen:

16 Variable, Daten, Ausdrucke bei READ, INPUT, PRINT, DISPLAY, DATA nicht richtig getrennt oder abgeschlossen.
17 Kein richtiges Argument bei PRINT FOR oder TAB oder Zahl paßt nicht ins F-Format.
18 Die Variable nach NEXT entspricht nicht der Lauf-Variablen der aktiven FOR-Schleife.
19 Die DATA-Liste ist erschöpft. READ findet keine Daten mehr.
20 Nach LET, DEF, FOR fehlt der Operator =.
21 Nach IF, GOTO, GOSUB steht keine oder keine existierende Zeilennummer.
22 Kein richtiger Ausdruck nach LET oder PRINT.
23 In FOR ist die Schrittweite 0.
24 Nach DIM oder FOR fehlt eine Variable.
25 Nach IF fehlt THEN oder nach FOR fehlt TO bzw. STEP oder nach DEF fehlt FN.
26 Zu RETURN gibt es kein GOSUB.
27 Mehr als 6 Unterprogramme oder FOR-Schleifen tiefer als 10fach verschachtelt.
28 Gelöschte Bibliotheksfunktion wird verwendet.
29 Eine Funktion FN oder ein Name in CALL ist nicht definiert oder die Parameterzahl von CALL stimmt nicht mit der Listenangabe überein.
30 Fehlerhafte Angaben in DIM-Anweisungen.

31 Indizierte Variable ist in DIM nicht definiert.
32 Indizierte Variable hat falsche Dimensionsangabe oder Index überschreitet Dimensionsangaben.
33 Dimensionsangabe in DIM ist > 511.
34 Speicherüberlauf, da zu große Felder.
35 Speicherüberlauf, da zu viele Variablen.
36 Statement-Code existiert nicht.

Fehler bei Kommandos (m ist ohne Bedeutung):

37 Speicherüberlauf, da Programm zu lang.
38 Zeile länger als 72 Zeichen.
39 Zeilen-Nummer > 9999.
40 Zeilen-Nummer existiert nicht.
41 Kommando existiert nicht.

```
READ
LIST
0100   REM PROGRAMMBEISPIEL 2
0101   REM PLOTTEN EINER FUNKTION AUF DEM FERNSCHREIBER
0110   DEF FNF(X)=SIN(X)*EXP(-0.1*X)
0115   FOR I=0 TO 15 STEP .5
0120   PRINT TAB(30.5+15*FNF(I));"*"
0140   NEXT I
0150   END

RUN
                                *
                                        *
                                            *
                                              *
                                            *
                                        *
                                    *
                                *
                              *
                             *
                             *
                                 *
                                     *
                                         *
                                             *
                                             *
                                           *
                                         *
                                       *
                                     *
                                     *
                                      *
                                        *
                                       *
                                         *
                                           *
                                          *
                                          *
```

Library

VORBEMERKUNG

Im folgenden sind einige weitere Programmiersysteme und Unterprogrammpakete erwähnt, die zur Basis-Software der Computer MINCAL 513/523 gehören. Ausführliche Beschreibungen hierzu sind auf Wunsch erhältlich.

Betriebssysteme für periphere Geräte sind im Kapitel "Peripherie" erwähnt.

DOPPELWORT-PAKET

Dieses Paket enthält in Mikroprogramm-Technik realisierte Befehle für

Doppelwort - Laden
" - Transfer
" - Addition
" - Subtraktion
" - Multiplikation
" - Division
" - Dezimal-Binär-Konversion (Rechts- und Linkskomma)
" - Binär-Dezimal-Konversion (Rechts- und Linkskomma)

Es ist 512 Worte lang und kann entweder in den ersten 4 k des Kernspeichers abgelegt oder in Form einer Festspeicher-Einheit implementiert sein.

Die interne Zahlen-Darstellung ist 2 Worte lang (Vorzeichen + 36 Datenbits).

EINWORT-GLEITKOMMA-PAKET

Dieses Paket enthält Unterprogramme für die Ausführung folgender Befehle:

Gleitkomma - Addition
" - Subtraktion
" - Multiplikation
" - Division
" - Dezimal-Konversion
Festkomma - Gleitkomma-Konversion
Gleitkomma - Quadratwurzel

Es ist 300 Worte lang und kann entweder im Kernspeicher abgelegt oder - zusammen mit der Programmierhilfe - als Festspeicher-Einheit XØ1 implementiert werden.

Die interne Zahlen-Darstellung ist 1 Wort lang (Vorzeichen + 12 bit Mantisse, 6 bit Charakteristik).

DOPPELWORT-GLEITKOMMA-PAKET

Dieses Paket enthält Unterprogramme für

- die 4 Grundrechnungsarten, sowie Potentzierung
- Ein/Ausgabe-Konversionen
- Mathematische Standardfunktionen

} auf Wunsch

Die interne Zahlen-Darstellung ist 2 Worte lang: (Vorzeichen + 27 bit Mantisse, Vorzeichen + 9 bit Exponent). Dies entspricht einer Mantissen-Genauigkeit von ca. 10^{-8} und einem Zahlenbereich von ca. $10^{\pm 150}$.

C ALCULATOR

Der CALCULATOR ist ein Programm zum Betrieb des MINCAL 523 als Tischrechner. Er erlaubt die Berechnung von mathematischen Gleichungen, in denen die 4 Grundrechnungsarten, Potenzierung, 8 Standardfunktionen, beliebige Klammerung und Konstanten vorkommen dürfen. Zwischen- und Endergebnisse werden sofort ausgedruckt; außerdem ist die wiederholte Berechnung von Ausdrucken mit einer bei konstanter Schrittweite veränderten Variablen möglich.

EDITOR-BETRIEBSSYSTEM

Dieses Programm ist für MINCAL 523-Systeme mit Externspeicher (Trommel oder Platte) vorgesehen. Es dient zur Archivierung und zum Aufruf von Programmen in ASSEMBLER- oder Formalsprachen (FORTRAN, ALGOL), zu deren Ein- und Ausgabe und Änderung sowie zum Aufruf des Assemblers bzw. der Compiler.

Die Quellprogramme werden im Kernspeicher als Dateien dynamisch verwaltet und sind im Externspeicher durch den Benutzer änderbar.

Kernspeichererweiterung

Im Gehäuse des Computers MINCAL 523 können bis zu 8 kWorte Kernspeicher untergebracht werden. Wird eine darüberhinausgehende Kapazität benötigt, so sind zusätzliche Speicher-Einschübe erforderlich; es ist dann eine Erweiterung der Speicherkapazität auf insgesamt 32 kWorte möglich.

