# dietz 621

## Handbuch

DIETZ Computer SYSTEME

# dietz 621

DIETZ Computer SYSTEME

Heinrich Dietz
Industrie-Elektronik
D-4330 Mülheim a.d. Ruhr 13
Solinger Straße 9
Tel. (02133)485024
Telex 856770

# Handbuch 4/74

Hinweis für den Leser:

Das vorliegende Handbuch DIETZ 621 soll Ihnen einen Überblick über das Computer-System 621, seine Konfigurationsmöglichkeiten, seine Software und seine Peripherie geben.

Detaillierte Information kann der Benutzer dem Handbuch entnehmen, wenn es um technische Daten, die Struktur und Bedienung der Zentraleinheit, die Maschinenbefehle und die Programmierung in Assembler geht.

Andere Aspekte und Baugruppen des Systems, insbesondere die Peripherie, Betriebssysteme und Programmiersprachen, sind hier nur so weit beschrieben, wie es das Verständnis von Funktion und Leistungsfähigkeit erfordert. Benutzer-orientierte Dokumentation hierzu liegt in getrennter Form vor.

Ausgabe: April 1974

Herausgeber: Heinrich Dietz
Industrie-Elektronik
D-4330 Mülheim a.d. Ruhr 13
Solinger Straße 9
Telefon (02133) 48 50 24
Telex 0856770

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

# Inhalt

# 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 Antrieben 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 den festprogrammierten Computer MINCAL 513 und sein freiprogrammiertes Gegenstück, den MINCAL 523.

1972 wird der Multibyte-Computer MINCAL 621 auf dem Markt eingeführt. Er hat sich schnell zu einem der erfolgreichsten Minicomputer auf dem europäischen Markt entwickelt und bildet den Kern des heutigen Systems DIETZ 621.

1973 kommt der DIETZ 1600 hinzu, ein außerordentlich leistungsfähiges System mit umfangreicher Software für Prozeßanwendungen und technisch-wissenschaftliche Datenverarbeitung im Foreground-/Background- oder Timesharing-Betrieb.

Am unteren Ende des Produktionsspektrums liegt der 1974 eingeführte Mikrocomputer DIETZ 211, der vor allem Aufgaben der Steuerung, Meßwerterfassung und -verarbeitung kostengünstig löst.

Heute entwickeln und fertigen in Mülheim 250 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 DIETZ 621 ist ein Teil dieser Gesamtkonzeption.

# Über den DIETZ 621

Das Konzept des DIETZ 621 berücksichtigt die Erfahrungen langjähriger erfolgreicher Computer-Entwicklung und verwendet modernste Technologien. Das günstige Preis-/Leistungsverhältnis ist nicht durch Weglassen wichtiger Funktionen erreicht worden, sondern durch eine neuartige Konzeption, die im Bereich der Kleincomputer ganz neue Maßstäbe setzt.

Hier in Kurzform die wichtigsten Eigenschaften des Computers:

## MULTIBYTE-KONZEPT

Der DIETZ 621 paßt sich optimal der gestellten Aufgabe an, weil er mit variabler Wortlänge arbeitet: 8 bit für Textverarbeitung und Zeichen-Ein/Ausgabe, 16 bit für Prozeßdaten, 32 bit für Rechengrößen.

Die Bytes werden durch die Computer-Hardware zu Daten beliebiger Länge verkettet - schnell, speichersparend, einfach.

## MULTIPROGRAMMING - HARDWAREUNTERSTÜTZT

Echtzeit-Aufgaben verlangen von einem Computer Multiprogramming-Eigenschaften: Programm-Unterbrechungen durch Abläufe mit hoher Priorität, gleichzeitige Bearbeitung mehrerer Programme.

Der DIETZ 621 hat bis zu 16 unabhängige Programmebenen mit eigenen Registern und Datenkanälen. Wechsel der Aufgaben bedeutet Wechsel der Ebene, und dieser erfolgt praktisch verzögerungsfrei, weil hardwaregesteuert.

## MULTIREGISTER-STRUKTUR

Jede Programmebene verfügt über bis zu 256 Register - eine für Computer dieser Klasse ungewöhnlich hohe Zahl. Viele Universalregister bedeuten kompakte, mit hoher Geschwindigkeit laufende Programme; sie ergänzen Multibyte- und Multiprogramming-Struktur zu einem einzigartigen Computer-Konzept.

## UNIVERSAL-BUS

Zentraleinheit, Kernspeicher, Peripherie und 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. Das BUS-Konzept erlaubt den Anschluß unterschiedlich schneller Speicher.

HALBLEITER-SPEICHER

Jeder 621 enthält - neben dem Hauptspeicher - einen Halbleiter-Speicher mit extrem kurzer Zugriffszeit, in dem sich Register, Datenpuffer und schnellaufende Programme befinden.

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 Urladeprogramm ist in einem ROM gespeichert und durch Tastendruck aufrufbar. Da dieses Programm nicht im Kernspeicher liegt, kann es auch nicht durch Programmierfehler zerstört werden.

ZUSÄTZLICHE OPTIONEN

In das Rechnergehäuse können zusätzlich eingebaut werden:
2 Interfaces für Standard-Peripherie; Real-Time-Clock; 4, 8, 16 oder 32 Kbyte Kernspeicher oder bis zu 8K reprogrammierbarer Festspeicher; Batteriepufferung für den Halbleiterspeicher.

COMPUTER-PERIPHERIE

Wie der Rechner selbst, so ist auch seine Peripherie modular ausbaufähig. Jede Anlage wird aus den Komponenten zusammengestellt , die zur Lösung der jeweiligen Aufgabe benötigt werden. Wächst die Aufgabe, wird das System erweitert, z.B. durch Kernspeicher-

erweiterung bis 80 Kbyte, Festkomma- oder Gleitkomma-Prozessoren, Platten-Systeme, Magnetband, Fernschreiber, Datensichtgeräte, Schnelldrucker, Lochstreifenleser/-Stanzer, Lochkartenleser, Plotter, XY-Schreiber, Datenfernübertragungseinrichtungen und Prozeß-Ein/Ausgänge für die unterschiedlichsten digitalen und analogen Signale.

## MODULARER AUFBAU

Zentraleinheit und periphere Erweiterungen sind so modular aufgebaut, daß sie in jedem Anwendungsfall kosten- und funktionsgerecht konfiguriert werden können.

# Über das DIETZsystem 621

Das DIETZsystem 621 ist ein universelles Hardware-/Software-System, das auf der Zentraleinheit 621 aufbaut und mit dem gesamten Peripherie- und Software-Katalog ausgerüstet werden kann.

## UNIVERSELLER EINSATZ

Die Einsatzmöglichkeiten des DIETZsystems 621 sind nahezu unbegrenzt, vor allem dort, wo es auf Echtzeit-Verhalten, Benutzerfreundlichkeit und Flexibilität ankommt: Datenerfassung, Meßwertverarbeitung, Steuerung und Regelung - die klassischen Aufgaben von Prozeßrechnern. Experimentsteuerung, Automatisierung analytischer Meßgeräte, Laborautomation, medizinische Technik - ein bedeutendes Gebiet für den Computer-Einsatz.

Technisch-wissenschaftliche Datenverarbeitung, Unterrichts-Unterstützung - für Forschung und Lehrbetrieb.

Datenfernverarbeitungs-Systeme und intelligente Terminals - mit Fähigkeiten vollwertiger Computer.

Interaktive, dialogfähige Systeme für die Erfassung und Verarbeitung von kommerziellen, administrativen und Fertigungsdaten - mit schnellem Zugriff im Echtzeit-Betrieb.

## DIETZdisk - EIN NEUES SPEICHERMEDIUM

Jedes DIETZsystem 621 ist mit einem neuartigen Speichermedium ausgestattet: der DIETZdisk.

Ein flexibler Plattenspeicher in einer robusten, völlig geschlossenen, handlichen Kassette. Berührungslos gelesen und beschrieben, präformatiert und daher unempfindlich und austauschbar, mit hoher Kapazität und schnellem Zugriff.

Die DIETZdisk steht dem Benutzer als Programm- und Datenträger zur Verfügung. Ein sicheres, bequem handhabbares Medium, mit 256 Kbyte sofort zugreifbarer Kapazität. Eine Lösung, die Lochstreifen, Bandkassetten und Floppy Disks in den Schatten stellt.

Im DIETZsystem 621 C ist ein zweites DIETZdisk-Laufwerk als Systemspeicher enthalten. Kurze Zugriffszeit und hohe Transferrate bieten die Eigenschaften eines plattenorientierten Systems - zu den Kosten eines normalen Computers.

## 2.4 MBYTE SYSTEMSPEICHER

Das DIETZsystem 621 D enthält als Systemspeicher eine Wechselplatte mit 2.4 Mbyte Kapazität - für erhöhte Ansprüche an Speichervolumen und Zugriffsgeschwindigkeit. Die Erweiterung auf bis zu 9.6 Mbyte ist vorgesehen.

## MODULARE PERIPHERIE

Zum DIETZsystem 621 gehört ein umfangreicher Katalog von Periphergeräten, Prozeß-Interfaces und Datenübertragungs-Schnittstellen. Aufgrund der modularen Konzeption des Peripherie-Systems lassen sich optimale Konfigurationen für alle Einsatzfälle zusammenstellen.

Ein Universal-BUS verbindet die Peripherie-Anschlüsse mit dem Grundsystem; System-Erweiterungen sind daher problemlos durchführbar. Und: Die gesamte Peripherie wird von der Software unterstützt.

## BENUTZERFREUNDLICHES BETRIEBSSYSTEM

Das Betriebssystem faßt alle Hardware- und Software-Ressourcen zusammen.
Es unterstützt:
Benutzer/System-Dialog über das Konsolgerät; Behandlung von und Zugriff zu Dateien; Overlay von Programmen; Verarbeitung von Interrupt- und Zeitaufträgen; Verwaltung von Programmen; Edition und Übersetzung von Quellprogrammen; Erkennung von Systemfehlern.

Diese Eigenschaften werden garantiert durch

- DBOS Plattenorientiertes Basis-Betriebssystem
- DFMS Datei-Verwaltungssystem
- MPOS Multiprogramming-Betriebssystem
- RTOS Echtzeit-Betriebssystem

## FÜR JEDES PROGRAMM DIE GEEIGNETE SPRACHE

Dem Benutzer des DIETZsystems 621 steht eine Vielzahl von Computer-Sprachen zur Verfügung.

- MINCASS 600 - die Assembler-Sprache für speicher- und laufzeit-optimale Programmierung
- MARS 600 - ein Makro-Assembler für Realtime-Systeme, der dem Benutzer die Beschreibung des Echtzeitverhaltens und die Behandlung der Peripherie leicht macht
- BASIC - die Dialogsprache für technisch-wissenschaftliche Probleme
- BASEX - ein interaktives, benutzerfreundliches Programmiersystem für Echtzeit-Anwendungen jeder Art, das im Multiprogramming arbeitet und die gesamte System-Peripherie unterstützt
- C-BASIC - eine interaktive Programmiersprache für kommerziell-administrativen Einsatz
- FORTRAN IV - die leistungsfähige Programmiersprache für technisch-wissenschaftliche Anwendungen.

SOFTWARE-PAKETE

Für das DIETZsystem 621 steht eine ständig wachsende Zahl anwendungsorientierter Programm-Pakete zur Verfügung:

Basis-Programme für Meßwertverarbeitung; mathematische und statistische Programme; Datenfernübertragungs-Prozeduren und Terminal-Emulationen; Commercial Package für kommerzielle Systeme.

