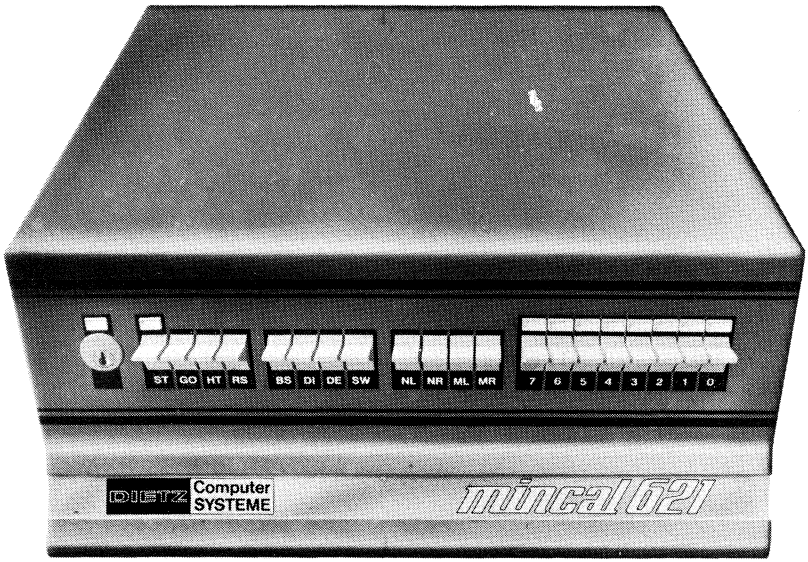
HANDBUCH

# mincal 621

DIETZ Computer SYSTEME




# mincal 621

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

* Dietz bietet immer etwas mehr

Ausgabe: Oktober 1971

Herausgeber: Heinrich Dietz
INDUSTRIE-ELEKTRONIK
433 Mülheim a.d. Ruhr, Kölner Straße 115
Telefon: (0 21 33) 48 85 41 · Telex: 08 567 70

Druck: Hoppe + Werry KG, Mülheim a.d. Ruhr

## Inhalt


| | Seite |
|---|---|
| DIETZ in Kurzform | 4 |
| Über den neuen MINCAL | 5 |
| Computer-Fibel | 9 |
| Struktur | 32 |
| Maschinenbefehle | 52 |
| ASSEMBLER | 76 |
| LIBRARY | 95 |
| MONITOR | 104 |
| Programmier-Hinweise | 108 |
| Bedienung | 115 |
| Aufbau | 118 |
| Anhang | 121 |

# 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 beiden Eigenschaften finden sich auch bei dem MINCAL-500-System, das 1969 auf den Markt gebracht wird. Dieses erfolgreiche Prozeßrechner-System umfaßt heute den festprogrammierten Computer MINCAL 513 und sein speicherprogrammiertes Gegenstück, den MINCAL 523.

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. Auch der MINCAL 621 ist ein Teil dieser Gesamtkonzeption.

# Über den neuen MINCAL

Mit dem MINCAL 621 stellt DIETZ einen neuen Computer der MINCAL-Serie vor.

Das Konzept dieses Computers berücksichtigt die Erfahrungen 7-jähriger erfolgreicher Computer-Entwicklung und verwendet modernste Technologien. Der attraktive Preis ist nicht durch Weglassen wichtiger Funktionen erreicht worden, sondern durch eine neuartige Konzeption, die im Bereich der Kleincomputer ganz neue Maßstäbe setzt.

Multibyte-Struktur: Die gesamte Verarbeitung einschließlich der arithmetischen Befehle ist auf eine beliebige Anzahl von (bis zu 256) Bytes bezogen. Die Länge der Verarbeitung wird durch einen DO-Befehl bestimmt, der vor dem Multibyte-Befehl steht. Diese Hardware-Lösung ist flexibel, speichersparend und vor allem schnell.

MOS-Speicher: Es stehen 256 Universalregister zur Verfügung, die als Akkumulator, Indexregister, schneller Datenspeicher und auch als schneller Programmspeicher verwendet werden können. Bei einem 16-Ebenen-Rechner kann dieser Satz von 256 Universalregistern sogar 16-mal vorhanden sein.

Diese maximal 4096 Register sind in einem superschnellen MOS-Speicher realisiert, dessen Inhalt vor Verlust bei Netzausfall geschützt werden kann.

Multiprogramming-Struktur: Der MINCAL 621 bietet dem Benutzer bis zu 16 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 maximal 256 eigene Register. Multiprogramming ist extrem einfach, denn für jede Aufgabe gibt es eine völlig unabhängige Ebene.

Universal-BUS: Kernspeicher, Peripherie, Recheneinheit und eventuelle Massenspeicher verkehren über einen Universal-BUS miteinander. Programmgesteuerter Datenverkehr und direkter Speicherzugriff bedienen sich des gleichen Datenkanals. Die Peripherie und der Speicher werden völlig gleich behandelt, so daß sich die Programmierung von Ein- und Ausgaben vereinfacht und trotzdem an Flexibilität gewinnt. Bei direktem Speicherzugriff wird die Zentraleinheit nicht berührt und kann bis zu ihrem nächsten Speicherzugriff intern weiterarbeiten. Natürlich erlaubt das BUS-Konzept den Anschluß beliebig schneller Speicher.

Flexible Adressierung: Die BUS-bezogenen Befehle können vielfältig adressiert werden: CONSTANT (die Adreß-Bytes werden unmittelbar als Operand verwertet), REGISTER (die ebenen-zugehörigen Register werden angesprochen), RELATIVE (der Operand steht bis zu 127 bytes vor bzw. nach dem Befehl), ABSOLUTE (der gesamte Speicherbereich kann durch eine 16-bit-Adresse angesprochen werden. Zusätzlich kann die so gebildete Adresse noch indiziert werden, über eines von maximal 127 Indexregistern.

Bedingter Sprung: Alle Entscheidungen werden durch bedingte Sprünge gefällt, bei denen das Programm relativ um bis zu 127 byte vorwärts oder rückwärts verzweigt. Besondere Befehle testen beliebige Bits oder Bitgruppen auf 0- oder 1-Zustand.

Zweiadreß-Befehle: Alle BUS-bezogenen Instruktionen sind Zweiadreß-Befehle, ein bei Kleincomputern ungewöhnlicher Komfort.

Hardware-Bootstrap: Ein Bootstrap-Programm ist in einem ROM gespeichert und ist durch Tastendruck aufrufbar.

Zusätzliche Optionen: Interfaces für Standard-Peripherie im Rechner-Gehäuse. Real-Time-Clock. Netzausfallschutz. Memory Parity. 4, 8 oder 16k byte Kernspeicher im Rechner-Gehäuse.

Software: Assembler und Makro-Assembler. Bibliothek mit Ein/Ausgabe-Paket, Doppelwort- und Gleitkomma-Paket. Test-Monitor.

| | |
|---|---|
| Typ: | Universal-Computer für Prozeßanwendungen, technisch-wissenschaftliche Zwecke und allgemeine Datentechnik |
| Wortlänge: | 8 bit (1 byte)<br>Ein- und Mehrbyte-Verarbeitung vorgesehen<br>(Einzelbefehle 1- bis 256-mal ausführbar) |
| Arbeitsspeicher (Pool): | MOS-RAM mit 0,25k byte<br>erweiterbar auf 4k byte<br>Zugriffszeit 0,2 us<br>Vollzyklus 0,4 us<br>auf Wunsch batteriegepuffert<br>enthält Register und Datenplätze |
| Kernspeicher: | 4k, 8k oder 16k byte<br>extern erweiterbar auf 48k byte<br>Zugriffszeit 0,4 us (4k); 0,3 us (8, 16k)<br>Vollzyklus 1,0 us (4k); 0,65 us (8, 16k)<br>enthält Programm und Daten |
| Technologie: | integrierte Schaltkreise (TTL, TTL-MSI) |
| Instruktionen: | 9 BUS-bezogene Befehle mit Register-, relativer, indirekter oder absoluter sowie indizierter Adressierung<br>7 Konstantenbefehle<br>32 bedingte Sprungbefehle<br>4 Schiftbefehle<br>1 Mehrfachausführungsbefehl<br>5 Steuerbefehle<br>2 Zustandsabfragebefehle |
| Instruktionslänge: | 1...5 byte je nach Befehlstyp |
| Operationsdauer: | min. 1,7 us; max. 8,5 us (bei 1 us Speicherzyklus) |
| Arbeitsregister: | max. 254 je Ebene oder 1 fester Akku je Ebene (programmierbar) (1...254 byte lang) |
| Indexregister: | max. 127 je Ebene (2 byte oder 1 byte lang) |
| Ebenen: | 2 Programmebenen mit hierarchischer Priorität<br>erweiterbar auf 16 Ebenen (Option) |

| | |
|---|---|
| Interrupt: | Wechsel der Programmebene bei Ende jeder Operation möglich (DO-Befehl und folgender Befehl gelten als 1 Befehl) |
| Universal-BUS: | Standard-Schnittstelle mit 8-bit-Daten-Ein/Ausgang, 8-bit-Adreßausgang, Ebenenausgang und Interrupt-Eingang für Speicher und Peripherie |
| Rechner-Uhr: (Option) | 10 MHz-Quarz (Taktgenerator)<br>fest einstellbarer Untersetzer für 1 ms-, 10 ms-, 100 ms- und 1 s-Starts |
| Bedienungskonsole: (Option) | enthält zentrale 8-bit-Anzeige und zentralen 8-bit-Schaltersatz, ferner Tasten für Laden, Speichern, Start, Stop, Nullsetzen sowie Bootstrap-ROM |
| Größe: | 19"-Einschub<br>5 Einheiten hoch (ca. 225 mm)<br>ca. 500 mm tief |
| Netzanschluß: | 220 V, 50 Hz einphasig<br>ca. 150 VA |
| Interfaces in Rechner-Gehäuse (Option) | E/A-Interface für 8-Kanal-Fernschreiber (Teletype) ASCII-Code, 110 Bd<br>Interfaces für 8-Kanal-Streifenleser und -Streifenlocher |

# Computer-Fibel

Dies soll eine kleine Hilfe für alle die Benutzer des MINCAL 621 sein, die noch keine Erfahrung mit Computern und Computer-Terminologie haben. Ein Computer hat nichts Geheimnisvolles an sich; um seine Prinzipien zu verstehen und mit ihm umzugehen, muß man nur folgerichtig denken und diese Denkschritte sorgfältig formulieren können.

## BINÄRZAHLEN

Computer behandeln Zahlen anders, als wir es gewohnt sind. Alle Elemente in einem Rechner können nur zwei verschiedene Zustände unterscheiden und behandeln:

1. Positive Spannung
2. Keine Spannung

Zwischenwerte kennt ein Computer nicht. Der Zustand: "Es besteht die halbe positive Spannung" ist nicht möglich, es sei denn, der Computer streikt.

Alle Elemente, die nur zwei Zustände kennen, nennt man DIGITAL. Darum heißt der Computer auch Digitalrechner.

Der Einfachheit halber nennt man den einen Zustand "1" und den anderen "0".

Rechnen kann man mit 0 oder 1 erst, wenn man mehrere Elemente miteinander kombiniert. Kombinieren wir versuchsweise drei digitale Elemente, drei Lampen, und überlegen, wie viele Möglichkeiten es gibt, wenn jede Lampe leuchten oder dunkel sein kann:

| | | | |
|---|---|---|---|
| ○ ○ ○ | oder | 0 0 0 | 0 |
| ○ ○ ☼ | | 0 0 1 | 1 |
| ○ ☼ ○ | | 0 1 0 | 2 |
| ○ ☼ ☼ | | 0 1 1 | 3 |
| ☼ ○ ○ | | 1 0 0 | 4 |
| ☼ ○ ☼ | | 1 0 1 | 5 |
| ☼ ☼ ○ | | 1 1 0 | 6 |
| ☼ ☼ ☼ | | 1 1 1 | 7 |

"Lampe leuchtet" soll einer 1 und "Lampe ist dunkel" einer 0 entsprechen.

Das sind 8 verschiedene Kombinationen; allgemein gilt die Regel, daß bei n-Elementen $2^n$ Kombinationen möglich sind. In diesem Beispiel sind es $2^3 = 8$ Kombinationen, die wir (rechte Spalte) mit 0 bis 7 bezeichnen. Das sind Zahlen im üblichen Dezimalsystem, das die Ziffern von 0 bis 9 benutzt. Von diesen zehn Ziffern hat das System seinen Namen (lateinisch zehn = decem).

Links neben den Dezimalzahlen ist eine weitere Zahlenreihe. Jedes Element aber nimmt nur zwei verschiedene Zustände an (0 und 1). Deshalb spricht man hier von einem Dual-System (lateinisch zwei = duo). Gebräuchlich ist auch der Ausdruck BINÄR-Zahlen. Einzelne Binärelemente oder Binärstellen werden als BIT bezeichnet. Es ist ein Kunstwort aus dem Englischen: binary digit = Binärstelle. Als Hauptwort für die Bezeichnung eines Elementes wird es groß geschrieben (Bit), als Maßeinheit für die Anzahl von Binärstellen klein (bit).

Zählen und Rechnen mit Binärzahlen erfolgt nach den gleichen Gesetzmäßigkeiten wie im Dezimalsystem. Beim Zählen z.B. addiert man ganz rechts eine 1 so lange, bis die letztmöglichen Ziffern erreicht sind. Will man dann weiterzählen, so beginnt man mit der kleinsten Ziffer eine Spalte weiter links. Beim Dezimalsystem muß man nach der 9 eine neue Spalte "eröffnen", beim Dualsystem nach der 1. Die einzelnen Spalten haben nun eine unterschiedliche Wertigkeit. Beim Dezimalsystem sind es von rechts beginnend die Wertigkeiten 1, 10, 100, 1000 usw., oder anders ausgedrückt, die Potenzen zur Basis 10 ($10^0$, $10^1$, $10^2$, $10^3$ ...).

Beim Dualsystem haben die Spalten die Wertigkeiten $1 = 2^0$, $2 = 2^1$, $4 = 2^2$, $8 = 2^3$ usw. Hier sind es also die Potenzen zur Basis 2.

Binärzahlen haben beim Rechnen den großen Vorteil, daß das ganze Einmaleins heißt:

1 mal 0 = 0 und

1 mal 1 = 1

Ebenso einfach ist das Addieren und Subtrahieren. Der Nachteil besteht aber darin, daß Binärzahlen leicht sehr lang und unübersichtlich werden.
So sieht binär die Zahl 2819 so aus:

101100000011

Da dies sehr unübersichtlich und außerdem schwer zu behalten ist, greift man zur HEXA-DEZIMAL-Darstellung. Hierbei faßt man jeweils 4 Binärstellen zusammen:

1011 0000 0011

Jedes dieser Päckchen wird nun je nach seinem Inhalt durch eine der Ziffern 0...9 oder A...F ersetzt, wobei folgende Zuordnung gilt:

| | | | |
|---|---|---|---|
| 0000 | = | 0 | |
| 0001 | = | 1 | |
| 0010 | = | 2 | |
| 0011 | = | 3 | |
| 0100 | = | 4 | |
| 0101 | = | 5 | |
| 0110 | = | 6 | |
| 0111 | = | 7 | |
| 1000 | = | 8 | |
| 1001 | = | 9 | |
| 1010 | = | A | (= dezimal 10) |
| 1011 | = | B | ( 11) |
| 1100 | = | C | ( 12) |
| 1101 | = | D | ( 13) |
| 1110 | = | E | ( 14) |
| 1111 | = | F | ( 15) |

Die Binärzahl aus dem vorigen Beispiel heißt in hexa-dezimaler Schreibweise "B03":

| 1011 | 0000 | 0011 |
|---|---|---|
| B | 0 | 3 |

Natürlich arbeitet der Computer mit Binärzahlen, die hexa-dezimale Schreibweise ist nur eine Vereinfachung für den Benutzer.

## DAS RECHNEN MIT BINÄRZAHLEN

Üblicherweise kann ein Computer, wenn er rechnet, nur addieren. Die Subtraktion wird durch eine spezielle Addition ersetzt; Multiplikation wird durch wiederholtes Addieren, Division durch mehrfaches Subtrahieren erzielt. Alle anderen arithmetischen Operationen lassen sich auf die vier Grundrechenoperationen zurückführen.

Wie addiert und subtrahiert man Binärzahlen?

Nehmen wir 4-stellige Binärzahlen und rechnen 5 + 4 = 9:

| $2^3$ | $2^2$ | $2^1$ | $2^0$ | |
|---|---|---|---|---|
| 0 | 1 | 0 | 1 | (= 5) |
| 0 | 1 | 0 | 0 | (= 4) |
| ① | | | | ① = Übertrag |
| 1 | 0 | 0 | 1 | (= 9) |

oder 7 + 3 = 10

| | | | | |
|---|---|---|---|---|
| 0 | 1 | 1 | 1 | (= 7) |
| 0 | 0 | 1 | 1 | (= 3) |
| ① | ① | ① | | |
| 1 | 0 | 1 | 0 | (= 10) |

Wichtig hierbei ist, daß man beachtet:

1 + 1 = 0 + Übertrag und

1 + 1 + Übertrag = 1 + (neuer) Übertrag

Negative Zahlen werden als ZWEIER-KOMPLEMENT der entsprechenden positiven Zahl dargestellt. Das Zweierkomplement erhält man, indem man alle Bits in ihr Gegenteil verkehrt (aus einer 0 wird eine 1 und umgekehrt; hier spricht man vom EINERKOMPLEMENT) und anschließend rechts eine 1 addiert.

Beispiel:

| | | |
|---|---|---|
| | 0 0 0 0 0 0 0 1 | (= 1) |
| also: | 1 1 1 1 1 1 1 0 | (= Einerkomplement von 1) |
| | 1 1 1 1 1 1 1 0 | |
| + | 1 | |
| | 1 1 1 1 1 1 1 1 | (= Zweierkomplement von 1) |

Da das Zweierkomplement eine negative Zahl ist, müßte das Ergebnis bei einer Addition von +1 und -1 Null sein:

```
    0 0 0 0   0 0 0 1        (= +1)
+   1 1 1 1   1 1 1 1        (= -1)
-------------------------
(1) 0 0 0 0   0 0 0 0        (= 0)
```

Wie wir sehen, stimmt die Annahme allerdings nur, wenn man den vordersten Überlauf unberücksichtigt läßt.

Der Computer führt nun eine Subtraktion durch, indem er den Subtrahenden negativ macht und dann addiert.

Beispiel: 19 - 5 = 14

```
                    +5 = 0 0 0 0   0 1 0 1
Einerkomplement von 5  = 1 1 1 1   1 0 1 0
Zweierkomplement von 5 = 1 1 1 1   1 0 1 1
```

also:

```
    0 0 0 1   0 0 1 1        (= +19)
    1 1 1 1   1 0 1 1        (= -5)
(1)(1)(1)(1)   (1)(1)
-------------------------
    0 0 0 0   1 1 1 0        (= 14)
```

Negative Binärzahlen erkennt man daran, daß das äußerste linke Bit gleich 1 ist.

## DATEN UND WORTE

Zu den Hauptfunktionen, die ein Computer ausführen kann, gehört das Speichern von DATEN. Das sind Binärzahlen, aber auch Namen, Texte und Anweisungen, die der Programmierer dem Computer gibt, damit dieser weiß, was er zu tun hat. All diese Daten werden in binärer Form gespeichert als irgendein Bit-Muster.

Der Computer hat eine Reihe von Speichermedien. Da sind einmal die Flip-Flop-Speicher, sehr schnelle, aber dafür ziemlich teure elektronische Speicher. Dann gibt es den Magnetkernspeicher, der etwas langsamer ist, aber dafür sehr viele Bits speichern kann. Neuerdings setzt man auch hochintegrierte Flip-Flop-Speicher in MOS-Technik ein, die genau die Mitte zwischen Kernspeichern und Flip-Flops bilden.

Bei allen Speichern ist immer eine bestimmte Anzahl von Bits zusammengefaßt. Diese Bits werden auf einmal abgelegt, addiert oder anderweitig behandelt. Die Anzahl Bits, die so zusammengefaßt ist, ist von Computer zu Computer unterschiedlich. Innerhalb eines Computer-Typs ist sie aber für alle Speicher gleich und stellt eine wichtige Kenngröße dar, die WORTLÄNGE. Ein Päckchen zusammengefaßter Bits nennt man ein WORT.

Weit verbreitet ist das 8-bit-Wort; man bezeichnet es als "BYTE".

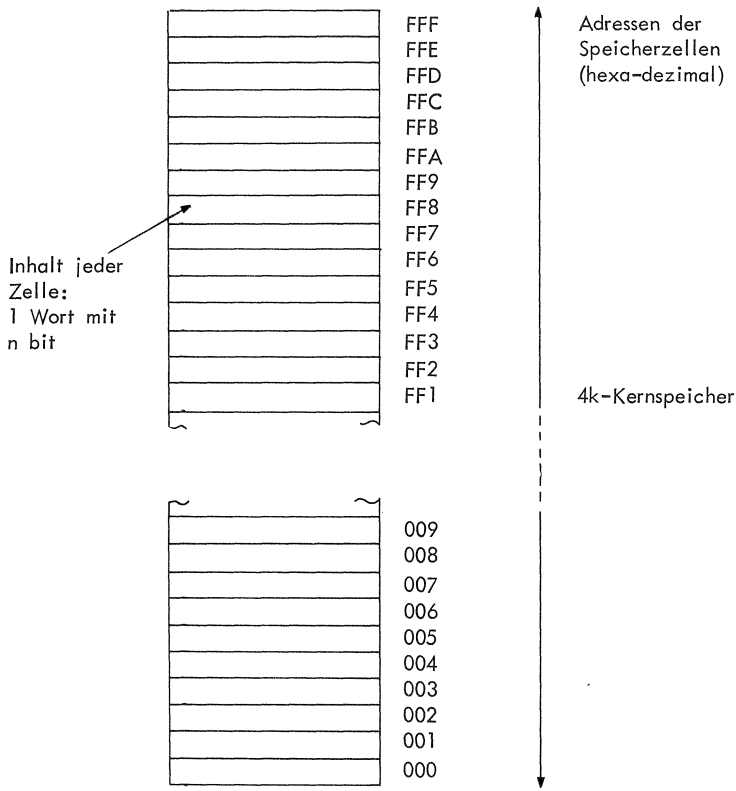
## REGISTER UND SPEICHER

Flip-Flop-Speicher von Wortlänge bezeichnet man als REGISTER. Allerdings kommt es auch vor, daß Register länger als ein Wort sind, z.B. 2-byte-Register (= 16-bit-Register).

Der KERNSPEICHER ist in der Lage, sehr viele Worte zu speichern. Im allgemeinen sind es $2^{12}$ = 4096 Worte (man spricht hier von 4k) oder ein Vielfaches hiervon. Will man ein bestimmtes Wort herausholen (lesen), so muß man dem Kernspeicher eine zusätzliche Information, die ADRESSE, geben, damit das richtige Wort gefunden wird. Jede SPEICHERZELLE hat also eine feste Adresse, aber einen variablen Inhalt von Wortlänge.

Adressen sind ebenfalls binär aufgebaut. Bei einem 4k-Speicher sind alle Adressen durch 12-stellige Binärzahlen - also 12 bit - darstellbar; übersichtlicher bezeichnet man sie mit 3-stelligen Hexa-Dezimalzahlen:

| | | | | |
|---|---|---|---|---|
| 1. Adresse | 0 0 0 0 | 0 0 0 0 | 0 0 0 0 | 000 |
| 2. Adresse | 0 0 0 0 | 0 0 0 0 | 0 0 0 1 | 001 |
| ... | | | | |
| usw. | | | | |
| ... | | | | |
| vorletzte Adresse | 1 1 1 1 | 1 1 1 1 | 1 1 1 0 | FFF |
| letzte Adresse | 1 1 1 1 | 1 1 1 1 | 1 1 1 1 | FFF |




Der MOS-Speicher ist genauso organisiert wie der Kernspeicher. Aber ein großer Teil der Adressen erfüllt die gleichen Funktionen, die früher von Flip-Flop-Registern erfüllt wurden. Deshalb ist es üblich, hier ebenfalls von REGISTERN zu sprechen. Register allerdings, deren Inhalt noch in wirkliche Flip-Flop-Register übertragen werden muß, bevor man mit ihm arbeiten kann.

## DAS PROGRAMM

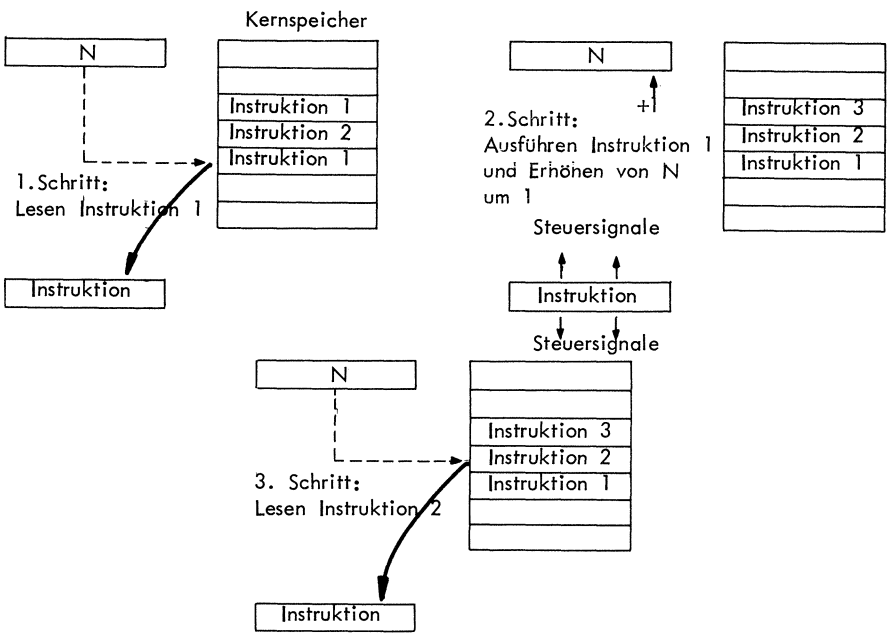
Speichern und Wiederauffinden von Daten ist zwar für einen Computer wesentlich, aber er kann noch mehr: Mit diesen Daten rechnen, sie manipulieren, ausgeben oder von außen aufnehmen. Aber all dies muß ihm genau vorgeschrieben werden. Dann führt er die gegebenen Anweisungen blitzschnell und sklavisch genau aus.

Das Erstellen solcher Anweisungen nennt man PROGRAMMIEREN. Eine Folge von Anweisungen ist ein PROGRAMM, und die einzelnen Anweisungen werden als INSTRUKTIONEN bezeichnet.

Ein fertiges Programm nimmt der Computer auf, indem er es Instruktion für Instruktion im Kernspeicher ablegt. Wenn man dann den Rechner startet, liest er die erste Instruktion aus dem Speicher und führt sie aus; dann liest er die zweite Instruktion, führt sie aus, dann die dritte und so fort, bis er schließlich eine Instruktion findet, die ihm sagt, daß er nun anhalten soll.

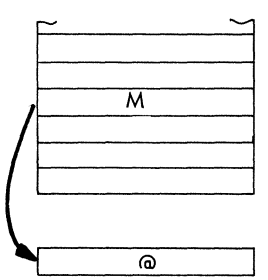
Um die Instruktionen aus dem Speicher zu lesen, benötigt der Computer eine Adresse. Diese Adresse muß natürlich "mitlaufen" und immer die Instruktion adressieren, die gerade ausgeführt werden soll. Dieses "Zählen" der Adressen übernimmt der INSTRUKTIONSZÄHLER (oder auch N-Register).

Die aus dem Speicher gelesenen Instruktionen werden in einem anderen Register gespeichert. Dieses Register erzeugt Steuersignale für das RECHENWERK (das eigentlich ausführende Organ des Computers) und bestimmt, mit welchen Daten gearbeitet werden soll.



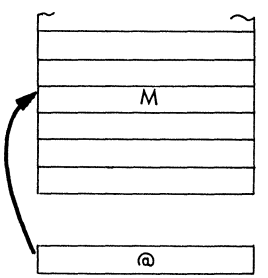
Im Kernspeicher des Computers stehen neben den Instruktionen auch die Daten, mit denen der Computer arbeitet. Natürlich kann nicht direkt im Kernspeicher gerechnet werden, sondern nur mit den Registern des Rechenwerkes. Das Haupt-Arbeitsregister ist der AKKUMULATOR (oder @ -Register). Besonders komfortable Computer verfügen über mehrere Akkumulatoren, die wahlweise benutzt werden können.

Ein wichtiger Arbeitsvorgang ist der Transport von Daten, z.B. aus dem Speicher in das @ -Register (LADEN, LOAD) oder aus dem @ -Register in den Speicher (SPEICHERN, STORE).




Speicher
Arbeitsregister

Laden aus Adresse M:
$\overline{<M>\rightarrow @}$




Speicher
Arbeitsregister

Speichern in Adresse M
$\overline{<@>\rightarrow M}$

@ ist das Symbol für das Arbeitsregister, M das für einen beliebigen Speicherplatz, und < ...> bedeutet "Inhalt von ...".

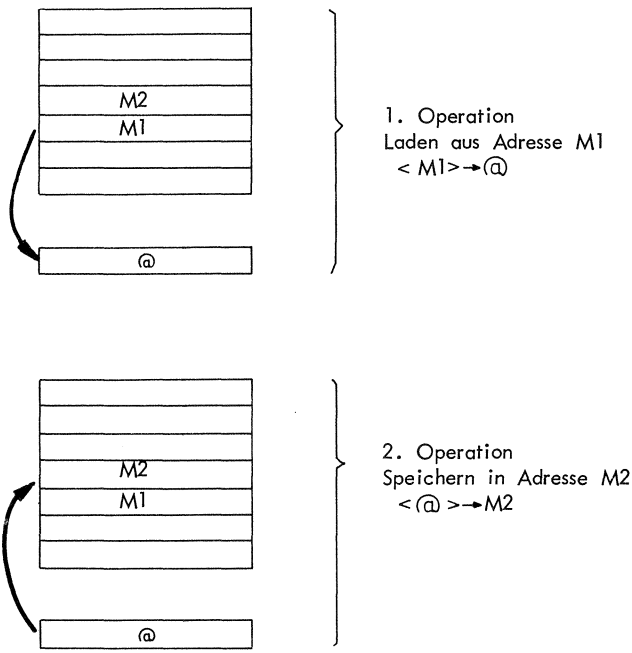
Bemerkenswert bei diesen Transportvorgängen ist, daß beim Datenempfang der alte Inhalt zerstört oder überschrieben wird, beim Senden aber erhalten bleibt. Beim Transport $<M>\rightarrow @$ haben anschließend M und @ den gleichen Inhalt, nämlich den ursprünglich nur in M gespeicherten.