Technische Daten

Kernspeicher:	1, 2 oder 3 Kernspeichereinheiten zu je 4 kWorten à 20 bit Zugriffszeit 0.6 us Vollzyklus 1.5 us Adressen: 20000...47777 (1. Speichereinheit) 50000...77777 (2. Speichereinheit)
Stromversorgung:	eingebaut
Netzanschluß:	220 V ± 10 % 50 Hz Leistungsaufnahme ca. 600 VA
Größe:	19"-Einschub (6 Einheiten) + Stromversorgung (4 Einheiten) } ca. 440 mm hoch 575 mm tief
Anschluß:	an Zentraleinheit MINCAL 523 Kernspeicher-Erweiterungs-Kanal über Flachkabel (in unmittelbarer Nähe)

Die Speichereinheit ist wie die Zentraleinheit MINCAL 513/523 aus einem vorderen und hinteren Rahmen aufgebaut. Die Kernspeicher belegen den vorderen Rahmen. Der hintere Rahmen kann zwei Massenspeicher-Interfaces aufnehmen (z.B. Magnettrommel-Interface + Magnetband-Interface).

Periphere Speichersysteme

VORBEMERKUNG

An den Computer MINCAL 523 können Magnettrommel-, Magnetplatten- und Magnetbandspeicher angeschlossen werden, die zur externen Speicherung von Daten und Programmen dienen. Trommel- und Plattenspeicher sind als Systemspeicher zu verstehen, während Magnetbandgeräte vor allem zur Archivierung großer Datenmengen dienen.

TROMMELSPEICHER-SYSTEM

Trommelspeicher:	Festkopf-Magnettrommelspeicher Kapazität 32 k oder 128 kWorte (19 bit + Parity) 32 oder 128 Spuren zu je 1 kWorten 4 Sektoren mit je 256 Worten je Spur mittlere Zugriffszeit 10 ms Transferrate ca. 1 MHz entsprechend ca. 50 kWorten/s 19"-Einschub, 13 Einheiten hoch mit eigener Stromversorgung
Controller:	Interface-Einheit zum Betrieb von 1 bis 4 Trommeln zum Anschluß an den Computer über X-Kanal und DMA-Kanal eingebaut in eine Speichereinheit
Software:	MINCDOS 500 Trommel-Betriebssystem mit Executive (Lese/Schreibsteuerung) und Operating (Bedienungssystem)

MINCDOS 500 EXECUTIVE

EXECUTIVE ist der Teil des Trommel/Platten-Betriebssystems, welcher die Übertragung von Blöcken beliebiger Länge und Lage zwischen Kernspeicher und Externspeicher ausführt.

Lesen (vom Externspeicher in den Kernspeicher) und Schreiben (in umgekehrter Richtung) wird jeweils durch eine spezielle Instruktion bewirkt, der 3 Parameter-Worte folgen:

```
542LL0      LESEN vom Externspeicher
IAAAAA      Kernspeicher-Basisadresse
JAAAAA      Externspeicher-Basisadresse
IAAAAA      Blocklänge

542LL4      SCHREIBEN auf Externspeicher
IAAAAA      Kernspeicher-Basisadresse
JAAAAA      Externspeicher-Basisadresse
IAAAAA      Blocklänge
```

Der erste Parameter gibt die erste Adresse des Speicherfeldes im Kernspeicher an; ist die 1. Oktalstelle I gleich 1, 2 oder 3, so wird zu AAAAA der Inhalt vom Indexregister 1, 2, oder 3 addiert. In gleicher Weise wird der dritte Parameter behandelt, der die Länge des zu übertragenden Datenblocks (in Worten) angibt.

Der zweite Parameter gibt die erste Adresse des Blocks im Externspeicher an; enthält die erste Oktalstelle J eine 4...7 (Bit 17), so wird zu den restlichen Bits der Inhalt vom Indexregister 1 addiert.

Nachdem die effektiven Basisadressen und die effektive Blocklänge ermittelt sind, prüft EXECUTIVE, ob direkter Transfer möglich oder indirekter Transfer nötig ist. Direkter Transfer bedeutet Lesen in bzw. Schreiben aus dem Feld mit der angegebenen Kernspeicher-Basisadresse; er ist nur möglich, wenn die Externspeicher-Basisadresse gleich einem Sektoranfang und die Blocklänge gleich 256 (= 400_8) bzw. ein Vielfaches davon ist. Andernfalls wird der Sektor, in dem sich adressierte Worte befinden, in das Datenfeld CTB (COMMON TRANSFER BLOCK) transferiert; beim Lesen werden die interessierenden Worte dann in das angegebene Speicherfeld übertragen, während beim Schreiben die betreffenden Worte im CTB überschrieben und der veränderte Block aus dem CTB auf den Sektor des Externspeichers zurücktransferiert wird.

Mit dieser Methode ist außer der Übertragung von ganzen Sektoren (die am schnellsten vor sich geht) auch der Transfer von Einzelworten und Blöcken beliebiger Lage und Länge möglich. Enthält ein Block einen oder mehrere ganze Sektoren, wendet EXECUTIVE direkten Transfer an, auch wenn am Anfang oder Ende des Blocks indirekter Transfer nötig ist.

Nach Auslösen des Lese- oder Schreibvorgangs geht EXECUTIVE zum Anfang und hält an. Nach Rückmeldung vom Externspeicher (Sektor transferiert) wird über das Überschreiben von CTB und erneuten Transfer entschieden; oder CTB wird ausgelesen bzw. der Einzeltransfer ist beendet. Ist der Block noch nicht fertig bearbeitet, so wird mit dem nächsten Sektor begonnen.

Sind mehrere Externspeicher angeschlossen (bis zu 4 möglich), so wird die Geräte-Nummer (0...3) in den letzten 2 Bits des Befehls angegeben, z.B.:

```
542LL2      LESEN vom Externspeicher 2
542LL7      SCHREIBEN auf Externspeicher 3
```

Im symbolischen Programm können die Befehle

RD	(READ FROM DRUM/DISC = LESEN) bzw.
WD	(WRITE ON DRUM/DISC = SCHREIBEN)

benutzt werden.

EXECUTIVE kann 4 Fehlerarten feststellen:

1 WRITE LOCKOUT	(zu beschreibende Spur ist geschützt)
2 PARITY	(gelesener Sektor enthielt Parity-Fehler)
3 STATUS	(Externspeicher nicht bereit)
4 KEIN CTB	(indirekter Transfer nötig, aber kein CTB vorgesehen)

Im Falle eines Fehlers wird auf dem Konsol-Fernschreiber eine Nachricht ausgedruckt, die mit DOS ERR beginnt und in Form von 7 6-stelligen Oktalstellen Fehlerart, Ursprungsebene, Transferinstruktion, Kernspeicher-, Externspeicher-Adresse und Blocklänge angibt. Das Programm in der Ursprungsebene wird nicht fortgesetzt.