DIETZsystem 621 C - GRUNDVERSION

Computer-System für Multiprogramming in 6 Benutzer-Ebenen. Plattenunterstützt durch 256 Kbyte DIETZdisk. Das Grundsystem umfaßt:

- Zentraleinheit mit
  Kernspeicher 32 Kbyte 650 ns
  Halbleiterspeicher 1 Kbyte 200 ns
  Netzausfallschutz, Echtzeituhr
- DIETZdisk-Doppellaufwerk 2 x 256 Kbyte
  mittlere Zugriffszeit 210 ms
- Konsoldrucker 50 Z/s 80 Z/Zl
  mit Tastatur
- 19"-Gestellschrank

DIETZsystem 621 D - GRUNDVERSION

Computer-System für Multiprogramming in 6 Benutzer-Ebenen. Plattenunterstützt durch 2.4 Mbyte Wechselplattenspeicher. Das Grundsystem umfaßt:

- Zentraleinheit mit
  Kernspeicher 32 Kbyte 650 ns
  Halbleiterspeicher 1 Kbyte 200 ns
  Netzausfallschutz, Echtzeituhr
- DIETZdisk-Einlaufwerk 256 Kbyte
  mittlere Zugriffszeit 210 ms
- Plattenspeicher-System 2.4 Mbyte
  mittlere Zugriffszeit 60 ms
- Konsoldrucker 50 Z/s 80 Z/Zl
- 19"-Gestellschrank

OPTIONEN

- Halbleiter-Speicher 2 Kbyte (Erweiterung auf 12 Benutzer-Ebenen)
- Kernspeicher 48 Kbyte und 80 Kbyte
- Gleitkomma-Prozessor 32 bit (max. 10 µs)
- Plattenspeicher 4.8, 7.2 und 9.6 Mbyte

PERIPHERGERÄTE

- Mosaikdrucker-Terminal 50 Z/s 80 Z/Zl
- Bildschirm-Terminal 1000 Zeichen
- Bildschirm-Terminal 2000 Zeichen
- 8-Kanal-Fernschreiber
- Schnelldrucker 200 Zl/min 132 Z/Zl
- Streifenleser 125 Z/s
- Streifenlocher 75 Z/s
- Kartenleser 400 K/min
- Graphische Bildschirmgeräte
- XY-Schreiber
- Digitalplotter
- Magnetband-Systeme

PROZESS-ANSCHLÜSSE

- 8/16-bit-Interrupteingänge
- 16-/32-bit-Digitaleingänge
- 16-/32-bit-Digitalausgänge
- 16-bit-Zähleingänge/Zeitausgänge
- 12-bit-Analogeingänge
- 10-bit-Analogausgänge
- Analog-Meßsysteme 30 kHz/250 kHz
- Integrierendes Meßsystem
- und viele weitere Prozeß-Interfaces

DFÜ-INTERFACES

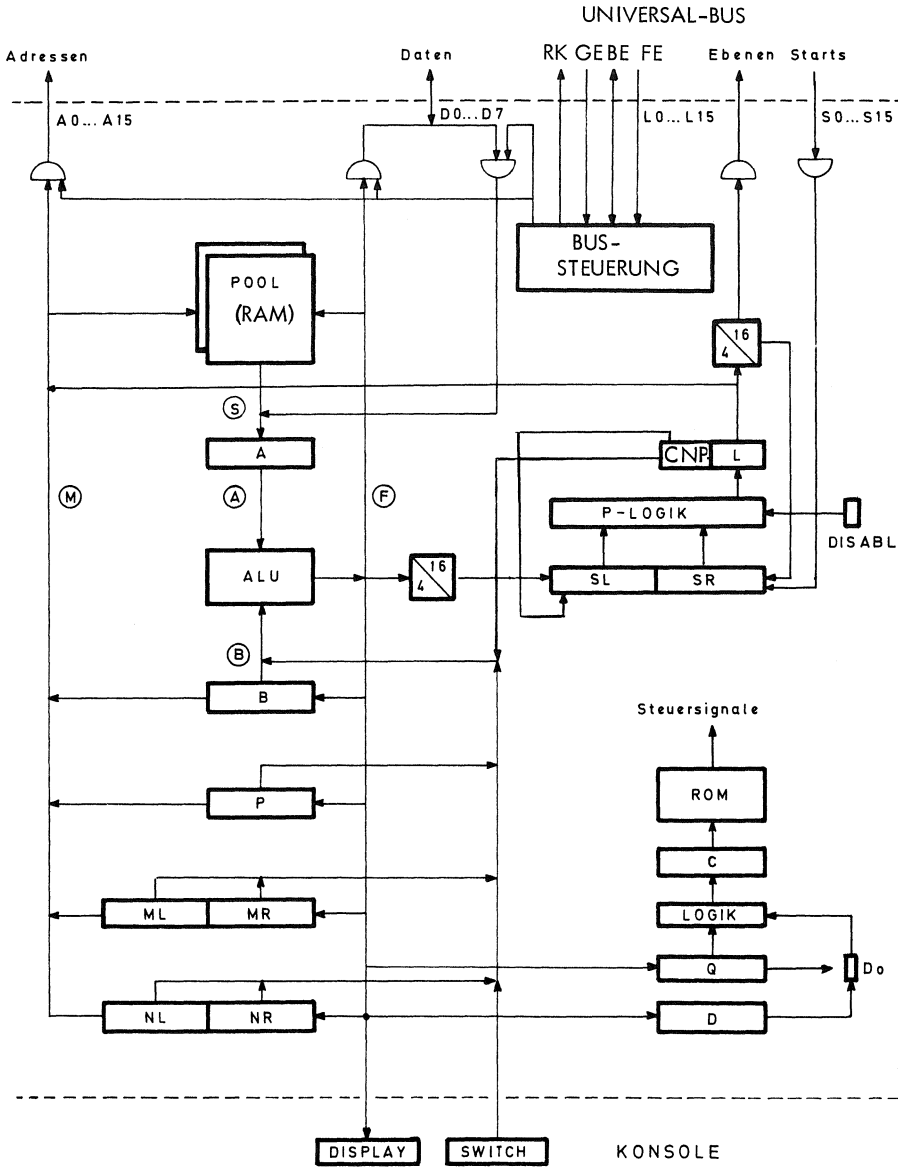
Synchrone und asynchrone Datenfernübertragungs-Schnittstellen, 110...9600 Baud.

# 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 DIETZ 621 ZENTRALEINHEIT

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 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: 8-bit-Register, bei dem in den Bits 0 bis 3 die laufende Ebene angegeben wird.

Die Bits 4 bis 7 dienen zur Identifikation eines Interrupts der CNP-Ebene.
Bit 4 wird bei Netzausfall, Bit 5 bei einer BUS-Belegung, die mit keinem FE-Signal quittiert wird, und Bit 6 bei Kernspeicher-Parity eingeschaltet. Bit 7 wird von dem Clock-Interrupt gesetzt.

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.

BUS-Steuerung: Logik zur Steuerung der BUS-Belegungsvorgänge der Zentraleinheit und zur Synchronisation der Zentraleinheit mit dem asynchronen BUS.

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 Halbleiter-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 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).

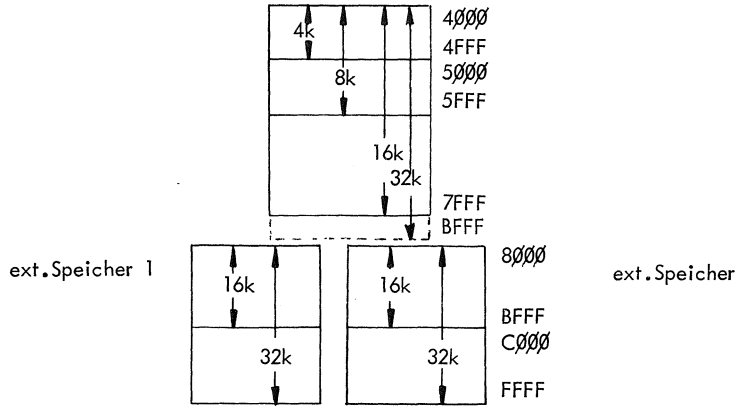
Ist der Pool nicht auf 256 byte Länge eingestellt, so reichen Pool-Adressen, die nicht mehr realisiert sind, in den Pool der nächsthöheren oder eventuell den Pool mehrerer höherer Ebenen. Die Pool-Bereiche schließen unmittelbar aneinander an.

## FESTSPEICHER

Anstelle des Kernspeichers kann ein reprogrammierbarer Halbleiter-Festspeicher eingesetzt werden, der in Stufen von 256 byte bis 8 kbyte ausbaufähig ist. Seine Adressen laufen von 4ØØØ bis 5 FFF.

KERNSPEICHER

Der Kernspeicher hat 4K, 8K, 16K oder 32K (bzw. bei externer Erweiterung bis 80 K) bytes Kapazität; seine Adressen laufen von 4ØØØ bis 4FFF, 5FFF, 7FFF oder BFFF (bzw. bis FFFF).




Bei einem Gesamtspeicherausbau bis 32K kann ein 32K-Modul als interner Speicher verwendet werden. Bei Ausbau über 32K ist die maximale Größe des internen Speichers auf 16K begrenzt.

Bei einem Speicherausbau über 48k (bis max. 80k) bytes sind entweder der externe Speicher 1 oder der externe Speicher 2 angewählt. Gleichzeitig können beide Speicher nicht angewählt sein.

Im Ausgangszustand kann mit dem externen Speicher 1 gearbeitet werden. Eine Umschaltung erfolgt mit einem TRA,ⓐ,'3FFD. Nach Ausführung des Befehls ist der externe Speicher 2 angewählt. Eine Rückumschaltung wird durch den Befehl TRA 'ⓐ '3FFE bewirkt.

Aus dem externen Speicher 1 und dem externen Speicher 2 kann zum RAM, zur Peripherie und zu dem ersten 16 k Kernspeicher zugegriffen werden.

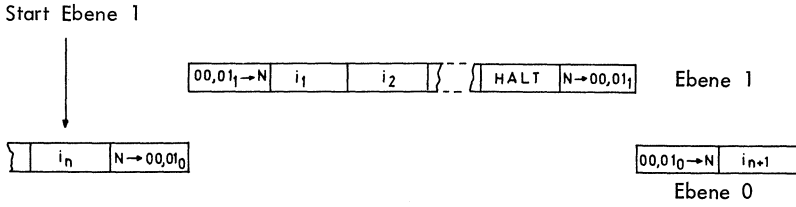
## PROGRAMMEBENEN

Es sind 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 DIETZ 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).

## UNIVERSAL-BUS

Der Universal-BUS ist ein Datenkanal zur Verbindung der einzelnen Komponenten des DIETZ 621-Systems Über diesen Datenkanal erfolgen

a) programmgesteuerte Ein- und Ausgaben und

b) Direkte Speicherzugriffe.

Ein Datentransfer über den Universal-BUS wird von einem aktiven Element initiiert (bei programmgesteuerter Ein-/Ausgabe von der Zentraleinheit, bei direktem Speicherzugriff z.B. von einem Plattencontroller), indem der BUS belegt wird (Signal BE) und über eine Adresse (Signale AØØ...A15 und gegebenenfalls LØØ...L15) ein passives Element (z.B. der Kernspeicher) angewählt wird. Über das Signal RK (Richtungskennzeichen) wird angegeben, ob Daten (DØ...D7) vom aktiven zum passiven Element oder vom passiven zum aktiven Element transportiert werden sollen. Das passive Element empfängt (oder sendet) nun die Daten und gibt außerdem mit dem Signal FE (Fertig) eine Quittung über den erfolgten Datentransport. Bleibt diese Quittung aus, weil z.B. ein nicht existierendes Element angesprochen wurde, erzeugt die BUS-Steuerung diese Quittung gleichzeitig mit einem System-Interrupt der CNP-Ebene.

Will ein passives Element von sich aus einen Datentransport initiieren, so sendet es einen Interrupt (SØØ...S15) zur Zentraleinheit, die als programmierte Reaktion hierauf als aktives Element einen Datentransport durchführt.

Konkurrieren mehrere aktive Elemente zur gleichen Zeit um eine BUS-Belegung, so wird dieser Konflikt über das Signal GE (Gewünscht) nach festgelegten Prioritäten gelöst. Der BUS enthält außerdem ein Nullstellsignal (N), das durch Betätigen der Taste RES der Bedienungskonsole oder bei einem Netzausfall erregt wird und das Signal STPX, das bei Fehlern der externen Speicher erregt wird.

BUS-Signale:

| | |
|---|---|
| DØ...D7 | Daten |
| AØØ...A15 | Adressen |
| LØØ...L15 | Levelleitungen |
| SØØ...S15 | Interrupt-Leitungen |
| RK | Richtungskennzeichen |
| BE | Belegt |
| GE | Gewünscht |
| FE | Fertig |
| N | Nullstellung |
| STPX | Fehlerstart |

## ADRESSIERUNG

Die BUS-bezogenen Befehle können wie folgt adressiert werden:

| | | |
|---|---|---|
| unmittelbar | (CONSTANT) | wahlweise |
| niveaugebunden | (REGISTER) | |
| relativ | (RELATIVE) | |
| voll | (ABSOLUTE) | |
| indirekt | (INDIRECT) | |
| 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..±32767 bei 2-byte-Indexregister; Ø...255 bei 1-byte-Indexregister). Mit 2-byte-Indexregistern lassen sich negative Indizes darstellen.

Indirekt bedeutet, daß der Inhalt des angegebenen Registers die Adresse bestimmt.