Will der Programmierer Daten aus einer Kernspeicheradresse in eine andere transportieren, so geht das nur über das Arbeitsregister. Seine beiden Anweisungen lauten dann in symbolischer Form:

```
LDA,@ ,M1
STA,@ ,M2
```

Symbolisch bedeutet hierbei, daß die Befehle (Laden, Speichern) durch Abkürzungen (LDA, STA) und die Adressen durch NAMEN (M1, M2) - anstelle von z.B. hexa-dezimalen Adressen - angegeben sind. Das Signal @ bestimmt, mit welchem Arbeitsregister gearbeitet werden soll. Nur Computer mit mehreren Arbeitsregistern benötigen daher diese Angabe.

Wenn der Computer diese beiden Instruktionen ausführt, geschieht folgendes:




An diesem Beispiel kann man erkennen, welche Angaben der Computer benötigt:

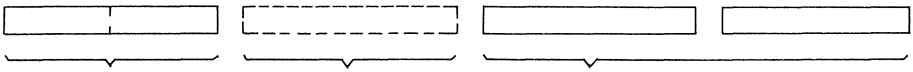
- Was ist zu tun? (Laden, Speichern)
  Diese Angabe nennt man den BEFEHL

- Um welchen Speicherplatz handelt es sich? (M1, M2)
  Diese Angabe nennt man die ADRESSE. Den Inhalt der Adresse, also der Wert, mit dem gearbeitet wird, bezeichnet man als OPERAND.

- Mit welchem Arbeitsregister soll gearbeitet werden? (@ )
  Diese Angabe nennt man die zweite Adresse.

Der Inhalt der (ersten) Adresse heißt OPERAND.

Beim MINCAL 621 sieht eine Instruktion so aus:

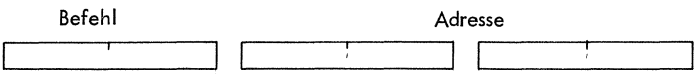



1. Byte: enthält den Befehl

2. Byte: enthält die Adresse des Arbeitsregisters. Sie wird nur dann angegeben, wenn man nicht mit dem Standard-Akkumulator arbeiten will.

3.+4. Byte: enthält die Operanden-Adresse.

Der Computer versteht nicht die symbolischen Befehle, sondern nur den MASCHINENCODE. Mit einem Übersetzungsprogramm, dem ASSEMBLER, wandelt er das SYMBOLISCHE Programm in Maschinencode um. Für die 2 obigen Befehle hätten wir folgendes Maschinencode-Ergebnis (Voraussetzung: M1 = Speicheradresse 01E2 und M2 = 01E3; @ ist der Standard-Akkumulator, deshalb keine 2. Adresse):

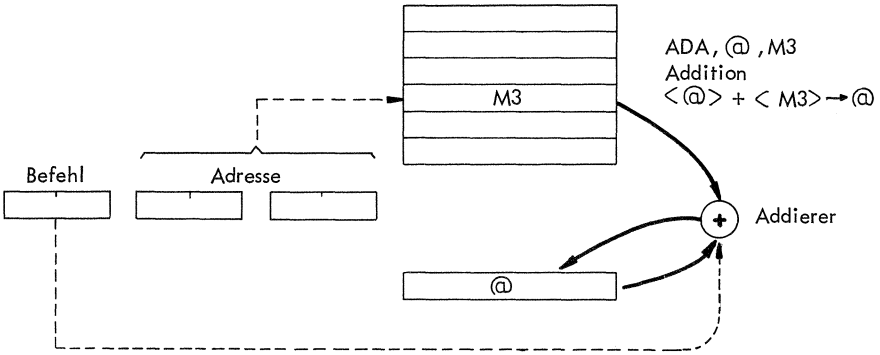



| | | | | | | |
|---|---|---|---|---|---|---|
| 8 | C | 0 | 1 | E | 2 | (LDA,@ ,M1) |
| E | C | 0 | 1 | E | 3 | (STA,@ ,M2) |

## MASCHINENBEFEHLE

Aber was kann der Programmierer dem Computer außerdem befehlen, was kann der Computer noch?

Da sind einmal die Befehle Addieren und Subtrahieren. Addieren bedeutet, daß die Binärzahl, die in einer Speicheradresse steht, zum Inhalt des Arbeitsregisters addiert wird:




Die Subtraktion läuft genauso ab, nur wird zwischen M3 und den Addierer ein Glied geschaltet, welches das Zweierkomplement des Operanden bildet.

Außer den arithmetischen Verknüpfungen zwischen Operand und Arbeitsregister gibt es noch die logischen Verknüpfungen

| | | | |
|---|---|---|---|
| Logisches UND: | 0 1 0 0 1 1 | < @ > | ANA, @ , Mn |
| | 0 1 1 1 0 1 | < Mn > | |
| | 0 1 0 0 0 1 | < @ > Ergebnis | |

Beim logischen UND erhält man pro Binärstelle als Ergebnis nur dann eine 1, wenn beide verknüpften Worte an dieser Stelle eine 1 enthielten. In allen anderen Fällen erhält man als Ergebnis eine 0.

| | | | |
|---|---|---|---|
| Inklusives ODER: | 0 1 0 0 1 1 | < @ > | ORA, @ , Mn |
| | 0 1 1 1 0 1 | < Mn > | |
| | 0 1 1 1 1 1 | < @ > Ergebnis | |

Bei inklusivem ODER erhält man pro Stelle als Ergebnis eine 1, wenn eines der Worte oder beide an dieser Stelle eine 1 enthielten. Nur wenn beide Bits 0 waren, erhält man als Ergebnis eine 0.

Exklusives ODER:

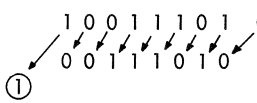
```
0 1 0 0 1 1     <@ >          EOA, @ , Mn
0 1 1 1 0 1     < Mn>
---------------
0 0 1 1 1 0     <@ >  Ergebnis
```

Beim exklusiven ODER erhält man pro Stelle als Ergebnis eine 1, wenn die verknüpften Worte an dieser Stelle ungleiche Binärziffern enthielten. Bei gleichen Binärziffern erhält man eine 0.

Diese logischen Verknüpfungen benötigt man zum Zerschneiden und Zusammensetzen von Daten und zum Feststellen, ob zwei Binärmuster gleich oder ungleich sind.

Außer den Befehlen, die einen Operanden mit dem Akkumulator verknüpfen, gibt es auch Befehle, die nur den Inhalt des Arbeitsregisters auf eine bestimmte Weise verändern. Hierzu gehören die Schiebebefehle. Der Inhalt des Akkus läßt sich rechts oder links verschieben, und das offen und geschlossen. Was hierbei passiert, kann man am besten an den Beispielen erkennen:

Schiften links offen

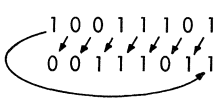



SLO, @

Ergebnis

Jedes Bit wird um eine Stelle nach links verschoben; das vorderste Bit geht verloren, und rechts wird eine 0 ergänzt.

Schiften links geschlossen (Rotieren)



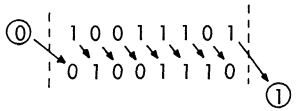

SLC, @

Ergebnis

Jedes Bit wird um eine Stelle nach links verschoben; das vorderste Bit wird in die rechts freiwerdende Stelle übertragen.

Entsprechend läuft das Schiften rechts ab:

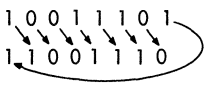
Schiften rechts offen




<@> SRO,@

<@> Ergebnis

Schiften rechts geschlossen (Rotieren)




<@> SRC,@

<@> Ergebnis

Außerdem kennt der Computer noch Instruktionen, mit denen er sich steuern läßt, z.B. Anhalten nach Erledigung der gestellten Aufgabe (Halt; HLT).

## EIN- UND AUSGABE

Der Computer kann Daten von außen aufnehmen oder seiner Umgebung vermitteln. Die PERIPHERIE, d.h. die mit dem Computer verbundene Umwelt, wird wie der Speicher behandelt. 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 GERÄTEADRESSE) zugeteilt, und Informationen werden in Form von Worten ausgetauscht, - wie beim Speicher. Die Verteilung der Daten erfolgt ebenfalls über das Arbeitsregister.

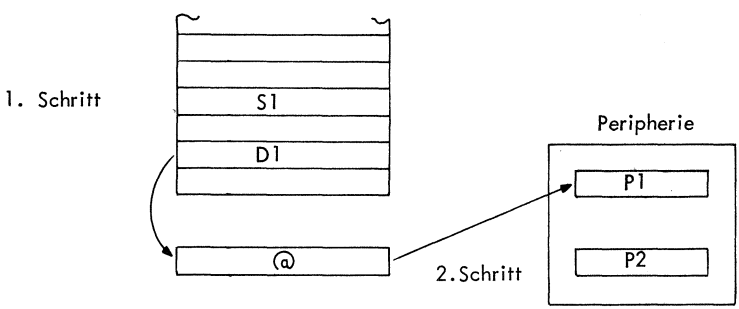
Eine typische Befehlsfolge für einen Ausgabevorgang sieht so aus:

1) Laden Datenwort aus Speicher — LDA, @ ,D1

2) Ausgabe Datenwort an Peripherie — STA, @ ,P1

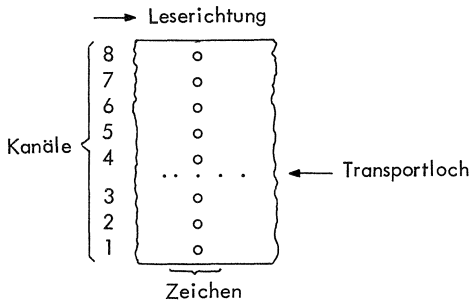
D1 = Kernspeicher-Adresse und

P1 = Externe Adresse.



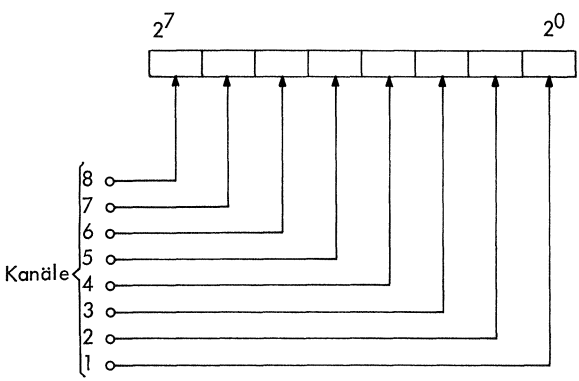
Normalerweise verstehen die Peripherie-Geräte nicht den Binärcode, mit dem der Computer rechnet. Sie haben ihren eigenen, z.B. den ASCII-Code. Dieser Code kommt auch in den Lochstreifen vor, die der Computer liest oder stanzt.

Ein Lochstreifen ist so aufgebaut:




Ein Zeichen auf dem Lochstreifen besteht aus 8 Lochreihen und einem kleineren Transportloch.

Die 8 Löcher (oder Nicht-Löcher) werden mit einem Mal gelesen und in das Arbeitsregister übernommen:



Jeder Buchstabe und jede Ziffer hat ein bestimmtes Code-Zeichen, zum Beispiel beim ASCII-Code:

| Kanal Bit | 8 7 6 5 4 3 2 1<br>$2^7 \leftarrow \quad \rightarrow 2^0$ | hexa-dez. Darstellung | ASCII-Code- Bedeutung | |
|---|---|---|---|---|
| | 0 0 1 1 0 0 0 0 | 30 | Ziffer | 0 |
| | 1 0 1 1 0 0 0 1 | B1 | " | 1 |
| | 1 0 1 1 0 0 1 0 | B2 | " | 2 |
| | 0 0 1 1 0 0 1 1 | 33 | " | 3 |
| | 1 0 1 1 0 1 0 0 | B4 | " | 4 |
| | 0 0 1 1 0 1 0 1 | 35 | " | 5 |
| | 0 0 1 1 0 1 1 0 | 36 | " | 6 |
| | 1 0 1 1 0 1 1 1 | B7 | " | 7 |
| | 1 0 1 1 1 0 0 0 | B8 | " | 8 |
| | 0 0 1 1 1 0 0 1 | 39 | | 9 |
| | 0 1 0 0 0 0 0 1 | 41 | Buchstabe | A |
| | 0 1 0 0 0 0 1 0 | 42 | " | B |
| | | ... | usw. | |

Der Kanal 8 trägt keine eigentliche Information. Er ist zur Kontrolle da und sorgt dafür, daß immer eine gerade Anzahl von Löchern gestanzt ist (PARITY-Bit).

Wird z.B. eine Ziffer eingelesen, so interessieren nur die rechten 4 Bit, denn sie entsprechen genau dem Binärcode. Also schneidet man die restlichen 4 Bit ab, indem man das ASCII-Zeichen und eine MASKE durch UND verknüpft:

| | | |
|---|---|---|
| 35 | 0 0 1 1 0 1 0 1 | = Ziffer 5 (ASCII) |
| OF | 0 0 0 0 1 1 1 1 | = Maske |
| 05 | 0 0 0 0 0 1 0 1 | = Ziffer 5 (Binär) |

Bei einer Ausgabe fügt man die fehlenden Bits durch inklusives ODER wieder hinzu:

| | | |
|---|---|---|
| 07 | 0 0 0 0 0 1 1 1 | = 7 (Binär) |
| B0 | 1 0 1 1 0 0 0 0 | = Ergänzung |
| B7 | 1 0 1 1 0 1 1 1 | = Ziffer 7 (ASCII) |

## WIR SCHREIBEN EIN PROGRAMM

Um nun all die gesammelten Erkenntnisse anzuwenden, wollen wir jetzt ein kleines Programm schreiben. Und zwar wollen wir 2 Zahlen eingeben, sie zueinander addieren und das Ergebnis anschließend wieder ausgeben.

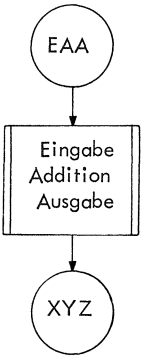
```
EAA:    LDA ,@ ,EXT     (Eingabe 1. Zeichen)
        ANA,@ ,MASK     (Maske)
        STA ,@ ,ZWS     (Zwischenspeichern)
        LDA ,@ ,EXT     (Eingabe 2. Zeichen)
        ANA,@ ,MASK     (Maske)
        ADA ,@ ,ZWS     (Addition)
        ORA ,@ ,ASCI    (Ergänzung)
        STA ,@ ,EXT     (Ausgabe Ergebnis)
        ...

MASK:   H   ,ØF

ASCI:   H   ,BØ

ZWS:    V

EXT:    Q   ,ØØF1       (Geräte-Adresse)
```

Hierzu ein paar Hinweise: Das Programm soll einen Namen haben: EAA (Eingabe/Addieren/Ausgabe), der als MARKE vor die erste Instruktion geschrieben wird.

Da im Programm Zwischenspeicher und Masken benutzt werden, müssen diese auch im Programm definiert werden, jeweils mit einer Linksmaske. Das geschieht mit den Symbolen V (VARIABLE) und H (Hexa-dezimaler FESTWERT). Mit Q wird dem symbolischen Gerätenamen EXT die Adresse ØØF1 zugewiesen. Die Ø (Null) wird durchgestrichen, um sie vom O (Oh) zu unterscheiden.

## SPRÜNGE UND SCHLEIFEN

Was geschieht, wenn der Computer dieses Programm ausgeführt hat? Er wird weiterlaufen und MASK als eine Instruktion auffassen. Das aber muß verhindert werden; andernfalls macht der Computer Unsinn. Ein ordnungsgemäßes Weiterlaufen erreicht man durch einen SPRUNG (oder VERZWEIGUNG) im Programm, z.B. zum Programmteil XYZ.

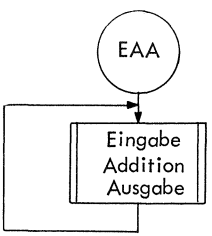



Dann muß an die Stelle der drei Pünktchen die Instruktion: JPA, , XYZ gesetzt werden. Damit wird der Instruktionszähler auf die Anfangsadresse des Programmteils XYZ gesetzt.

Oder man kann auch nach EAA zurückverzweigen:

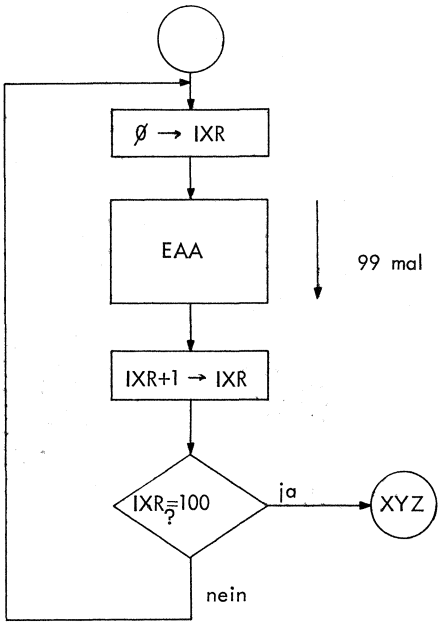
JPA, , EAA

Nun wiederholt sich der beschriebene Vorgang immer wieder.




Aus dieser Programmschleife kommt der Computer allerdings nie wieder heraus. Er liest Zahlen, adddiert sie und druckt das Ergebnis aus, und das ohne Ende.

Will man nur eine bestimmte Anzahl von Additionen durchführen, so muß ein Zähler mitzählen und bestimmen, wann aufgehört werden soll. Dieser Zähler ist das INDEXREGISTER. Es zählt jeden Durchlauf mit, und durch eine ABFRAGE stellt der Computer fest, ob schon der Endwert, z.B. 100, erreicht ist.




Wenn nein, geht das Programm nach EAA zurück, wenn ja - nach dem 99. Durchlauf - nach XYZ.

Das zugehörige Programm sieht so aus:

```
      LDC ,IXR,Ø
EAA   ...                    }   Programm EAA
      ...                    }
      ...                    }
      IEC ,IXR,100,XYZ
      JPA ,    EAA
```

LDC,IXR,Ø bedeutet, daß das Indexregister IXR mit einer Konstanten (CONSTANT) Ø geladen wird. Konstante heißt, daß die Adresse direkt als Operand genommen wird (und nicht ihr Inhalt!).

IEC,IXR,100,XYZ bedeutet: Addiere zu IXR eine 1 und springe, wenn der Inhalt gleich der Konstanten 100 ist, nach XYZ. Andernfalls laufe weiter auf die nächste Instruktion.

Solche Schleifenbildungen kommen in Programmen sehr häufig vor, und deshalb kann ein Computer gar nicht genug Indexregister haben.

In unserem Programm EAA stört aber noch, daß die Eingabe zweimal programmiert worden ist, was Platz kostet. Natürlich kann man das auch über eine Programmschleife erledigen. Besser ist für solche Fälle ein UNTERPROGRAMM, in das man über einen UNTERPROGRAMM-SPRUNG gelangt:

```
CSA,RET,EIN
```

Hierbei geschieht zweierlei: Erstens springt das Programm an die Stelle EIN, und insoweit verhält es sich wie ein normaler Sprung. Vorher aber wird der Instruktionszählerstand als RÜCKKEHRADRESSE in das Register RET übertragen. Am Ende des Unterprogramms, das die Befehle für Eingabe und Abspeichern enthält, ist ein RÜCKSPRUNG ins HAUPTPROGRAMM (an die Stelle nach dem Aufruf CS...) vorzusehen mit:

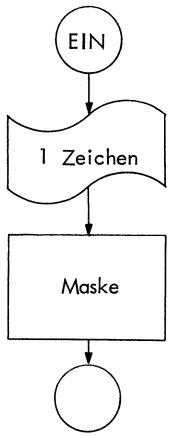
```
JPX,RET
```

Dies heißt: Springe indirekt über den Inhalt des Registers RET.

INDIREKT bedeutet, daß nicht zum Register RET gesprungen werden soll, sondern daß der Inhalt des Registers RET das Sprungziel angibt. Und hier steht ja die Rückkehradresse.

Hieran sieht man, daß die programmierte Adresse (nämlich RET) gar nicht die Adresse ist, mit der gearbeitet werden soll, Deshalb unterscheidet man auch die programmierte Adresse von der EFFEKTIVEN ADRESSE.

Unterprogramm EIN

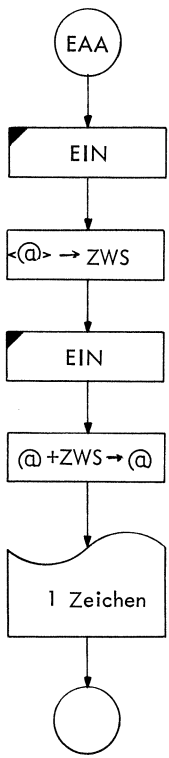



1 Zeichen lesen

Maskierung

Rücksprung

Das Programm EAA sieht nun so aus:




Unterprogramm EIN

Zwischenspeichern

Unterprogramm EIN

Addition

Ausgabe

## ADRESSIERUNG

Mit Ausnahme des JPX waren bisher programmierte Adresse und effektive Adresse gleich. Aber es gibt auch noch andere Fälle, wo beide Adressen nicht übereinstimmen; das liegt daran, daß der Computer verschiedene Arten der ADRESSIERUNG kennt.

In den ersten Beispielen wurde der Speicher und auch die Peripherie ABSOLUT addressiert, gekennzeichnet durch den Buchstaben A bei den Befehlen

| | | |
|---|---|---|
| | LDA | LOAD ABSOLUTE |
| oder | STA | STORE ABSOLUTE |

Die absolute Adresse ist 16 bit lang, und damit lassen sich 64k Speicherzellen adressieren. Die effektive Adresse ist gleich der programmierten Adresse.

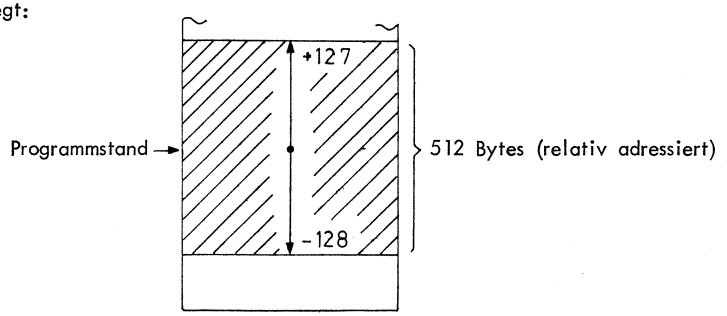
Außerdem haben wir beim Rücksprung aus dem Unterprogramm die INDIREKTE Adressierung kennengelernt. Hierbei ist die effektive Adresse gleich dem Inhalt der programmierten Adresse:

| | |
|---|---|
| LDX | LOAD INDIRECT |

Die "CONSTANT-Adressierung" ist ebenfalls schon erläutert worden. Hier ist die programmierte "Adresse" der Operand selbst:

| | |
|---|---|
| LDC | LOAD CONSTANT |

Neben diesen Adressierungsarten gibt es noch die RELATIVE (oder LATERALE) Adressierung. Hierbei geht man von der Annahme aus, daß viele Speicherplätze und Sprungziele in der Nähe der Instruktion stehen. Der Vorteil der relativen Adressierung ist, daß man mit einer 8-bit-Adresse auskommt; und dabei jeden Platz erreichen kann, der nicht weiter als 128 Adressen rückwärts bzw. 127 Adressen vorwärts liegt:

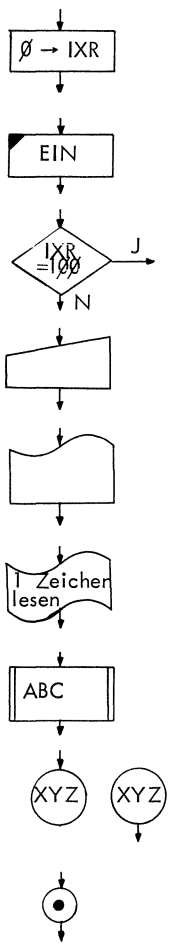


Bei relativer Adressierung ist die effektive Adresse gleich der Adresse der Instruktion ± programmierter Adresse:

LDL LOAD RELATIVE

## BLOCKDIAGRAMME

Nachzutragen bleibe eine Erklärung der grafischen Symbole. Sie sind Bestandteile von BLOCKDIAGRAMMEN (Programmablaufplänen) und sollen den Programmverlauf anschaulich darstellen. Die Kästchen werden durch Pfeile so miteinander verbunden, wie sie im Programm aufeinanderfolgen. Wichtige Symbole sind:




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), auch Unterprogramm

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

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

# Struktur

RECHENEINHEIT

Folgende Baugruppen bilden die wesentlichen Bestandteile der Recheneinheit:

A-Register: 8-bit-Register, in das alle gelesenen Daten gelangen und das als Rechenregister für arithmetische und logische Operationen dient. Der Inhalt des A-Registers wird angezeigt, sobald der Rechner angehalten wird.

B-Register: 8-bit-Register als zweites Rechenregister. Bei Indizierung enthält es die Indexregister-Adresse.

P-Register: 8-bit-Register zur Adressierung des Arbeitsregisters.

M-Register: 16-bit-Register für die effektive Adresse. Die beiden Hälften des M-Registers können angezeigt, und es können die Daten des Switch-Registers in das M-Register übertragen werden.

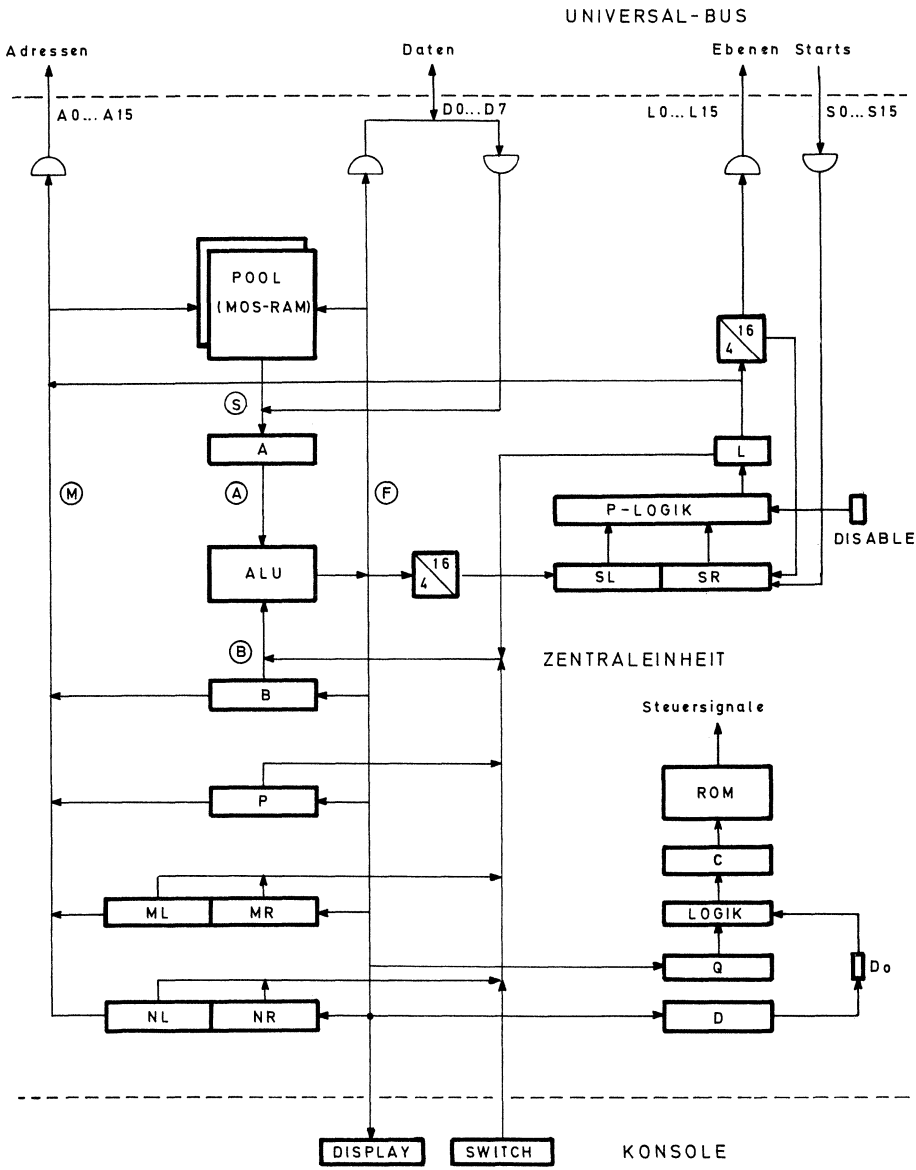
N-Register: 16-bit-Register, das als Instruktionszähler dient. Bei angehaltenem Rechner enthält das N-Register die Adresse des Befehls, der als nächster ausgeführt wird. Der Inhalt des N-Registers kann angezeigt, und es können die Daten des Switch-Registers in das N-Register übertragen werden.

SW-Register: 8-bit-Schaltersatz in der Bedienungskonsole (Option). Die Daten des SW-Registers können in das M-Register, N-Register oder eine Speicheradresse übertragen werden.

DISPLAY: 8-bit-Lampenfeld in der Bedienungskonsole (Option) zeigt den Zustand des F-Kanals an. Bei Stop wird der Inhalt des A-Registers angezeigt, wenn nicht über spezielle Schalter das M-Register oder das N-Register angewählt ist.

ALU: Arithmetisch-logische Einheit für 8 bit. Die ALU ist der zentrale Verknüpfungspunkt des Rechners.

Q-Register: 8-bit-Register für das Befehls-Byte der Instruktion.




BLOCKBILD MINCAL 621

D-Register: 8-bit-Register, das in einer DO-Schleife die Zahl der Ausführungen zählt.

DO-Register: 4-bit-Speicher für die Steuerinformationen eines DO-Befehls.

C-Register: 5-bit-Register für die Adresse der im ROM gespeicherten Mikroschritte.

ROM: TTL-Read Only Memory, das die Steuersignale (Mikroschritte) für den Rechner erzeugt.

S-Register: 16-bit-Register (Option) zur Speicherung der Ebenenstarts. Starts können vom Universal-BUS kommen oder programmiert sein. Zurückgestellt werden die Bits des S-Registers nur vom Programm.

L-Register: 4-bit-Register, das die laufende Ebene angibt.

DISABLE: Das DISABLE-Flip-Flop (1-bit-Register) verhindert Ebenenwechsel bzw. läßt im ausgeschalteten Zustand einen Ebenenwechsel zu.