Die Fehlerausgabe läuft in Ebene 00; ein etwa dort in Gang befindliches Programm wird unterbrochen; danach wird der Programmstand wieder auf den alten Stand gebracht, die Ebene 00 wird jedoch nicht gestartet.

MINCDOS 500 OPERATING

OPERATING ist der zweite Teil des Trommel/Platten-Betriebssystems; er dient zum Laden der Externspeicher über den schnellen Lochstreifenleser, zum Ausstanzen des Externspeicher-Inhalts auf dem schnellen Locher, zum Aufruf von Programmen aus dem Extern- in den Kernspeicher und zum Ablegen von Kernspeicher-Inhalten auf den Externspeicher.

Jedes eingelesene, ausgelochte, aufgerufene oder abgelegte Programm hat einen Namen, der im PROGRAM DIRECTORY (Programmverzeichnis) auf dem Externspeicher vermerkt sein muß; mit OPERATING kann dieses Verzeichnis geführt und ausgedruckt werden.

Das PROGRAM DIRECTORY befindet sich auf der ersten Spur des Externspeichers (Adressen 000000...001777 = 1 kWorte) und kann bis zu 256 Programm-Namen einschließlich der zugehörigen Parameter aufnehmen, wobei jeweils 4 Worte zu einem Programm gehören:

1. Wort	Programm-Name
2. Wort	Kernspeicher-Basisadresse (für Aufruf bzw. Ablegen)
3. Wort	Externspeicher-Basisadresse (Lage des Programms)
4. Wort	Programmlänge (in Worten)

Programm-Namen sind aus 3 alphanumerischen Zeichen (alle 64 druckbare Zeichen des ASCII-Codes einschließlich SPACE) aufgebaut, wobei jede Kombination erlaubt ist; die Kombination "@@@" jedoch gilt als Leerstelle im Verzeichnis und ist als Name verboten. Basisadressen und Programmlänge werden in binärer bzw. oktaler Form dargestellt.

Es ist zweckmäßig, jedes Programm bzw. Teilprogramm, jede Tabelle usw., die sich im Externspeicher befinden, auf diese Weise im PROGRAM DIRECTORY zu vermerken.

Bedienung und Betriebsarten

OPERATING wird durch Start der Programmebene 00 begonnen, z.B. mit Betätigen der Taste STA am Rechner-Bedienungsfeld. Der Konsol-Fernschreiber beginnt eine neue Zeile mit dem Ausdrucken der Buchstaben DOS.

Der Bediener wählt eine der 6 OPERATING-Betriebsarten durch Eintasten der Buchstaben L, F, R, P, C oder S an; jedes andere Zeichen wird durch erneutes Ausdrukken von DOS moniert; durch Eingabe von E jedoch wird OPERATING beendet.

Es gibt folgende Betriebsarten:

LIST	Ausdrucken des Programmverzeichnisses
FIND	Programm im Verzeichnis suchen (um es zu ändern, einzugeben oder zu löschen)
PUNCH	Auslochen Programm auf schneller Lochstreifeneinheit
READ	Einlesen eines Programms über die schnelle Lochstreifeneinheit
CALL	Aufrufen eines Programms
SAVE	Ablegen eines Programms

PLATTENSPEICHER-SYSTEM

Plattenspeicher:	Magnetwechselplattenspeicher mit beweglichem Lese/Schreibkopf Kapazität 0.8 Mbyte (19 bit + Parity) 2mal 200 Spuren zu je 2 kWorten 8 Sektoren zu je 256 Worten je Spur mittlere Zugriffszeit 60 ms Transferrate ca. 1.6 MHz entsprechend ca. 80 kWorten/s 19"-Einschub, 4 Einheiten hoch mit zusätzlicher Stromversorgung.
Controller:	Interface-Einheit zum Betrieb von 1 bis 4 Platten zum Anschluß an den Computer über X-Kanal und DMA-Kanal zum Lesen/Schreiben von 1...8 Sektoren einer Spur eingebaut in eine Speichereinheit
Software:	MINCEDOS 500 Platten-Betriebssystem mit Executive (Lese/Schreibsteuerung) und Operating (Bedienungssystem)

MINCEDOS 500

Das MINCEDOS-Betriebssystem entspricht weitgehend dem Trommel-Betriebssystem MINCDOS. Der einzige Unterschied besteht darin, daß nur ein Teil der Platte so angesprochen werden kann wie die Trommel.

Wegen der großen Adreßkapazität gibt es zwei Möglichkeiten der Plattenadressierung:

1. Jedes Wort wird adressiert (bis zu maximal 128 k möglich)
2. Die Sektoren werden adressiert (wird durch ein Minuszeichen in der Plattenadresse angegeben).

MAGNETBAND-SYSTEME

Bandeinheit:	9-Spur, 800 oder 1600 bpi Schreibdichte 12.5...25 ips Bandgeschwindigkeit Read-After-Write mit Stromversorgung Spulendurchmesser: 7 " (7200" Bandlänge) 8.5" (14400" ") 10.5" (28800 " ")	(19"-Einbau, 5 Einheiten hoch) (" , 7 " ") (" ,14 " ")
Controller:	Interface-Einheit zum Betrieb von 1 bis 4 Bändern zum Anschluß an den Computer über X-Kanal und DMA-Kanal eingebaut in eine Speichereinheit	
Software:	MINCTOS 500 Band-Betriebssystem	

MINCTOS 500

Das Band-Betriebssystem MINCTOS 500 ist ein speicherresidentes Programm, das 1/4 kWorte im Kernspeicher einnimmt. Es ist relativ adressiert, kann also beliebig im Speicher abgelegt werden.

MINCTOS 500 kann maximal 4 Bandeinheiten bedienen.

Es sind nachstehende Befehle möglich:

Lesen vom Band
Schreiben auf das Band
Zurückspulen bis zum Anfang (Rewind)
Schreiben einer Bandmarke
Definierter Vorlauf
Definierter Rücklauf

Lesen und Schreiben kann entweder binär oder byteweise erfolgen.

Binär heißt, daß das ganze Wort übertragen wird (3 bytes). Byteweise heißt, daß jedes Wort als 2 Bytes zu je 8 bit angesehen wird; die Bits 16...19 werden daher nicht übertragen.

Die Übertragung erfolgt im Cycle-Stealing.