## 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 Operandenadresse (>&) 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, @ , CON,ADR
```

Sprung nach ADR, wenn der Inhalt des Akkus und der drei folgenden Bytes gleich der Konstanten CON ist.

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

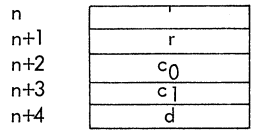
```
4<& BER, @ , REG,ADR
```

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 dritte Register-Byte mit der dritten Konstanten usw. verglichen. Verzweigt wird, wenn in allen Fällen Gleichheit besteht. Voraussetzung: DO mit Inkrementieren beider Adressen:

```
2=& BEC,REG,CON,ADR
```

Sprung nach ADR, wenn Register- und Konstanten-String gleich.

Die Instruktion sieht in diesem Falle folgendermaßen aus:




Befehl
Basisadresse Register
Konstante (niedriges Byte = Basis-Byte)
Konstante (hohes Byte)
Sprungweite

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

```
2=& BER ,REG1,REG2,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:

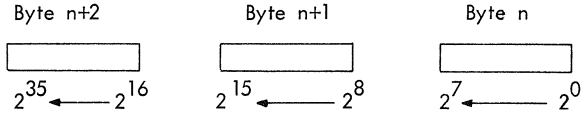
| | |
|---|---|
| 2=*ADL,REG,ADR | Doppel-Byte-Addition |
| 3=* SBL ,REG,ADR | 3-byte-Subtraktion |

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
```

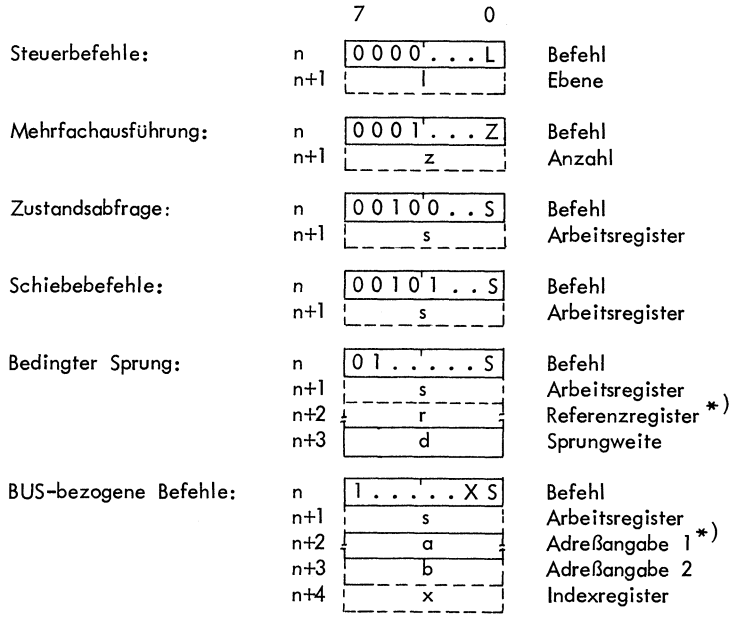
Sinngemäß läßt sich der DO-Befehl auch in Verbindung mit logischen Verknüpfungen verwenden:

```
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.

# 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.

Wird die CNP-Ebene durch Netzausfall, nicht quittierten BUS-Aufruf oder durch Speicher-Parity gestartet, wird gleichzeitig eine eventuell anstehende Unterbrechungssperre aufgehoben.

**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 | | I | | Ebene

Funktion: Die im folgenden Byte als rechtsbündige Hexazahl Ø...F angegebene Ebene I 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 | | I | | Ebene

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

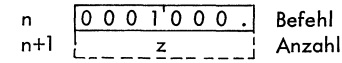
Eine DO-Instruktion bewirkt, daß die folgende Instruktion mehrfach ausgeführt wird. Die Anzahl der Ausführungen z kann von 2 bis 256 gewählt werden. Bei z = 2 ist nur das Befehlsbyte vorhanden; darüberhinaus steht z als rechtsbündige Binärzahl im folgenden Byte, wobei zu berücksichtigen ist, daß Nullinhalt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

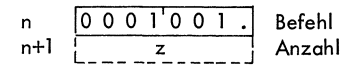
Zwischen einem DO-Befehl und beendeter Ausführung der nachfolgenden Instruktion kann das Programm nicht durch Wechsel der Programmebene unterbrochen werden.

z&  DO z-mal



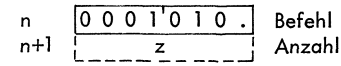

Funktion: Die folgende Instruktion wird z-mal ausgeführt. LINK wird nicht berücksichtigt; keine Adresse wird inkrementiert.

z*  DO z-mal mit Link



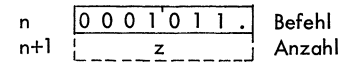

Funktion: Die folgende Instruktion wird z-mal ausgeführt. LINK wird berücksichtigt; keine Adresse wird inkrementiert.

z>&  DO z-mal
mit Inkrementieren Adresse



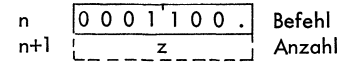

Funktion: Die folgende Instruktion wird z-mal ausgeführt. Nach jeder Ausführung wird die Operanden-Adresse M um 1 erhöht.

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



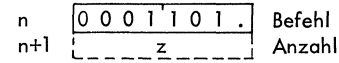

Funktion: Die folgende Instruktion wird z-mal ausgeführt. LINK wird berücksichtigt; nach jeder Ausführung wird die Operanden-Adresse M um 1 erhöht.

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.

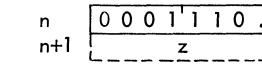
z<*  DO z-mal mit Link
mit Inkrementieren Register




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

z=&

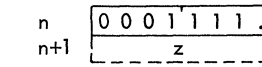
DO z-mal
mit Inkrementieren Register
mit Inkrementieren Adresse




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 Adresse




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.

Mit GL können außerdem Fehler- und Clock-Interrupt-Meldungen abgefragt werden.

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. Die linke Hälfte des Arbeitsregister-Inhalts hat folgende Bedeutung:

| | |
|---|---|
| <Bit 4> = 1 | Netzausfall |
| <Bit 5> = 1 | BUS-Belegung ohne Quittierung |
| <Bit 6> = 1 | Kernspeicher-Parityfehler |
| <Bit 7> = 1 | Clock-Interrupt. |

Diese Informationen kommen aus dem L-Register, dessen Bits 4 bis 7 nach einer GL-Instruktion in der CNP-Ebene automatisch gelöscht werden.

## SCHIEBEBEFEHLE

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

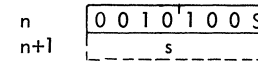
| | | | |
|---|---|---|---|
| Befehle: | SRO | Schieben rechts offen | . . . . . 0 0 . |
| | SRC | Schieben rechts zyklisch | . . . . . 0 1 . |
| | SLO | Schieben links offen | . . . . . 1 0 . |
| | SLC | Schieben links zyklisch | . . . . . 1 1 . |
| Arbeitsregister: | | Akkumulator @ | . . . . . 0 |
| | | 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

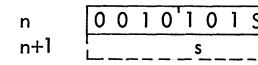



Funktion: Der Inhalt des Arbeitsregisters wird um 1 bit offen nach rechts verschoben. Bit 7 wird zu Null, und der vorherige Inhalt von Bit 0 geht verloren.

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

z<* SRO 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).

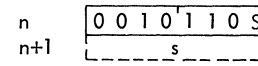
SRC Schieben rechts zyklisch




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

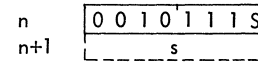



Funktion: Der Inhalt des Arbeitsregisters wird um 1 bit offen nach links verschoben. Bit 0 wird zu Null, und der vorherige Inhalt von Bit 7 geht verloren.

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

z<* SLO 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.

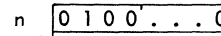
Dabei sind die Basis-Adressen der Register-Strings anzugeben.

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

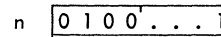
Bei Vergleichs- und Testbit-Abfragen ist die Bedingung insgesamt erfüllt, wenn sie in allen Bytes erfüllt ist.

Inkrementierung bezieht sich auch bei vorgeschaltetem DO nur auf das Basis-Byte des betreffenden Registers. Ein Überlauf wird nicht berücksichtigt.

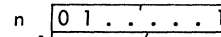
Länge: Abfrage auf Null/positiv: 2 byte (@)



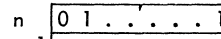

3 byte (s)




übrige Befehle: 3 byte (@)



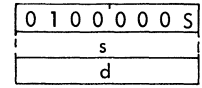

4 byte (s)




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.

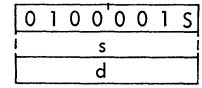
BZ Sprung wenn Null




Befehl
Arbeitsregister
Sprungweite

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

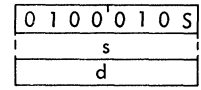
IZ Inkrementieren, Sprung wenn Null




Befehl
Arbeitsregister
Sprungweite

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

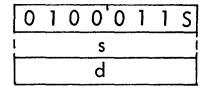
BP Sprung wenn positiv




Befehl
Arbeitsregister
Sprungweite

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

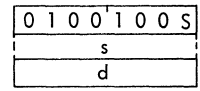
IP Inkrementieren, Sprung wenn positiv




Befehl
Arbeitsregister
Sprungweite

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

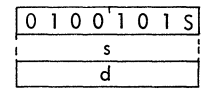
BNZ Sprung wenn nicht Null




Befehl
Arbeitsregister
Sprungweite

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

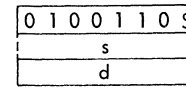
INZ Inkrementieren, Sprung wenn nicht Null




Befehl
Arbeitsregister
Sprungweite

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

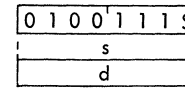
BNP Sprung wenn nicht positiv




Befehl
Arbeitsregister
Sprungweite

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

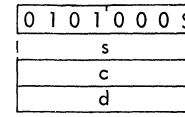
INP Inkrementieren, Sprung wenn nicht positiv




Befehl
Arbeitsregister
Sprungweite

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

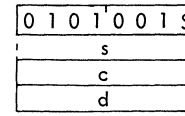
BEC Sprung wenn gleich Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

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

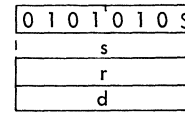
IEC Inkrementieren, Sprung wenn gleich Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

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

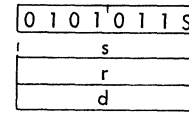
BER Sprung wenn gleich Register




Befehl
Arbeitsregister
Register
Sprungweite

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

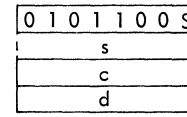
IER — Inkrementieren, Sprung wenn gleich Register




Befehl
Arbeitsregister
Register
Sprungweite

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

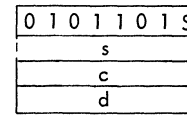
BNEC — Sprung wenn nicht gleich Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

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

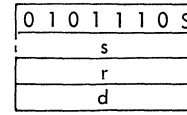
INEC — Inkrementieren, Sprung wenn nicht gleich Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

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

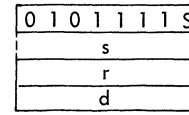
BNER — Sprung wenn nicht gleich Register




Befehl
Arbeitsregister
Register
Sprungweite

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

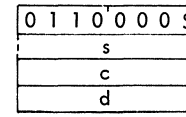
INER — Inkrementieren, Sprung wenn nicht gleich Register




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.

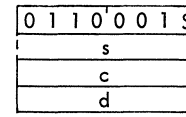
BZC Sprung wenn alle Testbits Null maskiert mit Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

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

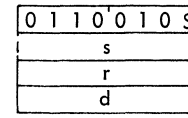
IZC Inkrementieren, Sprung wenn alle Testbits Null maskiert mit Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

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

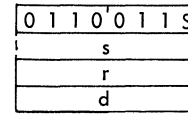
BZR Sprung wenn alle Testbits Null maskiert mit Register




Befehl
Arbeitsregister
Register
Sprungweite

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

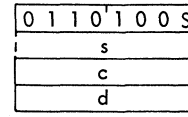
IZR Inkrementieren, Sprung wenn alle Testbits Null maskiert mit Register




Befehl
Arbeitsregister
Register
Sprungweite

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

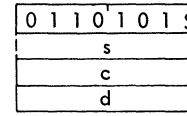
BNZC Sprung wenn nicht alle Testbits Null maskiert mit Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

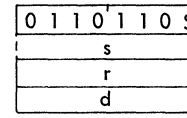
Funktion: Das Programm verzweigt, wenn nicht alle durch die Konstante c vorgegebenen Testbits des Arbeitsregisters Null sind.

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



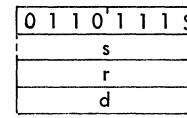

Funktion: Das Arbeitsregister wird inkrementiert. Das Programm verzweigt, wenn nicht alle durch die Konstante c vorgegebenen Testbits des Arbeitsregisters Null sind.

BNZR — Sprung wenn nicht alle Testbits Null maskiert mit Register



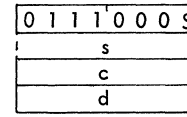

Funktion: Das Programm verzweigt, wenn nicht alle durch das Referenzregister r vorgegebenen Testbits des Arbeitsregisters Null sind.

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



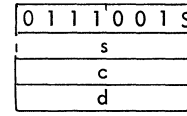

Funktion: Das Arbeitsregister wird inkrementiert. Das Programm verzweigt, wenn nicht alle durch das Referenzregister r vorgegebenen Testbits des Arbeitsregisters Null sind.

BOC — Sprung wenn alle Testbits Eins maskiert mit Konstante



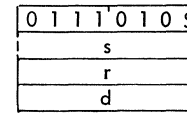

Funktion: Das Programm verzweigt, wenn alle durch die Konstante c vorgegebenen Testbits des Arbeitsregisters Eins sind.