P-Logik: Die Prioritätslogik ermittelt die höchste gestartete Ebene und setzt das L-Register entsprechend.

Alle genannten Register sind in Form integrierter Schaltkreise in der Recheneinheit enthalten. Sie sind für das Verständnis der Rechner-Struktur wichtig, jedoch für die Programmierung - mit Ausnahme von N- und SW-Register - nicht von Bedeutung, da der Benutzer keinen Zugriff zu ihnen hat. Vielmehr arbeitet der Benutzer mit Speicherplätzen im Arbeitsspeicher (Pool), die "seine" Register darstellen.

## ARBEITSSPEICHER (POOL)

Der Arbeitsspeicher ist ein MOS-RAM mit 256 bytes Kapazität (erweiterbar auf insgesamt 4k bytes). Er dient als schneller Datenspeicher; insbesondere aber enthält er die Arbeits- und Indexregister.

Jeder Programmebene werden (fest einstellbar) 16, 32, 64, 128 oder 256 bytes zugeteilt; diese Plätze bilden den "Pool". Registeradressen beziehen sich auf diese Bereiche, d.h. je nachdem, in welcher Ebene das Programm läuft, werden unterschiedliche Pools benutzt. (Will man diese Niveau-Bindung nicht, benutze man "absolute" Adressierung).

Die MOS-RAM-Adressen laufen von ØØØ bis ØFF bei 0,25k (bzw. ØØØ bis FFF bei 4k).

Register- und Pool-Adressen beziehen sich auf den Anfang des jeweiligen Pools. Im Prinzip sind alle Pool-Adressen von ØØ bis max. FF anzusprechen (als Pool-Adressen, spezifizierte Arbeitsregister und Indexregister), jedoch beachte man folgendes:

> Pool-Adressen ØØ und Ø1 nehmen bei Ebenenwechsel den augenblicklichen Programmstand auf und sind daher anderweitig nicht benutzbar.
>
> Pool-Adresse Ø2 (und - bei Mehrbyte-Operationen - die folgenden) stellt den "Akku" dar (für den Fall, daß kein spezifiziertes Arbeitsregister angegeben ist).

Ist der Pool nicht auf 256 byte Länge eingestellt, so reichen Pool-Adressen, die nicht mehr realisiert sind, in den Pool der nächsthöheren oder eventuell den Pool mehrerer höherer Ebenen. Die Pool-Bereiche schließen unmittelbar aneinander an.

## KERNSPEICHER

Der Kernspeicher hat 4k, 8k oder 16k (bzw. bei externer Erweiterung bis 48k) bytes Kapazität; seine Adressen laufen von 4ØØØ bis 4FFF, 5FFF oder 7FFF (bzw. bis FFFF).

Der Kernspeicher enthält Programm und Daten in beliebiger Weise.

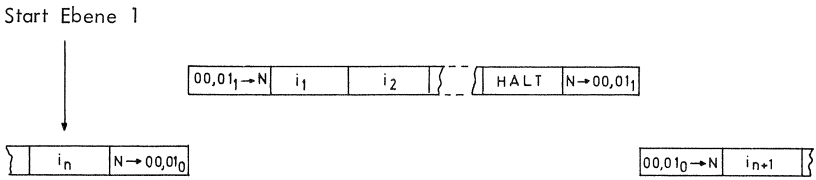
## PROGRAMMEBENEN

Es sind 2 oder 16 verschiedene Programmebenen vorgesehen. Jede Ebene ist dadurch gekennzeichnet, daß

ihr ein eigener Bereich (Pool) im Arbeitsspeicher zugeordnet ist, auf den sich die programmierten Register-Adressen beziehen,

für sie ein eigener E/A-Kanal aufgebaut werden kann, auf den sich die Geräteadressen beziehen (bei Benutzung der Ebenenausgänge als zusätzliche Adreßinformation).

Das Programm in einer Ebene wird von seinem äußeren Signal, durch die Fertigmeldung eines ihr zugeordneten Peripheriegerätes, oder durch das in einer anderen Ebene laufende gestartet. Läuft das Programm in keiner anderen oder in einer niedrigeren Ebene, so wird die gestartete Ebene sofort bzw. mit Ende der laufenden Operation aktiv und führt die nächste Instruktion aus (deren Adresse in Platz ØØ/Ø1 des Pools gespeichert war). Mit einem Halt wird das Programm angehalten, und eine niedrigere Ebene kann weiterlaufen.




$00,01_0$ = Pool-Adressen 00,01 der Ebene 0

$00,01_1$ = " 00,01 " " 1

$i_n$ = Instruktion n eines Programms

Durch Setzen von DISABLE kann der Ebenenwechsel (d.h. die Unterbrechung des Programms durch Start einer höheren Ebene) verhindert und wieder zugelassen werden.

Die Ebenen-Struktur erlaubt einfache Multiprogrammierung.

Hat der Computer mehrere, völlig unabhängige Aufgaben zu erledigen, so wird man jeder Aufgabe eine Ebene zuteilen. Man hat nur darauf zu achten, daß Aufgaben, die eine besonders schnelle Reaktion verlangen, einer Ebene mit hoher Priorität zugeordnet werden. Jede Ebene hat ihr eigenes Programm und eigene Datenspeicher.

Bei Ein- und Ausgaben muß durch Anhalten der Ebene auf das Peripheriegerät gewartet werden, damit die Zentraleinheit für die anderen Ebenen frei wird.

Besonders geeignet ist die Struktur des MINCAL 621 auch für die Bearbeitung von mehreren völlig gleichen Aufgaben, bei denen nur die Peripheriegeräte und die Daten unterschiedlich sind. In diesem Falle genügt es, ein Programm zu haben, das alle Ebenen benutzen. Nur bei Ansprechen der Peripheriegeräte benötigt man eine zusätzliche Information, denn bei gleichem Programm haben die Peripheriegeräte auch die gleichen Adressen. Diese zusätzliche Information liefert der Ebenen- (Level-) Ausgang, der als zusätzliche Adreßinformation verwertet wird. Bei den Datenspeichern erhält man ebenengebundene Adressen durch Register-Adressierung.

Da die Rückkehradressen der Unterprogramme ebenengebunden abgelegt werden, können auch Unterprogramme von mehreren Ebenen benutzt werden.

Will man von einer Ebene aus andere Ebenen steuern, so geschieht dies von der höchsten Ebene aus.

## REGISTER

Mit "Registern" sind Speicherplätze im jeweiligen Pool gemeint. Sie werden verwendet als:

Arbeitsregister: Die Mehrzahl der Befehle bezieht sich auf ein Arbeitsregister, das verändert, verglichen, geladen, transferiert oder sonstwie behandelt wird. Hierfür dient entweder der "Akkumulator" @ (Pool-Adresse Ø2) oder das in einem besonderen Byte "spezifizierte" Register (mit einer beliebigen Pool-Adresse). Damit stehen max. 254 Arbeitsregister bereit.

Indexregister: Wenn BUS-bezogene Befehle indiziert sind, dient die in einem besonderen Byte angegebene Pool-Adresse und folgende Adresse als Indexregister (2-byte-Indexregister). Ist die angegebene Pool-Adresse ungerade, wird nur diese Adresse als Indexregister verwendet (1 byte-Indexregister). Damit stehen max. 127 Indexregister zur Verfügung.

Rückkehradressen: Sie werden beim Unterprogrammsprung im spezifizierten Register sowie dem darauffolgenden Platz aufgehoben und beim Rücksprung dort wiedergeholt.

Pool-Adressen: Niveaugebundene Adressen (bei BUS-bezogenen Befehlen).

## ADRESSIERUNG

Die BUS-bezogenen Befehle können wie folgt adressiert werden:

| | | |
|---|---|---|
| unmittelbar | (CONSTANT) | wahlweise |
| niveaugebunden | (REGISTER) | |
| relativ | (RELATIVE) | |
| voll | (ABSOLUTE) | |
| nicht-indiziert | | wahlweise |
| indiziert | | |

CONSTANT bedeutet, daß die Adreßbytes als Konstanten verwendet werden. In Verbindung mit einem DO-Befehl kann es auch ein String von Konstanten sein.

REGISTER bezieht sich auf den Pool der jeweiligen Ebene.

RELATIVE bedeutet um einen Betrag von maximal -128 bzw. +127 bytes verschoben, bezogen auf das Byte, in dem die Adreßdifferenz angegeben ist.

ABSOLUTE bedeutet, daß jedes Byte des gesamten Speicherbereichs durch eine 16-bit-Adresse erreicht werden kann.

Indizierung bedeutet Addition des Indexregister-Inhaltes zur berechneten Adresse (Ø...65535 bei 2-byte-Indexregister; Ø...255 bei 1-byte-Indexregister). Mit 2-byte-Indexregistern lassen sich negative Indizes darstellen.

Indizierung in Verbindung mit CONSTANT führt zu indirekter Adressierung mit dem angegebenen Indexregister als indirekte Adresse.

## MEHRFACHAUSFÜHRUNG (DO-BEFEHL)

Ein besonderer Befehl (DO) erlaubt es, die folgende Instruktion 2- bis 256-mal auszuführen. Eine eventuell zum Befehl gehörende Adreßrechnung wird allerdings nur einmal durchgeführt.

Ergänzend kann man angeben, ob die Registeradresse (<&), die Operationsadresse (>&) oder beide (=&) bei jedem Durchlauf inkrementiert werden, ferner ob das Übertragungsbit (LINK) berücksichtigt werden soll (* statt &).

Der DO-Befehl kann sinnvoll auf Befehlsgruppen angewendet werden:

Schiebebefehle
Bedingte Sprungbefehle
BUS-bezogene Befehle

Schiebebefehle werden durch vorgeschaltetes "DO" zum 1-byte-Mehrbitschieben benutzt, indem die Registeradresse nicht inkrementiert (also die Mehrfachausführung auf 1 byte abgewendet wird) und kein Überlauf berücksichtigt wird.

```
4&SLO , @          4-bit-1-byte-Linksschieben (Akku)
```

1-byte-Mehrbit-Rotieren wird durch DO ohne Registerinkrement und ohne Berücksichtigung des Überlaufs erzielt:

```
2&SRC ,@           2-bit-1-byte-Rechts-Rotieren (Akku)
```

Mehrbyte-1-bit-Schieben erhält man, indem der DO-Befehl die Registeradresse inkrementiert und außerdem den Überlauf berücksichtigt:

```
2<*SRO,REG         1-bit-2-byte-Rechtsschieben (2 Registerplätze)
```

Wichtig hierbei ist, daß beim Linksschieben als erstes Byte (Basis-Byte) das mit der niedrigeren Adresse (und den niedrigwertigen Stellen) geschoben wird und dann das nächsthöhere Byte.

Beim Rechtsschieben wird dagegen beim Byte mit der höchsten Adresse (und den höherwertigen Stellen) begonnen und dann mit dem nächstniedrigeren Byte weitergearbeitet. Im Falle des Rechtsschiebens (und nur dann) dekrementiert der Rechner die Registeradresse. Im Maschinencode sind also die Basis-Bytes bei Rechts- und Linksschieben unterschiedlich (nicht dagegen bei Benutzung des Assemblers).

Will man beim Schieben von einem oder mehreren Bytes einen Gesamtüberlauf berücksichtigen, so muß 1 Byte mehr als gewünscht geschoben werden. Der Inhalt dieses Bytes ist vorher zu löschen. Anschließend kann in diesem Byte der Überlauf abgefragt werden.

Programmiert man beim Mehrbyte-Schieben den Überlauf nicht, so werden die benachbarten Bytes unabhängig voneinander um 1 bit verschoben.

Bedingte Sprungbefehle mit vorgestelltem DO dienen zu folgenden Zwecken:

Abfrage eines Strings von Register - Bytes auf Null/nicht Null.
Hierzu ist DO mit Inkrementieren der Registeradresse zu programmieren (der Überlauf ist hierbei irrelevant):

```
3<& BZ,REG,ADR     Sprung nach ADR, falls sowohl REG als
                   auch die beiden folgenden Bytes Nullinhalt haben
```

Vergleich eines Register-Strings mit einer im Befehl angegebenen Konstanten und Verzweigung, falls der Inhalt aller Register-Bytes gleich der Konstanten ist. Zu programmieren ist: DO mit Inkrementieren Registeradresse:

| | |
|---|---|
| 4<& BEC, @ ,ADR | Sprung nach ADR, wenn der Inhalt des Akkus und der drei folgenden Bytes gleich der Konstanten ist. |

Will man einen String von Register-Bytes mit einem anderen Register vergleichen, programmiert man:

4<& BER, @ , ADR

Vergleich eines Register-Strings mit einem Konstanten-String (im Befehl). Hierbei wird das erste Register-Byte mit der ersten Konstanten, das zweite Register-Byte mit der zweiten Konstanten, das zweite Register-Byte mit der zweiten Konstanten usw. verglichen. Verzweigt wird, wenn in allen Fällen Gleichheit besteht. Voraussetzung: DO mit Inkrementieren beider Adressen:

| | |
|---|---|
| 2=& BEC,REG,ADR | Sprung nach ADR, REG, wenn Register- und Konstanten-String gleich. |

Die Instruktion sieht in diesem Falle folgendermaßen aus:

| | | |
|---|---|---|
| n | | Befehl |
| n+1 | r | Basisadresse Register |
| n+2 | $c_0$ | Konstante (niedriges Byte = Basis-Byte) |
| n+3 | $c_1$ | Konstante (hohes Byte) |
| n+4 | d | Sprungweite |

Will man einen String von Register-Bytes mit einem anderen Register-String vergleichen, programmiert man:

2=& BER ,REG ,ADR

BUS-bezogene Befehle mit vorgestelltem DO erfüllen z.B. folgende Funktion:

Arithmetische Mehrbyte-Operationen sind durch einen DO-Befehl mit Inkrementieren sowohl der Registeradresse als auch der Operandenadresse sowie mit Berücksichtigung des Überlaufs darstellbar:

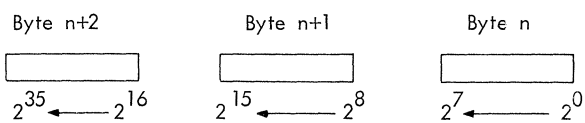
| | |
|---|---|
| 2=*ADL,REG,ADR | Doppel-Byte-Addition |
| 3=* SBL ,REG,ADR | 3-byte-Subtraktion |

Will man hierbei ein mögliches Überschreiten des Zahlenbereichs in positiver oder negativer Richtung berücksichtigen (Gesamt-Überlauf), so muß je ein Byte mehr verarbeitet werden.

Nicht möglich ist es allerdings, auf diese Weise zu 2 Bytes ein einzelnes Byte zu addieren.

Programmiert man z.B. nur das Inkrementieren der Registeradresse, aber nicht der Operandenadresse, wird derselbe Operand zunächst zum ersten Byte, anschließend zum zweiten Byte und eventuellen weiteren einzeln addiert.

Bei der Verarbeitung werden die Adressen grundsätzlich inkrementiert (Ausnahme: Rechtsschieben). Das führt dazu, daß das Byte mit der niedrigeren Adresse auch die niedriger wertigen Stellen enthalten muß:




Statt des Operanden kann in allen obengenannten Fällen auch mit einem Konstanten-String gearbeitet werden.

Blockweiser Transfer läßt sich durch Laden oder Speichern mit vorangestelltem DO (mit Inkrementieren Register und Operandenadresse) realisieren; er führt zur Übertragung von Registerbereichen in den Kernspeicher oder umgekehrt:

| | |
|---|---|
| 256=& STA,ØØ,ADR | Speichern gesamter Pool in Kernspeicher |
| 256=& LDA,ØØ,ADR | Laden gesamter Pool aus dem Kernspeicher |

Wird die Registeradresse nicht inkrementiert, so läßt sich z.B. ein Kernspeicherbereich mit dem gleichen Inhalt laden:

```
      LDC,REG,Ø   }   Löschen von 256 byte
256>& STA ,REG,ADR }  Kernspeicher
```

Anstelle von Operanden kann auch mit Konstanten oder Konstanten-Strings gearbeitet werden:

```
2Ø<& LDC,REG,Ø    Löschen von 20 bytes im Pool
```

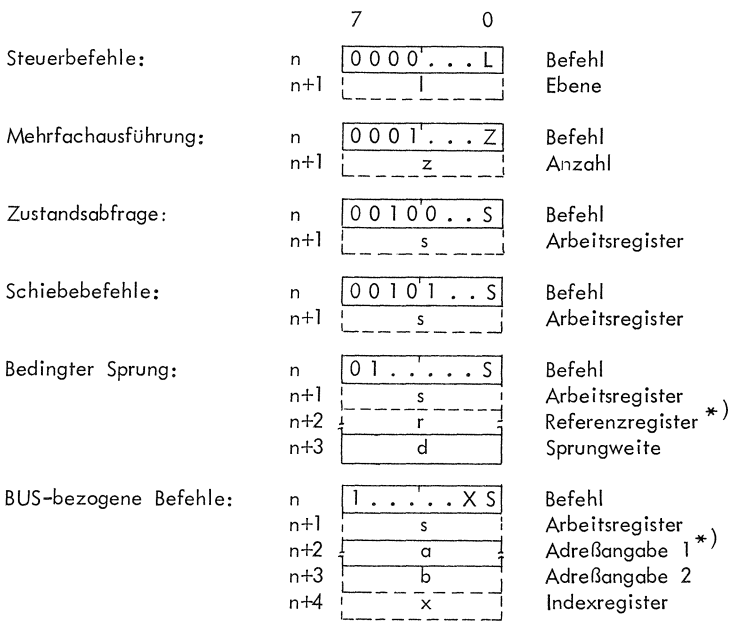
Sinngemäß läßt sich der DO-Befehl auch in Verbindung mit logischen Verknüpfungen verwenden:

```
2=& ANA, @ ,ADR    2 byte UND-Verknüpfung
7>& ORL ,REG,ADR   7 byte des Kernspeichers werden mit dem
                   Register REG durch ODER verknüpft
3<& ANC,REG,ADR    Maskierung von 3 Register-Bytes mit der
                   gleichen Maske ADR
```

## BEFEHLSAUFBAU

Instruktionen werden beim MINCAL 621 durch ein oder mehrere aufeinanderfolgende Bytes dargestellt. Das erste Byte enthält den Befehl sowie ggfs. einige zusätzliche Angaben, z.B. über die Art der Adressierung; in den Folgebytes stehen weitere Angaben, welche die Instruktion näher beschreiben, z.B. Register- und Operandenadressen, Konstanten, Sprungweiten usw.

Der Befehlsaufbau ist für die Befehlsgruppen im folgenden kurz skizziert; näheres entnehme man dem Abschnitt MASCHINENBEFEHLE.




*) oder Konstante bzw. Konstanten-String von z byte Länge bei vorgeschaltetem DO-Befehl

Eine DO-Instruktion muß unmittelbar vor dem Befehlsbyte der Instruktion liegen, deren Mehrfachausführung er bewirken soll.

## UNIVERSAL-BUS

Der Universal-BUS ist ein Datenkanal, der alle Systemkomponenten des Computers MINCAL 621 miteinander verbindet. Dieser eine Datenkanal erfüllt sowohl Aufgaben des programmgesteuerten Datentransfers und des direkten Speicherzugriffs. Die Datenübertragung erfolgt bit-parallel jeweils zwischen einem aktiven Element und einem passiven Element des Universal-BUS.

Das aktive Element belegt den Universal-BUS - bei mehreren aktiven Elementen entscheidet eine Rangordnung, ob eine Belegung möglich ist -, adressiert ein passives Element und bestimmt die Art der Datenübertragung. Das passive Element quittiert durch ein Fertig-Signal den Aufruf, und erst dann gibt das aktive Element den BUS frei.

Will ein passives Element eine Anforderung stellen, so kann es das nur über einen Interrupt (Ebenenstart). Hierauf wird die Datenverbindung von der CPU (Rechner-Zentraleinheit) zum passiven Element hergestellt.

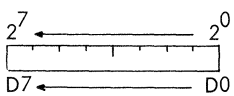
Zu den aktiven Elementen des Universal-BUS' zählen die CPU und alle Interfaces, die einen direkten Speicherzugriff haben. Diese Interface-Schaltungen wirken jedoch als passive Elemente, wenn sie die Ausgangsdaten und eine Arbeitsanweisung von der CPU erhalten.

Alle anderen Interfaces und auch die Speicher gelten grundsätzlich als passive Elemente.

Außer einigen Steuerleitungen sind alle Signalleitungen bidirektional; Eingangs- und Ausgangssignal sind somit identisch.

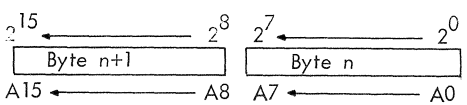
Der Universal-BUS hat folgende Signalleitungen:

D0...D7 — 8 Datenleitungen zum Datentransport zwischen einem aktiven und einem passiven Element.




Zuordnung Wertigkeit und Datenleitungen

A0...A15 — 16 Adreßleitungen zur Anwahl von Systemkomponenten durch ein aktives Element. Einzelne Komponenten können mehrere Adressen haben (z.B. Kernspeicher).

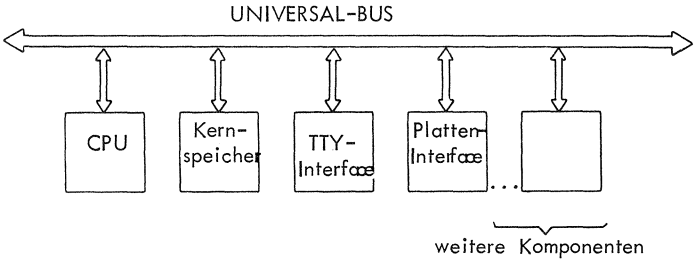


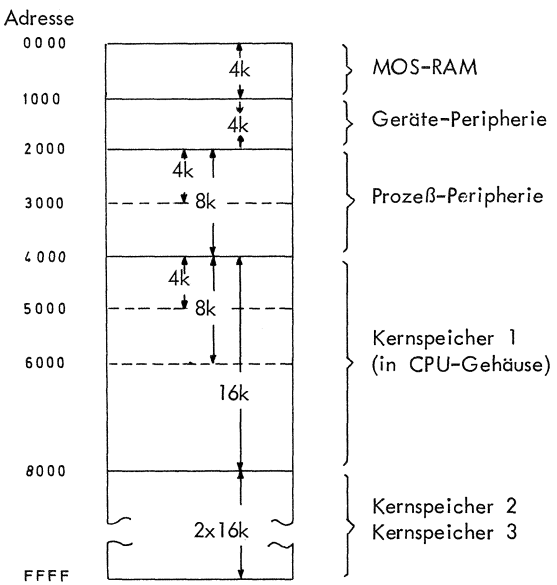

Zuordnung Wertigkeit und Adreßleitungen

| | |
|---|---|
| L0...L15 | 16 Level-Leitungen zur Anwahl von ebenen-gebundenen Komponenten. Diese Leitungen gehen nur von der CPU aus; sie sind nicht bidirektional. |
| S0...S15 | 16 Startleitungen (Interrupt-Leitungen) zum Starten einer Rechner-Ebene. Diese Leitungen gehen nur zur CPU. Sie sind nicht bidirektional. |
| RK | 1 Richtungskennzeichen gibt an, ob der Datenfluß von dem aktiven zum passiven Element verlaufen soll oder in entgegengesetzter Richtung (0 V = aktiv → passiv). |
| N | 1 Nullstelleitung. Mit dieser Leitung werden alle Flip-Flops in den Ausgangszustand gebracht. Sie kommt von der CPU (Taste in Bedienungsfeld bzw. Nullstellung bei wiederkehrendem Netz). |
| F | 1 Fertigleitung. Über die Fertigleitung quittiert das passive Element den Datenverkehr. |
| BE | 1 Belegt-Leitung. Diese Leitung wird von einem aktiven Element beschaltet, solange es den BUS belegt. |
| GE | 1 Gewünscht-Leitung. Über diese Leitung meldet ein aktives Element einen Belegungswunsch an. |

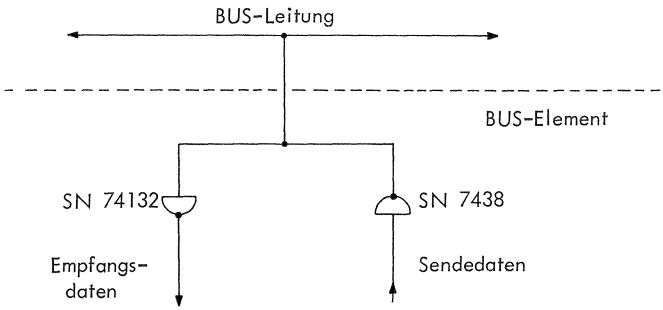
Alle Leitungen haben bei nicht belegtem BUS ein positives Potential. Bei belegtem BUS haben die Signale folgende Bedeutung:

0 V = 1
+5 V = 0



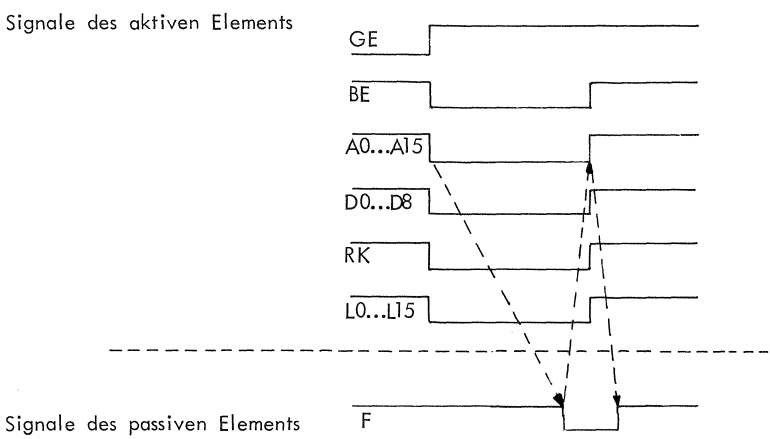

Adreßschema des MINCAL 621



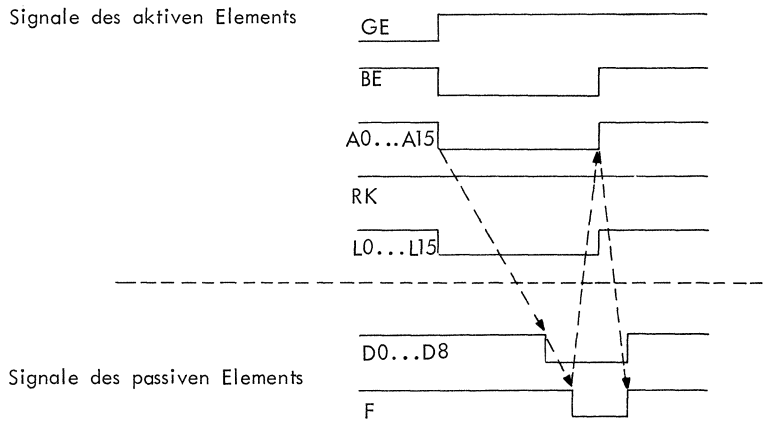

Für die Ankopplung an den BUS werden als Empfangsschaltung Schmitt-Trigger (SN 74 132) und für die Sendeschaltung Open-Collector-Buffer (SN 7438) verwendet.

Nachstehende Zeitdiagramme sollen den Datenverkehr zwischen aktivem und passivem Element erläutern. Dabei ist vorausgesetzt, daß der BUS nicht belegt ist und auch kein Belegungswunsch eines anderen aktiven Elements vorliegt.

Senden des aktiven Elements:

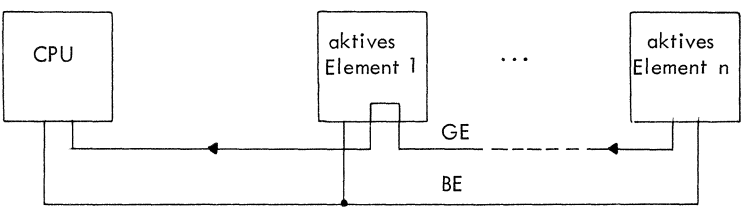


Senden des passiven Elements:

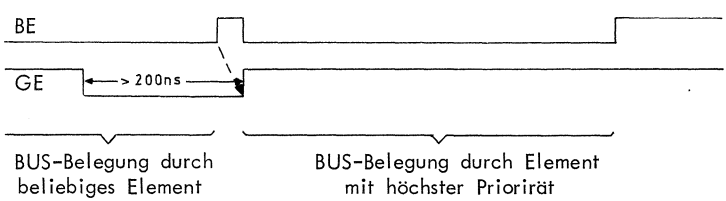


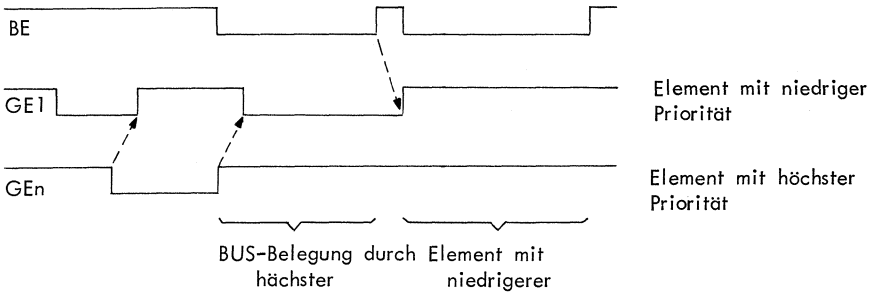

Das Signal "Fertig" wird erzeugt, nachdem die Adressen stabil anstehen und eine eingestellte Zeit abgelaufen ist. Die Daten werden mit der Adresse direkt durchgeschaltet.

Prioritäten bei mehreren aktiven Elementen werden durch die GE-Leitung ermittelt. Diese Leitung beginnt bei dem aktiven Element mit der höchsten Priorität und endet bei der CPU, die immer die niedrigste Priorität hat.