Auf jeden Befehl müssen immer Parameter-Worte folgen. Er hat die Form:

542LLX
YAAAAA
ØØBBBB
CCCCCC

LL	Ebene, in der MINCTOS 500 läuft
X	Nummer des Befehls
Y	Nummer der Bandeinheit

A...A	Kernspeicher-Basisadresse
B...B	Blocklänge
C...C	Sonderausgang

Hinter jedem Befehl ist eine symbolische Schreibweise angegeben; in der Befehlstabelle des Assemblers ist der entsprechende Maschinenbefehl einschließlich der MINCTOS 500-Ebene (LL) zu vermerken.

Lesen byteweise	542LLØ = RTF	Read Tape Format
Lesen binär	542LL1 = RTB	Read Tape Binary
Schreiben byteweise	542LL2 = WTF	Write Tape Format
Schreiben binär	542LL3 = WTB	Write Tape Binary
Zurückspulen	542LL4 = RTB	Rewind Tape
Schreiben einer Bandmarke	542LL5 = WTM	Write Tape Mark
Definierter Vorlauf	542LL6 = FTM	Forward until Tape Mark
Definierter Rücklauf	542LL7 = BTM	Bach until Tape Mark

Der Sonderausgang wird in folgenden Fällen angesprungen, wobei im W-Register eine Schlüsselzahl vermerkt ist:

W-Register = 1	Bandeinheit nicht bereit
" = 2	gelesener Block zu lang
" = 3	Bandmarke gelesen
" = 4	Blocklänge größer als 4095
" = 5	Bandende gefunden
" = 6	Parity-Fehler
" = 7	Schreibversuch trotz Schreibsperre
" = 8	Bandanfang gefunden
" = 9	Basisadresse + Blocklänge ungleich Zählerstand

Peripherie-Interfaces

Geräte- und Prozeßperipherie-Interfaces dienen zum Anschluß der Computer MINCAL 513/523 an Bedienungsperipherie und Prozeß-Ein/Ausgänge.

Sie bestehen aus einer oder mehreren Bausteinen im Format der doppelten Europakarte (225 mm hoch, 160 mm tief; Adapterkarten sind 270 mm tief). Der Anschluß der Karten geschieht über zwei 64-polige Stecker. Zu den Geräten oder zum Prozeßanschluß dienen Adapterkarten mit 20- oder 30-poligen DIN-Steckern.

Die Interfaces werden - soweit sie nicht in der Zentraleinheit eingebaut sind - in einem oder mehreren Zusatzeinschüben (Anschlußeinheit) untergebracht.

Anschlußeinheit

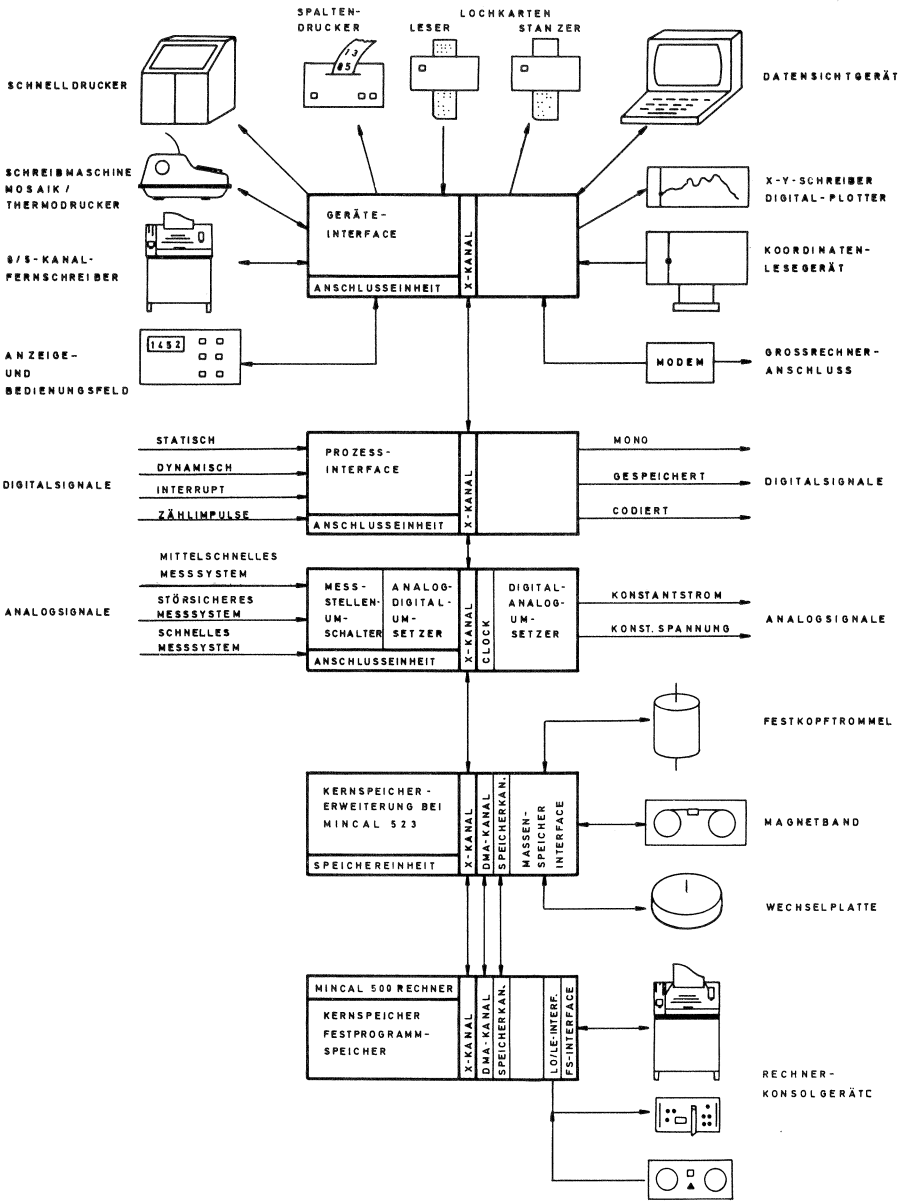
Diese Einheit kann (je nach Baugröße der Interface-Einheiten) ca. 15 Interfaces aufnehmen; die Karten werden von hinten in senkrechter Lage eingesteckt. Der Anschluß der Peripherie erfolgt rückseitig über 20- oder 30-polige DIN-Stecker auf speziellen Adapterkarten.

Die Einheit enthält eine für alle Interfaces ausreichende Stromversorgung; ferner 2 Paar Anschlußstecker mit Adapterkarten für den X-Kanal (zur Verbindung mit dem Computer sowie evtl. weiterer Einschübe).