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




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

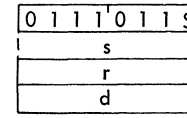
BOR — Sprung wenn alle Testbits Eins maskiert mit Register




Befehl
Arbeitsregister
Register
Sprungweite

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

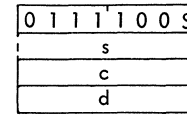
IOR — Inkrementieren, Sprung wenn alle Testbits Eins maskiert mit Register




Befehl
Arbeitsregister
Register
Sprungweite

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

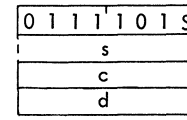
BNOC — Sprung wenn nicht alle Testbits Eins maskiert mit Konstante




Befehl
Arbeitsregister
Konstante
Sprungweite

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

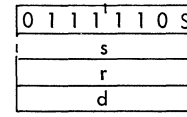
INOC — Inkrementieren, Sprung wenn nicht alle Testbits Eins maskiert mit Konstante




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.

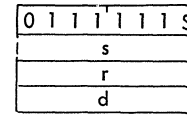
BNOR — Sprung wenn nicht alle Testbits Eins, maskiert mit Register




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 . . . . . X S |
|---|---|---|---|
| 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 RAM's (Pool), 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 RAM's.

Adressierungsmöglichkeiten sind:

| | | |
|---|---|---|
| ..C | Konstante: | Der Operand steht als Konstante in der Instruktion (nicht möglich bei ST, JP und CS) |
| ..X | indirekt: | Die effektive Adresse steht in einem Indexregister x |
| ..R | Register: | Die effektive Adresse ist ein Register r (nicht möglich bei JP und CS) |
| .. 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...+32767).

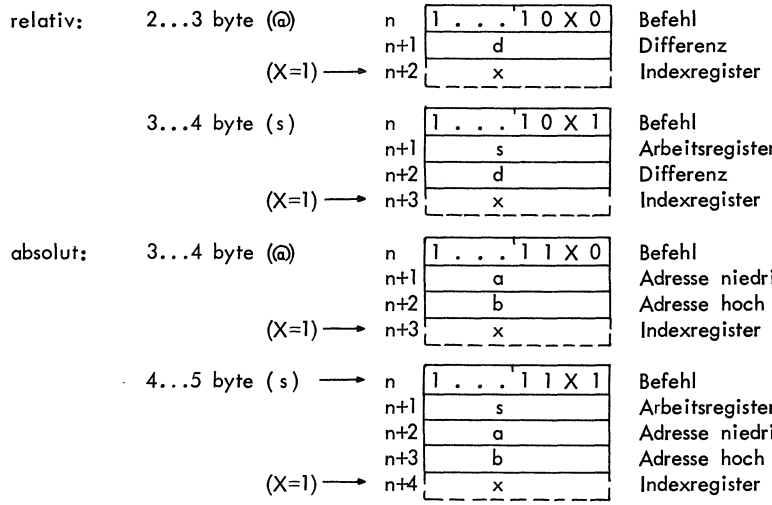
Ist dagegen eine ungerade Adresse x angegeben, so wird nur das Byte x als Index verwendet. Der Index ist eine 8-bit-Zahl, mit der nur positive Indizierung möglich ist (0...255).

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 mit der jeweiligen Ebenen-Adresse in logische ODER-Verknüpfung gebracht. Dies gilt auch bei Registeradressierung und indirekter Registeradressierung.

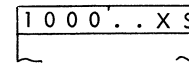
DO-Befehle: Die Anwendung von DO-Instruktionen auf die Befehle LD, AD, SB, AN, OR, EO und ST ist sinnvoll; die häufigsten Anwendungen sind später im einzelnen angegeben.

| Länge: | | | | | | |
|---|---|---|---|---|---|---|
| | Konstante: | 2 byte (@) | n | 1...0000 | Befehl |
| | | | n+1 | c | Konstante |
| | | 3 byte (s) | n | 1...0001 | Befehl |
| | | | n+1 | s | Arbeitsregister |
| | | | n+2 | c | Konstante |
| | indirekt: | 2 byte (@) | n | 1...0010 | Befehl |
| | | | n+1 | x | Indexregister |
| | | 3 byte (s) | n | 1...0011 | Befehl |
| | | | n+1 | s | Arbeitsregister |
| | | | n+2 | x | Indexregister |
| | Register: | 2...3 byte (@) | n | 1...01X0 | Befehl |
| | | | n+1 | r | Register |
| | | (X=1) ⟶ | n+2 | x | Indexregister |
| | | 3...4 byte (s) | n | 1...01X1 | Befehl |
| | | | n+1 | s | Arbeitsregister |
| | | | n+2 | r | Register |
| | | (X=1) ⟶ | n+3 | x | Indexregister |




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

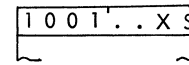



Funktion: Das Arbeitsregister wird mit dem Inhalt der effektiven Adresse (Operand) geladen.

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.

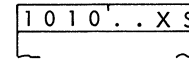
AD.. Addieren




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.

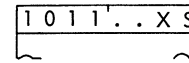
SB.. Subtrahieren




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

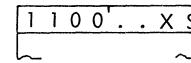



Funktion: Der Arbeitsregister-Inhalt wird mit dem Inhalt der effektiven Adresse (Operand) in UND-Verknüpfung gebracht; das Ergebnis steht im Arbeitsregister.

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

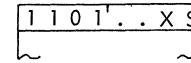



Funktion: Der Arbeitsregister-Inhalt wird mit dem Inhalt der effektiven Adresse (Operand) in inklusive ODER-Verknüpfung gebracht; das Ergebnis steht im Arbeitsregister.

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

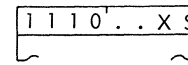



Funktion: Der Arbeitsregister-Inhalt wird mit dem Inhalt der effektiven Adresse (Operand) in exklusive ODER-Verknüpfung gebracht; das Ergebnis steht im Arbeitsregister.

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

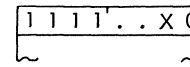



Funktion: Der Inhalt des Arbeitsregisters wird in der effektiven Adresse abgespeichert.

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

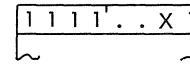
z=& ST.. Ein vorgeschalteter DO-Befehl z=& bewirkt Speichern eines Arbeitsregister-Inhalts von z Byte Länge in einem gleich langen Adreß-String.

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.

BEFEHLSLISTE

| Befehl | | | | Ausführungszeit | Bedeutung |
|---|---|---|---|---|---|
| NOP | | | | 2.1 | No Operation |
| SEL | LEV | | | 2.8 | Set Level |
| HLT | | | | 2.1 | Halt |
| HSL | LEV | | | 2.8 | Halt, Set Level |
| ECL | | | | 2.1 | Enable Change Level |
| DCL | | | | 2.1 | Disable Change Level |
| (DO) | (NUM)... | | | 1.9 | Do next ... times |
| GL | (REG) | | | 1.9 | Get Level |
| GS | (REG) | | | 1.9 | Get Switch |
| SRO | (REG) | | | 2.5/1.0 | Shift right open |
| SRC | (REG) | | | 2.5/1.0 | Shift right closed |
| SLO | (REG) | | | 2.5/1.0 | Shift left open |
| SLC | (REG) | | | 2.5/1.0 | Shift left closed |
| BZ | (REG) | | ADR | 3.2/0.6 | Branch if Zero |
| BNZ | (REG) | | ADR | 3.2/0.6 | Branch if Non Zero |
| BP | (REG) | | ADR | 3.2/0.6 | Branch if Positive |
| BNP | (REG) | | ADR | 3.2/0.6 | Branch if Non Positive |
| IZ | (REG) | | ADR | 4.0/0.6 | Increment, Branch if Zero |
| INZ | (REG) | | ADR | 4.0/0.6 | Increment, Branch if Non Zero |
| IP | (REG) | | ADR | 4.0/0.6 | Increment, Branch if Positive |
| INP | (REG) | | ADR | 4.0/0.6 | Increment, Branch if Non Positive |
| BEC | (REG) | CON | ADR | 4.3/1.7 | Branch if Equal Constant |
| BER | (REG) | REG | ADR | 4.9/1.2 | Branch if Equal Register |
| BNEC | (REG) | CON | ADR | 4.3/1.7 | Branch if Non Equal Constant |
| BNER | (REG) | REG | ADR | 4.9/1.2 | Branch if Non Equal Register |
| IEC | (REG) | CON | ADR | 5.1/1.7 | I, Branch if Equal Constant |
| IER | (REG) | REG | ADR | 5.7/1.2 | I, Branch if Equal Register |
| INEC | (REG) | CON | ADR | 5.1/1.7 | I, Branch if Non Equal Constant |
| INER | (REG) | REG | ADR | 5.7/1.2 | I, Branch if Non Equal Register |
| BZC | (REG) | CON | ADR | 4.3/1.7 | Branch if all Testbits Zero, Constant-Mask |
| BZR | (REG) | REG | ADR | 4.9/1.2 | Branch if all Testbits Zero, Register-Mask |
| BNZC | (REG) | CON | ADR | 4.3/1.7 | Branch if not all Testbits Zero, Constant-Mask |
| BNZR | (REG) | REG | ADR | 4.9/1.2 | Branch if not all Testbits Zero, Register-Mask |
| IZC | (REG) | CON | ADR | 5.1/1.7 | I, Branch if all Testbits Zero Constand-Mask |
| IZR | (REG) | REG | ADR | 5.7/1.2 | I, Branch if all Testbits Zero Register-Mask |
| INZC | (REG) | CON | ADR | 5.1/1.7 | I, Branch if not all Testbits Zero Const.-Mask |
| INZR | (REG) | REG | ADR | 5.7/1.2 | I, Branch if not all Testbits Zero Reg.-Mask |

| Befehl | | | | Ausführungszeit | Bedeutung |
|---|---|---|---|---|---|
| BOC | (REG) | CON | ADR | 4.3/1.7 | Branch if all Testbits ONE, Constant-Mask |
| BOR | (REG) | REG | ADR | 4.9/1.2 | Branch if all Testbits ONE, Register-Mask |
| BNOC | (REG) | CON | ADR | 4.3/1.7 | Branch if not all Testbits ONE, Const.Mask |
| BNOR | (REG) | REG | ADR | 4.9/1.2 | Branch if not all Testbits ONE, Reg.-Mask |
| IOC | (REG) | CON | ADR | 5.1/1.7 | I, Branch if all Testbits ONE, Const.-Mask |
| IOR | (REG) | REG | ADR | 5.7/1.2 | I, Branch if all Testbits ONE, Reg.-Mask |
| INOC | (REG) | CON | ADR | 5.1/1.7 | I, Branch if not all Testbits ONE, C-Mask |
| INOR | (REG) | REG | ADR | 5.7/1.2 | I, Branch if not all Testbits ONE, R-Mask |
| LDC | (REG) | | CON | 2.5/1.5 | LOAD Constant |
| ADC | (REG) | | CON | 2.9/1.9 | ADD Constant |
| SBC | (REG) | | CON | 2.9/1.9 | SUBTRACT Constant |
| ANC | (REG) | | CON | 2.9/1.9 | AND Constant |
| ORC | (REG) | | CON | 2.9/1.9 | OR Constant |
| EOC | (REG) | | CON | 2.9/1.9 | Excl. OR Constant |
| LDX | (REG) | | IXR | 4.2/1.0 | Load indirect |
| ADX | (REG) | | IXR | 4.6/1.4 | Add indirect |
| SBX | (REG) | | IXR | 4.6/1.4 | Subtract indirect |
| ANX | (REG) | | IXR | 4.6/1.4 | AND indirect |
| ORX | (REG) | | IXR | 4.6/1.4 | OR indirect |
| EOX | (REG) | | IXR | 4.6/1.4 | Excl. OR indirect |
| STX | (REG) | | IXR | 4.2/1.0 | Store indirect |
| JPX | | | IXR | 4.0 | Jump indirect |
| CSX | REG | | IXR | 5.5 | Call Subroutine indirect |
| LDR | (REG) | ADR | (IXR) | 3.6/1.0 | Load Register |
| ADR | (REG) | ADR | (IXR) | 4.0/1.4 | Add Register |
| SBR | (REG) | ADR | (IXR) | 4.0/1.4 | Subtract Register |
| ANR | (REG) | ADR | (IXR) | 4.0/1.4 | AND Register |
| ORR | (REG) | ADR | (IXR) | 4.0/1.4 | OR Register |
| EOR | (REG) | ADR | (IXR) | 4.0/1.4 | Excl. OR Register |
| STR | (REG) | ADR | (IXR) | 3.6/1.0 | Store Register |
| LDL | (REG) | ADR | (IXR) | 4.1/1.5 | Load relative |
| ADL | (REG) | ADR | (IXR) | 4.5/1.9 | Add relative |
| SBL | (REG) | ADR | (IXR) | 4.5/1.9 | Subtract relative |
| ANL | (REG) | ADR | (IXR) | 4.5/1.9 | AND relative |
| ORL | (REG) | ADR | (IXR) | 4.5/1.9 | OR relative |
| EOL | (REG) | ADR | (IXR) | 4.5/1.9 | Excl. OR relative |
| STL | (REG) | ADR | (IXR) | 4.1/1.5 | Store relative |
| JPL | | ADR | (IXR) | 3.4 | Jump relative |
| CSL | REG | ADR | (IXR) | 4.9 | Call Subroutine relative |

| Befehl | | | | Ausführungszeit | Bedeutung |
|---|---|---|---|---|---|
| LDA | (REG) | ADR | (IXR) | 4.7/1.0 | Load absolute |
| ADA | (REG) | ADR | (IXR) | 5.1/1.4 | Add absolute |
| SBA | (REG) | ADR | (IXR) | 5.1/1.4 | Subtract absolute |
| ANA | (REG) | ADR | (IXR) | 5.1/1.4 | AND absolute |
| ORA | (REG) | ADR | (IXR) | 5.1/1.4 | OR absolute |
| EOA | (REG) | ADR | (IXR) | 5.1/1.4 | Excl. OR absolute |
| STA | (REG) | ADR | (IXR) | 4.7/1.0 | Store absolute |
| JPA | | ADR | (IXR) | 4.5 | Jump absolute |
| CSA | REG | ADR | (IXR) | 6.0 | Call Subroutine absolute |

Bemerkung:

Die erste Zeitangabe ist die gesamte Befehlsausführungszeit, die zweite Angabe gibt die Zeit für eine Mehrfachausführung bei vorgeschaltetem DO-Befehl an.