Das aktive Element mit der höchsten Priorität kann den BUS belegen, nachdem es eine bestimmte Zeit (mindestens 200 ns) die GE-Leitung beschaltet hat. Auf jeden Fall aber muß das GE-Signal so lange erzeugt werden, bis ein eventuell anstehendes BE-Signal verschwindet. Dann kann der BUS mit einem BE-Signal belegt und GE abgeschaltet werden

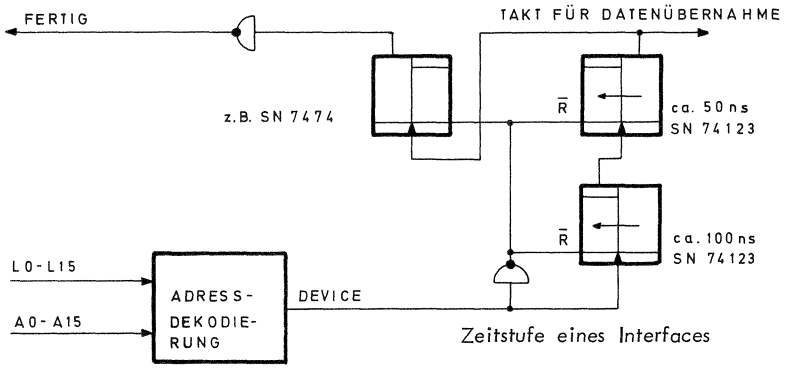




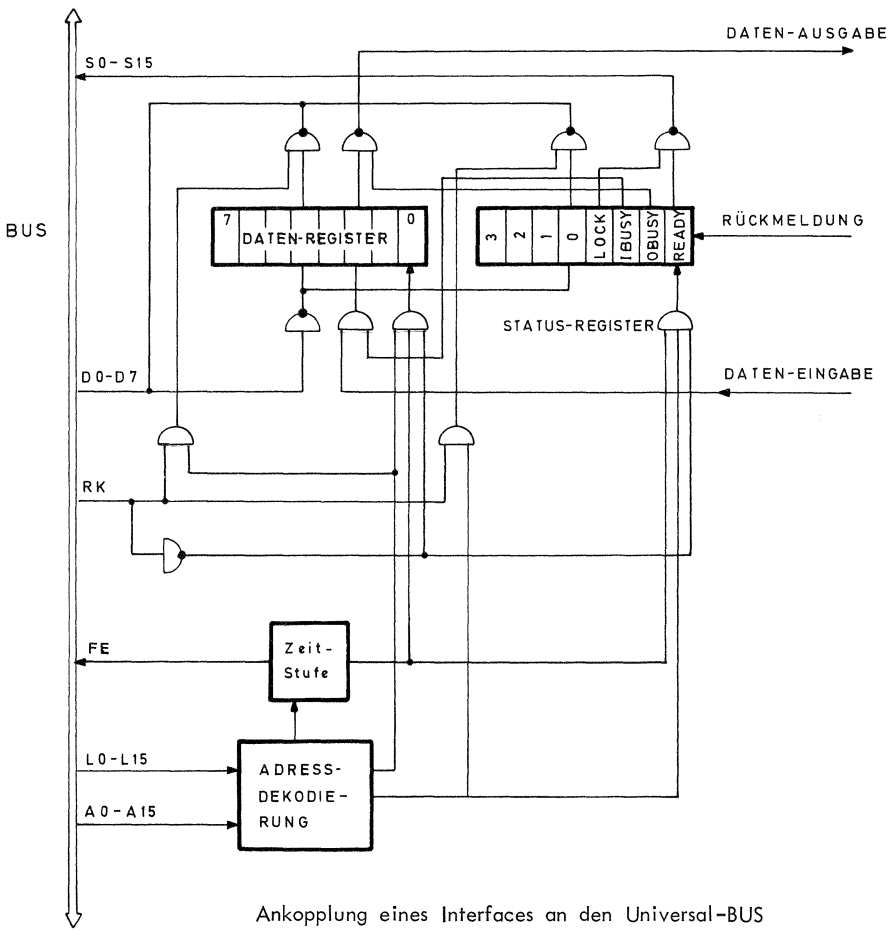

Ein aktives Element, das nicht die höchste Priorität hat, darf den BUS nur belegen, wenn es mindestens 200 ns das GE-Signal erzeugt hat und hierbei nicht durch ein GE eines aktiven Elements mit höherer Priorität unterbrochen worden ist.

Sobald solch ein Signal empfangen wird, muß GE weggeschaltet werden. Es darf erst dann weider erzeugt werden, wenn kein höherwertiges GE mehr vorliegt. Erst nach mindestens 200 ns erfolgreichem Beschalten von GE darf der BUS belegt werden.

Die CPU erzeugt kein GE-Signal. Sie belegt den BUS, wenn kein GE- und kein BE-Signal ansteht.




Zeitstufe eines Interfaces




Ankopplung eines Interfaces an den Universal-BUS

## INTERFACE-SCHALTUNGEN

Als Interfaces werden Schaltungen bezeichnet, die an den Universal-BUS angeschlossen sind und irgendwelche peripheren Geräte steuern, z.B. Fernschreiber, Locher und Leser, aber auch Meßgeräte und andere Prozeßperipherie.

Je nach angeschlossenem Gerätetyp sind die Interface-Schaltungen unterschiedlich; jedoch gibt es, vor allem auf der dem BUS zugewandten Seite, gemeinsame Merkmale, die im folgenden kurz beschreiben sind.

Wesentliche Funktionsgruppen sind:

Datenregister: 8-bit-Flip-Flop-Speicher, vom BUS aus einschreib- und abfragbar. Hält Daten für das periphere Gerät bereit (Ausgang) bzw. übernimmt sie von ihm (Eingabe).

Statusregister: 8-bit-Flip-Flop-Speicher, vom BUS aus einschreib- und abfragbar, mit den Funktionen:

- READY (Fertigmeldung vom Gerät, löst Start der Ebene aus)
- OBUSY (Ausgabe)
- IBUSY (Eingabe)
- LOCK (verhindert Ebenenstart durch READY)
- bis zu 4 weitere Funktionen je nach Art des peripheren Geräts

Adreßdekodierung: Entschlüsselt die Adresse A; verknüpft sie u.U. mit einer Programmebene L und wählt damit die beiden Register an, deren Adressen bis auf die letzte Stelle (A0) identisch sind (Statusregister: A0 = 0; Datenregister: A0 = 1).

Zeitstufe: Bestimmt die Dauer der Transferzeit zwischen CPU und Interface über den BUS.

Eine Datenausgabe zu einem Gerät erfolgt, sobald OBUSY eingeschaltet und READY ausgeschaltet ist. Die Geräte-Rückmeldung schaltet READY ein. Hiermit wird die Ausgabe unterbunden und ein START erzeugt.

Die Eingabe erfolgt sinngemäß durch Setzen von IBUSY.

Nur, wenn die Adressen mindestens 100 ns anstehen, wird ein Übernahmetakt und das Signal FE erzeugt. Verschwinden die Adressen, bevor die 100 ns abgelaufen sind, so werden die Monoflops zurückgestellt, und ein Übernahmetakt wird nicht erzeugt.
Bei langem BUS empfiehlt es sich, die Wartezeit zu verlängern.

# Maschinenbefehle

STEUERBEFEHLE

| 0 0 0 0 | . . . . |
|---|---|

| | | | |
|---|---|---|---|
| Befehle | NOP | Keine Operation | 0 0 0 0 0 0 0 0 |
| | SEL | Start Ebene | 0 0 0 0 0 0 0 1 |
| | HLT | Halt | 0 0 0 0 0 0 1 0 |
| | HSL | Halt, Start Ebene | 0 0 0 0 0 0 1 1 |
| | ECL | Unterbrechung zulassen | 0 0 0 0 0 1 0 0 |
| | DCL | Unterbrechung verhindern | 0 0 0 0 1 0 0 0 |

Länge 2 byte (SEL, HSL)
1 byte (übrige)

Die Steuerbefehle bewirken Start einer Programmebene, Anhalten des laufenden Programms sowie Aus- und Einschalten des DISABLE-Zustands. Der Befehl NOP (leeres Befehlsbyte) hat keine Funktion und wird übersprungen.

Die Anwendung der DO-Instruktion auf Steuerbefehle ist nicht sinnvoll.

**NOP** Keine Operation

n | 0 0 0 0 | 0 0 0 0 | Befehl

Funktion: Dieses Befehlsbyte wird übersprungen.

SEL Start Ebene

n | 0 0 0 0 | 0 0 0 1 | Befehl
n+1 | | 1 | | | Ebene

Funktion: Die im folgenden Byte als rechtsbündige Hexazahl Ø...F angegebene Ebene 1 wird gestartet.

HLT Halt

n | 0 0 0 0 | 0 0 1 0 | Befehl

Funktion: Das Programm in der laufenden Ebene wird angehalten.

HSL Halt, Start Ebene

n | 0 0 0 0 | 0 0 1 1 | Befehl
n+1 | | 1 | | | Ebene

Funktion: Das Programm in der laufenden Ebene wird angehalten. Die im folgenden Byte als rechtsbündige Hexazahl Ø...F angegebene Ebene 1 wird gestartet.

ECL Unterbrechung zulassen

n | 0 0 0 0 | 0 1 0 0 | Befehl

Funktion: Dieser Befehl stellt den Normalzustand her, in dem das Programm der jeweiligen Ebene durch den Start jeder höheren Ebene unterbrochen werden kann.

DCL Unterbrechung verhindern

n | 0 0 0 0 | 1 0 0 0 | Befehl

Funktion: Dieser Befehl stellt den DISABLE-Zustand her, in dem das Programm der jeweiligen Ebene nicht durch Aktivieren einer höheren Ebene unterbrochen werden kann. Durch ECL wird dieser Zustand beendet.

Vor jedem Halt-Befehl (HLT, HSL) ist der DISABLE-Zustand durch ECL zu verhindern, da sonst der Halt nicht wirksam wird.

## MEHRFACHAUSFÜHRUNG (DO)

| 0 0 0 1 . . . . |
|---|

| | | | |
|---|---|---|---|
| Befehle: | z& | | 0 0 0 1 0 0 0 . |
| | z * | mit L | 0 0 0 1 0 0 1 . |
| | z>& | M+1 | 0 0 0 1 0 1 0 . |
| | z>* | M+1, mit L | 0 0 0 1 0 1 1 . |
| | z<& | R+1 | 0 0 0 1 1 0 0 . |
| | z<* | R+1, mit L | 0 0 0 1 1 0 1 . |
| | z=& | R+1, M+1 | 0 0 0 1 1 1 0 . |
| | z=* | R+1, M+1, mit L | 0 0 0 1 1 1 1 . |
| Anzahl: | z = 2 | | . . . . . 0 |
| | z = 3...256 | | . . . . . 1 |

Länge:

1 byte (z = 2):

| | | |
|---|---|---|
| n | 0 0 0 1 . . . 0 | Befehl |

oder

2 byte (z = 3...256):

| | | |
|---|---|---|
| n | 0 0 0 1 . . . 1 | Befehl |
| n+1 | z | Anzahl |

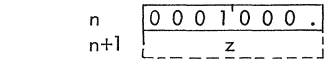
Eine DO-Instruktion bewirkt, daß die folgende Instruktion mehrfach ausgeführt wird. Die Anzahl der Ausführungen z kann von 2 bis 256 gewählt werden. Bei z = 2 ist nur das Befehlsbyte vorhanden; darüberhinaus steht z als rechtsbündige Binärzahl im folgenden Byte, wobei zu berücksichtigen ist, daß Nullinhalt des Folgebytes 256malige Ausführung bedeutet.

Im DO-Befehl kann angegeben werden, ob das LINK-Bit (L) berücksichtigt wird (Überlauf bei Schiebebefehlen, Mehrbyte-Addieren und Subtrahieren), und ob bei jeder Ausführung die Operanden-Adresse (M) oder die Arbeitsregister-Adresse (R) um 1 erhöht wird. Beim Befehl SR (Schieben rechts) wird die Arbeitsregister-Adresse um 1 erniedrigt, wenn R angegeben ist.

Der DO-Befehl führt nur zur mehrmaligen Wiederholung des Ausführungsteils der folgenden Instruktion, nicht zur Wiederholung der vorher ablaufenden Adreßrechnung.

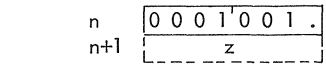
Zwischen einem DO-Befehl und beendeter Ausführung der nachfolgenden Instruktion kann das Programm nicht durch Wechsel der Programmebene unterbrochen werden.

z&  DO z-mal



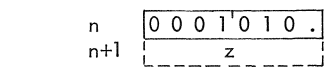

Funktion: Die folgende Instruktion wird z-mal ausgeführt. LINK wird nicht berücksichtigt; keine Adresse wird inkrementiert.

z*  DO z-mal mit Link



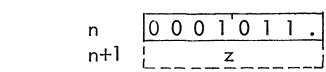

Funktion: Die folgende Instruktion wird z-mal ausgeführt. LINK wird berücksichtigt; keine Adresse wird inkrementiert.

z>&  DO z-mal
mit Inkrementieren Adresse



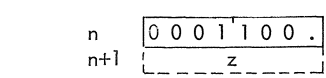

Funktion: Die folgende Instruktion wird z-mal ausgeführt. Nach jeder Ausführung wird die Operanden-Adresse M um 1 erhöht.

z>*  DO z-mal mit Link
mit Inkrementieren Adresse



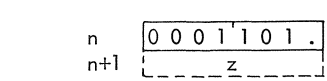

Funktion: Die folgende Instruktion wird z-mal ausgeführt. LINK wird berücksichtigt; nach jeder Ausführung wird die Operanden-Adresse M um 1 erhöht.

z<&  DO z-mal
mit Inkrementieren Register




Funktion: Die folgende Instruktion wird z-mal ausgeführt. Nach jeder Ausführung wird die Adresse des Arbeitsregisters um 1 erhöht.

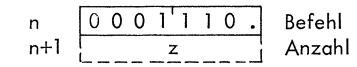
z<*  DO z-mal mit Link
mit Inkrementieren Register




Funktion: Die folgende Instruktion wird z-mal ausgeführt. LINK wird berücksichtigt; nach jeder Ausführung wird die Adresse des Arbeitsregisters um 1 erhöht.

z=&

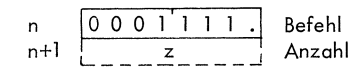
DO z-mal
mit Inkrementieren Register
mit Inkrementieren Adresse




Funktion: Die folgende Instruktion wird z-mal ausgeführt. Nach jeder Ausführung werden die Adresse des Arbeitsregisters und die Operanden-Adresse um 1 erhöht.

z=*

DO z-mal mit Link
Inkrementieren Register und




Funktion: Die folgende Instruktion wird z-mal ausgeführt. LINK wird berücksichtigt; nach jeder Ausführung werden die Adresse des Arbeitsregisters und die Operanden-Adresse um 1 erhöht.

## ZUSTANDSABFRAGE

| 0 | 0 | 1 | 0 | 0 | . | . | S |
|---|---|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| Befehle: | GS | Schalter abfragen | | 0 0 1 0 0 0 1 . | |
| | GL | Ebene abfragen | | 0 0 1 0 0 1 0 . | |
| Arbeitsregister: | | Akkumulator @ | | . . . . . 0 | |
| | | Spezifiziertes Register s | | . . . . . 1 | |
| Länge: | | 1 byte (@) | n | 0 0 1 0 0 . . 0 | Befehl |
| | | oder | | | |
| | | 2 byte (s) | n | 0 0 1 0 0 . . 1 | Befehl |
| | | | n+1 | s | Arbeitsregister |

Diese Befehlsgruppe überträgt Informationen von den Konsol-Tasten bzw. die Nummer der laufenden Programmebene in das Arbeitsregister. Als Arbeitsregister kann entweder der Akkumulator @ oder ein beliebig spezifiziertes Register s angegeben werden, dessen Adresse dann im Folgebyte steht.

Die Anwendung der DO-Instruktion auf diese Befehle ist nicht sinnvoll.

GS Schalter abfragen

| | | |
|---|---|---|
| n | 0 0 1 0 0 0 1 S | Befehl |
| n+1 | s | Arbeitsregister |

Funktion: Das über die 8 Daten-Schalter 7...0 der Rechnerkonsole eingegebene Byte wird in das Arbeitsregister übertragen.

Nur wirksam, wenn Rechner mit Konsole ausgestattet.

GL Ebene abfragen

| | | |
|---|---|---|
| n | 0 0 1 0 0 1 0 S | Befehl |
| n+1 | s | Arbeitsregister |

Funktion: Die Nummer der laufenden Programmebene wird als rechtsbündige Hexa-Zahl Ø...F in das Arbeitsregister übertragen.

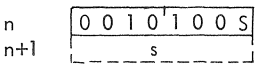
## SCHIEBEBEFEHLE

| | | | | |
|---|---|---|---|---|
| | | | | 0 0 1 0 1 . . S |
| Befehle: | SRO | Schieben rechts offen | | . . . . . 0 0 . |
| | SRC | Schieben rechts zyklisch | | . . . . . 0 1 . |
| | SLO | Schieben links offen | | . . . . . 1 0 . |
| | SLC | Schieben links zyklisch | | . . . . . 1 1 . |
| Arbeitsregister: | | Akkumulator @ | | . . . . . 0 |
| | | Spezifiziertes Register s | | . . . . . 1 |
| Länge: | | 1 byte (@) | n | 0 0 1 0 1 . . 0 — Befehl |
| | | oder | | |
| | | 2 byte (s) | n | 0 0 1 0 1 . . 1 — Befehl |
| | | | n+1 | s — Arbeitsregister |

Diese Befehlsgruppe bewirkt offenes oder zyklisches Schieben des Arbeitsregister-Inhalts um 1 bit nach rechts oder links. Als Arbeitsregister kann entweder der Akkumulator oder ein beliebig spezifiziertes Register s angegeben werden, dessen Adresse dann im Folgebyte steht.

In Verbindung mit einer geeigneten DO-Instruktion ist offenes und zyklisches Mehrbit-Schieben eines Register-Bytes möglich, sowie offenes Schieben des Inhalts eines Mehrbyte-Register-Strings um 1 bit.

SRO Schieben rechts offen

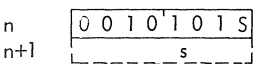



Funktion: Der Inhalt des Arbeitsregisters wird um 1 bit offen nach rechts verschoben. Bit 7 wird zu Null, und der vorherige Inhalt von Bit 0 geht verloren.

z& SR Ein vorgeschalteter DO-Befehl z& bewirkt offenes Rechts-Schieben des Arbeitsregister-Inhalts um z bit.

z<* SR Ein vorgeschalteter DO-Befehl z bewirkt offenes Rechts-Schieben eines Register-Srings von z byte Länge um 1 bit. Als Arbeitsregister-Adresse ist die um (z-1) erhöhte Basis-Adresse des Register-Strings anzugeben (gilt nicht für symbolische Programmierung).

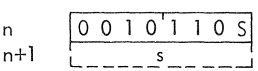
SRC Schieben rechts zyklisch




Funktion: Der Inhalt des Arbeitsregisters wird um 1 bit zyklisch nach rechts verschoben. Bit 7 erhält den vorherigen Inhalt von Bit 0.

z& SRC Ein vorgeschalteter DO-Befehl z& bewirkt zyklisches Rechts-Schieben des Arbeitsregister-Inhalts um z bit.

SLO Schieben links offen

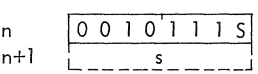



Funktion: Der Inhalt des Arbeitsregisters wird um 1 bit offen nach links verschoben. Bit 0 wird zu Null, und der vorherige Inhalt von Bit 7 geht verloren.

z& SL Ein vorgeschalteter DO-Befehl z& bewirkt offenes Links-Schieben des Arbeitsregister-Inhalts um z bit.

z<* SR Ein vorgeschalteter DO-Befehl z bewirkt offenes Links-Schieben eines Register-Srings von z byte Länge um 1 bit. Als Arbeitsregister-Adresse ist die Basis-Adresse des Register-Strings anzugeben.

SLC Schieben links zyklisch




Funktion: Der Inhalt des Arbeitsregisters wird um 1 bit zyklisch nach links verschoben Bit 0 erhält den vorherigen Inhalt von Bit 7.

z<* SLC Ein vorgeschalteter DO-Befehl z bewirkt zyklisches Links-Schieben des Arbeitsregister-Inhalts um z bit.

## BEDINGTE SPRUNGBEFEHLE

| | | 0 1 . . ' N . I S |
|---|---|---|
| Befehlsgruppen: | Abfrage auf Null/Positiv | 0 1 0 0 . . . . |
| | Abfrage auf Gleichheit | 0 1 0 1 . . . . |
| | Abfrage auf Testbits | 0 1 1 . . . . . |
| Sprung: | wenn Bedingung erfüllt | . . . . 0 . . . |
| | wenn Bedingung nicht erfüllt | . . . . 1 . . . |
| Inkrementieren: | nein | . . . . . 0 . |
| | ja | . . . . . 1 . |
| Arbeitsregister: | Akkumulator @ | . . . . . 0 |
| | Spezifiziertes Register s | . . . . . 1 |

Diese Befehlsgruppe fragt den Inhalt des Arbeitsregisters auf bestimmte Kriterien ab. Je nachdem, ob sie erfüllt sind oder nicht, verzweigt das Programm auf eine entfernte Stelle; andernfalls wird es mit der folgenden Instruktion fortgesetzt.

Abfrage-Kriterien sind:

Nullinhalt (alle Bits sind 0),
positiver Inhalt (Bit 7 ist 0);

Gleichheit mit einer Konstanten c;
Gleichheit mit Inhalt eines Referenzregisters r;

Vorhandensein bestimmter Bitmuster (Testbits), wobei deren Stellung durch eine Maske vorgegeben wird, die als Konstante c oder als Inhalt eines Referenzregisters r vorhanden ist.

Vor Abfrage kann das Arbeitsregister inkrementiert, d.h. sein Inhalt um 1 erhöht werden.

Durch diese Befehle wird der Inhalt des Arbeitsregisters - abgesehen von der eventuellen Inkrementierung - nicht verändert.

Die relative Sprungweite d steht im letzten Byte der Instruktion und bezieht sich auf dessen Adresse, wobei d eine Zweierkomplementzahl bildet. Damit kann das Programm maximal um 128 byte zurück bzw. 127 byte vorwärts springen.

DO-Befehle: Eine vorgeschaltete DO-Instruktion z<* (bei Abfrage auf Nullinhalt eines Registers) bzw. z=* (bei den übrigen Befehlen) bewirkt Abfrage eines Arbeitsregister-Strings von z byte Länge, bzw. dessen Vergleich mit einem ebenso langen Konstanten- oder Register-String.

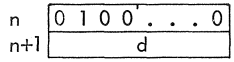
Dabei sind die Basis-Adressen der Register-Strings anzugeben.

Bei Abfrage auf Null ist die Bedingung erfüllt, wenn alle Bytes Nullinhalt haben.

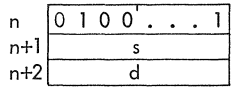
Bei Vergleichs- und Testbit-Abfragen ist die Bedingung insgesamt erfüllt, wenn sie in allen Bytes erfüllt ist.

Inkrementierung bezieht sich auch bei vorgeschaltetem DO nur auf das Basis-Byte des betreffenden Registers.

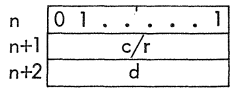
Länge: Abfrage auf Null/positiv: 2 byte (@)



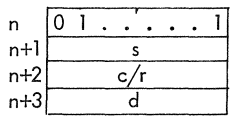

3 byte (s)




übrige Befehle: 3 byte (@)



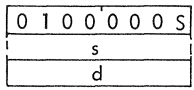

4 byte (s)




Bei einem vorgeschalteten DO-Befehl z=* ist statt einer Konstanten c ein Konstanten-String von z Bytes in der Instruktion enthalten, vom Basis-Byte an in Richtung aufsteigender Adressen.

Vereinbarung: Befehle mit Verneinung ("Sprung wenn nicht ...") führen in all den Fällen zum Verzweigen, wo der entsprechende nicht verneinte Befehl das Programm unverzweigt weiterlaufen läßt, und umgekehrt.

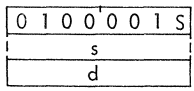
BZ — Sprung wenn Null




Befehl
Arbeitsregister
Sprungweite

Funktion: Das Programm verzweigt, wenn das Arbeitsregister Nullinhalt hat.

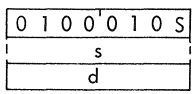
IZ — Inkrementieren, Sprung wenn Null




Befehl
Arbeitsregister
Sprungweite

Funktion: Das Arbeitsregister wird inkrementiert. Das Programm verzweigt, wenn das Arbeitsregister Nullinhalt hat.

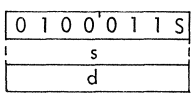
BP — Sprung wenn positiv




Befehl
Arbeitsregister
Sprungweite

Funktion: Das Programm verzweigt, wenn das Arbeitsregister positiven Inhalt hat.

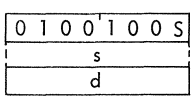
IP — Inkrementieren, Sprung wenn positiv




Befehl
Arbeitsregister
Sprungweide

Funktion: Das Arbeitsregister wird inkrementiert. Das Programm verzweigt, wenn das Arbeitsregister positiven Inhalt hat.

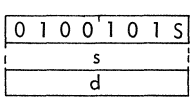
BNZ — Sprung wenn nicht Null




Befehl
Arbeitsregister
Sprungweite

Funktion: Das Programm verzweigt, wenn das Arbeitsregister nicht Nullinhalt hat.

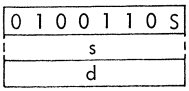
INZ — Inkrementieren, Sprung wenn nicht Null




Befehl
Arbei tsregister
Sprungweite

Funktion: Das Arbeitsregister wird inkrementiert. Das Programm verzweigt, wenn das Arbeitsregister nicht Nullinhalt hat.

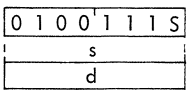
BNP Sprung wenn nicht positiv




Befehl
Arbeitsregister
Sprungweite

Funktion: Das Programm verzweigt, wenn das Arbeitsregister nicht positiven Inhalt hat.

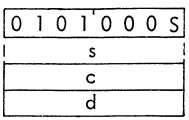
INP Inkrementieren, Sprung wenn nicht positiv




Befehl
Arbeitsregister
Sprungweite

Funktion: Das Arbeitsregister wird inkrementiert. Das Programm verzweigt, wenn das Arbeitsregister nicht positiven Inhalt hat.

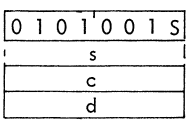
BEC Sprung wenn gleich Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

Funktion: Das Programm verzweigt, wenn der Inhalt des Arbeitsregisters gleich der Konstanten c ist.

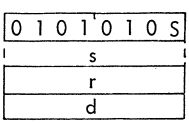
IEC Inkrementieren, Sprung wenn gleich Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

Funktion: Das Arbeitsregister wird inkrementiert. Das Programm verzweigt, wenn der Inhalt des Arbeitsregisters gleich der Konstanten c ist.

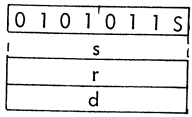
BER Sprung wenn gleich Register




Befehl
Arbeitsregister
Register
Sprungweite

Funktion: Das Programm verzweigt, wenn der Inhalt des Arbeitsregisters gleich dem des Referenzregisters r ist.

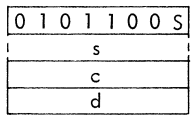
IER — Inkrementieren, Sprung wenn gleich Register




Befehl
Arbeitsregister
Register
Sprungweite

Funktion: Das Arbeitsregister wird inkrementiert. Das Programm verzweigt, wenn der Inhalt des Arbeitsregisters gleich dem des Referenzregisters r ist.

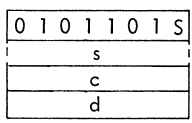
BNEC — Sprung wenn nicht gleich Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

Funktion: Das Programm verzweigt, wenn der Inhalt des Arbeitsregisters nicht gleich der Konstanten c ist.

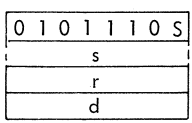
INEC — Inkrementieren, Sprung wenn nicht gleich Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

Funktion: Das Arbeitsregister wird inkrementiert. Das Programm verzweigt, wenn der Inhalt des Arbeitsregisters nicht gleich der Konstanten c ist.

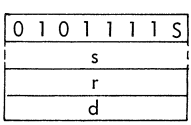
BNER — Sprung wenn nicht gleich Register




Befehl
Arbeitsregister
Register
Sprungweite

Funktion: Das Programm verzweigt, wenn der Inhalt des Arbeitsregisters nicht gleich dem des Referenz-Registers r ist.

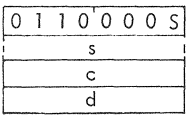
INER — Inkrementieren, Sprung wenn nicht gleich Register




Befehl
Arbeitsregister
Register
Sprungweite

Funktion: Das Arbeitsregister wird inkrementiert. Das Programm verzweigt, wenn der Inhalt des Arbeitsregisters nicht gleich dem des Referenz-Registers r ist.

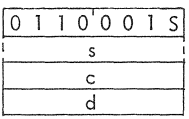
BZC — Sprung wenn alle Testbits Null maskiert mit Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

Funktion: Das Programm verzweigt, wenn alle durch die Konstante c vorgegebenen Testbits des Arbeitsregisters Null sind.

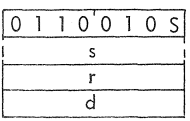
IZC — Inkrementieren, Sprung wenn alle Testbits Null maskiert mit Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

Funktion: Das Arbeitsregister wird inkrementiert. Das Programm verzweigt, wenn alle durch die Konstante c vorgegebenen Testbits des Arbeitsregisters Null sind.

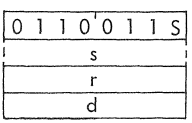
BZR — Sprung wenn alle Testbits Null maskiert mit Register




Befehl
Arbeitsregister
Register
Sprungweite

Funktion: Das Programm verzweigt, wenn alle durch das Referenzregister r vorgegebenen Testbits des Arbeitsregisters Null sind.

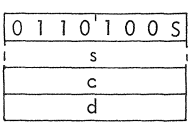
IZR — Inkrementieren, Sprung wenn alle Testbits Null maskiert mit Register




Befehl
Arbeitsregister
Register
Sprungweite

Funktion: Das Arbeitsregister wird inkrementiert. Das Programm verzweigt, wenn alle durch das Referenzregister r vorgegebenen Testbits des Arbeitsregisters Null sind.

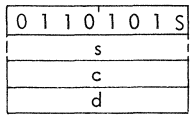
BNZC — Sprung wenn nicht alle Testbits Null maskiert mit Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

Funktion: Das Programm verzweigt, wenn nicht alle durch die Konstante c vorgegebenen Testbits des Arbeitsregisters Null sind.

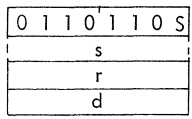
INZC — Inkrementieren, Sprung wenn nicht alle Testbits Null maskiert mit Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

Funktion: Das Arbeitsregister wird inkrementiert. Das Programm verzweigt, wenn nicht alle durch die Konstante c vorgegebenen Testbits des Arbeitsregisters Null sind.