Die Baugruppen werden über eine gemeinsame Device-Selection angewählt; auf dieser Karte werden X-Kanal-Adressen und Programmebenen den Interfaces zugeordnet.

Steckplatz-Einheiten:	82	
Stromversorgung:	+5 V/5.5 A; -5 V/1 A; +15 V/5 A; +24 V/3 A	
Netzanschluß:	220 V $\pm$10 % 50 Hz Leistungsaufnahme max. 200 VA	
Größe:	19"-Einschub 7 Einheiten hoch + Stromversorgung 2 Einheiten hoch	ca. 400 mm

MINCAL 500 PROZESSRECHNER-SYSTEM



Geräte-Interfaces

Diese Interfaces dienen zum Anschluß von Peripheriegeräten (insbesondere Bedienungsperipherie). Folgende Interfaces sind verfügbar:

Fernschreib-Interface
8-Kanal-E/A-Schnittstelle
110 Bd, 11 Schritte/Zeichen
mit Formatisierung und Parity-Erzeugung/-Prüfung
für Linienstrom 20...40 mA

Fernschreib-Interface
8-Kanal/5-Kanal-E/A-Schnittstelle
50/75/110 Bd, 7.5 oder 11 Schritte/Zeichen
für Linienstrom 20...40 mA

Streifenleser-Interface
8-Kanal-E-Schnittstelle
für 5/8-Kanal-Lochstreifenleser 125 Z/s

Streifenleser-Interface
8-Kanal-E-Schnittstelle
für 5/8-Kanal-Lochstreifenleser 330 Z/s

Streifenlocher-Interface
8-Kanal-A-Schnittstelle
für 5/8-Kanal-Lochstreifenstanzer 50 Z/s

V24-Interface für Geräte-Anschluß
8-Kanal-E/A-Schnittstelle
110...1200 Bd, 10 oder 11 Schritte/Zeichen
Schnittstelle nach V24
für Display, Mosaikdrucker, Teletype

Interface für Kugelkopf-Schreibmaschine
E/A-Schnittstelle
für IBM 735 BCD, 15 Z/s

Spaltendrucker-Interface
A-Schnittstelle
(für Kienzle-Spaltendrucker)

Schnelldrucker-Interface
A-Schnittstelle
(für Kettendrucker MDS 4030)

Analog-XY-Interface
mit 2 Analogausgängen 0...1 V
mit 2 10-bit-Digital-Analog-Umsetzern und 2 Kaskadenregistern mit Steuerung für Z-Koordinate
(für XY-Schreiber und graphische Displays)

Digital-Plotter-Interface
mit Impulsausgängen für XY-Koordinaten und Federsteuerung
(für Digital-Plotter)

Kartenleser-Interface
12-Kanal-Schnittstelle
geeignet für Start-Stop-Betrieb

Interfaces für Prozeßsignale

Diese Interfaces dienen zur Ein- oder Ausgabe von digitalen oder analogen Signalen, wie sie für Prozeßanschlüsse typisch sind. Folgende Interfaces sind verfügbar:

18/36-bit-Digitaleingang statisch/TTL
Prozeß-Interface zur statischen Abfrage von 18/36 digitalen Eingangssignalen
TTL-Schnittstelle (5 V)

18/36-bit-Digitaleingang statisch/HTL
Prozeß-Interface zur statischen Abfrage von 18/36 digitalen Eingangssignalen
HTL-Schnittstelle (12...30 V)

18/36-bit-Digitaleingang statisch/Relais
Prozeß-Interface zur statischen Abfrage von 18/36 digitalen Eingangssignalen über Relais ($\geqslant$12 V, 15 mA)
16 Spulenanschlüsse + 1 gemeinsame Rückleitung

18-bit-Digitaleingang dynamisch/TTL
Prozeß-Interface zur Speicherung und Abfrage von 18 digitalen Eingangssignalen
mit Differenziereingang, 18-bit-Speicher und Interrupt-Auslösung
(6-bit-weise verriegelbar)
TTL-Schnittstelle (5 V)

18-bit-Digitaleingang dynamisch/HTL
Prozeß-Interface zur Speicherung und Abfrage von 18 digitalen Eingangssignalen
mit Differenziereingang, 18-bit-Speicher und Interrupt-Auslösung
(6-bit-weise verriegelbar)
HTL-Schnittstelle (12...30 V)

18-bit-Digitaleingang dynamisch/Relais
Prozeß-Interface zur Speicherung und Abfrage von 18 digitalen Eingangssignalen über Relais ($\geqslant$ 12 V, 15 mA) mit Differenziereingang, 18-bit-Speicher und Interrupt-Auslösung
(6-bit-weise verriegelbar)

Prozeß-Interface für Absolutdrehgeber
für max. 30 Spuren
mit Hardware-Umschlüßlung für V-Logik zum Anschluß eines Absolut-Drehgebers (Dual) mit Lampensteuerung über Referenzdiode

Prozeß-Interface für Absolutdrehgeber
beliebigen Codes 12 bit
mit Abfrage-Logik und Zweifadenlampen-Steuerung

18/36-bit-Digitalausgang/TTL
Prozeß-Interface zur Speicherung und Ausgabe von 18/36 Ausgangssignalen
TTL-Schnittstelle (5...30 V, max. 80 mA)

18/36-bit-Digitalausgang/Relais
Prozeß-Interface zur Speicherung und Ausgabe von 18/36 Ausgangssignalen
18/36 Kontaktausgänge von Reed-Relais
1 Arbeitskontakt je bit mit gemeinsamer Rückleitung
max. 110 V 0.5 A 10 W bei ohmscher Last

18/36-bit-Digital-Impulsausgang/HTL
Prozeß-Interface zur Ausgabe von 18/36 Ausgangssignalen eingestellter Zeitdauer
HTL (5...30 V, 80 mA)
T = 1...500 ms

18-bit-Digital-Impulsausgang/HTL
Prozeß-Interface zur Ausgabe von 18 Ausgangssignalen eingestellter Zeitdauer
HTL (5...30 V 1 A)
T = 1...500 ms

18-bit-Digital-Impulsausgang/Relais
Prozeß-Interface zur Ausgabe von 18 Ausgangssignalen eingestellter Zeitdauer
Reed-Relais: 1 Arbeitskontakt je bit mit gemeinsamer Rückleitung
max. 110 V 0.5 A 10 W bei ohmscher Last
T = 1...500 ms

Analogausgang 10 bit/10 V/1 V
Prozeß-Interface mit 10-/0-bit-Register, Digital-Analog-Umsetzer und Verstärker
Ausgangsspannung 0...10 V (0...1 V) niederohmig