Die angegebenen Ausführungszeiten der Befehle gehen von folgenden Voraussetzungen aus:

1) Das Programm steht im Kernspeicher (0.65 µs Zykluszeit). Sollte das Programm im Halbleiterspeicher stehen, können je Byte der Instruktion 0.5 µs von den angegebenen Zeiten abgezogen werden.

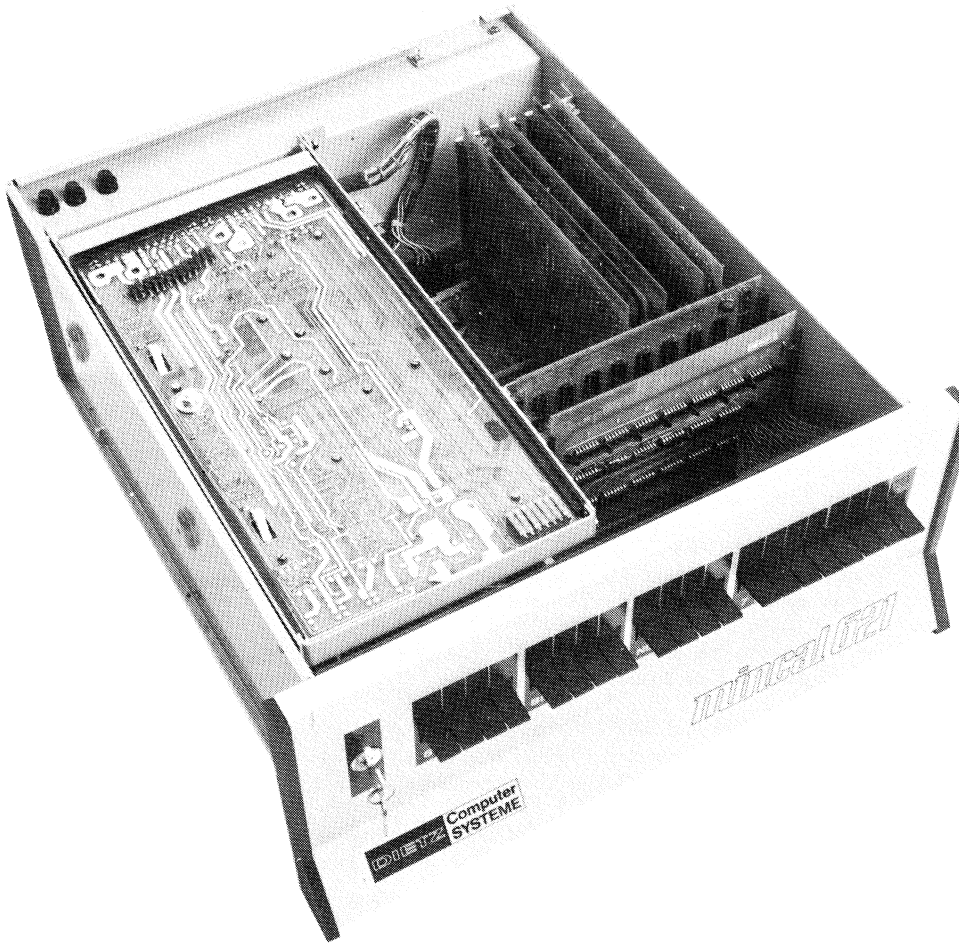
2) Ist bei einem DO-Befehl die angegebene Wiederholzahl > 2 oder ist als Register nicht der Standard-Akku ((a)) angegeben, so muß zu den angegebenen Zeiten 0.7 µs addiert werden. Dies gilt nicht für den CS-Befehl.

3) Bei Indizierung mit einem 1-byte-Indexregister sind 1.7 µs, bei einem 2-byte-Indexregister 2.3 µs zu addieren.

4) Bei CONSTANT- und RELATIVE-Adressierung steht die Konstante oder der adressierte Operand im Kernspeicher. Sollten sich diese Daten im Pool befinden, sind von den angegebenen Zeiten 0.5 µs zu subtrahieren.

5) Bei INDIRECT-, REGISTER- und ABSOLUTE-Adressierung steht der adressierte Operand im Pool. Steht er im Kernspeicher, so sind zu den angegebenen Zeiten 0.5 µs zu addieren.

6) Der UNIVERSAL-BUS des MINCAL 621 ist ein asynchroner BUS. Alle Zeiten hängen von der physikalischen Länge des BUS und von der Reaktionszeit der passiven Elemente ab. Die angegebenen Zeiten können sich deshalb in geringem Maße von den tatsächlichen Zeiten unterscheiden.


Computer
SYSTEME

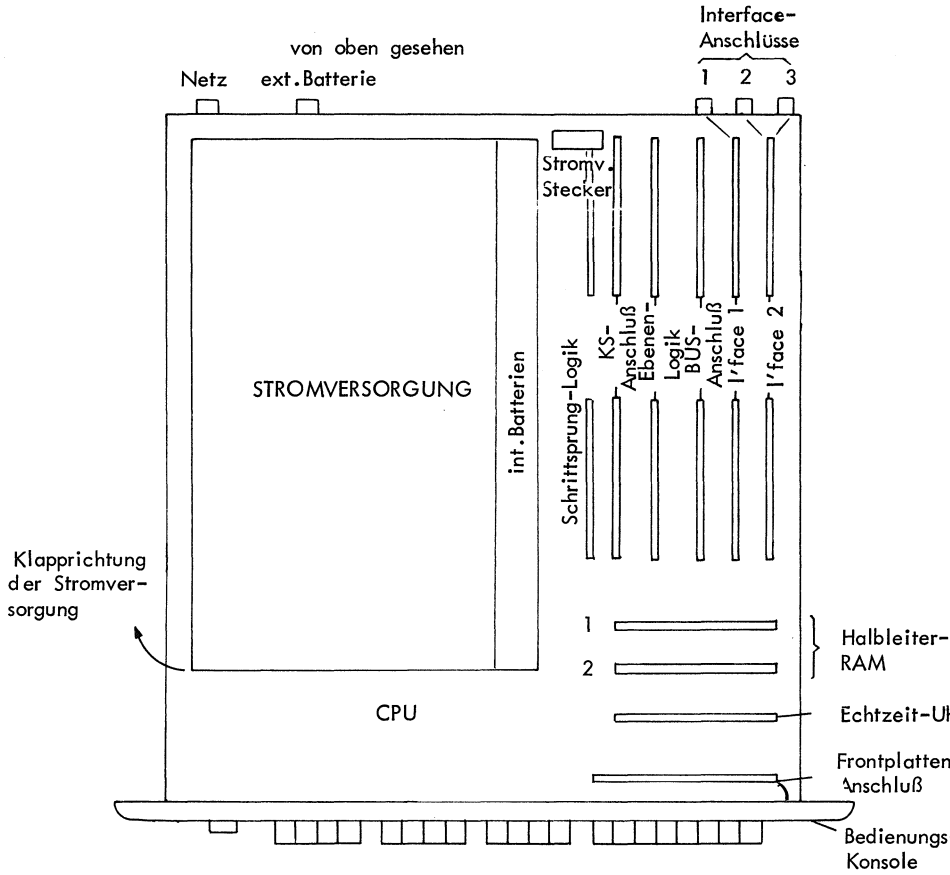
# Aufbau

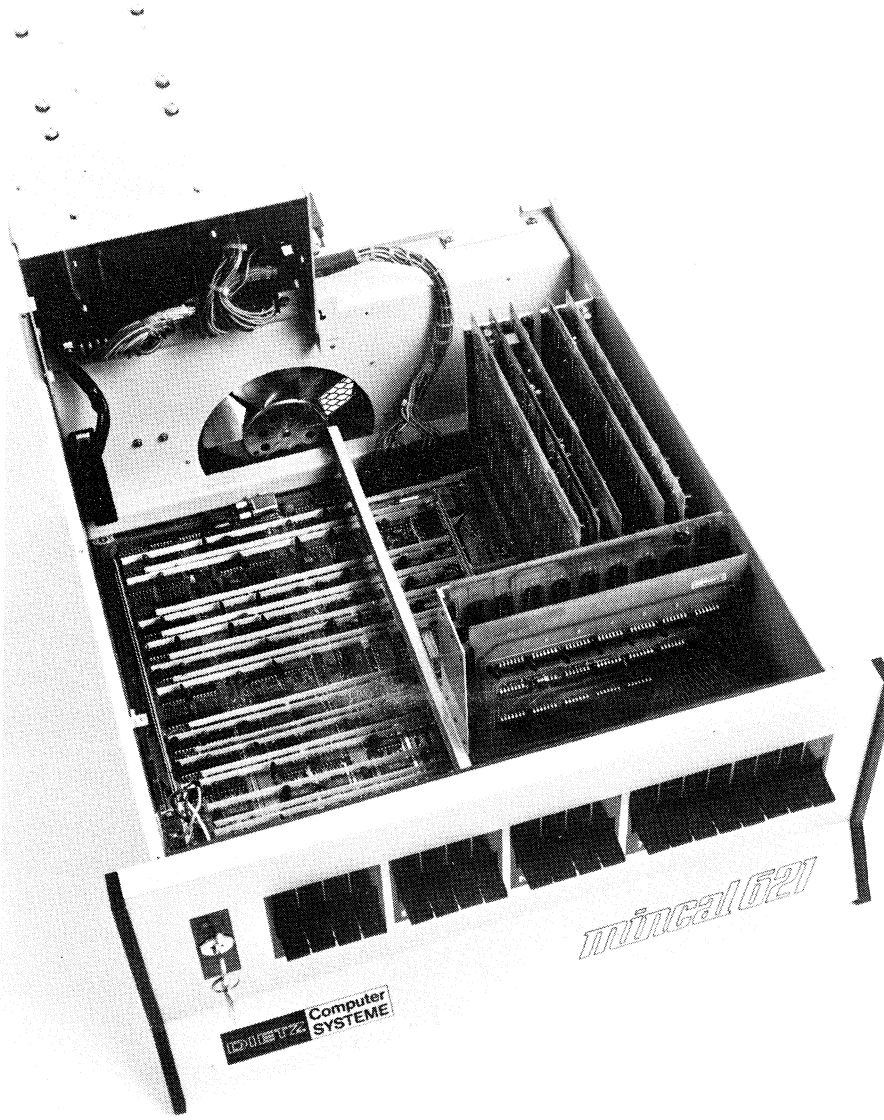
Der DIETZ 621 besteht aus einem geschlossenen Gehäuse, das vorne von der Bedienungs-Konsole (oder einer Blindplatte) abgeschlossen wird. Hinten befindet sich der Netzanschlußstecker. Hier werden auch die Kabel eingebauter Interfaces und das BUS-Kabel herausgeführt.

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 die Uhr, für die RAMs, die Ebenenlogik, die Schrittsprung-Logik, die Interfaces und den BUS-Anschluß enthält.

Über der CPU befindet sich die hochklappbare Stromversorgung.






DIETZ 621

mit aufgeklappter Stromversorgung

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

Auf der Leiterseite sind die wichtigsten Meßpunkte durch Beschriftung gekennzeichnet.