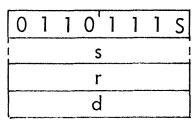
BNZR — Sprung wenn nicht alle Testbits Null maskiert mit Register




Befehl
Arbeitsregister
Register
Sprungweite

Funktion: Das Programm verzweigt, wenn nicht alle durch das Referenzregister r vorgegebenen Testbits des Arbeitsregisters Null sind.

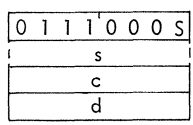
INZR — Inkrementieren, Sprung wenn nicht alle Testbits Null maskiert mit Register




Befehl
Arbeitsregister
Register
Sprungweite

Funktion: Das Arbeitsregister wird inkrementiert. Das Programm verzweigt, wenn nicht alle durch das Referenzregister r vorgegebenen Testbits des Arbeitsregisters Null sind.

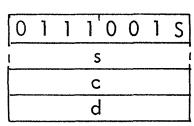
BOC — Sprung wenn alle Testbits Eins maskiert mit Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

Funktion: Das Programm verzweigt, wenn alle durch die Konstante c vorgegebenen Testbits des Arbeitsregisters Eins sind.

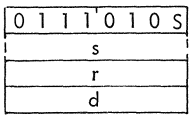
IOC — Inkrementieren, Sprung wenn alle Testbits Eins maskiert mit Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

Funktion: Das Arbeitsregister wird inkrementiert. Das Programm verzweigt, wenn alle durch die Konstante c vorgegebenen Testbits des Arbeitsregisters Eins sind.

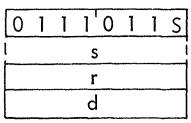
BOR — Sprung wenn alle Testbits Eins maskiert mit Register




Befehl
Arbeitsregister
Register
Sprungweite

Funktion: Das Programm verzweigt, wenn alle durch das Referenzregister r vorgegebenen Testbits des Arbeitsregisters Eins sind.

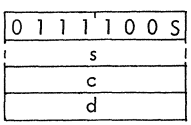
IOR — Inkrementieren, Sprung wenn alle Testbits Eins maskiert mit Register




Befehl
Arbeitsregister
Register
Sprungweite

Funktion: Das Arbeitsregister wird inkrementiert. Das Programm verzweigt, wenn alle durch das Referenzregister r vorgegebenen Testbits des Arbeitsregisters Eins sind.

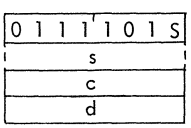
BNOC — Sprung wenn nicht alle Testbits Eins maskiert mit Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

Funktion: Das Programm verzweigt, wenn nicht alle durch die Konstante c vorgegebenen Testbits des Arbeitsregisters Eins sind.

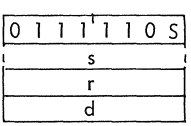
INOC — Inkrementieren, Sprung wenn nicht alle Testbits Eins maskiert mit Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

Funktion: Das Arbeitsregister wird inkrementiert. Das Programm verzweigt, wenn nicht alle durch die Konstante c vorgegebenen Testbits des Arbeitsregisters Eins sind.

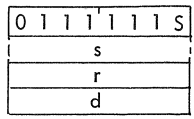
BNOR — Sprung wenn nicht alle Testbits Eins, maskiert mit Register




Befehl
Arbeitsregister
Register
Sprungweite

Funktion: Das Programm verzweigt, wenn nicht alle durch das Referenzregister r vorgegebenen Testbits des Arbeitsregisters Eins sind.

INOR — Inkrementieren, Sprung wenn nicht alle Testbits Eins maskiert mit Register




Befehl
Arbeitsregister
Register
Sprungweite

Funktion: Das Arbeitsregister wird inkrementiert. Das Programm verzweigt, wenn nicht alle durch das Referenzregister r vorgegebenen Testbits des Arbeitsregisters Eins sind.

Bei den bedingten Sprungbefehlen mit Testbit-Abfrage (BZC...INOR) werden die Bits des Arbeitsregisters überprüft, die in der "Maske" gleich 1 sind, wobei die Maske als Konstante im Befehl oder als Variable in einem Referenzregister enthalten ist. Bits mit Nullinhalt in der Maske spielen keine Rolle.

Beispiele:

| | | | |
|---|---|---|---|
| a) | Maske | 0 0 0 1 1 0 1 0 | |
| | Arbeitsregister | 1 0 1 0 0 0 0 1 | alle Testbits Ø |
| | | . . . | |
| b) | Maske | 0 0 0 1 1 0 1 0 | |
| | Arbeitsregister | 0 1 1 0 1 0 0 0 | nicht alle Testbits Ø |
| | | . . . | nicht alle Testbits 1 |
| c) | Maske | 0 0 0 1 1 0 1 0 | |
| | Arbeitsregister | 0 1 0 1 1 1 1 0 | alle Testbits 1 |
| | | . . . | |

Für die einzelnen Befehle bedeutet dies:

| | | | |
|---|---|---|---|
| BZ.. /IZ... : | Programm verzweigt bei | a) , | läuft weiter bei b)c) |
| BNZ../INZ..: | " " | b)c), | " " " a) |
| BO.. /IO.. : | " " | c) , | " " " a)b) |
| BNO../INO..: | " " | a)b), | " " " c) |

## BUS-BEZOGENE BEFEHLE

| 1 . . . . XS |
|---|

| | | | |
|---|---|---|---|
| Befehle: | LD | Laden | 1 0 0 0 . . . . |
| | AD | Addieren | 1 0 0 1 . . . . |
| | SB | Subtrahieren | 1 0 1 0 . . . . |
| | AN | UND | 1 0 1 1 . . . . |
| | OR | Inklusives ODER | 1 1 0 0 . . . . |
| | EO | Exklusives ODER | 1 1 0 1 . . . . |
| | ST | Speichern | 1 1 1 0 . . . . |
| | JP | Sprung | 1 1 1 1 . . . 0 |
| | CS | Unterprogramm-Sprung | 1 1 1 1 . . . 1 |
| Adressierung: | | | |
| | ..C | Konstante (unmittelbar) | . . . . 0 0 0 . |
| | ..X | indirekt (über Register x) | . . . . 0 0 1 . |
| | ..R | Register | . . . . 0 1 . . |
| | ..L | relativ | . . . . 1 0 . . |
| | ..A | absolut | . . . . 1 1 . . |
| Indizierung: | | nicht indiziert | . . . . . 0 . |
| | | indiziert über Indexregister x | . . . . . 1 . |
| Arbeits-register: | | Akkumulator @ | . . . . . 0 |
| | | Spezifiziertes Register s | . . . . . 1 |

Diese Befehlsgruppe setzt das Arbeitsregister mit einer BUS-Adresse (effektive Adresse) in Beziehung, d.h. mit einem Byte des MOS-RAM-Pools, des Kernspeichers oder Festspeichers, oder mit der Peripherie des Rechners. Hierzu gehören außerdem Sprung und Unterprogramm-Sprung auf eine beliebige Adresse des Kern- oder Festspeichers sowie des MOS-RAMs.

Adressierungsmöglichkeiten sind:

| | | |
|---|---|---|
| ..C | Konstante: | Der Operand steht als Konstante in der Instruktion (nicht sinnvoll bei JP und CS) |
| ..X | indirekt: | Die effektive Adresse steht in einem Indexregister X |
| ..R | Register: | Die effektive Adresse ist ein Register r |
| .. L | relativ: | Die effektive Adresse ist um die Differenz d entfernt von dem Byte, in dem d steht (maximal 128 byte zurück bzw. 127 byte vorwärts) |
| .. A | absolut: | Die effektive Adresse ist in Form von 2 Bytes (16 bit) in der Instruktion angegeben. |

Indizierung ist bei Register-, relativer und absoluter Adressierung möglich. In diesem Falle wird der Inhalt des Indexregisters x zur effektiven Adresse addiert. Dabei ist folgendes zu beachten:

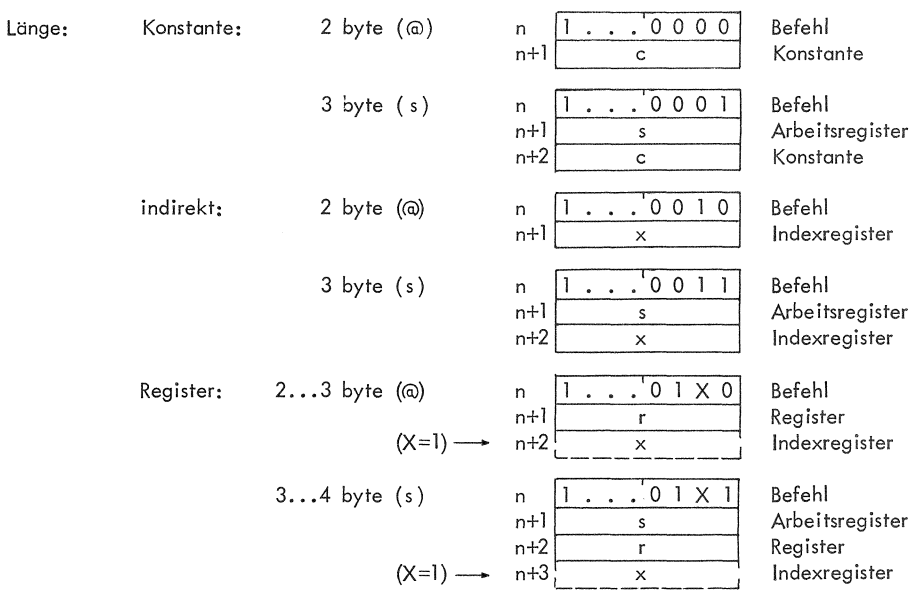
Ist für das Indexregister eine gerade Adresse x angegeben, so wird das Doppelbyte x (niedrige Stellen) und x+1 (hohe Stellen) als Index verwendet. Der Index ist eine 16-bit-Zweierkomplement-Zahl; daher ist positive und negative Indizierung möglich (-32768...65535).

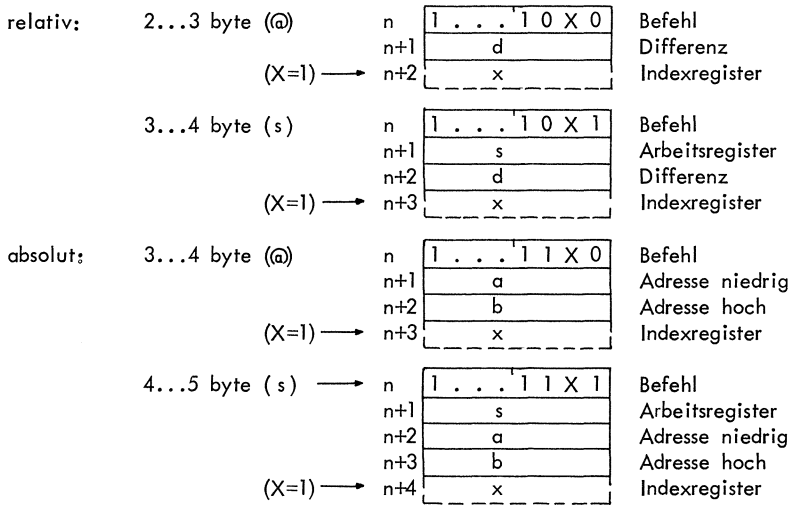
Ist dagegen eine ungerade Adresse x angegeben, so wird nur das Byte x als Index verwendet. Der Index ist eine 8-bit-Zahl, mit der nur positive Indizierung möglich ist (0...127).

Für die indirekte Adressierung bedeutet das, daß im ersten Falle eine volle 16-bit-Adresse in x, x+1 enthalten ist, während im zweiten Falle nur die 8 bit in x wirksam sind, d.h. hiermit können nur die absoluten Adressen ØØØØ...ØØFF angesprochen werden.

Zu beachten ist, daß angegebene Register und Indexregister ebenen-gebunden sind, d.h. ihre absolute Adressen sind um die jeweilige Ebenen-Adresse gegenüber der angegebenen Adresse verschoben.

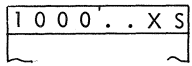
DO-Befehle: Die Anwendung von DO-Instruktionen auf die Befehle LD, AD, SB, AN, OR, EO und ST ist sinnvoll; die häufigsten Anwendungen sind später im einzelnen angegeben.






Bei einem vorgeschalteten DO-Befehl ist statt einer Konstanten c ein Konstanten-String von z Bytes in der Instruktion enthalten, vom Basis-Byte aus in Richtung aufsteigender Adressen.

LD Laden

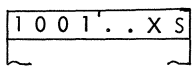



Funktion: Das Arbeitsregister wird mit dem Inhalt der effektiven Adresse (Operand) geladen.

z<& LD.. Ein vorgeschalteter DO-Befehl z<& bewirkt Laden der z Bytes eines Arbeitsregister-Strings mit stets demselben Operanden-Byte.

z=& LD.. Ein vorgeschalteter DO-Befehl z=& bewirkt Laden eines Arbeitsregister-Strings von z Byte Länge mit einem Operanden-String derselben Länge.

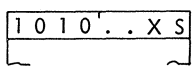
AD.. Addieren




Funktion: Zum Arbeitsregister-Inhalt wird der Inhalt der effektiven Adresse (Operand) addiert.

z=*AD.. Ein vorgeschalteter DO-Befehl z=* bewirkt Addieren eines Operanden-Strings von z Byte Länge zu einem Arbeitsregister derselben Länge.

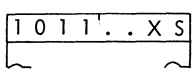
SB.. Subtrahieren




Funktion: Vom Arbeitsregister-Inhalt wird der Inhalt der effektiven Adresse (Operand) subtrahiert.

z=* SB.. Ein vorgeschalteter DO-Befehl z=* bewirkt Subtrahieren eines Operanden-Strings von z Byte Länge von einem Arbeitsregister der gleichen Länge.

AN.. UND

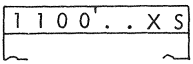



Funktion: Der Arbeitsregister-Inhalt wird mit dem Inhalt der effektiven Adresse (Operand) in UND-Verknüpfung gebracht; das Ergebnis steht im Arbeitsregister.

z<& AN.. Ein vorgeschalteter DO-Befehl z<& bewirkt, daß die z Bytes eines Arbeitsregister-Strings mit stets demselben Operanden-Byte in UND-Verknüpfung gebracht werden.

z=& AN.. Ein vorgeschalteter DO-Befehl z=& bewirkt UND-Verknüpfung zwischen einem Arbeitsregister- und einem Operanden-String von jeweils z Byte Länge.

OR..  Inklusives ODER

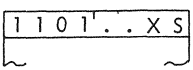



Funktion: Der Arbeitsregister-Inhalt wird mit dem Inhalt der effektiven Adresse (Operand) in inklusive ODER-Verknüpfung gebracht; das Ergebnis steht im Arbeitsregister.

z<& OR.. Ein vorgeschalteter DO-Befehl z<& bewirkt, daß die z Bytes eines Arbeitsregister-Strings mit stets demselben Operanden-Byte in inklusive ODER-Verknüpfung gebracht werden.

z=& OR.. Ein vorgeschalteter DO-Befehl Z=& bewirkt inklusive ODER-Verknüpfung zwischen einem Arbeitsregister- und einem Operanden-String von jeweils z Byte Länge.

EO..  Exklusives ODER

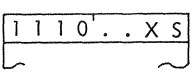



Funktion: Der Arbeitsregister-Inhalt wird mit dem Inhalt der effektiven Adresse (Operand) in exklusive ODER-Verknüpfung gebracht; das Ergebnis steht im Arbeitsregister.

z<& EO.. Ein vorgeschalteter DO-Befehl z<& bewirkt, daß die z Bytes eines Arbeitsregister-Strings mit stets demselben Operanden-Byte in exklusive ODER-Verknüpfung gebracht werden.

z=& EO.. Ein vorgeschalteter DO-Befehl z=& bewirkt exklusive ODER-Verknüpfung zwischen einem Arbeitsregister- und einem Operanden-String von jeweils z Byte Länge.

ST..  Speichern

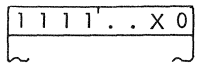



Funktion: Der Inhalt des Arbeitsregisters wird in der effektiven Adresse abgespeichert. Bei Konstanten-Adressierung (STC) wird der Inhalt innerhalb der Instruktion abgespeichert (an die Stelle eines Konstanten-Bytes).

z>& ST.. Ein vorgeschalteter DO-Befehl z>& bewirkt Speichern des immer gleichen Arbeitsregister-Inhalts in z aufeinanderfolgenden Bytes eines Adreß-Strings.

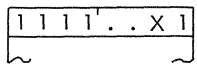
z=& ST.. Ein vorgeschalteter DO-Befehl z=& bewirkt Speichern eines Arbeitsregister-Inhalts von z Byte Länge in einem gleich langen Adreß-String.

JP..  Sprung




Funktion: Das Programm verzweigt zur angegebenen Adresse, indem der Instruktionszähler auf deren Wert gesetzt wird.

CS..  Unterprogramm-Sprung




Funktion: Der Inhalt des Instruktionszählers, bezogen auf das letzte Byte der Instruktion und um 1 erhöht, wird im Arbeitsregister s gespeichert (Rückkehradresse = Adresse des Befehlsbytes der auf CS folgenden Instruktion). Dann verzweigt das Programm zur angegebenen Adresse.

Die Rückkehr aus dem Unterprogramm (zur auf CS folgenden Instruktion) wird an dessen Ende durch einen indirekten Sprungbefehl über das Rückkehr-Adreßregister s (JPX, s) programmiert.

# ASSEMBLER

## VORBEMERKUNG

Der Assembler MINCASS 600 ist ein Programm zur Übersetzung von symbolischen Programmen in die Maschinensprache des Computers MINCAL 621. Er steht dem Benutzer in Form eines Lochstreifens zur Verfügung; nach Einlesen des Lochstreifens in den Kernspeicher (ab Adresse 4ØØØ) und Betätigen der Taste ST (Start) an der Rechner-Konsole ist das System zur Programmumwandlung bereit.

Es sind zwei Assembler-Versionen verfügbar:

Einfacher Assembler MINCASS 600
für Systeme mit 4k byte Kernspeicher und Teletype

Makro-Assembler MINCASS 600 M
für Systeme ab 8k byte Kernspeicher mit Teletype und schneller Lochstreifenausrüstung

Funktionen, die nur vom Makro-Assembler MINCASS 600 M ausgeführt werden, sind im folgenden mit +) gekennzeichnet.

## PROGRAMMAUFBAU

Ein symbolisches Programm besteht aus einer Folge von Anweisungen (Statements). Es gibt verschiedene Typen von Anweisungen:

Steueranweisungen
Wertzuweisungen
Belegungsanweisungen
Maschinen-Instruktionen
Makro-Instruktionen
Kommentare

Anweisungen werden in der Reihenfolge geschrieben, wie sie im Programm nacheinander benötigt werden; der Assembler übersetzt das Programm in gleicher Reihenfolge in Maschinensprache. Zur Niederschrift benutze man die MINCAL 600 Programm-Formblätter.

Jede Anweisung besteht aus einer Folge von Buchstaben, Ziffern und Symbolen, wobei - soweit nicht im einzelnen eingeschränkt - alle 64 druckbaren Zeichen des ASCII-(ISO-7-) Codes zulässig sind. Leerschritte (Space) werden im allgemeinen vom Assembler überlesen. Steuerzeichen, wie z.B. Wagenrücklauf (CR) und Zeilenwechsel (LF) werden ebenfalls überlesen. Anweisungen werden voneinander durch Semikolon (;) getrennt und vom Assembler fortlaufend numeriert (0000...9999).

Innerhalb der Anweisungen sind die Zeichen zu Worten zusammengefaßt, welche die notwendigen Einzelangaben darstellen. Die Worte werden durch Trennzeichen separiert. Der generelle Aufbau einer Anweisung ist wie folgt:

| LABEL | NUMBER | INSTR | SPEC | OPERAND | SUPPL |
|---|---|---|---|---|---|
| {Marke}: | {Anzahl} $\begin{matrix}*\\&\end{matrix}$ | {Befehl}, | {Spezifikation}, | {Operand}, | {Ergänzung}; |

Im Einzelfalle, insbesondere auch je Anweisungstyp, können bestimmte Worte entfallen; die Kommentar-Anweisung besteht nur aus Text, eingeleitet durch einen Schrägstrich (/). Die Worte einer Anweisung haben folgende Bedeutung:

Marke: Hier ist ein Name einzutragen, wenn an anderer Stelle im Programm auf die Anweisung Bezug genommen wird. Als Trennzeichen steht hinter der Marke ein Doppelpunkt (:).

Anzahl: Hier ist eine Dezimalzahl z (2...256) einzutragen, wenn Variablen- oder Konstanten-Strings von mehr als 1 Byte Länge vereinbart werden, oder wenn Mehrfachausführung (DO-Befehl) vorgesehen ist. Im letzteren Falle ist ggfs. auch eines der Zeichen > , < oder = hinter z notwendig. Statt der Zahl z kann ein Name stehen, dem vorher ein Wort zugewiesen wurde. Als Trennzeichen steht (*) oder (&) hinter der Anzahl.

# MINCAL 600 PROGRAM FORM

DIETZ Computer SYSTEME

PROGRAM POTENZEN VON 2 DATE 4-10-71 NAME FRANZ MEYER SHEET 1

```
/ LABEL : NUMBER * INSTR , SPEC , OPERAND , SUPPL  FOR COMMENTS                    NOTES
1 2   5  6 7   11 12 13  16 17 18  21 22 23  28 29 30  33 34  40  45  50  55  60  65  70
/PROGRAMM ZUR BERECHNUNG DER POTENZEN VON ZWEI
/MIT EXPONENTEN VON Ø BIS 15
/
/
          0     ,'4ØØØ
VAR  :    Q     ,'2Ø
EXP  :    Q     ,'21
CTR  :    Q     ,'22
TELE :    Q     ,'ØØ
/
ANF  :  2<*LDC   ,EXP  ,Ø
          LDC   ,VAR  ,1
A1   :    INEC  ,CTR  ,8     ,A2
/WAGENRUECKLAUF
        2 *WTC   ,TELE ,'ØAØD
          LDC   ,CTR  ,Ø
/LINKSSCHIEBEN
A2   :  2<*SLO   ,VAR
/AUSGABE
        6 *WDRD  ,TELE ,VAR
/ENDE   ?
          INEC  ,EXP  ,16    ,A1
          HLT
          JP    ,     ,ANF
/
/
          Z
```

Befehl: Jede Anweisung muß einen "Befehl" enthalten, der sie kennzeichnet. Er besteht für Steuer- und Belegungsanweisungen sowie für Wertzuweisungen aus einem Buchstaben, für Maschinen- und Makro-Instruktionen aus einem Buchstaben, gefolgt von 1 bis 3 weiteren Buchstaben oder Ziffern. Es sind nur solche Befehle zulässig, die in der Befehlsliste des Assemblers vermerkt oder vom Benutzer als Makrobefehle definiert sind (s. später). Als Trennzeichen steht dahinter ein Komma (,).

Spezifikation: Enthält notwendige Zusatzangaben. Bei Maschinen-Instruktionen sind dies die gestarteten Ebenen oder das Arbeitsregister; im letzteren Falle können Namen, Hexa-Zahlen oder das Akkumulator-Symbol @ verwendet werden. Als Trennzeichen steht dahinter ein Komma (,).

Operand: Enthält bei Maschinen-Instruktionen die Operanden-Adresse oder eine Konstante. Je nach Bedarf können Namen, Dezimalzahlen, Hexa-Zahlen, Text-Zeichen oder das Symbol @ verwendet werden. Als Trennzeichen steht dahinter ein Komma (,).

Ergänzung: Enthält bei bedingten Sprungbefehlen die Sprungadresse (Name), bei BUS-bezogenen Befehlen das Indexregister (Name, Hexa-Zahl, @ ).

Den Abschluß einer Anweisung bildet ein Semikolon (;). Es kann unmittelbar auf das letzte Wort folgen. Wenn Marke oder Anzahl nicht vorgesehen ist, entfallen die zugehörigen Trennzeichen; für hinter dem Befehl stehende Worte, die "leer" bleiben, müssen Kommata als Trennzeichen vorgesehen werden, wenn danach noch ein "ausgefülltes" Wort folgt.

## WORTELEMENTE

Worte innerhalb von Anweisungen können aus folgenden Elementen bestehen:

Namen: Namen sind symbolische Ersatzbezeichnungen für Adressen, Festwerte oder andere Angaben. Sie bestehen aus einem Buchstaben, dem von bis zu 3 weitere Buchstaben oder Ziffern folgen können.
Beispiele: A, AB, ABC, ABCD, X1, X999, H1T, OØ3P.
Jedem Namen muß im Programm ein bestimmter Wert zugewiesen werden. Das geschieht durch Eintragen des betreffenden Namens als Marke in einer Anweisung, wodurch ihm die (Basis-) Adresse der betreffenden Instruktion oder Belegung zugeteilt oder - im Falle der Wertzuweisung Q - ein bestimmter Wert zugewiesen wird. Ein Name darf in jedem Programm nur einmal definiert sein.

Dezimalzahlen: Dezimalzahlen bestehen aus 1 bis 5 Ziffern, vor denen ein Minuszeichen stehen kann. Der Assembler erzeugt aus ihnen die entsprechende binäre Ganzzahl bzw. deren Zweierkomplement.
Beispiele: 1, 99, 255, 32767, -1, -128, -32768.

Hexa-Zahlen: Hexa-Dezimalzahlen bestehen aus 2, 4 oder mehr Zeichen (Ziffern 0...9, Buchstaben A...F), vor denen ein Apostroph (') steht. Je 2 Hexa-Zeichen faßt der Assembler zu einem Byte zusammen.
Beispiele: 'ØØ, 'F3, '1A77, '12ØØFF.

Text-Zeichen: Textzeichen oder -Strings bestehen aus einem oder mehreren druckbaren ASCII-Zeichen; davor muß ein Anführungszeichen (") stehen. Der Assembler reserviert ein Byte je Zeichen. Leerschritte werden in diesem Falle nicht überlesen, sondern als Byte berücksichtigt.
Beispiele: "1, "TEXT, "+12␣/␣ABC.

Akkumulator: Für den Akkumulator ist statt der Register-Adresse 'Ø2 das Zeichen @ zu verwenden.

Steueranweisungen

Stehen am Anfang und Ende eines Programms. Sie belegen keinen Speicherplatz.

O Ursprung Programm

Definiert die Adresse der nächstfolgenden speicherbelegenden Anweisung. Zu Beginn jedes Programms muß eine O-Anweisung stehen.

Spezifikation: 4-stellige Hexa-Zahl

Beispiel: O, '4F12

Z Ende Programm

Schließt das symbolische Programm ab.

Wertzuweisung

Bewirkt Zuweisung eines Wortes zu einem Namen. Belegt keinen Speicherplatz. Muß im Programm (beliebig weit) vor der Stelle stehen, an welcher der Name benutzt wird.

Q Wertzuweisung

Weist einem Namen, der als Marke vor Q steht, den danach spezifizierten Wert zu.

Marke: Vorgeschrieben (Name)

Anzahl: Angabe notwendig, wenn der Wert die Kapazität eines Bytes überschreitet. Es wird die Anzahl der benötigten Bytes angegeben (2...256).

Spezifikation: Dezimalzahl,
Hexa-Zahl,
Text,
Name,
Name + Dezimalzahl, oder
Name + Hexa-Zahl,
Name + Name

Beispiele:

```
A    :     Q,  12
ZH15 : 2*  Q,  -32768
REG3 :     Q,  'F3
ADR  : 2*  Q,  'ØFA6
TX1  :     Q,  "/
TX2  : 12* Q,  "ALPHABET␣!!!
NAM1:      Q,  NAM2
XY1  :     Q,  XY1+1
XYD1 : 2*  Q,  XYD+999
A1B  :     Q,  AB+'ØF
SUM  :     Q,  X1+X2
```

Belegungsanweisungen

Belegen Speicherplätze mit Nullinhalt oder Festwerten. Die Anweisungen können mit Namen als Marken versehen werden. Der Name bezieht sich dann auf die Speicheradresse bzw. auf die Basis-Adresse des Byte-Strings.
+) zur Definition von Gleitkommazahlen siehe Abschnitt STANDARD PACKS.

V Variable

Reserviert ein Byte bzw. einen Byte-String im Speicher. Nach dem Assemblieren haben mit V reservierte Bytes Nullinhalt.

Anzahl: Angabe notwendig, wenn mehr als ein Byte reserviert werden soll. Es wird die Länge des Byte-Strings angegeben (2...256).

Beispiele:

```
    V
2*  V
256* V
```

D Dezimalzahl

Reserviert ein oder zwei Bytes im Speicher und belegt sie mit der Binärzahl, die der angegebenen Dezimalzahl entspricht.

Anzahl: Angabe notwendig, wenn 2 Bytes belegt werden.

Spezifikation: positive oder negative dezimale Ganzzahl

Beispiele:

```
    D,  1
    D,  255
    D,  -35
2*  D,  9999
2*  D,  -32768
2*  D,  65535
```

A Adresse

Reserviert ein oder zwei Bytes im Speicher und belegt sie mit einer Adresse. Anzahl: Angabe, wenn 2 Bytes belegt werden. Spezifikation: Name

Beispiele:

```
2* A,ADDR
   A,REG7
```

H Hexa-Zahl

Reserviert ein Byte bzw. einen Byte-String im Speicher und belegt sie mit 2 Hexa-Zahlen je Byte.

Anzahl: Angabe notwendig, wenn mehr als ein Byte reserviert wird. Es wird die Länge des Byte-Strings angegeben (2...256).

Spezifikation: 2- ...512-stellige Hexa-Zahl.

Beispiele:

```
   H,   'FF
2* H,   'Ø2B3
4* H,   '778899AA
```

T Text

Reserviert ein Byte bzw. einen Byte-String im Speicher und belegt sie mit einem druckbaren ASCII-Zeichen je Byte.

Anzahl: Angabe notwendig, wenn mehr als ein Byte reserviert wird. Es wird die Länge des Byte-Strings angegeben, die gleich der Zeichenzahl ist (2...256).

Spezifikation: 1 bis 256 druckbare Zeichen (einschließlich Leerschritt).