Analog-Ausgang 12 bit/20 V
Prozeß-Interface mit 12-bit-Register, Digital-Analog-Umsetzer und Verstärker
Ausgangsspannung 0...20 V niederohmig

Analogausgang 10 bit/20 mA
Prozeß-Interface mit 10-bit-Register, Digital-Analog-Umsetzer und Verstärker
Ausgangsstrom 0...20 mA, max. Bürde 450 Ohm/850 Ohm

Ziffern-Anzeige-Treiber
Register- und Treiberschaltungen für BCD-Ziffernanzeigen
(3.5...30 V)
für 4 x 2 (bzw. 2 x 3 oder 2 x 4) Dekaden, mit Vorzeichen

Prozeß-Interface für codierte Digital-Ausgabe
4 Dekaden, mit Registern

Analog-Eingang 12 bit/10 V
Prozeß-Interface mit Analog-Digital-Umsetzer 12 bit
Konversionszeit ca. 30 us
Eingangsspannung 0...$\pm$10 V, nicht potentialfrei

Analog-Meßsysteme

VORBEMERKUNG

Zur Erfassung insbesondere einer Vielzahl von analogen Meßsignalen stehen drei Meßsysteme zur Verfügung, die sich durch ihre Auflösung und ihre Meßgeschwindigkeit unterscheiden.

Alle Systeme sind modular aufgebaut und in sich abgeschlossene Einheiten, die über den X-Kanal bzw. DMA-Kanal mit dem Computer MINCAL 513/523 verbunden werden.

Mittelschnelles Analog-Meßsystem

Auflösung:	12 bit (Stufenverschlüßler)
Konversionszeit:	ca. 25 us
Meßbereich:	0...+10 V - 5...+ 5 V (Option) -10...+10 V (Option)
Potentialtrennung:	zwischen Meßkreis und Logik durch Fotokoppler
Betriebsarten:	programmgesteuert oder selbstgesteuert (Option) für Messung mehrerer Meßwerte bzw. Meßkanäle und Ablage im Kernspeicher vom Programm vorwählbar: Speicher-Basisadresse, Meßkanal-Basisadresse, Blocklänge, Einkanal/inkrementierende Mehrkanal-Messung
Meßfrequenz:	max. 29 kHz im selbstgesteuerten Betrieb
Meßkanäle:	MOS-FET-Multiplexer (Option) 1...4 Bausteine mit 8 oder 16 Eingängen (max. 64 Kanäle)
Sample-and-Hold:	Option
Stromversorgung:	eingebaut
Größe:	19"-Einbaurahmen, offen (abgeschirmt), konvektionsbelüftet Höhe 3 Einheiten (ca. 135 mm) Tiefe ca. 250 mm
Meßanschlüsse:	über rückseitige Steckverbindungen
Netzanschluß:	220 V $\pm$ 10 % 50 Hz

Schnelles Analog-Meßsystem

Auflösung:	8 bit (Stufenverschlüßler) Konversionszeit 0.95 us, oder 10 bit (Stufenverschlüßler) " 1.2 us
Meßbereich:	0...+10 V 0...+ 5 V (Option)
Potentialtrennung:	zwischen Meßkreis und Logik durch Fotokoppler
Betriebsart:	selbstgesteuert für Messung mehrerer Meßwerte bzw. Meßkanäle und Ablage im Kernspeicher vom Programm vorwählbar: Speicher-Basisadresse, Meßkanal-Basisadresse, Blocklänge, Einkanal/inkrementierende Mehrkanal-Messung
Meßfrequenz:	200 kHz
Meßkanäle:	MOS-FET-Multiplexer (Option) 1...4 Bausteine mit 8 Eingängen (max. 32 Kanäle)
Sample-and-Hold:	Option
Stromversorgung:	eingebaut
Größe:	19"-Einbaurahmen, offen (abgeschirmt), konvektionsbelüftet Höhe 3 Einheiten (ca. 135 mm) Tiefe ca. 250 mm
Meßanschlüsse:	über rückseitige Steckverbindungen
Netzanschluß:	220 V $\pm$ 10 % 50 Hz

Störsicheres Analog-Meßsystem

Auflösung:	+ 120000 Ziffernschritte zu je 10 uV Integrierendes Meßverfahren Integrationszeit 20 ms
Meßfolge:	20 bzw. 100 Messungen/s
Meßbereich:	1.2 V
Potentialtrennung:	mit Meßkreisschirm (guard)
Betriebsart:	programmgesteuert
Eingangswiderstand:	1000 MOhm
Meßfehler:	±0.01 % v.M., ±0.003 % v.E.
Meßkanäle:	3-polig schaltende Reed-Relais in Schutzschirmtechnik. Pro Einschub 8...64 Kanäle
Größe:	19"-Einbaurahmen Höhe 199 mm Tiefe 407 mm
Netzanschluß:	220 V +10 %/-15 % 50 Hz

Periphergeräte

Zu den MINCAL 513/523 ist eine große Anzahl von Periphergeräten verfügbar, die über die im Kapitel "Geräte-Interfaces" beschriebenen Schnittstellen betrieben werden.

- 8-Kanal-Fernschreiber
  Teletype ASR 33, mit angebautem Streifenlocher und -leser
  10 Z/s, 72 Z/Zeile
  mit V24-Anschluß oder mit Linienstrom-Anschluß

- 5-Kanal-Fernschreiber
  Siemens T100
  10 Z/s, 64 Z/Zeile
  mit Linienstrom-Anschluß

- 8 (5)-Kanal-Streifenleser
  optischer Leser mit Schrittmotor für Vor/Rückwärts-Betrieb
  125 Z/s
  mit Spuleinrichtung (Spulen 152 mm Ø)
  19"-Einbaugerät, Höhe 5 Einheiten

- 8 (5)-Kanal-Streifenleser
  optischer Leser mit Andruckrolle und Bremse, 330 Z/s
  19"-Einbaugerät, Höhe 3 Einheiten
  Spuleinrichtung, Höhe 4 Einheiten

- 8-Kanal-Streifenlocher
  50 Z/s
  mit Vorratsspule
  19"-Einbaugerät, Höhe 4 Einheiten

- Bildschirm-Terminal
  Datensichtgerät mit 12" Bildschirm
  27 Zeilen zu je 74 Zeichen
  mit Vordergrund/Hintergrund-Speicher
  mit V24-Anschluß (110...4800 Bd)
  mit Eingabetastatur
  auf Wunsch mit Hardcopy-Einheit (Thermodrucker)
  auf Wunsch mit Doppel-Magnetbandkassette