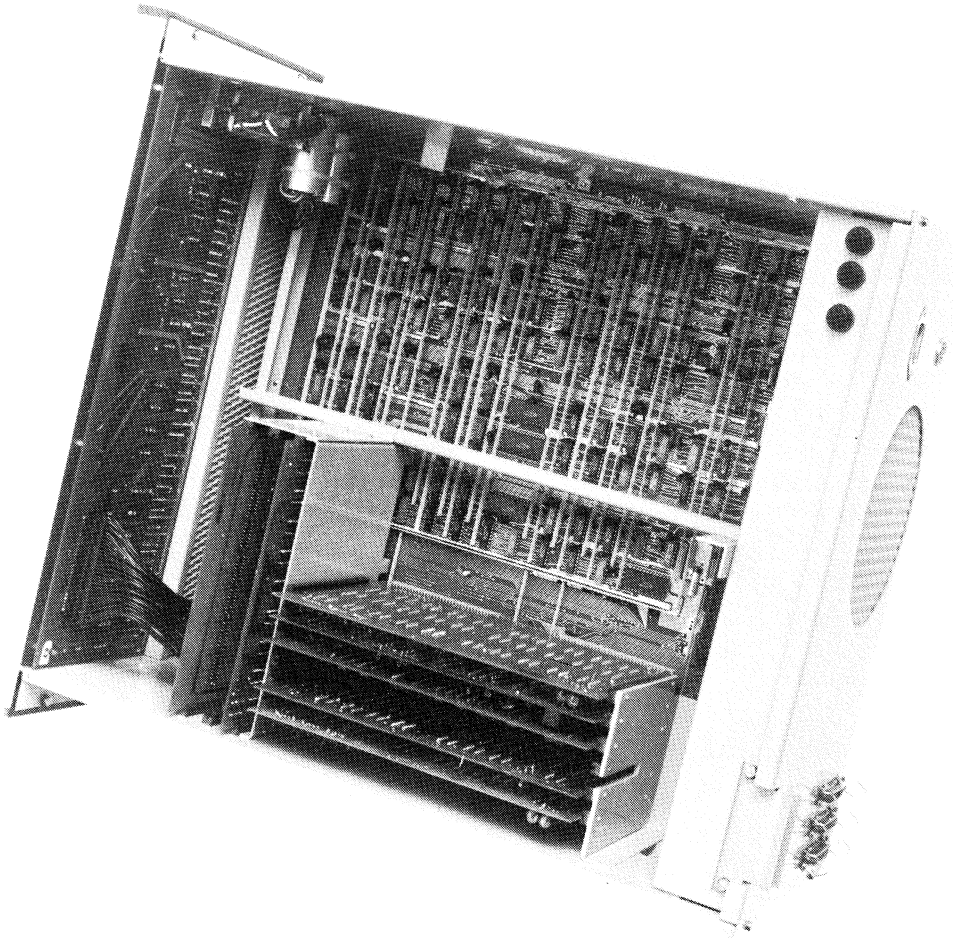
Bei der Inbetriebnahme des Rechners sind folgende Meßpunkte auf ihre Sollwerte gegen den Massemeßpunkt ("⊥" Telefonbuchse) zu überprüfen:

| Bezeichnung | Meßwert | Toleranz |
|---|---|---|
| +Z | +5 V | + 2 % |
| +T | +12 V | ∓ 5 % |
| +R | +12 V | ∓ 5 % |
| -R | -12 V | ∓ 5 % |
| $+Z_B$ | + 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 rechts nach links folgende Reihenfolge:

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

* Diese Potentiometer dürfen nicht verstellt werden!



Blick in die Zentraleinheit
(Stromversorgung entfernt)

Folgende Spannungen werden auf Überspannung durch Schutzschalter überwacht:

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

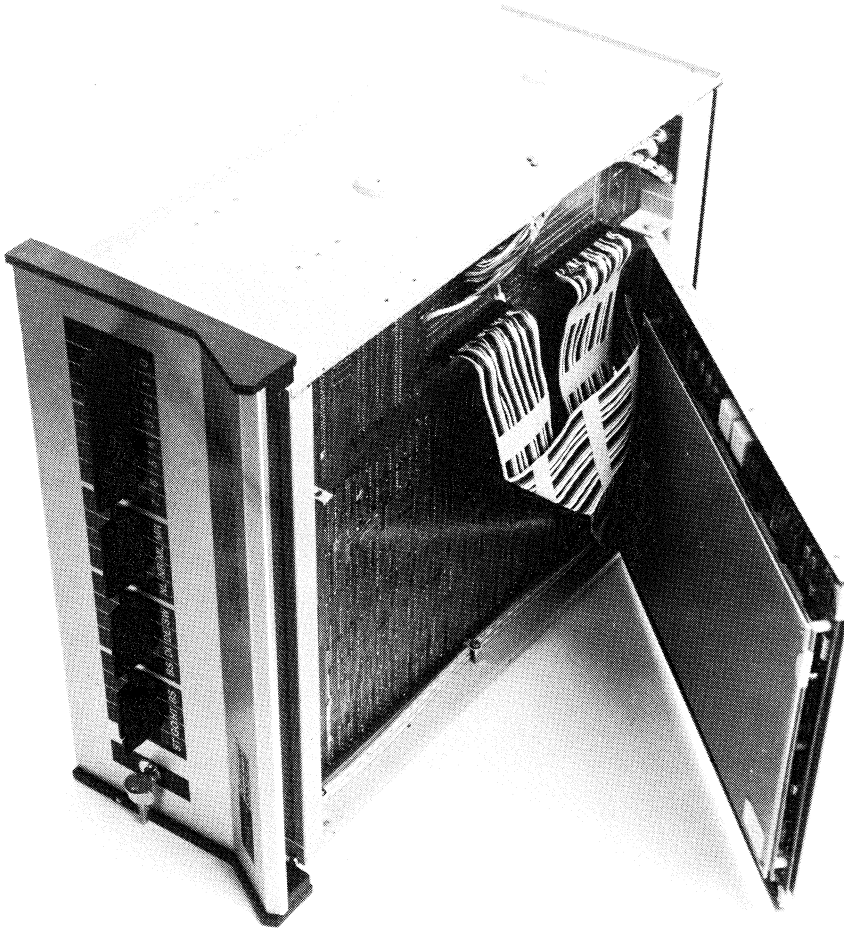
Netzsicherungen:

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

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

Die zweite Sicherung ist für den Transformator Tr.2 (-R; +B; +H; -H).

Die rechte Sicherung ist zum Schutz der Batterien vorgesehen.



DIETZ 621

mit ausgeklapptem Kernspeicher

# 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.

Durch Rechtsdrehen des Schlüsselschalters wird der Computer eingeschaltet. Die Lampe über dem Schalter leuchtet, und der Computer ist betriebsbereit. Über die Taste ST oder über einen externen Interrupt kann der Computer gestartet werden, wenn ein entsprechendes Programm im Speicher des Computers gespeichert ist.

Will man ein Programm in den Speicher einlesen, so kann über den eingebauten Bootstrap ein Ladeprogramm (Lader) eingelesen werden, mit dem dann ein beliebiger Hexa-Lochstreifen an beliebige Stellen des Kernspeichers geladen werden kann.

Bedienung Bootstrap: Schalter BS einlegen. Lader-Lochstreifen (im RUBOUT-Bereich) in Teletype-Leser oder in schnellen Leser einlegen (im letzteren Falle zusätzlich Schalter 4 einlegen). Taste STA betätigen; jetzt wird der Lader in die Plätze 'Ø6 bis 'FF des RAMs eingelesen.

Bedienung Lader: Schalter BS und 4 in Normalstellung bringen. Einzulesenden Hexa-Lochstreifen (im RUBOUT-Bereich) in Teletype-Leser oder schnellen Leser einlegen. Taste STA betätigen. Teletype führt Wagenrücklauf aus und druckt # aus. Nun

| | |
|---|---|
| aaaa-bbbb ISH | (bei Teletype-Leser) oder |
| aaaa-bbbb IFH | (bei schnellem Leser) |
| aaaaS | (Start bei Adresse aaaa) |

aaaa ist die Anfangs-, bbbb die Endadresse des Kernspeicherbereichs, in den der Streifen gelesen werden soll (4-stellige Hexazahlen). Der Streifen wird eingelesen. Am Ende wird # ausgedruckt. Danach kann der Vorgang mit weiteren Streifen wiederholt werden (neuen Streifen einlegen und o.a. Eingaben über den Teletype machen).

Der Lader prüft den zu ladenden Streifen und bricht bei einem Fehler den Vorgang mit der Nachricht ERR ab.

Hat man einen Lochstreifen, der einen Lader als Vorspann enthält, vereinfacht sich die Bedienung:
Schalter BS einlegen. Lochstreifen (im RUBOUT-Bereich) in Teletype-Leser oder in schnellen Leser einlegen (im letzteren Falle zusätzlich Schalter 4 einlegen). Taste STA betätigen; jetzt wird der Lochstreifen eingelesen. Wenn der Lesevorgang aufhört, Schalter BS und 4 in Normalstellung bringen und erneut Taste STA betätigen. Nun wird der gesamte Lochstreifen eingelesen.

Die Schalter und Tasten der Bedienungskonsole haben im einzelnen folgende Funktionen:

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 und durch NR und NL die laufende Ebene 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 (Ausnahme: Switch-Register).

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

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.

# Technische Daten

| | |
|---|---|
| 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 256mal ausführbar) |
| Arbeitsspeicher: | Halbleiter-RAM 0.25 K bis 4 Kbyte<br>Zugriffszeit 200 ns, Vollzyklus 400 ns<br>auf Wunsch batteriegepuffert<br>als Register-, Daten- und Programmspeicher |
| Hauptspeicher: | Kernspeicher 4 Kbyte bzw. 8 K, 16 K oder 32 Kbyte<br>Zugriffszeit 400 bzw. 300 ns, Vollzyklus 1 us bzw. 650 ns<br>oder<br>Festspeicher (RPROM) 0.25 K bis 8 Kbyte<br>extern auf 80 Kbyte erweiterbar |
| Technologie: | Integrierte Schaltkreise (TTL-MSI) |
| Instruktionen: | 5 Steuerbefehle<br>1 Mehrfach-Ausführungsbefehl<br>2 Zustandsabfragebefehle<br>4 Schiebebefehle<br>32 bedingte Sprungbefehle<br>9 BUS-bezogene Befehle mit Register-, relativer, absoluter oder indirekter sowie indizierter Adressierung<br>6 Konstanten-Befehle |
| Instruktionslänge: | 1...5 byte je nach Befehlstyp |
| Operationsdauer: | min. 1.9 us; max. 8.3 us |
| Arbeitsregister: | max. 254 sowie 1 fester Akku je Ebene (1...254 byte lang) |
| Indexregister: | max. 127 je Ebene (1 oder 2 byte lang) |
| Ebenen: | 2 oder 16 Programmebenen mit getrennten Registern und hierarchischer Priorität |
| Interrupts: | Durch Wechsel der Programmebene bei Ende jeder Operation möglich |

| | |
|---|---|
| Universal-BUS: | Standard-Schnittstelle mit 8 bit-Daten-Ein/Ausgang, 16 bit-Adreßausgang, Ebenen-Ausgang und Ebenenstart-Eingang für den Anschluß von externen Speichern und der Peripherie, ermöglicht programm- und fremdgesteuerten Datenverkehr |
| Bedienungskonsole: | Option |
| Echtzeit-Uhr: | Option |
| Interfaces: | Option (Raum für 2 Einkarten-Interfaces) |
| Netzanschluß: | 220 V $\pm$10 % 50 Hz einphasig<br>Leistungsaufnahme ca. 400 VA (4k KS)<br>ca. 675 VA (8/16k KS) |
| Größe: | 19"-Einschub, allseitig geschlossen, zwangsbelüftet<br>5 Einheiten hoch (ca. 225 mm)<br>525 mm tief |
| Gewicht: | ca. 35 kg |

## GRUNDAUSRÜSTUNG

Der MINCAL 621 ist ein System-Computer mit modularem, der jeweiligen Aufgabe anzupassendem Aufbau. Zur Grundausrüstung der Zentraleinheit gehören:

| | |
|---|---|
| Prozessor: | Vollständiger Prozessor für alle Maschinenbefehle |
| Stromversorgung: | Netzteil für Prozessor, Kernspeicher und Optionen |
| Arbeitsspeicher: | 0.25 Kbyte RAM |
| Programmebenen: | Logik für 16 Programmebenen |
| BUS-Anschluß: | Steckerplatz für Universal-BUS-Anschluß |
| Weitere Anschlüsse: | Netzstecker, Stecker für externe RAM-Pufferung und 3 Interface-Stecker in Gehäuse-Rückseite |
| Frontplatte: | Einfache Frontplatte ohne Bedienungs- und Anzeigeelemente |

## OPTIONEN

Die Zentraleinheit MINCAL 621 kann darüberhinaus folgende Optionen enthalten:

| | |
|---|---|
| Arbeitsspeicher: | Erweiterung um jeweils 0.25 Kbyte bis insgesamt 4 Kbyte RAM-Kapazität |
| Pufferung: | Pufferung der RAMs bei Netzausfall durch 2 eingebaute Batterien (bis 1 Kbyte RAM; Ladestromversorgung ist im Netzteil vorgesehen.<br>Versorgungsdauer: 10 h (0.25 K); 5 h (0.5 K); 2 h (1 Kbyte RAM).<br>Bei 2 Batterien verdoppeln sich die Zeiten. |
| Hauptspeicher: | Kernspeicher 4 Kbyte (400 ns Zugriffszeit, 1 us Vollzyklus) mit Netzausfallschutz-Logik, oder<br><br>Kernspeicher 8 Kbyte (300 ns Zugriffszeit, 650 ns Vollzyklus) mit Netzausfallschutz- und Parity-Logik, oder<br><br>Kernspeicher 16 Kbyte (300 ns Zugriffszeit, 650 ns Vollzyklus) mit Netzausfallschutz- und Parity-Logik, oder<br><br>Kernspeicher 32 Kbyte; sonst wie 16 Kbyte, oder |

| | |
|---|---|
| | Reprogrammierbarer Festspeicher (ca. 1 us Zugriffszeit)<br>Kapazität max. 8 Kbyte, in Stufen von 0.25 Kbyte ausbaufähig<br>mit Netzausfallschutz-Logik |
| Echtzeit-Uhr: | Untersetzer (vom Prozessor-Quarz gesteuert)<br>löst in Abständen von 1, 10, 100 oder 1000 ms (fest eingestellt) einen Ebenen-Start aus (CNP-Ebene) |
| Bedienungskonsole: | Frontplatte mit Bedienungs- und Anzeigeelementen<br>(anstelle der einfachen Frontplatte) |
| Interfaces: | Einbauraum und Steckplätze für 2 Einkarten-Interfaces sind vorgesehen |