Beispiele:

```
   T,   "Z
9* T,   "+12␣/␣ ABC
```

Maschinen-Instruktionen

Die hierzu gehörenden Anweisungen beziehen sich auf die Maschinenbefehle des MINCAL 621 (siehe dort). Der Maschinencode wird vom Assembler entsprechend der Befehlsstruktur und in der Reihenfolge der Anweisungen erzeugt. Zur Mehrfach-Ausführung einer Instruktion (DO-Befehl) ist in der Anweisung die Anzahl z (2...256) einzutragen, gefolgt von der Angabe, welche Adresse inkrementiert wird, sowie von einem Trennzeichen, das zugleich die Berücksichtigung des LINK-Bits angibt:

| | | | |
|---|---|---|---|
| z & | n-fache Ausführung | | |
| z>& | " | , Operanden-Adresse wird inkrementiert | LINK wird nicht berücksichtigt |
| z<& | " | , Register-Adresse " | |
| z =& | " | , beide Adressen " | |
| z * | " | | |
| z< * | " | , Operanden-Adresse " | LINK wird berücksichtigt |
| z> * | " | , Register-Adresse " | |
| z=* | " | , beide Adressen " | |

Zu Beginn der Anweisung kann als Marke ein Name stehen. Ihm wird die Adresse des Befehls-Bytes zugewiesen, bei Mehrfachausführung die Adresse des Befehlsbytes des davorstehenden DO-Befehls.

Eine vollständige Anweisung sieht z.B. so aus:

```
MARK:2=*ADA,REG1,ADR,IXRG
```

Folgende symbolische Befehle sind vorgesehen, geordnet nach Gruppen:

NOP Ebenso: HLT, ECL, DCL
Steuerbefehle
Keine weiteren Angaben.

SEL Ebenso: HSL
Steuerbefehle mit Start einer Programmebene.

Spezifikation: Nummer der gestarteten Programmebene.

Beispiele:
```
SEL,'ØB
HSL,LEV3
```

GS Ebenso: GL
Abfrage Konsolschalter bzw. laufende Programmebene.

Spezifikation: Arbeitsregister.

Beispiele:
```
GS ,@
GL ,1A
GL ,REG7
```

SRO Ebenso: SRC, SLO, SLC
Schiebebefehle.

Spezifikation: Arbeitsregister.

Beispiele:
```
SRO,@
SRC,'1A
SLO, REG7
```

BZ Ebenso: IZ, BP, IB, BNZ, INZ, BNP, INP
Bedingter Sprung mit Abfrage Register-Inhalt auf Null oder Vorzeichen.

Spezifikation: Arbeitsregister.

Operand: Nicht zulässig (jedoch Komma vorsehen).

Ergänzung: Sprungadresse (Name).

Beispiele:

```
BZ , @  , , SPRG
IZ ,'1A  , , AD6
BNP, REG7, , X2
```

BEC Ebenso: IEC, BER, IER, BNEC, INEC, BNER, INER,
BZC, IZC, BZR, IZR, BNZC, INZC, BNZR, INZR
BOC, IOC, BOR, IOR, BNOC, INOC, BNOR, INOR
Bedingter Sprung mit Vergleich zwischen Arbeitsregister-Inhalt einerseits und Konstante oder Vergleichsregister-Inhalt andererseits.

Spezifikation: Arbeitsregister.

Operand: Konstante oder Vergleichsregister.

Ergänzung: Sprungadresse (Name).

Beispiele:

```
BEC  , @  ,12  ,SPRG
INEC ,'1A ,'FF ,AD6
BZC  ,REG7,MA3 ,X2
IER  , @  ,RG17,N1A
BNOR,1A  ,'A3 ,AD7
INOR ,REG7,, @  ,L
```

LD... Ebenso: AD..., SB..., AN..., OR..., EO..., ST...
BUS-bezogene Befehle (außer JP und CS).
Als dritter Buchstabe des Befehls ist je nach Adressierungsart C, X, R, L oder A anzugeben.

Spezifikation: Arbeitsregister.

Operand: Konstante oder Operanden-Adresse (außer bei ...X).

Ergänzung: Indexregister (wenn indiziert).

Beispiele:

```
LDC  , @  ,25    ,
ADX  ,'1A ,      ,IND2
SBR  ,REG7,'FF   ,
ANL  , @  ,ADR   ,
ORA  ,'1A ,'40A2 ,'1F
```

JP... Sprung
Dritter Buchstabe siehe LD...

Spezifikation: nicht zulässig (jedoch Komma vorsehen)

Operand: Sprungadresse (außer bei ...X)

Ergänzung: Indexregister (wenn indiziert)

Beispiele:
```
JPX  ,      ,      ,IND2
JPL  ,      ,SPRG
JRA  ,      ,'4ØA2,'1F
```

CS... Unterprogramm-Sprung
Dritter Buchstabe siehe LD...

Spezifikation: Arbeitsregister (Rückkehradresse)

Operand: Sprungadresse (außer bei X)

Ergänzung: Indexregister (wenn indiziert)

Beispiele:
```
CSX  , @    ,      ,IND2
CSL  ,'1A   ,SPRG
CSA  ,RET3  ,'4ØA2,'1F
```

Bemerkung: Bei Konstanten-Befehlen der Gruppe BEC und LD... können Konstanten von bis zu 2 Byte Länge dezimal oder hexa-dezimal als Operanden eingetragen werden, z.B.:

```
2=&INEC,@,'ØØFF,SPRG
2=*ADC ,'1A,4Ø96
```

Für längere Konstanten sind Namen vorzusehen, denen über eine Q-Anweisung die entsprechenden Werte zugewiesen werden.

+) Makro-Instruktionen

Symbolische Makro-Anweisungen dienen zur Programmierung von komplexen Befehlen, die nicht durch einfache Maschinen-Instruktionen des MINCAL 621 darstellbar sind. Die Makro-Anweisung ruft ein Unterprogramm auf, welches diesen komplexen Befehl ausführt; danach wird die folgende Anweisung ausgeführt.

Der Benutzer kann Makro-Anweisungen auf zweierlei Art gebrauchen:

Standard-Makros zu den MINCAL 600-Bibliotheksprogrammen (LIBRARY)

Selbstdefinierte Makros zu vom Benutzer erstellten Unterprogrammen.

Die Standard-Makros sind in der Befehlsliste des MINCASS 600 M Makro-Assemblers vermerkt und im Abschnitt "Programm-Bibliothek" ausführlich beschrieben.

Will der Benutzer eigene Makro-Anweisungen definieren, so ist folgendes zu beachten:

Die erforderlichen Unterprogramme sind entweder getrennt zu erstellen und zu testen; sie sind dann später in vorbestimmte Speicherbereiche einzulesen. Oder sie sind an irgendeiner Stelle im Programm symbolisch programmiert.

Zu Beginn eines Programms, das davon Gebrauch macht, sind die Makro-Anweisungen zu definieren, und zwar noch vor der O-Anweisung. Dies wird eingeleitet durch eine M-Anweisung (Buchstabe M ohne weitere Angaben). Dann folgt eine Liste der im Programm benutzten Makrobefehle, zusammen mit der absoluten Einsprungadresse in das Unterprogramm.

Beispiel:

```
M
XY  ,PACK
AB3Z,'4FØØ
HJK ,A2Ø5
```

Dies bedeutet, daß ein Makrobefehl (AB3Z) vorkommt, der einen Unterprogrammsprung zur festen Adresse '4FØØ bewirkt, während zwei weitere Makrobefehle (XY und HJK) Unterprogramme aufrufen, die im symbolischen Programm definiert sind, wobei PACK und A2Ø5 die Namen für die Einsprungstellen sind.

Als Makrobefehle können beliebige Folgen von 2, 3 oder 4 Zeichen verwendet werden, allerdings mit Einschränkungen:

Das erste Zeichen muß ein Buchstabe sein; R, K, W sind nicht zulässig.

Die Zeichenfolge darf keiner Anweisung für Maschinen-Instruktionen und keiner Makro-Anweisung für die Bibliotheksprogramme entsprechen.

Als erste 2 Zeichen sind LD, ST, AD, SB, MP und DV verboten.

Als drittes Zeichen sind C, X R, L und A, als viertes Zeichen D, F und G bei der Definition verboten; im Programm können sie vorkommen und bewirken dann Holen eines Operanden vor Eintritt in das Unterprogramm (s. später).

Wird z.B. im Programm der Makrobefehl HJK verwendet ohne weitere Angaben, so wird ein Unterprogrammsprung mit der symbolischen Entsprechung

```
CSA,RET,A2Ø5
```

erzeugt, was einem Unterprogrammsprung nach A2Ø5 ohne Übergabe eines Operanden entspricht; die Rückkehradresse steht in RET.

Stattdessen kann eine Makro-Anweisung mit Übergabe eines Operanden programmiert werden, dessen Adressierungsart und Länge durch Ergänzen zweier Buchstaben zum definierten Makrobefehl (der dann 2 Zeichen haben muß) bestimmt wird; hinzu kommen Angaben über Operandenadresse und eventuelle Indizierung. So erzeugt die Makro-Anweisung

```
XYm ,      ADR,IXR
```

eine Befehlsfolge mit der symbolischen Entsprechung

```
LDm  ,OPD  ,ADR,IXR
CSA  ,RET  ,PACK
```

wobei in m je nach Adressierungsart der Buchstabe C, X, R, L oder A steht. Ist ein viertes Zeichen I vorhanden,

```
XYml ,     ADR,IXR
```

so entsteht die Befehlsfolge

```
z=*LDm  ,OPD  ,ADR,IXR
   CSA  ,RET  ,PACK
```

wobei z = 2, 3 oder 4 ist, wenn für I der Buchstabe D, F oder G eingesetzt wird. Diese Form von Makrobefehlen erlaubt es, vor Eintritt in das Unterprogramm einen Operanden von 1...4 byte Länge zu übergeben, der beliebig adressierbar ist.

OPD und RET sind festgelegte Registerplätze bzw. -Strings, die auch von den Makrobefehlen der Bibliothek benutzt werden. Es ist daher nicht zweckmäßig, in vom Benutzer definierten Unterprogrammen Makrobefehle der Bibliothek zu benutzen, ohne den Inhalt dieser Register vorher zu retten.

## Kommentare

Kommentare dienen zur verbalen Erklärung des Programms. Sie können an beliebigen Stellen des Programms stehen und haben für das Programm selbst keine Bedeutung.

Eine Kommentar-Anweisung beginnt mit einem Schrägstrich (/), gefolgt vom Text aus beliebigen druckbaren Zeichen, wobei alle Leerschritte berücksichtigt werden. Der Kommentar wird mit Semikolon (;) beendet; es ist daher innerhalb des Kommentars nicht zulässig.

## +) KORREKTUREN

Beim Erstellen von symbolischen Programmen, z.B. off-line mit Hilfe des Teletype, können Fehler entstehen, die bereits beim Eintippen erkannt werden. Hierfür sind Korrekturmöglichkeiten vorgesehen:

Eingabe eines waagerechten Pfeils (←) macht das vorangehende Zeichen ungültig. Danach das richtige Zeichen eingeben.

Eingabe eines senkrechten Pfeils (↑) macht die gesamte Anweisung bis zum letzten Semikolon ungültig. Danach die Anweisung neu eingeben.

## HANDHABUNG DES ASSEMBLERS

Jede Programmumwandlung erfordert mindestens 2 Läufe des Assemblers:

ASS dient zum Aufbau der Markenliste sowie zur Erkennung von formalen Fehlern

EXC dient zur Erzeugung eines Lochstreifens, der das Programm in Maschinencode enthält.

Der Benutzer wählt den Lauf durch Eingabe der Bezeichnung ASS bzw. EXC über die Tastatur des Teletype vor; darauf ist einzugeben, worüber das Quellprogramm eingelesen wird und wohin das Resultat abgelegt wird:

| | | | | |
|---|---|---|---|---|
| ASS | Eingabe: | IKB | Tastatur des Teletype | symbolisches Programm |
| | | ISB | Langsamer Leser (Teletype) | |
| | | +) IFB | Schneller Leser | |
| | | +) ICB | Kernspeicher | |
| | Ausgabe: | OSH | Langsamer Locher (Teletype) | symbolisches Programm |
| | | +) OFH | Schneller Locher | |
| | | +) OCH | Kernspeicher | |
| EXC | Eingabe: | ISB | Langsamer Leser (Teletype) | symbolisches Programm |
| | | +) IFB | Schneller Leser | |
| | | +) ICB | Kernspeicher | |
| | Ausgabe: | OSH | Langsamer Locher (Teletype) | Maschinencode-Programm |
| | | +) OFH | Schneller Locher | |

Beispiele:
```
ASS  ,IKB,ISB
+) ASS  , ISB , OFB
+) ASS  , IFB , OCB
   EXC  , ISB , OSH
+) EXC,  IFB , OFH
+) EXC,  ICB , OFH
```

Nach Vorwahl der Betriebsart ist "Wagenrücklauf" einzugeben, und der Lauf beginnt. Das symbolische Programm, gleichgültig ob über die Tastatur eingegeben, als Lochstreifen eingelesen oder erzeugt oder im Kernspeicher abgelegt, hat das vom Assembler vorgeschriebene symbolische Format. Ausgelochte Maschinencode-Streifen haben Hexa-Format (s. Anhang).

Varianten dieser Betriebsarten sind solche, bei denen nur die Eingabe vorgeschrieben, die Ausgabe aber weggelassen wird, z.B.:

```
      ASS,ISB
      EXC,ISB
+)    EXC,ICB
```

Hierbei erfolgt keine Ausgabe; jedoch werden alle Anweisungen, die formale oder Adressierungsfehler enthalten, zusammen mit der Anweisungs-Nummer und einem Fehlercode auf dem Teletype ausgedruckt (siehe Fehlerliste).

Im Normalfall wird zu Beginn jedes ASS-Laufs die Markenliste gelöscht; jedoch hat der Benutzer die Möglichkeit, dies zu verhindern; er gibt dann ein:

```
ASS,SAV,...
```

Eine weitere Betriebsart bewirkt Drucken eines Protokolls auf dem Teletype. Hierzu ist PRO und die Quelle des symbolischen Programms einzugeben, z.B.:

```
PRO,ISB
PRO,IFB
PRO,ICB
```

Das gedruckte Protokoll hat je Zeile folgendes Format (1 Zeile = 1 Anweisung):

| | | |
|---|---|---|
| Anweisungs-Nummer | (4 Ziffern) | |
| Leerschritt | | |
| Fehlercode | (2 Ziffern oder 2 Leerschritte) | |
| (Basis-) Adresse | (4 Hexa-Ziffern) | |
| Leerschritt | | |
| Marke | (4 Zeichen) | } falls vorhanden |
| . | | |
| . | | |
| Anzahl | (4 Zeichen) | } falls vorhanden |
| Zusatzzeichen | ( >,< , = oder Leerschritt) | |
| Trennzeichen | (* oder &) | |
| Befehl | (4 Zeichen) | |
| , | | } oder längere Spezifikation |
| Spezifikation | (4 Zeichen) | |
| , | | |
| Operand | (6 Zeichen) | |
| , | | |
| Ergänzung | (4 Zeichen) | |
| Leerschritt | | |
| Maschinencode | (max. 8 Hexa-Ziffern-Paare, durch je 1 Leerschritt getrennt; die Paare entsprechen erzeugten Bytes in aufsteigender Adreßreihenfolge) | |

+) Spezifikationen mit mehr als 16 Zeichen und Maschinencode-Strings mit mehr als 8 byte werden in Folgezeiten spaltengerecht fortgesetzt.

Kommentare werden unter Weglassung des einleitenden Schrägstrichs mit Beginn der Markenspalte ausgedruckt.

+) Der Protokollausdruck erfolgt abschnittweise, so daß das Endlospapier in DIN A4-Blätter geschrieben werden kann. Am Kopf jedes Blattes wird dessen Nummer sowie der Programmname gedruckt.

Assembler-Läufe können auch zur Korrektur von symbolischen Programmen benutzt werden.

Hierzu wird vor dem betreffenden Lauf das Kommando

COR

eingegeben. Mit dieser Betriebsart wird ein Pufferbereich im Speicher angesprochen, der die notwendigen Korrekturen in Form symbolischer Anweisungen enthält. Weitere Kommandos sind in diesem Zusammenhang:

| | | | | |
|---|---|---|---|---|
| | L | Ausdruck | aller Anweisungen | im Pufferbereich |
| +) | m,L | " | Anweisung m | " |
| +) | n,m,L | " | Anweisungen m bis n | " |
| | C | Löschen | aller Anweisungen | im Pufferbereich |
| +) | m,C | " | Anweisung m | " |
| +) | m,n,C | " | Anweisungen m bis n | " |
| +) | m,D | Löschen | Anweisung m | im Quellprogramm |
| +) | m,n,D | " | Anweisungen m bis n | " |
| | m,A | Ändern | Anweisung m | im Quellprogramm |
| | n,m,A | " | Anweisungen m bis n | " |
| +) | m,I | Einfügen einer Anweisung nach m | | im Quellprogramm |

danach neue Anweisungen eingeben

\# Ende der Betriebsart

m, n sind jeweils vierstellige Anweisungs-Nummern.

Nach der Betriebsart C wird ein neuer Assembler-Lauf gewählt (ASS, evtl. auch EXC); die im Pufferbereich enthaltenen Korrekturen werden dabei berücksichtigt.

+) Zu Beginn des Assembler-Betriebs kann der Benutzer durch Eingeben 4-stelliger Hexa-Zahlen die maximale vom Assembler benutzte Speicheradresse (ADR), die Größe des Pufferbereichs (BUF) und die Länge der Markenliste (LAB) verändern den Programmnamen (NAM) mit max. 16 ASCII-Zeichen eingeben.

+) Nach einem EXC-Lauf kann mit LIB ein Streifen eingelesen und ein Duplikat erstellt werden, das zusätzlich die Bibliotheksprogramme enthält. Diese schließen sich damit an das eigentliche Programm an, wobei jedoch nur die PACKS angehängt werden, die von im Programm enthaltenen Makrobefehlen benutzt werden.

FEHLERLISTE

| Fehlercode | Bedeutung |
|---|---|
| ØØ | Allgemeiner Syntaxfehler |
| Ø1 | " " am Befehlsbeginn |
| Ø2 | Marke mehrfach definiert |
| Ø3 | Überlauf Markenliste |
| Ø4 | Anzahl zu groß |
| Ø5 | Syntaxfehler Anzahl |
| Ø6 | Name für Anzahl nicht definiert |
| Ø7 | Befehl nicht zulässig |
| Ø8 | Syntaxfehler im Operanden |
| 1Ø | Registeradresse größer als 1 Byte |
| 11 | " nicht definiert |
| 12 | Operandenadresse syntaktisch zu lang |
| 13 | " zu lang entsprechend DO-Befehl |
| 14 | " nicht definiert |
| 16 | Indexregister-Adresse größer als 1 Byte |
| 17 | " nicht definiert |
| 2Ø | Überlauf allgemein |
| 22 | Spezifikation fehlt |
| 23 | " hat falsche Länge |
| 24 | Marke vor Q-Anweisung fehlt |

# LIBRARY

## VORBEMERKUNG

Die Bibliothek (LIBRARY) des MINCAL 621 Computers besteht aus Unterprogrammen, die umfangreichere Funktionen erfüllen als einzelne Maschinenbefehle. Die Unterprogramme werden durch Makro-Anweisungen aufgerufen; hierfür ist der Makro-Assembler MINCASS 600 M zu benutzen. Jedoch können sie auch - für den Benutzer umständlicher - mit Unterprogrammaufrufen unter Übergabe des Operanden mit einfachen Assemblerbefehlen bedient werden.

Die Unterprogramme der Bibliothek sind entsprechend ihrer Funktion zu Paketen (PACKS) zusammengefaßt:

READ/WRITE PACK : (Ein/Ausgabe von Text und anderen Zeichen)
DOUBLE BYTE PACK : (Doppelbyte-Arithmetik mit Ein/Ausgabe)
FLOATING POINT PACK: (Gleitkomma-Arithmetik mit Ein/Ausgabe)

Das zweite Paket bedingt Vorhandensein des ersten; das letzte setzt beide anderen voraus.

Die Unterprogramme benutzen als Variablenspeicher die Register der jeweiligen Ebene; sie können daher im Multiprogramming von beliebig vielen Ebenen gleichzeitig benutzt werden. Je Ebene muß ein Pool von 32 Bytes (Register-Adressen ØØ...1Ø) zur Verfügung stehen; dieser Bereich, einschließlich dem Akkumulator @, kann durch die Unterprogramme verändert werden.

## READ/WRITE PACK

Dieses Unterprogramm-Paket dient zur Ein- und Ausgabe von Text und Hexa-Zahlen in Verbindung mit Fernschreibern, Lochstreifengeräten und anderen, zeichenweise arbeitenden Periphergeräten im ASCII-Code.

Der Aufbau der zugehörigen Makro-Anweisungen ist:

⟨Anzahl⟩ * ⟨Befehl⟩ , ⟨Gerät⟩ , ⟨Operand⟩ , ⟨Indexregister⟩

Folgende Befehle sind vorgesehen:

| | |
|---|---|
| RTm | Lesen Text |
| WTm | Schreiben Text |
| RHm | Lesen Hexa |
| WHm | Schreiben Hexa |

Lesen bedeutet Eingabe Periphergerät und Abspeichern in der effektiven Adresse, Schreiben den umgekehrten Vorgang. Für m ist einer der Buchstaben C, X, R, L oder A entsprechend der gewünschten Adressierungsart einzusetzen; die Operanden-Adresse wird wie üblich programmiert, ebenso das eventuelle Indexregister, mit dessen Inhalt sie modifiziert wird.

Bei Text (T) wird ein Byte unverändert als ASCII-Zeichen behandelt; im Falle von Hexa (H) werden die linke und rechte Hälfte (in dieser Reihenfolge) eines Bytes zwei ASCII-Zeichen (Ø...9, A...F) zugeordnet, indem die zusätzlichen 4 Bit des ASCII-Codes abgeschnitten bzw. ergänzt werden.

Die Gerätenummer wird als Spezifikation dem Befehl mitgegeben, wobei dort entweder eine zweistellige Hexa-Zahl oder ein Name steht, der entsprechend definiert ist. Wird z.B. 'F3 als Gerätenummer programmiert, so wird das Gerät mit der BUS-Adresse ØF3Ø angesprochen.

Vor dem Befehl kann die Anzahl der ein- oder ausgegebenen Zeichen bestimmt werden (2...256), wenn es sich um mehr als eins handelt. Bei Text-Befehlen entspricht dem ein gleich langer Operanden-String, wobei das Basis-Byte, welches auch das zeitlich zuerst behandelte Zeichen enthält, programmiert wird. Bei Hexa-Befehlen ist die Länge des Operanden-Strings halb so groß.

Text- und Hexa-Zeichen werden, als Konstanten verwendet, zweckmäßig mit den Definitionen T und H programmiert. Für die Ausgabe von ASCII-Steuerzeichen (z.B. Wagenrücklauf) ist Text-Ausgabe von Konstanten zweckmäßig, die als Hexa-Zahlen eingegeben werden.

Beispiele:

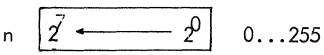
```
    RTA ,'F3 ,ADDR      Lesen 1 Textzeichen von 'F3 nach ADDR
6 * RHR ,'ØØ ,REG1      "     6 Hexa-Zeichen von ØØ nach REG1
    WTL ,DEV,CHAR       Schreiben 1 Textzeichen aus CHAR nach DEV
2 * WTC,FS2 ,'ØAØD      Ausgabe Wagenrücklauf/Zeilenvorschub auf FS2
    WTC,FS2 ,"X         Schreiben "X" auf FS2
```

Die Ausgabe der Zeichen erfolgt im ASCII-(ISO-7-) Code mit geradzahliger Parität. Bei der Eingabe wird auf diese Parität geprüft; fehlerhafte Zeichen werden zwar abgelegt, jedoch wird dann ein Register-Byte ERR auf 1 gesetzt; der Benutzer kann dieses Byte nach Ablauf des Makrobefehls im Hauptprogramm abfragen.

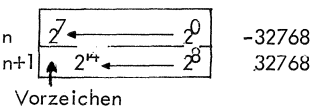
# DOUBLE BYTE PACK

Mit diesem Unterprogramm-Paket können Doppelbyte-Ganzzahlen airhtmetisch behandelt sowie ein- und ausgegeben werden, einschließlich der Umwandlungen von Binär- in Dezimalzahlen und umgekehrt. Hinzu kommen Konversions- und Ein/Ausgabebefehle für EinbyteGanzzahlen.

Einbyte-Ganzzahlen sind stets positiv:




Doppelbyte-Ganzzahlen sind positiv oder negativ (Zweierkomplement) und umfassen 16 bit (niedriges Byte = Basis-Byte):

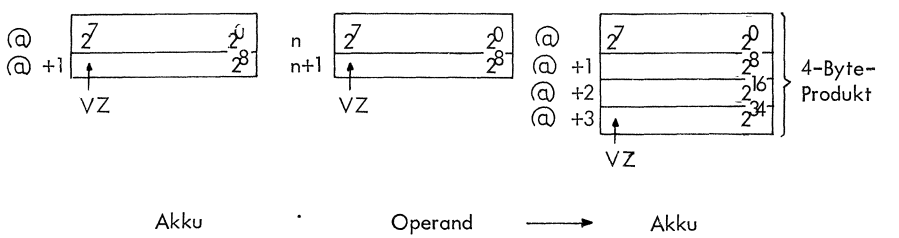



Folgende arithmetischen Befehle sind für Doppelbyte-Ganzzahlen vorgesehen:

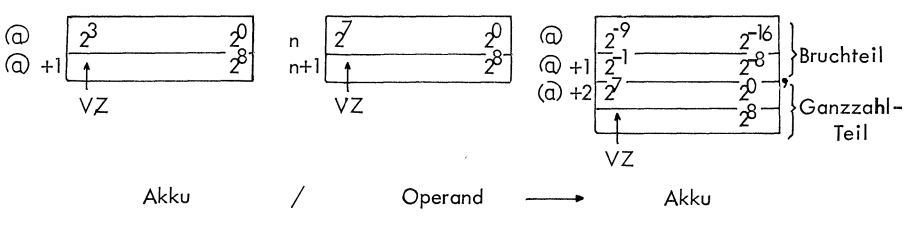
| | |
|---|---|
| MPmD | Multiplizieren Doppelbyte |
| DVmD | Dividieren " |

Als Arbeitsregister wird stets der Akkumulator benutzt. Der Operand kann wie bei BUS-bezogenen Befehlen üblich addiert werden; für m ist C, X R, L oder A einzusetzen.

Die Multiplikation ergibt ein 4-byte-Produkt:




Bei der Division ergibt sich ein 4-Byte-Quotient mit Mittelkomma:



Beide Operationen laufen vorzeichenrichtig ab.

Um dem Benutzer einen symbolisch vollständigen Satz von Doppelbyte-Befehlen an die Hand zu geben, sind im Makro-Assembler noch folgende Befehle vorgesehen:

| | | | |
|---|---|---|---|
| LD mD | Laden | Doppelbyte | (Operand → Akku) |
| ST mD | Speichern | " | (Akku → Adresse) |
| ADmD | Addieren | " | (Akku + Operand → Akku) |
| SB mD | Subtrahieren | " | (Akku - Operand → Akku) |

Es werden jedoch keine Unterprogramme hierfür benutzt; vielmehr erzeugt der Assembler hieraus Maschinenbefehle mit vorgeschaltetem DO-Befehl.

Die Angabe einer Anzahl ist nicht zulässig, so daß sich für diese Anweisungs-Gruppe folgender Aufbau ergibt:

⟨ Befehl ⟩ , @ , ⟨ Operand ⟩ , ⟨ Indexregister ⟩

Beispiele für Doppelbyte-Anweisungen:

| | |
|---|---|
| LDCD , @ , -25ØØØ | -25000 → Akku |
| STXD , @ , IND | Akku → < IND > |
| ADRD , @ , REG7 , IXR1 | Akku + Operand → Akku |
| SBLD , @ , CONS | Akku - Operand → Akku |
| MPAD , @ , ADDR , 'AB | Akku · Operand → Akku |
| DVRD , @ , 'A1 | Akku : Operand → Akku |

Zum Doppelbyte-Paket gehören ferner folgende Ein/Ausgabe- und Konversionsbefehle:

| | | |
|---|---|---|
| RD mD | Lesen Doppelbyte-Ganzzahl | |
| WD mD | Schreiben Doppelbyte-Ganzzahl | |
| RA mD | Konversion ASCII → Binär | Doppelbyte-Ganzzahl |
| WA mD | " Binär → ASCII | |
| RB mD | " BCD → Binär | |
| WB mD | " Binär → BCD | |

Die ersten beiden Befehle haben den Aufbau

⟨ Anzahl ⟩ * ⟨ Befehl ⟩ , ⟨ Gerät ⟩ , ⟨ Operand ⟩ , ⟨ Indexregister ⟩

und bewirken das Lesen eines ASCII-Zeichen-Strings mit Ganzzahl-Bedeutung, Umwandlung in eine binäre Doppelbyte-Zahl und Abspeichern in der angegebenen (sowie der nächsthöheren) Adresse; beziehungsweise beim Schreiben den umgekehrten Vorgang. Dabei ist das Peripheriegerät sowie die Zahl der ASCII-Zeichen anzugeben, die gelesen bzw. ausgegeben werden sollen:

```
- 1 2 3 4 5          Anzahl = 6
- 2                    "    = 2
␣ ␣ ␣ ␣ 3 5           "    = 6
␣ ␣ ␣ - 3 2 7 6 8     "    = 9
```

Beim Schreiben werden führende Nullen mit Leerschritten unterdrückt; für positives Vorzeichen steht ein Leerschritt. Gelesen wird höchstens die angegebene Stellenzahl; jede Nicht-Ziffer nach einer Ziffer führt jedoch schon zur Beendigung des Lesevorgangs.

Hinsichtlich Paritäts-Erzeugung und -Prüfung des ASCII-Codes gelten die Bemerkungen des vorigen Abschnitts.

Mit den restlichen 4 Befehlen, die den Aufbau

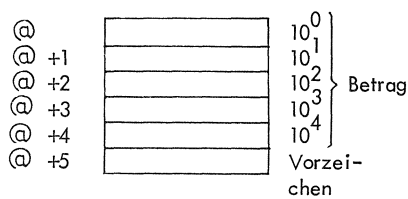
‹ Befehl › , @ , ‹ Operand › , Indexregister

haben, können Ganzzahlen, die als ASCII- oder BCD-Zeichen im Akkumulator und den nächsthöheren 5 Bytes stehen, in binäre Doppelbyte-Zahlen umgewandelt und in der effektiven Adresse abgelegt werden; ebenso ist der umgekehrte Vorgang möglich.