- Thermodrucker
  10...30 Z/s oder 40 Z/s, 72 Z/Zeile

- Mosaikdrucker
  180 Z/s, 132 Z/Zeile

- XY-Schreiber
  DIN A4 oder DIN A3

- Digital-Plotter
  DIN A4
  oder für Endlospapier (11" Schreibbreite)
  300 Schritte/s

- Lochkartenleser
  für 80-spaltige Lochkarten, Stapelleser
  0...200 Spalten/s

- LSK-Abrufleser
  Siemens-Lochstreifenkartenleser 61, 10 Z/s

- LSK-Locher
  Siemens-Lochstreifenkartenlocher 159, 10 Z/s

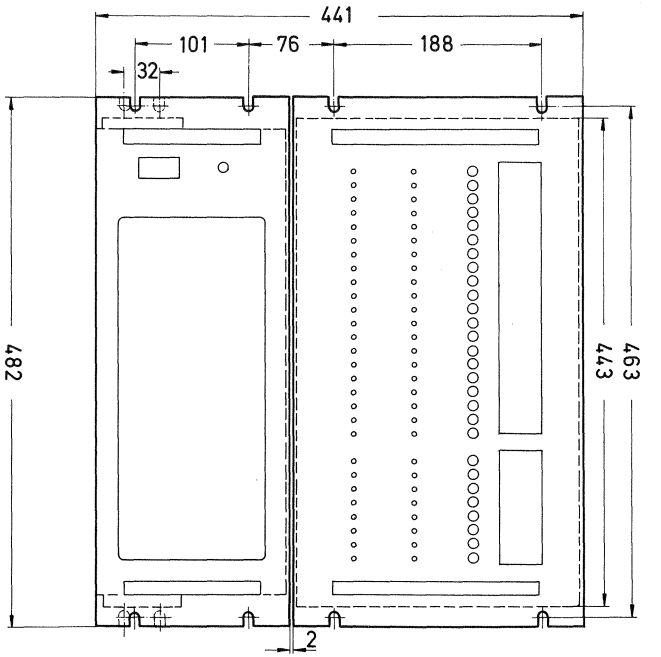
- FS-Rundschreibknotenstelle codegesteuert
  Siemens Zentraleinheit 9530
  adressen-gesteuerter Datenverteiler für max. 10 Endstellen

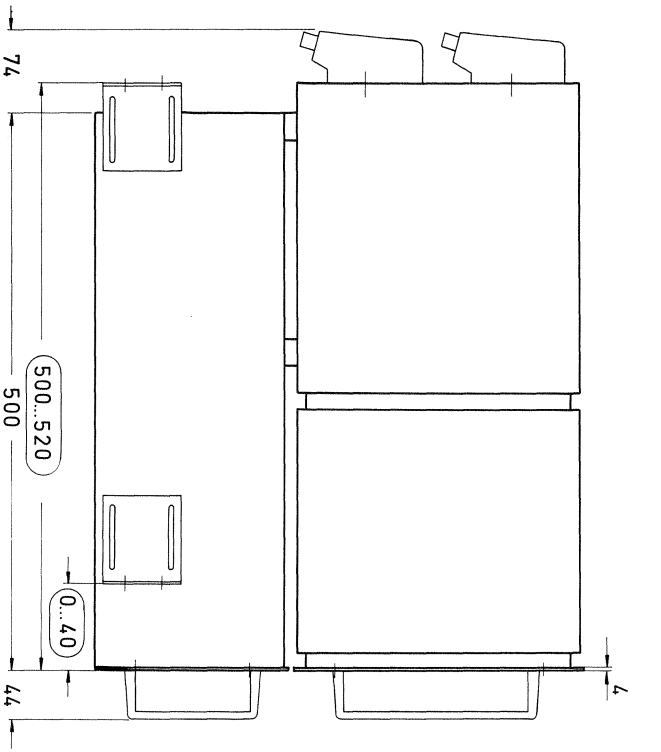
- Computer-Display-Terminal
  für graphische und alphanumerische Darstellungen auf einer Speicherröhre

- Kugelkopf-Schreibmaschine
  IBM 735 BCD, 15 Z/s
  mit codegesteuerter Rot-Schwarz-Umschaltung, 120 Z/Zeile

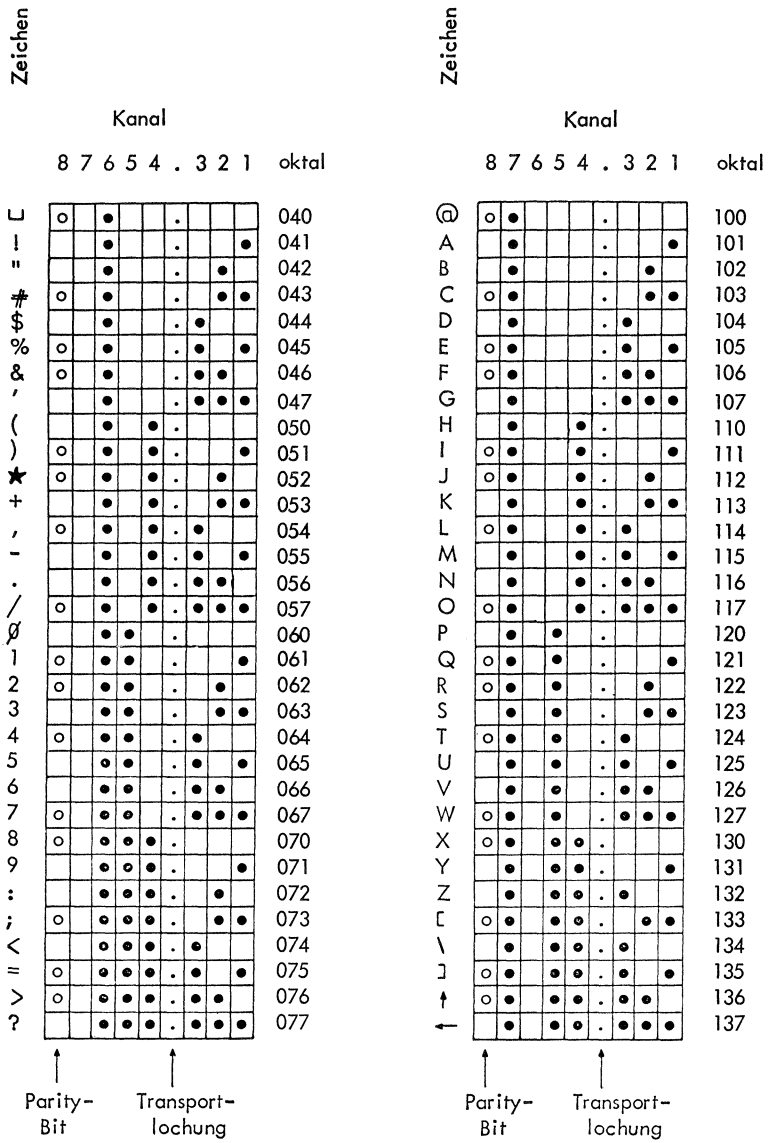
- Spaltendrucker
  Digitaldrucker mit Springwagen
  5-stellig mit Tabulator
  (Kienzle D44-SW 5)