MODIFIKATIONEN

Folgende Parameter werden vom Hersteller oder Benutzer festgelegt bzw. sind nachträglich modifizierbar:

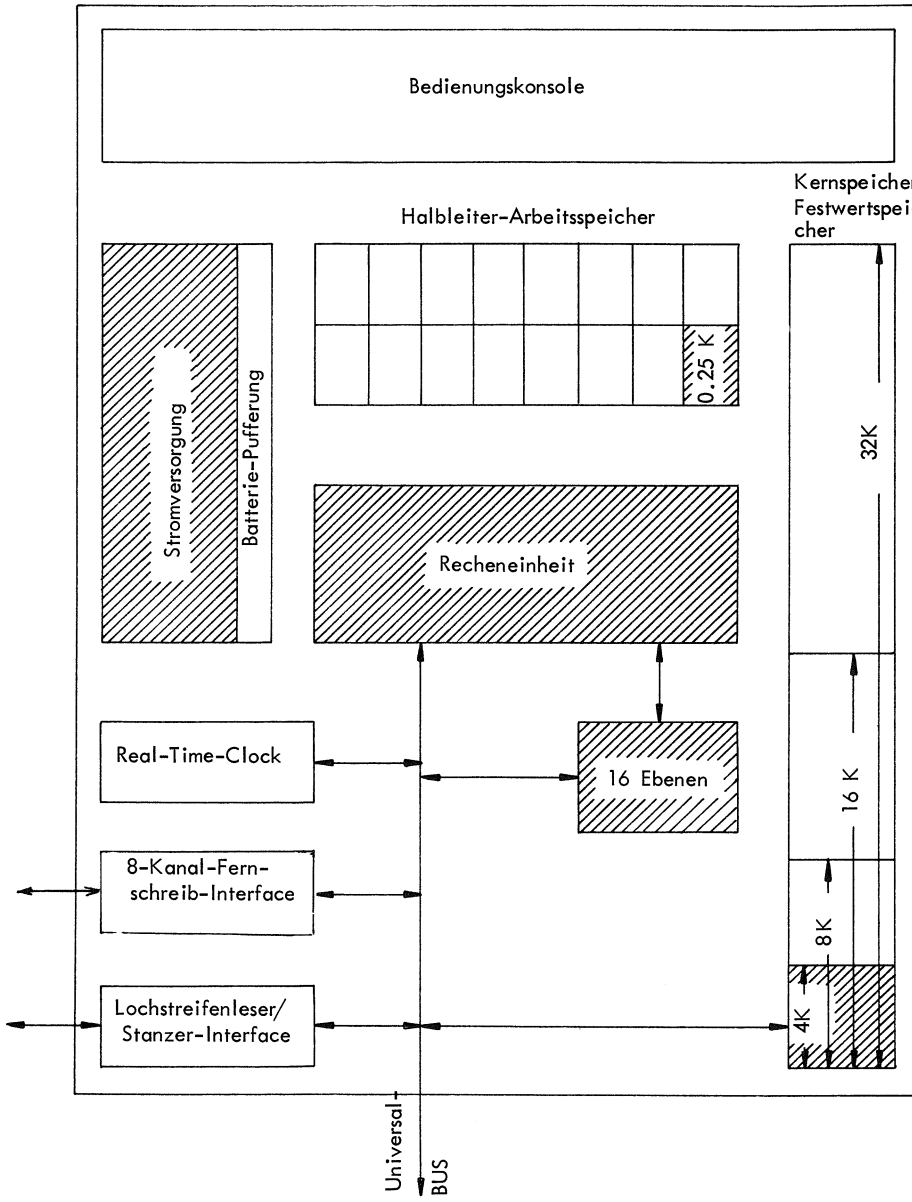
| | |
|---|---|
| Pool-Größe: | Festlegung auf 16, 32, 64, 128 oder 256 byte pro Ebene<br>(auf Ebenenlogik-Baustein) |
| CNP-Ebene: | Zuordnung des Starts für Fehlermeldungen und Echtzeituhr zu einer bestimmten Programmebene (CNP-Ebene)<br>(auf Ebenenlogik-Baustein) |
| | Festlegung des Intervalls für den Echtzeituhr-Start auf 1, 10, 100 oder 1000 ms<br>(auf Clock-Baustein) |
| Ebene Interface 2: | Rückmeldung des Interfaces auf beliebigen Ebenen-Start Sx, statt auf SØ<br>(auf CPU-Grundplatte). |

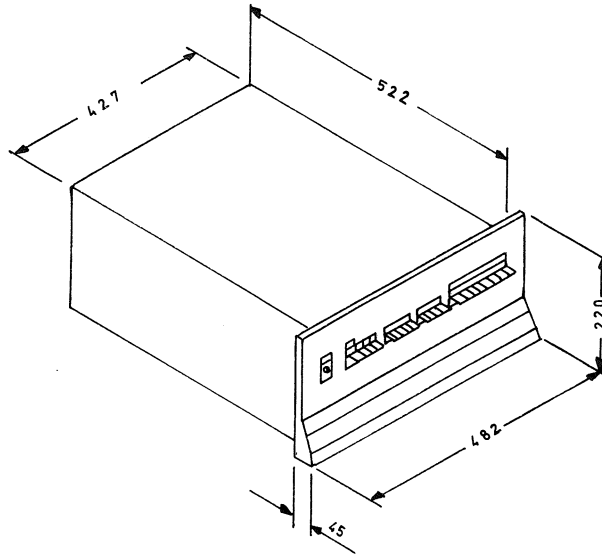
## ANSCHLUSS- UND UMGEBUNGSBEDINGUNGEN

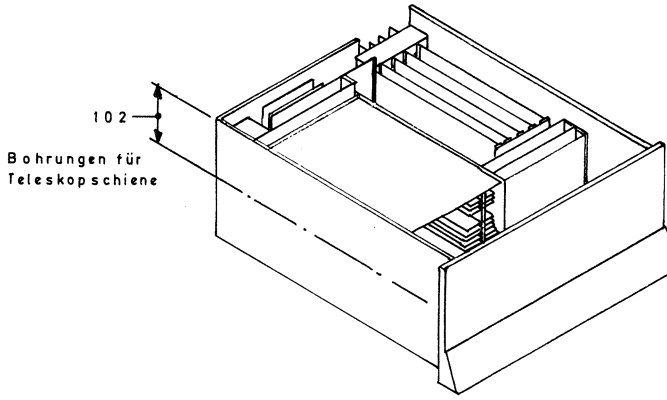
| | |
|---|---|
| Spannung | 220 V Wechselspannung, einphasig mit Schutzleiter |
| Spannungsschwankung | $\pm$10 % |
| Frequenz | 50 Hz, +5 %, -2 % |
| Klirrfaktor | $\leq$ 6 % |
| Zulässige Kurzzeiteinbrüche | $\leq$ 4 ms (Abstand $\geq$ 1 s) |
| Umgebungstemperatur | 0°C bis +50°C |
| Luftfeuchtigkeit | 0 bis 95 % ohne Kondensation |
| Staubgehalt der Luft | Filterklasse B |

Diese Angaben gelten nicht für alle Peripheriegeräte bzw. Erweiterungen des Systems. Je nach Anlagenkonfiguration sind andere Anschluß- und Umgebungsbedingungen einzuhalten.

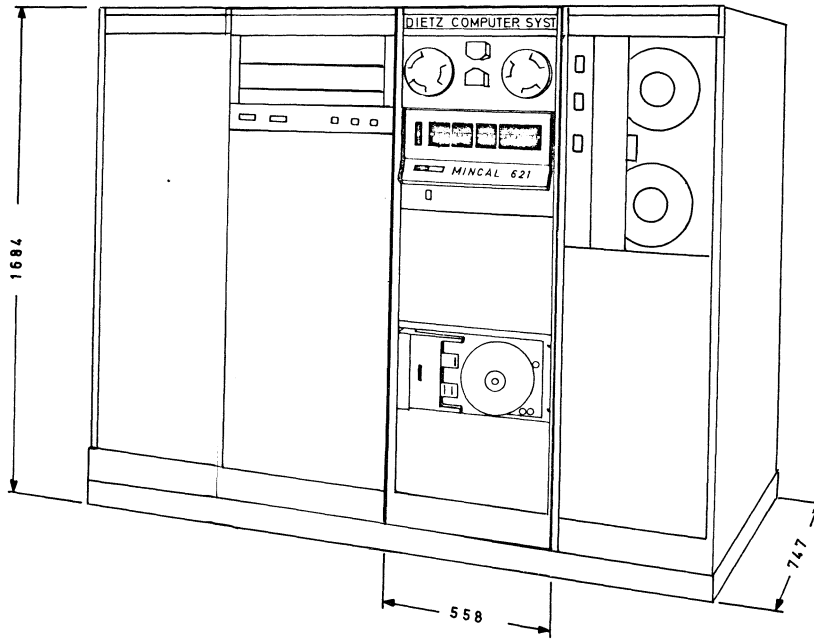
(Grundausrüstung schraffiert)






Abmessungen DIETZ 621




Schrankaufbau und -abmessungen des
Computer-Systems DIETZ 621

# Peripherie-System

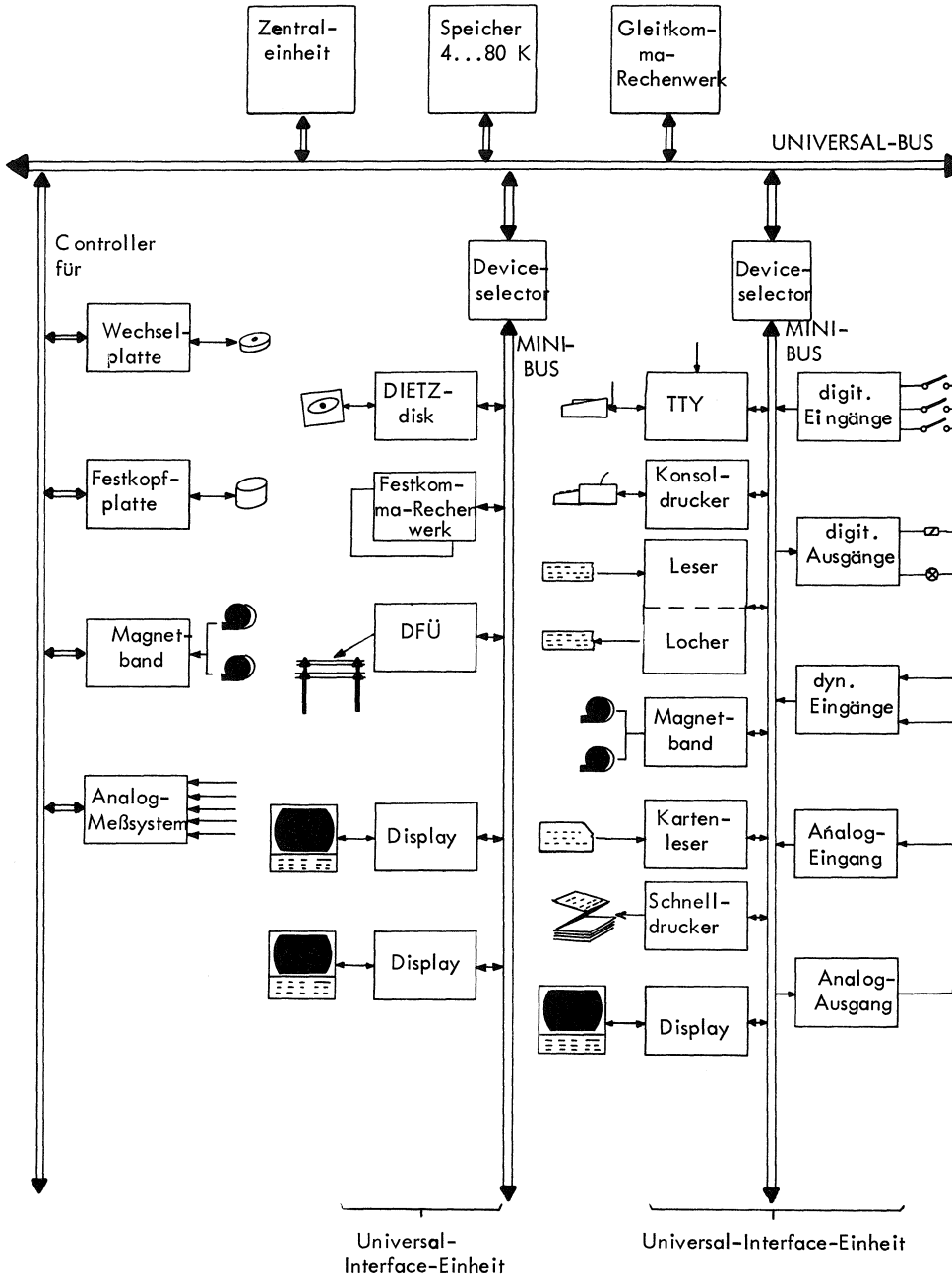
Die Zentraleinheit kann durch Anschluß von zusätzlichen Baugruppen zu einem Computer-System erweitert werden.