Lage der Zeichen im Akkumulator:

Inhalt: ASCII-Zeichen
bzw. BCD ('Ø...'9)

Vorzeichen: - oder Leerschritt (ASCII)
bzw. '5 oder 'Ø (BCD)




Beispiele für Doppelbyte-Ein/Ausgabe- und Konversionsbefehle:

```
6xRDAD ,DEV,ADDR        6-Zeichen-Zahl von DEV nach ADDR (binär)
9xWDCD,'F3 ,-32768      -32768 auf 'F3 9-stellig ausgeben

 RAXD , @ , , IND       @(ASCII) ⟶ <IND>(binär)
 WARD, @ , REG7         REG7 (binär) ⟶ @(ASCII)
 RBAD , @ , '2FØ8       @(BCD) ⟶ '2FØ8 (binär)
 WBLD , @ , VAR,IXR     Operand (binär) ⟶ @(BCD)
```

Bestandteil des Doppelbyte-Pakets sind schließlich noch Ein/Ausgabe- und Konversionsbefehle für Einbyte-Ganzzahlen:

| | | |
|---|---|---|
| RI m | Lesen Einbyte-Ganzzahl | |
| WI m | Schreiben Einbyte-Ganzzahl | |
| RA m | Konversion ASCII → Binär | Einbyte-Ganzzahl |
| WAm | " Binär → ASCII | |
| RB m | " BCD → Binär | |
| WBm | " Binär → BCD | |

Sie entsprechen völlig den Doppelbyte-Befehlen; jedoch werden nur positive Zahlen behandelt, die maximal 3 geltende Ziffern haben, nur 3 Akkumulator-Bytes (@ bis @+ 2) belegen und in binärer Form ein Byte einnehmen.

Beispiele hierfür:

| | |
|---|---|
| 2*RIA ,DEV,ADDR | 2-stellige Ganzzahl von DEV nach ADDR |
| 6*WIC,'F3,125 | ␣ ␣ ␣ 1 2 5 auf 'F3 ausgeben |

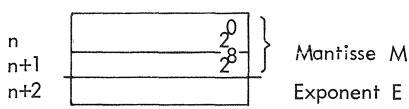
RAX , @ , , IND
WAR, @ ,REG7
RBA , @ , '2FØ8
WBL , @ , VAR, IXR

} siehe Doppelbyte-Beispiele

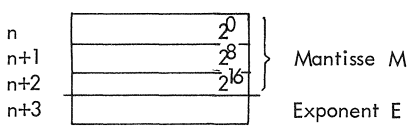
## FLOATING POINT PACK

Dieses Unterprogramm-Paket gibt dem Benutzer die Möglichkeit, mit Gleitkomma-Zahlen zu rechnen, sie zu konvertieren sowie ein- und auszugeben. Es gibt 2 interne Darstellungen von Gleitkomma-Zahlen mit unterschiedlicher Genauigkeit:

F-Typ (2-Byte-Mantisse):




G-Typ (3-Byte-Mantisse):



Die Mantissen sind Doppelbyte- bzw. 3-Byte-Ganzzahlen; sie können positiv oder negativ sein. Exponenten sind positive oder negative Einbyte-Ganzzahlen (Bereich -128 ... 127) zur Basis 2. Eine Gleitkommazahl hat daher den Wert

$$M \cdot 2^E \qquad (M = \text{Mantisse}; E = \text{Exponent})$$

Das niedrige Mantissen-Byte ist stets das Basis-Byte.

Als arithmetische Befehle sind vorgesehen:

| | |
|---|---|
| ADmF | Addieren Gleitkomma F-Typ |
| SB mF | Subtrahieren " " |
| MPmF | Multiplizieren " " |
| DVmF | Dividieren " " |
| | |
| ADmG | Addieren Gleitkomma G-Typ |
| SB mG | Subtrahieren " " |
| MPmG | Multiplizieren " " |
| DVmG | Dividieren " " |

Dazu gibt es noch 3- und 4-byte-Transportbefehle, die jedoch nicht als Unterprogramme existieren, sondern vom Makro-Assembler als Maschineninstruktionen mit vorgeschaltetem DO-Befehl erzeugt werden:

| | |
|---|---|
| LDmF | Laden Gleitkomma F-Typ |
| STmF | Speichern " " |
| | |
| LDmG | Laden Gleitkomma G-Typ |
| STmG | Speichern " " |

Der Aufbau der Anweisungen ist in allen Fällen:

⟨ Befehl ⟩ , @ , ⟨ Operand ⟩ , ⟨ Indexregister ⟩

und entspricht hinsichtlich der Adressierungsart den Doppelbyte-Befehlen. Als Arbeitsregister kann wiederum nur der Akkumulator @ angegeben werden; das niedrigste Byte der Mantisse belegt @ selbst; es folgen das bzw. die höheren Bytes und schließlich der Exponent in ( @ + 2) bzw. (@ + 3).

Einige Beispiele:

```
LDLF  ,@ ,CONS
STXG  ,@ ,,IND
ADRG  ,@ ,REG7
SBLF  ,@ ,VAR,IXR
MPAG,@ ,ADDR
DVRF  ,@ ,'A1
```

Zur Definition von Gleitkomma-Festwerten kennt der Makro-Assembler folgende Belegungs-Anweisungen:

| | |
|---|---|
| F | Belegt 3 Bytes mit einer Gleitkomma-Zahl vom F-Typ |
| G | Belegt 4 Bytes mit einer Gleitkomma-Zahl vom G-Typ |

Als Spezifikation steht dahinter entweder eine beliebige Dezimalzahl (F-Format) oder eine solche mit einer Zehnerpotenz (E-Format); zum Beispiel:

| | | |
|---|---|---|
| F ,-123.45 | | (F-Format) |
| F ,.31415E-Ø1 | (= $0.31415 \cdot 10^{-1}$) | (E-Format) |
| F ,2Ø. | | (F-Format) |
| G,2859.67Ø2 | | (F-Format) |
| G,-2.8596ØØ2EØ3 | | (E-Format) |

Folgende Ein/Ausgabe-Befehle sind vorgesehen:

| | | | | |
|---|---|---|---|---|
| RF mF | Lesen | Gleitkomma-Zahl | F-Typ | im F-Format |
| WFmF | Schreiben | " | " | " |
| RE mF | Lesen | " | " | im E-Format |
| WEmF | Schreiben | " | " | " |
| RF mG | Lesen | Gleitkomma-Zahl | G-Typ | im F-Format |
| WFmG | Schreiben | " | " | " |
| RE mG | Lesen | " | " | E-Format |
| WEmG | Schreiben | " | " | " |

Prinzipiell entsprechen sie den Befehlen für die Doppelbyte-Ein/Ausgabe; der Benutzer hat jedoch die Wahl zwischen zwei externen Darstellungen (F- und E-Format). Außerdem ist neben der Anzahl der insgesamt gelesenen oder geschriebenen Zeichen (w) noch die Zahl der Stellen hinter dem Dezimalpunkt (d) anzugeben, in diesem Falle mit Ø als Trennzeichen, also:

w · d Ø

Beispiele hierfür:

```
 7.2$RFAF ,DEV,ADDR      Lesen 7 Zeichen, 2 Dezimalstellen, F-Format
14.7$WERG,'F3 ,REG7,IXR  Schreiben 14 Zeichen, 7 Dezimalstellen, E-Format
```

Hinzu kommen als Konversionsbefehle:

| | | | | |
|---|---|---|---|---|
| RA mF | Konversion | ASCII | Binär | Gleitkommazahl F-Typ |
| WAmF | " | Binär | ASCII | |
| RB mF | " | BCD | Binär | |
| WB mF | " | Binär | ASCII | |
| RA mG | Konversion | ASCII | Binär | Gleitkommazahl G-Typ |
| WAmG | " | Binär | ASCII | |
| RB mG | " | BCD | Binär | |
| WB mG | " | Binär | BCD | |

Im Aufbau und Funktion entsprechen diese Befehle denen für Doppelbyte-Ganzzahlen. Die Lage der Stellen im Akkumulator-String ist wie folgt: Die wichtigste Mantissenstelle belegt @ , es folgen 4 Bytes (F) bzw. 7 Bytes (G) für die höheren Mantissenstellen, dann das Vorzeichen der Mantisse, zwei Bytes für die beiden Exponentenstellen und das Vorzeichen des Exponenten.

Beispiele für Gleitkomma-Konversionsbefehle:

```
RAAF , @ ,ADDR,IXR      @ (ASCII) → Adresse (binär, F-Typ)
WBXG, @ , ,IND          Operand (binär, G-Typ) → @ (BCD)
```

Bezüglich Paritätserzeugung und -Prüfung des ASCII-Codes siehe READ/WRITE PACK.

## LESEN OHNE INITIATE

Alle gerätebezogenen Lesebefehle R... der LIBRARY beziehen sich auf Eingaben, die ein INITIATE erfordern (s. HINWEISE FÜR DIE PROGRAMMIERUNG), also z.B. den Leser am Fernschreiber oder schneller Lochstreifenleser.

Wo dies nicht erforderlich ist, z.B. bei Eingaben über die Tastatur des Fernschreibers, ist im Makrobefehl der Buchstabe K anstelle von R zu verwenden, zum Beispiel:

```
    KTA  ,'F3 , ADDR
 6*KDAD,DEV,ADDR,IXR
7.2$ KFRF  ,DEV,REG7
```

# MONITOR

## VORBEMERKUNG

Der MONITOR ist ein Programm zum Austesten von Programmen, die im Kernspeicher des MINCAL 621 abgelegt sind. Das zu testende Programm läuft unter Steuerung des MONITORS ab, 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 Kernspeicher-Inhalts über Konsol-Fernschreiber und schnelle Lochstreifengeräte.

MONITOR setzt 32 oder mehr Register je Ebene voraus.

## TESTBEGINN

Zunächst wird das MONITOR-Programm in einen freien Kernspeicher-Bereich geladen, das N-Register der Ebene Ø über die Bedienungskonsole auf die Anfangsadresse dieses Bereichs gesetzt und der Rechner gestartet. Der MONITOR meldet sich durch Klingelzeichen zum Zeichen, daß eine Eingabe erwartet wird. (Das geschieht für alle folgenden Eingaben).

Der Bediener gibt am Konsol-Fernschreiber mit

LEV l (cr)

die Programmebene ein, in welcher der MONITOR laufen soll; für l ist eine der Hexa-Ziffern Ø...F einzusetzen. Der MONITOR muß stets auf der gleichen Ebene laufen wie das zu testende Programm.

Zur Bestätigung ist dann "Wagenrücklauf" (cr) einzugeben; jedes andere Zeichen erklärt die Eingabe für ungültig. (Das gilt auch für alle im folgenden genannten Kommandos).

Dann kann mit

BUF nnnn (cr)

die Länge eines Pufferbereichs angegeben werden, in den später einzufügende Maschinencode-Bytes eingegeben werden können; für nnnn ist eine entsprechende 4-stellige Hexa-Zahl zu wählen. Andernfalls wird eine Standard-Pufferlänge benutzt.

Schließlich wird das zu testende Programm über eine der Einlese-Betriebsarten in den Kernspeicher gelesen (siehe EIN/AUSGABE).

## STEUERKOMMANDOS

Das zu testende Programm wird mit

aaaa S (cr)

gestartet, wobei für aaaa die Startadresse als 4-stellige Hexa-Zahl einzugeben ist.

Läuft das Programm später auf einen vom MONITOR vorgewählten Halt, so kann es durch eines der folgenden Kommandos wieder angestoßen werden:

| | |
|---|---|
| N (cr) | Nächste Instruktion ausführen, dann wieder anhalten |
| G (cr) | Weiterlaufen bis zum nächsten Monitor-Halt |

Das Kommando

E (cr)

beendet den Monitor-Betrieb. Er kann durch Starten über die Rechner-Konsole wieder aufgenommen werden.

## MONITOR-HALT

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

aaaa H (cr)

vorbereitet, wobei aaaa die Haltadresse ist. Sie muß einem Befehlsbyte bzw. dem eines vorgeschalteten DO-Befehls entsprechen (d.h. dem Basis-Byte eines Instruktions-Strings laut Assembler-Protokoll). Das Programm hält dann nach Ausführung des davorliegenden Befehls an. Halts nach unbedingten Sprüngen und Ebenenwechsel-Befehlen (zu höheren Ebenen) sind wirkungslos; ebenfalls nach bedingten Sprüngen, wenn verzweigt wird.

Es können bis zu 5 Haltbefehle eingebaut werden; in Programmschleifen sind stets mindestens 2 Halts vorzusehen.

Haltbefehle kann man einzeln mit

aaaa D (cr)

wieder eliminieren. Durch

D (cr)

werden sämtliche vorgesehenen Halts wieder gelöscht.

## ABFRAGEN, ÄNDERN UND EINFÜGEN

Sobald das zu testende Programm auf einen Monitor-Halt läuft, wird ein Kommando des Bedieners erwartet.

Nach Eingabe von

aaaa L (cr)

wird der Inhalt eines Register- oder Speicherplatzes aaaa ausgedruckt; nach Eingabe von

aaaa bbbb L (cr)
xx

der Inhalt sämtlicher Adressen von aaaa bis bbbb einschließlich. Je Byte wird eine zweistellige Hexa-Zahl gedruckt; ein Leerschritt trennt sie vom nächsten Byte.

Nach Ausgabe eines Einzelbytes (aaaa L) kann der Bediener durch Betätigen der Taste WRU Ausdrucken des nächsten Byte-Inhalts anfordern; das kann beliebig wiederholt werden, wobei jeweils die nächsthöhere Adresse abgefragt wird. Beendet wird dieser Vorgang durch Eingeben von # .

Durch das Kommando

aaaa A (cr)
xx

wird der Inhalt der Adresse aaaa durch xx ersetzt; durch

aaaa bbbb A (cr)
xx yy ... #

die Adressen aaa bis bbb durch einen String (xx yy ...). Einzugeben sind je Byte zwei Hexa-Ziffern, mit oder ohne Leerschritte zwischen den Bytes. Wird der String vorzeitig durch # beendet, so bekommen die restlichen Bytes Nullinhalt.

Nach Ändern eines Einzelbytes (aaa A) kann durch Taste WRU der nächste Platz aufgerufen und mit einer zweistelligen Hexa-Zahl geändert werden. Auch dieser Vorgang ist beliebig fortzusetzen; er wird durch Eingabe von # beendet.

Schließlich besteht die Möglichkeit, durch

```
aaaa I (cr)
xx yy ...#
```

an beliebiger Stelle im Programm (beginnend bei Adresse aaaa) einen Byte-String (xx yy ...) einzufügen; er wird in dem eingangs erwähnten Pufferbereich abgelegt. Es können mehrere Einfügungen vorgenommen werden; ihre Zahl ist nur durch die Größe des Pufferbereichs begrenzt.

Mit dem Kommando

```
aaaa U (cr)
```

kann jede einzelne Einfügung rückgängig gemacht werden; die Eingabe

```
U (cr)
```

löscht alle Einfügungen.

## EIN/AUSGABE

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

| | |
|---|---|
| aaaa bbbb ISH | Einlesen über langsamen Leser (Konsol-Teletype) |
| aaaa bbbb IFH | " " schnellen Leser |
| aaaa bbbb OSH | Ausstanzen auf langsamen Locher (Konsol-Teletype) |
| aaaa bbbb OFH | " " schnellen Locher |

Mit aaaa ist die erste, mit bbbb die letzte Adresse des Kernspeicher-Bereichs gemeint.

Gelesene und gelochte Streifen haben Hexa-Format (s. Anhang).

# Programmier- Hinweise

## VORBEMERKUNG

Dieser Abschnitt enthält einige Hinweise für die Programmierung des MINCAL 621, die sich aus der Multiprogramming-Struktur und der Art der Peripherie-Schaltungen ergeben und beachtet werden sollten.

## PROGRAMM-ANFANG

Jedes Programm sollte mit der Instruktion

ECL

beginnen; damit wird der Ebenenwechsel freigegeben.

Bei Rechnen mit Netzausfallschutz ist eine weitere Maßnahme vorzusehen. Sobald das Netz wiederkehrt, werden alle Flip-Flop-Schaltungen nullgesetzt. Das N-Register jedoch wird auf die Adresse '4ØØØ gesetzt, das ist das erste Byte im ersten Kernspeicher. Dort ist eine Anfangsroutine vorzusehen, die aus folgenden Elementen besteht:

- DCL-Befehl (Befehlsbyte auf '4ØØØ)
- Laden der Programmstand-Speicherregister[1] aller benutzten Ebenen mit den erwünschten Anfangsadressen
- Indirekter Sprung über das Programmstand-Speicherregister

Wird irgendeine Ebene gestartet, entweder bei Wiederkehr des Netzes automatisch die Ebene Ø (wenn Netzausfallschutz so beschaltet) oder von außen bzw. durch die Rechner-Uhr irgendeine andere Programmebene, so läuft dieses Programm in der jeweiligen Ebene richtig und springt dann auf die Anfangsadresse des Programms der gestarteten Ebene. Dort muß (und dies ist je Ebene vorzusehen) entweder sofort oder nach weiteren, vom Benutzer zu bestimmenden Instruktionen der Befehl ECL stehen, um den DISABLE-Zustand wieder aufzuheben und Multiprogrammierung zu ermöglichen.

[1] Register-Adressen ØØ/Ø1 der einzelnen Ebenen

Beispiel für eine Anfangsroutine:

```
ANF     DCL                          Unterbrechung verhindern
        2=*LDC ,@ ,'4Ø4Ø   }
        2=*STA ,@ ,'ØØØØ   }  für Ebene Ø
        2=*LDC ,@ ,'4Ø8Ø   }
        2=*STA ,@ ,'ØØ1Ø   }   "    "  1
        2=*LDA ,@ ,STAF    }
        2=*LDA ,@ ,'ØØFØ   }   "    "  F
            JPX ,   ,      ,'ØØ      Sprung zum Anfang
```

ANF liegt auf Adresse 'ØØØØ. Die Startadressen der Ebenen Ø bzw. 1 werden auf feste Werte ('4Ø4Ø bzw. '4Ø8Ø) gesetzt, während für Ebene F der Inhalt des Speicherplatzes STAF (+ folgender) als Startadresse maßgebend ist. Für die Lage der Programmstandspeicher im Pool ist angenommen, daß jede Ebene 16 byte als Registerplätze hat; daraus ergeben sich die absoluten Adressen 'ØØØØ, 'ØØ1Ø bzw. 'ØØFØ für die Ebenen Ø, 1 bzw. F.

## RECHNER-UHR

Die Zentraleinheit des MINCAL 621 kann eine Rechner-Uhr (real-time-clock) erhalten. Sie besteht aus einer Untersetzerschaltung, die vom quarzgesteuerten Taktgenerator des Rechners betrieben wird und in Abständen von wahlweise 1, 10, 100 oder 1000 ms den Start einer Ebene bewirkt. Taktabstand und gestartete Ebene werden durch Beschaltung in der Zentraleinheit festgelegt.

Die Uhr kann vom Programm her blockiert und freigegeben werden, indem man die BUS-Adresse '3FFF mit Ø oder einer rechtsbündigen 1 belegt:

```
LDC,@ ,Ø          }
STA ,@ ,'3FFF     }   Uhr AUS (unwirksam)

LDC,@ ,1          }
STA ,@ ,'3FFF     }   Uhr EIN (wirksam)
```

Durch die Taste RS und durch die Nullstellung bei Netzwiederkehr wird die Uhr in den AUS-Zustand gebracht.

## PROZESS-EIN/AUSGABEN

Für die Prozeßperipherie des MINCAL 621, d.h. alle Ein/Ausgabeschaltungen, die sich nicht auf Geräte, wie z.B. Fernschreiber, Leser, Locher usw. beziehen, sind die BUS-Adressen '2ØØØ...'3FFF vorgesehen, also 8192 verschiedene Adressen. (Bei eingebauter Uhr ist die letzte dieser Adressen - '3FFF - für diese reserviert).

Der Datentransfer zwischen CPU und Prozeßperipherie geschieht in der gleichen Weise wie der zwischen CPU und Kernspeicher, d.h. über den Universal-BUS und die BUS-bezogenen Befehle.

| | | |
|---|---|---|
| Beispiele: | LDA, @ ,'2ØØØ | Eingabe '2ØØØ nach @ |
| EXT3: | Q ,'2F78<br>LDA,REG7,'EXT3,IXR | } Eingabe von EXT (= '2F78) + IXR nach REG7 |
| 6=* | STA, @ ,'3ØØF | Ausgabe @ (+ 5 folgende Bytes) nach '3ØØF (+ 5 folgende Adressen) |

Je Adresse können Daten von 1 byte Länge ausgetauscht werden; Ein- und Ausgabe sind in Verbindung mit der gleichen Adresse möglich, wenn im Interface die entsprechenden BUS-Anschlüsse berücksichtigt werden. Außer dem Befehl LD... können auch die Befehle AD..., SB..., OR..., AN... und EO... verwendet werden, wenn dies zweckmäßig erscheint. Als Adressierungsarten kommen ...A (absolut) und ...X (indirekt) in Betracht; im letzteren Falle ist ein 2-byte-Indexregister (gerade Basisadresse) zu wählen, in dem die externe Adresse steht.

## GERÄTE-PERIPHERIE

Die Geräteperipherie des MINCAL 621 hat den BUS-Adreßbereich '1ØØØ...'1FFF. Dabei ist zu beachten, daß bei den meisten Gerätetypen mehr als ein Flip-Flop-Register von Byte-Länge im Interface enthalten und dementsprechend mehrere Adressen vorgesehen sind.

Jedem Gerät ist eine aus zwei Hexa-Ziffern bestehende Gerätenummer gg zugeordnet, jedem Interface-Register eine weitere Hexa-Ziffer f. Aus diesen baut sich die BUS-Adresse auf:

'1ggf = BUS-Adresse Register f für Gerät gg

Damit können maximal 64 Geräte mit je 16 Interface-Registern von Byte-Länge angesprochen werden. Jedes Interface und damit jedes Gerät kann, wenn entsprechend beschaltet, einer bestimmten Programmebene zugeordnet werden. Dadurch vervielfacht sich die Zahl der möglichen Geräte, denn in diesem Falle wird - bei gleicher

Geräte-Nummer - immer jeweils das Interface angesprochen, welches der jeweiligen Programmebene zugeordnet ist.

Typische Interface-Schaltungen wie die für den 8-Kanal-Fernschreiber (Teletype ASR 33) und für den schnellen Streifenleser und -locher haben jeweils 2 Interface-Register von Byte-Länge mit den Adressen:

'1ggØ = Datenregister
'1gg1 = Statusregister

Das Datenregister bewirkt den byte-weisen Datenaustausch zwischen Periphergerät und Universal-BUS.

Das Statusregister steuert die Ein/Ausgabe und hat folgende Einzelfunktionen:

Bit Ø = READY
1 = IBUSY
2 = OBUSY
3 = LOCK
4 = INITIATE
5...7 = (nicht benutzt)

IBUSY bzw. OBUSY leitet den Ein- bzw. Ausgabevorgang ein (wenn gleichzeitig READY ausgeschaltet ist). Ist der Ein- bzw. Ausgabevorgang beendet, z.B. ein Zeichen ausgedruckt, schaltet sich READY selbsttätig ein; es bewirkt einen Start der zugehörigen Programmebene, wenn nicht LOCK gesetzt ist. INITIATE hat eine besondere Funktion: Es löst z.B. bei Lesen den Transport des Streifens aus und muß daher beim angebauten langsamen Leser des Teletype und beim schnellen Leser zugleich mit IBUSY vom Programm eingeschaltet werden.

Zur Ein/Ausgabe über die Geräteperipherie werden im Normalfalle die Makrobefehle der LIBRARY ausreichen. Jeeoch kann der Benutzer anhand der folgenden Beispiele Ein/Ausgaben auch in Einzelschritten programmieren.

Ein/Ausgabe im Multiprogramming:

Diese auf die Struktur des MINCAL 621 zugeschnittene Betriebsart beruht darauf, daß nach Anstoß des Ein- oder Ausgabevorgangs die jeweilige Programmebene ausgeschaltet wird, um anderen Ebenen Gelegenheit zur Benutzung der Recheneinheit zu geben. Mit Ende des Vorgangs wird die auslösende Ebene wieder gestartet, und das Programm läuft weiter.

Ausgabe: Zunächst wird ein Register XOS mit dem Bitmuster ØØØØ Ø1ØØ (= 'Ø4) geladen, um bei jeder folgenden Ausgabe das Statusregister im Interface richtig zu bedienen (OBUSY ein, alle anderen aus):

```
LDC,XOS,'Ø4
```

Je Ausgabevorgang ist dann zu programmieren (gg = Geräte-Nummer):

```
STA,DAT,'1ggØ          (Datenregister laden)
STA,XOS,'1gg1          (Statusregister laden)
HLT
```

Der erste Befehl lädt das Datenregister des Interfaces mit dem im Register DAT stehenden Byte, der zweite stößt die Ausgabe an. Dann folgt ein Halt. Mit Ende des Ausgabezyklus' wird die Ebene wieder gestartet, und das Programm läuft weiter.

Eingabe: Zunächst wird ein Register XIS mit dem Bitmuster ØØØØ ØØ1Ø (= 'Ø2) geladen, entsprechend dem Statusregister-Inhalt bzw. den folgenden Eingabebefehlen (IBUSY ein, alle anderen aus):

```
LDC,XIS,'Ø2
```

Dies gilt z.B. für die Tastatur des Teletype [1]. Für dessen Leser, ebenso für den schnellen Leser [2], ist stattdessen das Bitmuster ØØØ1 ØØ1Ø (= '12) vorzusehen (zusätzlich INITIATE ein):

```
LDC,XIS,'12
```

Dann folgt je Eingabevorgang (gg = Geräte-Nummer):

```
STA,IXS ,'1gg1         (Statusregister laden)
HLT
LDA,DAT,'1ggØ          (Datenregister holen)
```

Der erste Befehl löst den Eingabevorgang aus; dann folgt ein Halt. Mit Ende des Eingabezyklus' wird die Ebene wieder gestartet, und der dritte Befehl transferiert den Inhalt des Datenregisters, d.h. das gelesene Byte, ins Register DAT.

1) Makrobefehle K... der LIBRARY

2) Makrobefehle R... der LIBRARY

Abschluß: Nach einer Folge von Ein- oder Ausgaben, in jedem Falle jedoch vor einem gewünschten Programm-Halt, muß das READY-Bit im benutzten Interface rückgesetzt werden, da sonst der Halt durch den infolge von READY dauernd anstehenden Programmstart überlaufen wird. Dies geschieht z.B. durch die Befehlsfolge (gg = Geräte-Nummer):

```
LDC,@ ,Ø
STA,@ ,'1gg1          (Nullstellen Statusregister)
```

Bemerkung: Während der oben beschriebenen Ein/Ausgaben darf der Rechner nicht im DISABLE-Zustand sein, da der Programm-Halt (HLT) nicht wirksam würde. Statt LD... können bei der Eingabe auch andere BUS-bezogene Befehle benutzt werden (mit dann anderer Funktion); statt absoluter kann indirekte Adressierung verwendet werden, ebenfalls Indizierung über ein Indexregister, sofern nur die effektive BUS-Adresse gleich der vom Status- oder Datenregister ist.

Ein/Ausgabe mit Warteschleifen:

Diese Betriebsart ist insbesondere dann von Nutzen, wenn ein Geräte-Interface, dessen Daten- und Statusregister keiner Ebene fest zugeordnet sind, von einer beliebigen Programmebene aus bedient werden soll.

Hierbei ist zunächst das LOCK-Bit des Statusregisters jedesmal zu setzen, um einen Start der Programmebene, auf die das READY-Flip-Flop des Interfaces im Normalfall auch bei den übrigen nicht ebenen-gebundenen Geräten wirkt, zu verhindern. Das bedeutet ein anderes Bitmuster beim Vorbereiten der Register XOS bzw. XIS:

```
      LDC,XOS,'ØC          (Ausgabe)
oder  LDC,XIS ,'ØA         (Eingabe ohne INITIATE)
oder  LDC,XIS ,'1A         (Eingabe mit INITIATE)
```

Im übrigen ist die Programmierung für Ein- und Ausgabe gleich denen für Multiprogramming-Betrieb, jedoch werden die Halt-Befehle (HLT) ersetzt durch die Befehlsfolge (gg = Geräte-Nummer):

```
LOOP: LDA  ,@ ,'1gg1          (Laden Statusregister)
      BNOC,@ ,'Ø1  ,LOOP      (Rückverzweigen bis READY gesetzt)
```

die so lange als Abfrageschleife läuft, bis mit READY der Vorgang beendet ist. Im Prinzip hat diese Betriebsart den gleichen Ablauf wie die Multiprogramming-Ein/Ausgabe; jedoch ist der Rechner währenddessen für alle Programmebenen mit niedrigerer Priorität gesperrt.

## KONSOL-PERIPHERIE

Ein 8-Kanal-Fernschreiber (Teletype ASR 33) mit eingebautem Streifenleser und -locher sowie ggfs. je ein schneller Streifenleser und -locher bilden die Standard-Peripherie eines MINCAL 621; sie werden als Konsol-Peripheriegeräte bezeichnet. Die Interfaces hierfür haben folgende Spezifikationen:

| | | |
|---|---|---|
| Fernschreiber: | Geräte-Nummer: | '00 |
| | BUS-Adresse Datenregister: | '1000 |
| | BUS-Adresse Statusregister: | '1001 |
| | Druckwerk: | Ausgabe programmieren |
| | Locher: | Ausgabe programmieren; Locher vorher manuell einschalten (Druckwerk läuft mit) |
| | Tastatur: | Eingabe programmieren ohne INITIATE [1] |
| | Leser: | Eingabe programmieren mit INITIATE [2] |
| Schnelle Lochstreifengeräte: | Geräte-Nummer: | '01 |
| | BUS-Adresse Datenregister: | '1010 |
| | BUS-Adresse Statusregister: | '1011 |
| | Locher: | Ausgabe programmieren |
| | Leser: | Eingabe programmieren mit INITIATE [2] |

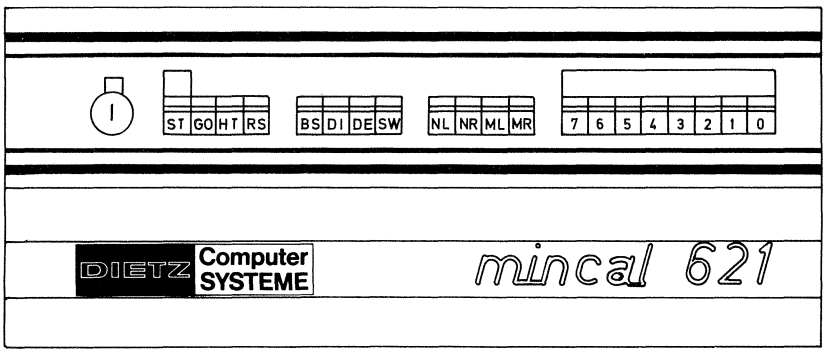
Die Interface-Register der Konsol-Peripherie sind nicht ebenen-gebunden; das bedeutet, daß sie von allen Programmebenen aus bedient werden können (und im übrigen, daß die Geräte-Nummern '00 und '01 an kein anderes Gerät gleich welcher Ebene vergeben werden dürfen). Der durch READY bewirkte Start (bei Ende Ein/Ausgabevorgang) startet jedoch stets Ebene 0.