- Schnelldrucker
  Kettendrucker MDS 4030
  300 Z/min, 64 Charaktere, 135 Z/Zeile


441
101
76
188
32
482
473
463
2


74
500...520
500
0...40
44
4

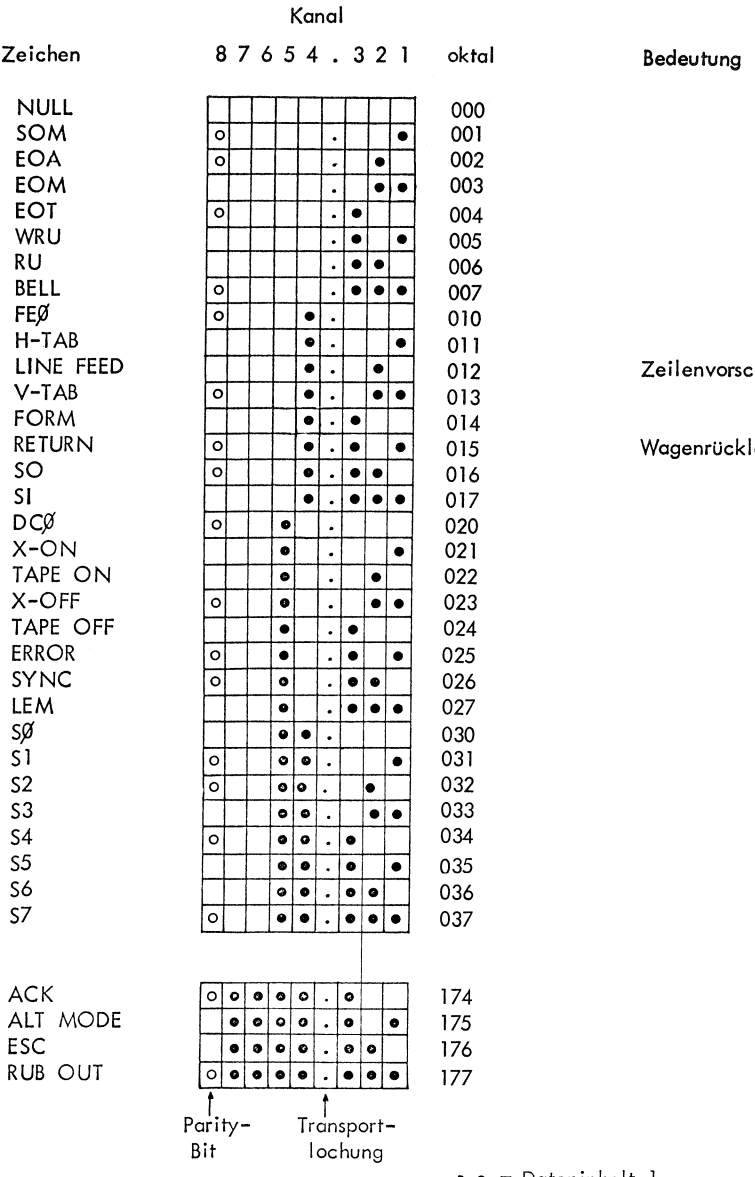
ASCII-Code Druckbare Zeichen



␣ (Zeichen Ø4Ø) bedeutet Leerschritt

o ● = Dateninhalt 1
= Lochung im Streifen
= Stromschritt (MARK)

ASCII-Code Steuerzeichen



	NOP	000000		MZR	.030AAA		HLT	540000
				MPO	.031AAA		HSL	542LL0
(e)	VBL	001400		MMO	.032AAA		HBR	544AAA
(e)	VBR	001440		MIC	.033AAA		STL	550LL0
(e)	VDL	001500		MDC	.034AAA		ECL	553000
(e)	VDR	001540		MCO	.035AAA		DCL	554000
	COD	00AAAA		MCI	.036AAA			
							GX	60XAAA
	SRLW	000010		LDR	.11RAAA		FX	61XAAA
	SRLD	000011		TRR	.15RAAA		OX	65XAAA
	SRLX	000012		ADR	.04RAAA			
	SRAW	000014		SBR	.05RAAA		GB	-60XAAA
	SRAD	000015		FOR	.06RAAA		FB	-61XAAA
	SRAX	000016		FAR	.07RAAA		IBG	-62XAAA
	SLLX	000020					IBF	-63XAAA
	SLLD	000021		LDC	.10RCCC		IB	-64XAAA
	SLLW	000022		ADC	.14RCCC		OB	-65XAAA
	SLAX	000024					IBH	-66XAAA
	SLAD	000025		LD	.12QAAA		OBH	-67XAAA
	SLAW	000026		TR	.16QAAA			
				AD	.20QAAA		RBL	-700DD0
(e)	SRLW	0100NN		SB	.22QAAA		SBL	-700DD1
(e)	SRLD	0110NN	(e)	MP	.30QAAA		RBR	-710DD0
(e)	SRLX	0120NN	(e)	DV	.32QAAA		SKB	-720DD0
(e)	SRAW	0140NN		FO	.24QAAA		SKR	-730DD0
(e)	SRAD	0150NN		FA	.26QAAA			
(e)	SRAX	0160NN		FE	.34QAAA	(e)	IBS	-74NDDF
(e)	SLLX	0200NN		CP	.36QAAA	(e)	OBS	-75NDDF
(e)	SLLD	0210NN						
(e)	SLLW	0220NN		BR	.40SAAA	(e)	IBB	-760DD0
(e)	SLAX	0240NN		BZ	.42RAAA	(e)	OBB	-770DD0
(e)	SLAD	0250NN		BP	.44RAAA			
(e)	SLAW	0260NN		BM	.46RAAA			
(e)	SRR	0130NN						
(e)	SLN	023000		CS	.50QAAA			
				CM	.52QAAA			

AAA = Adreßteil
CCC = Festwert
DD = Gerätenummer
F = Format
LL = Ebene
N = Anzahl Zeichen
NN = Anzahl Bits
Q = Ergänzung
R = Register
S = Sensor
X = Indexregister

·... = Vorzeichen von Bedeutung
..- = indirekt wenn ..A (+1 bei 2. Oktalstelle)
(e) = erweiterter Befehlsvorrat