[1] bzw. Makrobefehle K... der LIBRARY

[2] bzw. Makrobefehle R... der LIBRARY

# Bedienung

Zur Kontrolle des Computers ist eine Bedienungskonsole vorgesehen, über die die wichtigen Register und Zustände angezeigt werden und außerdem Daten eingegeben werden können. Der Computer kann aber auch ohne Bedienungskonsole betrieben werden.




Über ein 8-bit-Schalter-Register (Switch-Register) - Schalter 0...7 - können Daten in bestimmte Flip-Flop-Register, in Pool-Adressen und in BUS-Adressen gegeben werden.

Über dem Schalter-Register befindet sich ein 8-bit-Lampenfeld, das den Zustand von Flip-Flop-Registern, Pool-Adressen und BUS-Adressen anzeigt.

Links neben dem Switch-Register ist ein 4-bit-Schalterfeld, über das das N-Register (Instruktionszähler) und das M-Register angewählt werden können. Da beide Register 2-byte-Länge haben, wird jeweils die rechte Hälfte (NR, MR) mit den niedrigwertigen Bits oder die linke Hälfte (NL, ML) angewählt.

Die angewählten Register werden in dem Lampenfeld angezeigt. Bei Betätigen der Taste SW (aus dem Schalterfeld links neben der Registeranwahl) wird der Inhalt des Switch-Registers in das angewählte Register übertragen und gleichzeitig angezeigt. Sind weder ML, MR noch NL, NR angewählt, wird das A-Register angezeigt.

Durch gleichzeitiges Betätigen von NR und ML wird das B-Register und durch Betätigen von NR und MR das P-Register angezeigt.

Im dritten Schalterfeld von rechts sind außer der Taste SW (Switch) noch die Tasten DE (Deposit), DI (Display) und BS (Bootstrap) enthalten.

Durch Betätigen der Taste DE wird der Inhalt des Switch-Registers in die Adresse übertragen, die durch das M-Register angewählt wird.

Mit der Taste DI wird der Inhalt der Adresse angezeigt, die durch das M-Register angewählt ist (Voraussetzung: Tasten NL, NR, ML und MR sind in Ruhestellung).

Mit dem Schalter BS wird das eingebaute Bootstrap-Programm angewählt. Dieses Programm wird ausgeführt, wenn man zusätzlich die Taste ST (START) im Schalterfeld ganz links betätigt.

Im Schalterfeld ganz links gibt es folgende Tasten und Schalter:

RS (Reset): Hiermit werden alle Flip-Flops des Rechners in die Ausgangsstellung gebracht.

Schalter HT (Halt): Ein laufendes Programm kann mit diesem Schalter angehalten werden. Das N-Register, Pool- und BUS-Adressen lassen sich in diesem Zustand anzeigen und ändern.
Betätigt man dann die Taste GO (Go), so wird eine Instruktion ausgeführt; danach wird wieder angehalten.
Wird der Schalter HT wieder in die Ruhestellung gebracht, so läuft nach Betätigen der Taste GO das Programm weiter.
Läuft kein Programm (die Lampe über der Taste ST leuchtet in diesem Falle nicht), so führt ein Betätigen der Taste GO bei gleichzeitig eingelegtem Schalter HT zur Inkrementierung des M-Registers.

Mit der Taste ST (Start) wird die Ebene Ø des Computers gestartet. Die Lampe über dieser Taste leuchtet auf, sobald eine Ebene gestartet wurde und das Programm läuft.

Links auf der Bedienungskonsole ist ein Schlüsselschalter mit 3 Stellungen: In der 1. Stellung ist der Computer ausgeschaltet, in der 2. Stellung ist das Netz eingeschaltet, und die Lampe über dem Schalter leuchtet. In der 3. Stellung ist das Netz eingeschaltet (Lampe leuchtet), aber alle Schalter und Tasten der Bedienungskonsole sind verriegelt.

Läuft das Programm, so sind auch bei nicht verriegelter Bedienungskonsole alle Schalter und Tasten wirkungslos (Ausnahme: HT).

Nach dem Einschalten der Spannung mit dem Schlüsselschalter (die Lampe über dem Schalter leuchtet) ist der Computer betriebsbereit, und ein Programm kann über die Taste ST oder von außen über einen BUS-Start gestartet werden; danach leuchtet die Lampe über der Taste ST.

Während das Programm läuft, wird über das Lampenfeld der F-Kanal des Rechners angezeigt. Ist der Schalter HT nach unten geschaltet, so hält das Programm an. Im N-Register steht die Adresse des Befehlsbytes der Instruktion, die als nächste ausgeführt wird.

Bei angehaltenem Rechner können alle Flip-Flop-Register ohne Einfluß auf das Programm verändert werden. Bei Ändern des N-Registers wird das Programm bei der neuen Adresse fortgesetzt. Der Inhalt von N muß das Befehlsbyte einer Instruktion adressieren.

Bei angehaltenem Rechner (oder wenn kein Programm läuft) können Pool- und BUS-Adressen angezeigt und geändert werden: Die niedrigwertigen 8 Bits der gewünschten Adresse werden im Switch-Register eingestellt (Schalter betätigt = 1). Danach wird MR angewählt und durch Betätigen der Taste SW der Inhalt des Switch-Registers nach MR übertragen. Dieser Wert wird gleich angezeigt. Dann stellt man die 8 höherwertigen Bits der Adresse im Switch-Register ein, bringt MR in die Ausgangsstellung und schaltet ML ein. Durch erneutes Betätigen der Taste SW wird dieser Wert übernommen und angezeigt. Danach wird auch ML in die Ruhelage gebracht. Durch Betätigen der Taste DI wird nun der Inhalt der eingegebenen Adresse im Lampenfeld angezeigt. Will man diesen Wert ändern, so stellt man den neuen Wert im Switch-Register ein und betätigt die Taste DE. Zur Kontrolle kann man anschließend noch DI betätigen.

Will man mehrere aufeinanderfolgende Adressen anzeigen oder ändern, kann man bei ausgeschaltetem Programm und nach Einlegen des Schalters HT mit der Taste GO das M-Register um jeweils 1 erhöhen. Mit SW wird nur die Ausgangsadresse in M eingegeben und anschließend auf die beschriebene Weise erhöht.

Über den eingebauten "Bootstrap" kann ein Ladeprogramm (Lader) eingelesen werden. Hierzu legt man den Lochstreifen mit dem Lader in den Leser des Teletype oder den schnellen Leser ein (jeweils auf den Zufuhrbereich). Nach Betätigen des Schalters BS und der Taste ST wird der Lochstreifen in die ersten 256 Bytes des MOS-RAMs eingelesen (bei der schnellen Lochstreifeneinheit muß außerdem der Schalter 4 des Switch-Registers eingelegt werden). Nach dem Einlesen des Laders ist die Adresse ØØØØ über die Bedienungskonsole mit dem Wert ØØØØ1ØØØ (binär) geladen. Startet man dann über die Taste ST den Rechner, wird das Ladeprogramm ausgeführt.

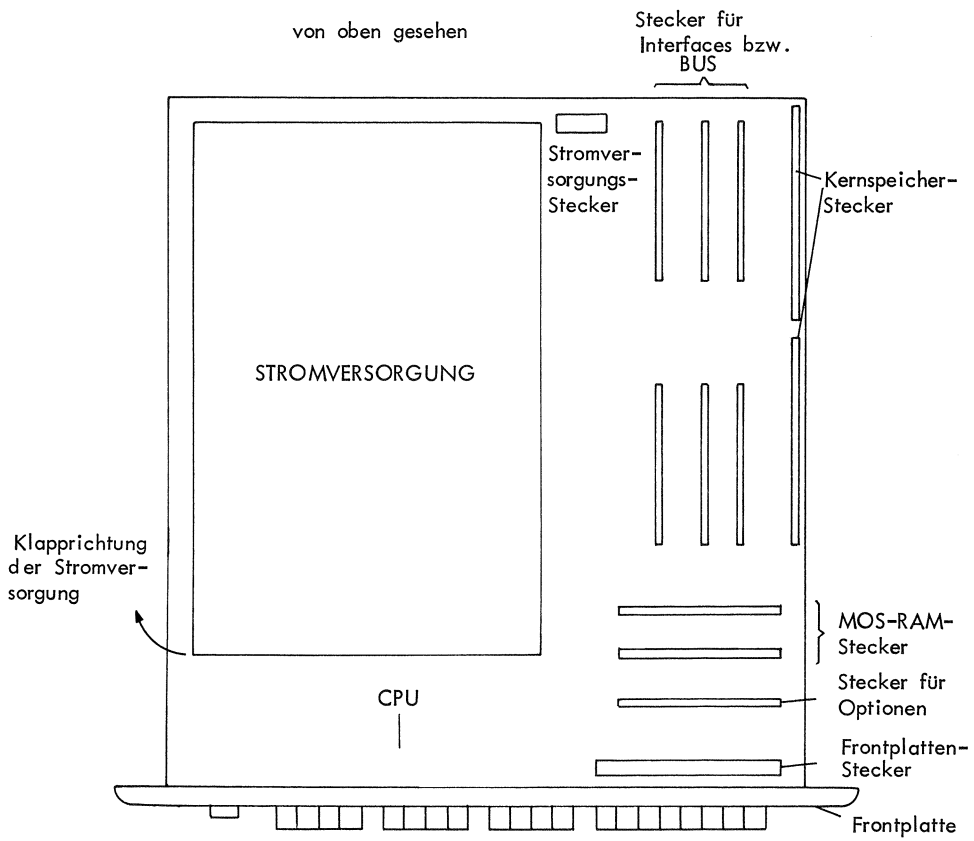
# Aufbau

Der MINCAL 621 besteht aus einem geschlossenen Gehäuse, das vorne von der Frontplatte (oder einer Blindplatte) abgeschlossen wird. Hinten befindet sich der Netzanschlußstecker. Hier werden auch die Kabel eingebauter Interfaces und das BUS-Kabel herausgeführt.

Die Kühlluft für den Rechner wird von vorn angesaugt (unter der Frontplatte). Sie geht durch einen Filter, das leicht gereinigt werden kann, an den Komponenten des Computers vorbei und wird nach hinten herausgeblasen.

In dem Gehäuse befindet sich unten der Kernspeicher (oder ein ROM), der waagerecht eingebaut ist. Darüber ist die CPU waagerecht montiert, die auch die Stecker für die Frontplatte, den Kernspeicher und die senkrecht gesteckten Leiterplatten für Optionen, für das MOS-RAM, die Ebenensteuerung und die Interfaces (bzw. den BUS) enthält.

Über der CPU befindet sich die hochklappbare Stromversorgung.



Bei der Stromversorgung blickt man auf die Leiterseite des Regelbausteins der Stromversorgung.

Auf der Leiterseite sind die wichtigsten Meßpunkte durch Beschriftung gekennzeichnet.

Bei der Inbetriebnahme des Rechners sind folgende Meßpunkte auf ihre Sollwerte gegen den Massemeßpunkt ("⊥" Telefonbuchse) zu überprüfen:

| Bezeichnung | Meßwert | Toleranz | Bemerkung |
|---|---|---|---|
| +Z | +5 V | + 2 % | Telefonbuchse |
| +T | +12 V | ∓ 5 % | Meßöse |
| +R | +18 V | ∓20 % | " |
| -R | -18 V | ∓20 % | " |
| $+Z_B$ | + 5 V | ∓ 2 % | " |
| $-Z_B$ | - 5 V | ∓ 2 % | " |
| +H | +15 V | ∓ 2 % | " |
| -H | -15 V | ∓ 2 % | " |

Für die Betriebsspannungen befinden sich Potentiometer an der zur Frontplatte zeigenden Leiterplattenkante. Von der Frontplatte aus gesehen haben diese Potentiometer von links nach rechts folgende Reihenfolge:

| Bezeichnung | Funktion |
|---|---|
| +Z | Spannungshöhe der +Z |
| S* | Strombegrenzung der +Z |
| N* | Ansprechschwelle des Netzausfallschutzes |
| +T | Spannungshöhe der +T |
| $+Z_B$ | " " $+Z_B$ |
| +B* | " " +B |
| $-Z_B$ | " " $-Z_B$ |
| -B* | " " -B |
| -H | " " -H |
| +H | " " +H |

* Diese Potentiometer dürfen nicht verstellt werden!

Folgende Spannungen werden durch Überspannungsschutzschalter überwacht:

| Bezeichnung | Einschaltstellung |
|---|---|
| +Z | Knebel zeigt zur Rückwand |
| +H | " " " " |
| -H | Stift ist eingedrückt |
| +T | " " " |
| $+Z_B$ | " " " |
| $-Z_B$ | " " " |

Netzsicherungen

Zwischen Stromversorgung und Rückwand befinden sich zwei Sicherungen.
Eine defekte Sicherung wird durch Aufleuchten der Sicherungsschraubkappe angezeigt.

Die rechte Sicherung (von der Frontplatte gesehen) ist für den Transformator Tr.1 (+Z; +T; +R).

Die linke Sicherung ist für den Transformator Tr.2 (-R; $+Z_B$; $-Z_B$; +H; -H).

Sicherungen zum Schutz der Batterien

Auf der Bestückungsseite des Regelbausteins befinden sich zwei Schmelzsicherungen.
Sie schützen die Batterien vor einem Kurzschluß im Entladezustand.

# Anhang

MINCAL 621 OPTIONEN (Zentraleinheit)

| | |
|---|---|
| Konsole: | Frontplatte mit Bedienungskonsole einschließlich Bootstrap-Loader |
| MOS-RAM: | Erweiterung um 1 bis 7 Einheiten zu je 256 Byte (bis 4k Byte) (je 2k auf einer MOS-RAM-Karte) |
| Kernspeicher: | 4k byte/1 us (erfordert Stromversorgung Typ 1)<br>8k byte/0.65 µs, 16k byte/0.65 µs } (erfordert Stromversorgung Typ 2) |
| Ebenen: | 2 oder 16 Programmebenen<br>Angabe der Register-Byte je Ebene erforderlich (16, 32, 64, 128 oder 256) |
| Parity/Clock: | Paritäts-Erzeugung und -Prüfung je Byte für Kernspeicher (9. Bit ist stets vorhanden)<br>Angabe, ob Rechner-Stop oder Start Ebene bei Parity-Fehler erfolgen soll<br>+<br>Echtzeit-Uhr (von 10 MHz der CPU abgeleitet)<br>Angabe über Periode (1, 10, 100, 1000 ms) und gestartete Ebene erforderlich |
| Pufferung: | Eingebaute Batterie-Pufferung für MOS-RAM bei Netzausfall (überbrückt 24h bei 1k MOS-RAM) |
| Netzausfallschutz: | Schützt Kernspeicher-Inhalt bei Netzausfall. Bei Wiederkehr des Netzes Wiederstart bei Adresse '1000 möglich (Ebene Ø) |
| Interfaces: | Die CPU enthält Einbauraum für 3 Interface-Karten, einschließlich einer eventuellen Anschlußkarte für den Universal-BUS |

# OPERATIONSZEITEN MINCAL 621

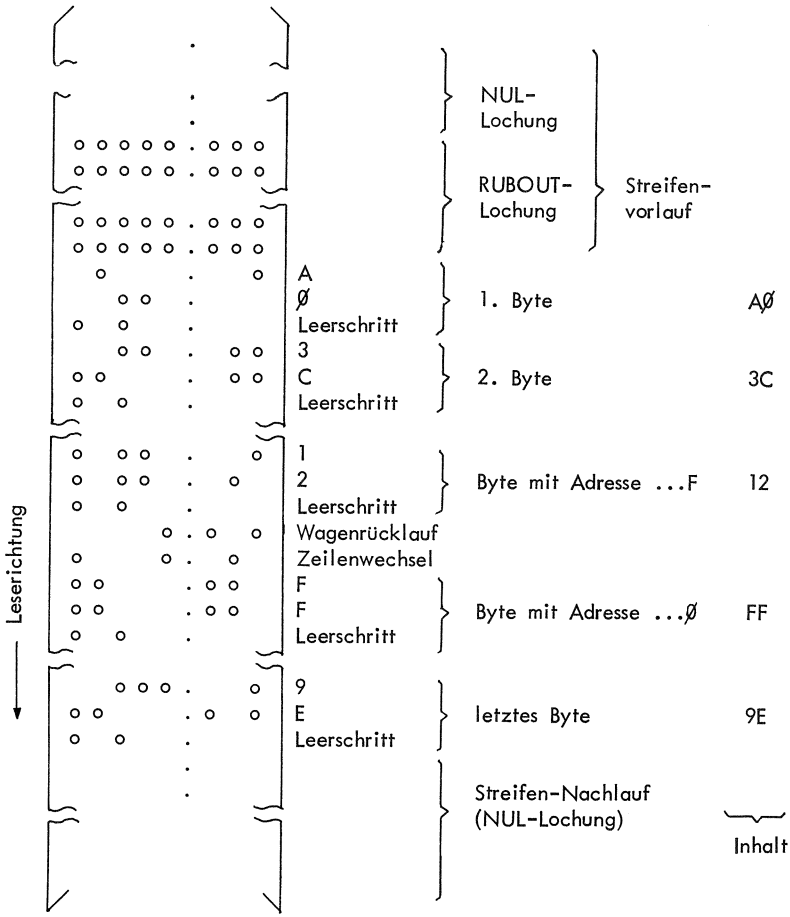
| Befehlsgruppe | Operationsdauer (us) | | + je Mehrfachausführung (us) | |
|---|---|---|---|---|
| | min. | max. | min. | max. |
| Steuerbefehle | 1.7 | 2.4 | | |
| DO | 1.7 | 2.4 | | |
| Zustandsabfrage | 1.9 | 2.6 | | |
| Schiebebefehle | 2.5 | 3.2 | 1.2 | 1.2 |
| Bedingter Sprung | 1.9 | 6.0 | 0.6 | 1.5 |
| BUS-bezogene Befehle | 2.8 | 8.5 | 1.2 | 1.7 |

Die Dauer der Befehle ist innerhalb der angegebenen Grenzen im wesentlichen von der Anzahl der abgefragten Bytes, d.h. der Länge der Instruktions-Strings abhängig; außerdem hängt sie davon ab, ob Operanden im POOL (MOS-RAM) oder im Kernspeicher liegen.

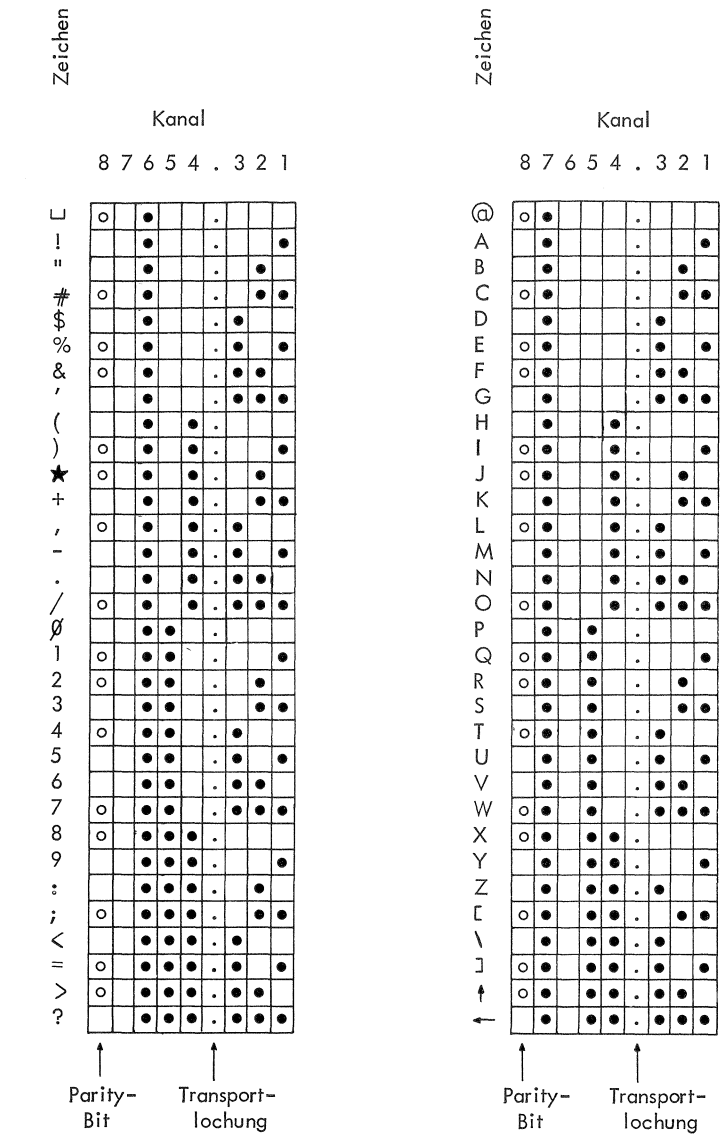
BUS-Befehle, die sich auf die Peripherie beziehen, verlängern sich um die je Interface eingestellte BUS-Transferzeit, wenn diese größer als 1 us ist.

Die oben angegebenen Zeiten gelten für den 4k-Kernspeicher mit 1 us Zykluszeit.

# HEXA-FORMAT-LOCHSTREIFEN



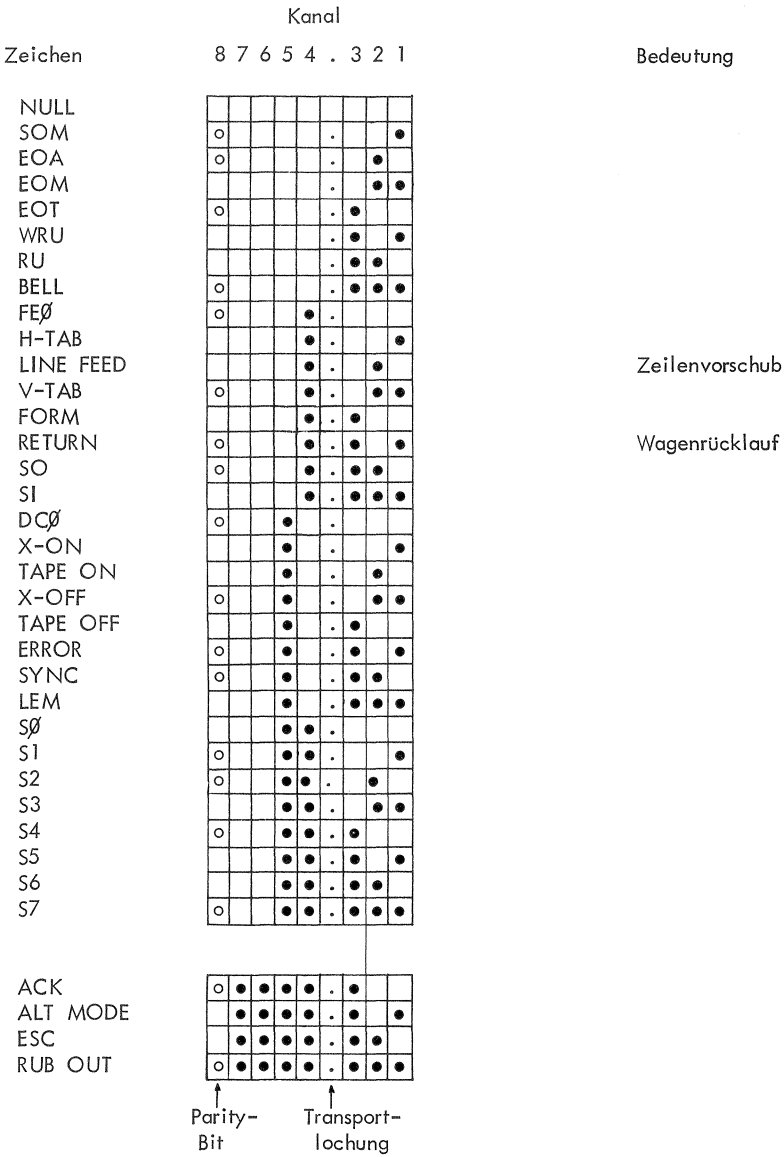
# ASCII-Code Druckbare Zeichen

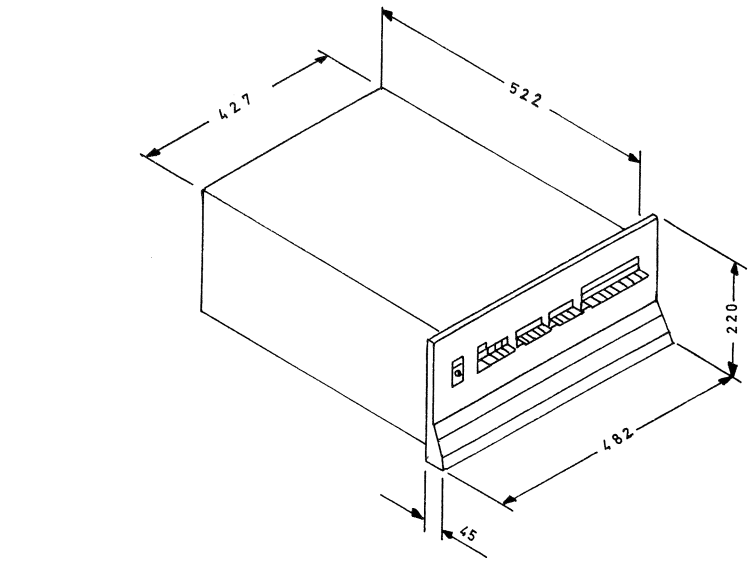


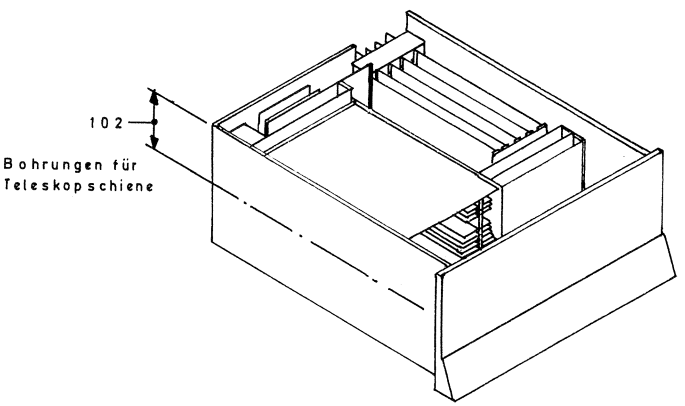

␣ (Zeichen Ø4Ø)
bedeutet Leerschritt

o ● = Dateninhalt 1
= Lochung im Streifen
= Stromschritt (MARK)

# ASCII-Code Steuerzeichen





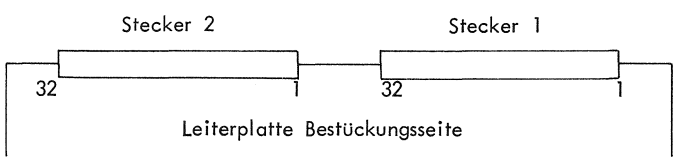




Abmessungen MINCAL 621

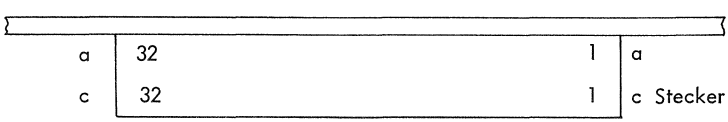
Steckerbelegung: Universal-BUS

| 2a | | 2c |
|---|---|---|
| 0 V | 1 | S3 |
| 0 V | 2 | S2 |
| 0 V | 3 | S1 |
| 0 V | 4 | S0 |
| BE | 5 | GE |
| FE | 6 | RK |
| N | 7 | frei |
| D7 | 8 | " |
| D6 | 9 | " |
| D5 | 10 | " |
| D4 | 11 | " |
| D3 | 12 | " |
| D2 | 13 | " |
| D1 | 14 | " |
| D0 | 15 | " |
| A15 | 16 | A15 |
| A14 | 17 | A14 |
| A13 | 18 | A13 |
| A12 | 19 | A12 |
| A11 | 20 | A11 |
| A10 | 21 | A10 |
| A 9 | 22 | A 9 |
| A 8 | 23 | A 8 |
| A 7 | 24 | A 7 |
| A 6 | 25 | A 6 |
| A 5 | 26 | A 5 |
| A 4 | 27 | A 4 |
| A 3 | 28 | A 3 |
| A 2 | 29 | A 2 |
| A 1 | 30 | A 1 |
| A 0 | 31 | A 0 |
| 0 V | 32 | 0 V |

| 1a | | 1c |
|---|---|---|
| +5 V | 1 | +5 V |
| +12 V | 2 | +12 V |
| -5 V | 3 | -5 V |
| -12 V | 4 | -12 V |
| L15 | 5 | L15 |
| L14 | 6 | L14 |
| L13 | 7 | L13 |
| L12 | 8 | L12 |
| L11 | 9 | L11 |
| L10 | 10 | L10 |
| L 9 | 11 | L 9 |
| L 8 | 12 | L 8 |
| L 7 | 13 | L 7 |
| L 6 | 14 | L 6 |
| L 5 | 15 | L 5 |
| L 4 | 16 | L 4 |
| L 3 | 17 | L 3 |
| L 2 | 18 | L 2 |
| L 1 | 19 | L 1 |
| L 0 | 20 | L 0 |
| 0 V | 21 | S15 |
| 0 V | 22 | S14 |
| 0 V | 23 | S13 |
| 0 V | 24 | S12 |
| 0 V | 25 | S11 |
| 0 V | 26 | S10 |
| 0 V | 27 | S 9 |
| 0 V | 28 | S 8 |
| 0 V | 29 | S 7 |
| 0 V | 30 | S 6 |
| 0 V | 31 | S 5 |
| 0 V | 32 | S 4 |




Leiterplatte



| | |
|---|---|
| Stecker für BUS-Anschluß: | ERNI Federleiste<br>Typ: STV-N-364<br>Best.-Nr.: 9722.343.001 |
| Stecker an Interface: | ERNI Messerleiste<br>Typ: STV-P-364<br>Best.-Nr.: 97.22.333.001 |