All diese Baugruppen werden über den UNIVERSAL-BUS mit der Zentraleinheit verbunden. Ausgenommen Erweiterungen der Zentraleinheit (Kernspeicher, Gleitkomma-Rechenwerk) gibt es zwei Einschübe, über die die Peripherie angeschlossen werden kann:

- UNIVERSAL-INTERFACE-EINHEIT
  19"-Einschub, 6 Einheiten hoch
  zur Aufnahme von Einkarten-Interfaces, die im programmgesteuerten Betrieb bedient werden

- AKTIVES ELEMENT
  19"-Einschub, 3 Einheiten hoch
  zur Aufnahme von Controllern und schneller Meßsystemen, die in direktem Speicherzugriff (DMA) betrieben werden.

Die Ankopplung der UNIVERSAL-INTERFACE-EINHEIT an den UNIVERSAL-BUS erfolgt über einen DEVICE-SELECTOR, der aus dem asynchronen UNIVERSAL-BUS den synchronen, reduzierten MINI-BUS erzeugt, an den die Interfaces angeschlossen werden.

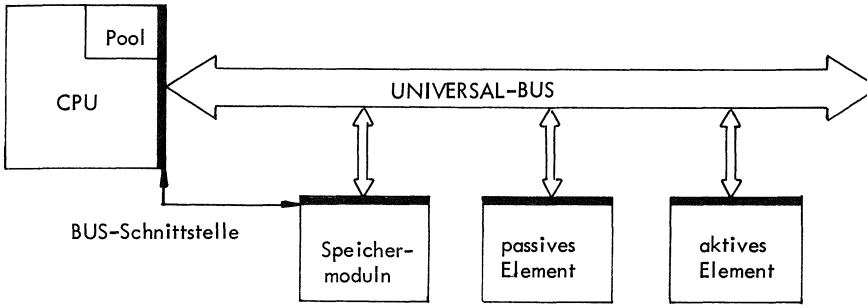

Zentral-einheit
Speicher 4...80 K
Gleitkom-ma-Rechenwerk
UNIVERSAL-BUS
Controller für
Device-selector
Device-selector
MINI-BUS
MINI-BUS
Wechsel-platte
Festkopf-platte
Magnet-band
Analog-Meßsystem
DIETZ-disk
Festkom-ma-Rechen-werk
DFÜ
Display
Display
TTY
Konsol-drucker
Leser
Locher
Magnet-band
Karten-leser
Schnell-drucker
Display
digit. Eingänge
digit. Ausgänge
dyn. Eingänge
Analog-Eingang
Analog-Ausgang
Universal-Interface-Einheit
Universal-Interface-Einheit

# Universal-BUS

ALLGEMEINE BESCHREIBUNG

Der Universal-BUS (kurz: BUS) des DIETZ 621 hat die Aufgabe, alle Komponenten eines Systems miteinander zu verbinden und einen bit-parallelen Datenaustausch zwischen ihnen zu ermöglichen. Die Systemkomponenten können sein:

eine CPU (Central Processor Unit)
Speichermoduln
passive Elemente (z.B. Interfaces)
aktive Elemente (z.B. Controller).




Der Halbleiterspeicher des DIETZ 621, der Pool, gehört nicht zu den vom BUS erreichbaren Elementen. Der Inhalt des Pool kann nur von der CPU selbst unter Programmkontrolle gelesen oder verändert werden.

Als passive Elemente sind die Komponenten definiert, die den BUS nicht selbsttätig belegen und für einen Datenaustausch mit einem anderen Element benutzen können; Speichermoduln sind somit grundsätzlich als passive Elemente anzusehen.

Aktive Elemente sind die Komponenten, die selbsttätig den BUS belegen und einen Datentransfer durchführen können; CPU oder Controller sind demnach aktive Elemente.

Mit Ausnahme der CPU sind aktive Elemente nicht ausschließlich aktiv; sie verhalten sich genau wie passive Elemente, wenn sie z.B. von der CPU Arbeitsanweisungen erhalten.

Ein Datentransfer kann nur zwischen einem aktiven und einem passiven Element erfolgen, wobei das aktive Element die Steuerung des Transfers übernimmt und das passive Element lediglich den vollzogenen Transfer quittiert.

Durch dieses Quittungsprinzip wird erreicht, daß der Datenaustausch unabhängig von der physikalischen Länge des BUS und von der Reaktionszeit der Elemente durchgeführt werden kann. Die max. Transferrate wird jedoch sowohl von der Länge des BUS bestimmt als auch von der Reaktionszeit; im günstigsten Fall kann alle 650 ns ein 8-bit-Wort übertragen werden, was 1.5 Millionen Bytes pro Sekunde entspricht.

Sind mehrere aktive Elemente an einen BUS angeschlossen, dann sorgt eine Prioritätsstruktur dafür, daß die BUS-Belegungen zeitlich gestaffelt erfolgen. Jedem aktiven Element wird deshalb eine Prioritätsstufe zugeteilt. Wollen zwei aktive Elemente gleichzeitig die BUS-Kontrolle übernehmen, so bekommt sie das Element mit der höheren Priorität zuerst.

Die Schnittstellen aller Elemente sind gleichartig aufgebaut, so daß ein aktives Element mit Speichern und Interfaces unter Ausnutzung der gleichen Signale korrespondieren kann.

Speziell für die CPU bedeutet das, daß alle Maschinenbefehle für den Speicherverkehr auch für den Verkehr mit Interfaces verwendet werden können (s. auch BUS-bezogene Befehle!). Interface-Register können somit genau so flexibel behandelt werden wie Kerspeicherzellen.

Der Universal-BUS des DIETZ 621 ist bidirektional aufgebaut, d.h. daß an der BUS-Schnittstelle eines Elements kein Unterschied zwischen Eingangs- und Ausgangsleitungen besteht. Die Richtung des Datentransfers wird durch ein spezielles Signal bestimmt, das Richtungskennzeichen RK.

Ein BUS hat den Vorteil, sowohl den programmgesteuerten Datentransfer als auch den direkten Kernspeicherzugriff (Direct Memory Access oder DMA) ohne zusätzlichen Aufwand zuzulassen.

Der Universal-BUS bietet zusätzlich die Möglichkeit, die Mehrebenenstruktur des DIETZ 621 innerhalb der Peripherie auszunutzen, sei es durch Bindung eines Interfaces an eine bestimmte Ebene oder durch Rückmeldungen von Interfaces in Form von Startsignalen (Interrupts).

## BUS-SIGNALE

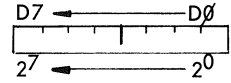
Der Universal-BUS des DIETZ 621 umfaßt insgesamt 63 Signalleitungen, deren Namen und Bedeutungen im folgenden Abschnitt erläutert sind.

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

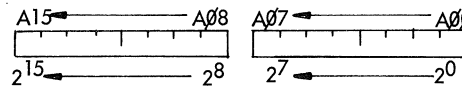
0 V...+0.5 V ≙ logisch 1 ≙ Signal vorhanden

+3 V...+5 V ≙ logisch Ø ≙ Kein Signal

DØ...D7 8 bidirektionale Datenleitungen zur bit-parallelen (byte-seriellen) Übertragung eines Datums. Zur Datensicherung durch ein Parity-Bit steht bei Bedarf eine neunte Datenleitung D8 zur Verfügung.
Die Zuordnung von Datenleitung und Wertigkeit des Datenbits zeigt folgende Abbildung:



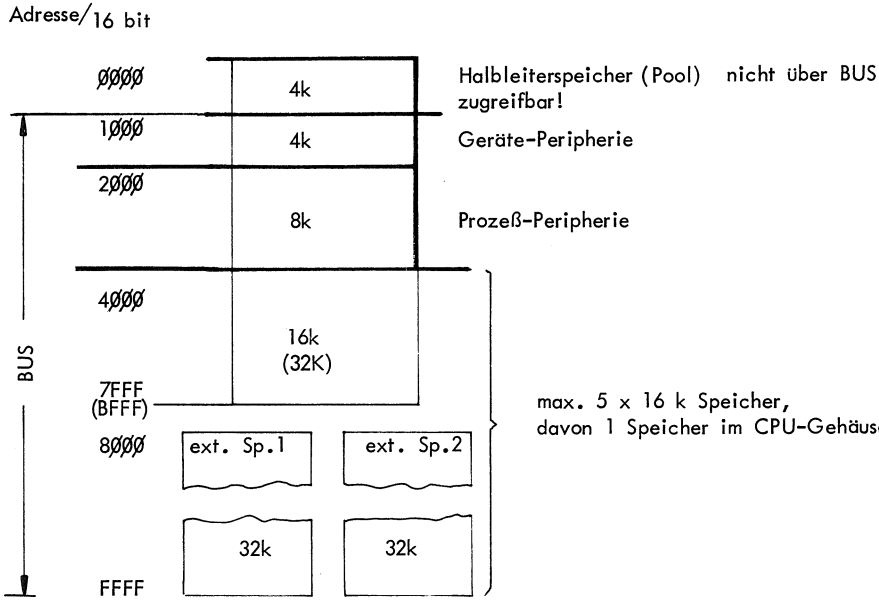

AØØ...A15 16 bidirektionale Adreßleitungen zur Anwahl von Interfaces und Speichern durch ein aktives Element:




Mit einem 16-bit-Adreßwort sind

$$64\ k = 65536_{10} = FFFF_{16}\ \text{Einzeladressen}$$

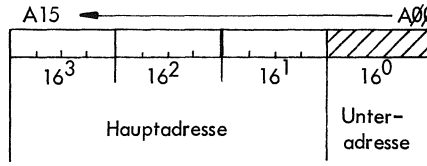
anwählbar, die nach folgendem Schema aufgeteilt sind:



Ein Adreßwort kleiner als $1ØØØ_{16}$ wählt eine Zelle des Halbleiterspeichers an, ein Adreßwort größer oder gleich $1ØØØ_{16}$ aktiviert den BUS.

Der Adreßbereich von $1ØØØ_{16}$ bis $3FFF_{16}$ einschließlich ist der Peripherie vorbehalten; Adressen größer als $3FFF_{16}$ sind den Speichern zugeordnet.

Ein Adreßwort für den Peripheriebereich ist unterteilt in die Hauptadresse und die Unteradresse:




Die Hauptadresse ist ein 12-bit-Wort und wählt das Interface an; die gleichzeitig angebotene Unteradresse ist ein 4-bit-Wort und wählt innerhalb des Interfaces eine bestimmte Baugruppe an, meist ein Register.

Diese Aufteilung des 16-bit-Adreßworts erlaubt den Anschluß von $768_{10}$ bzw. $3ØØ_{16}$ Interfaces mit je $16_{10}$ bzw. $F_{16}$ anwählbaren Unteradressen.

RK Das Richtungskennzeichen gibt an, in welcher Richtung ein Datentransfer ausgeführt werden soll. Bei RK = log 1 soll der Datentransfer vom aktiven zum passiven Element verlaufen. Dieser Transfer kann als "Schreiben" oder "Ausgabe" bezeichnet werden.

Bei RK = log Ø ist die Übertragungsrichtung vom passiven zum aktiven Element; es handelt sich um "Lesen" oder "Eingeben".

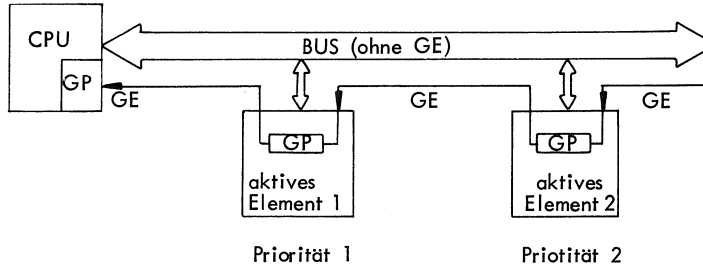
BE Das Signal "BELEGT" wird von einem aktiven Element auf die bidirektionale Signalleitung geschaltet und zeigt an, daß ein Datentransfer abläuft, der BUS also nicht frei für ein anderes aktives Element ist.

FE Das Signal "FERTIG" wird während eines Datentransfers von einem angewählten passiven Element erzeugt und hat die Funktion einer Quittung: dem aktiven Element wird mitgeteilt, daß der Datentransfer "fertig" ist.
Die Signalleitung FE ist bidirektional ausgelegt.

GE Das Signal "GEWUENSCHT" wird von einem aktiven Element erzeugt, wenn es einen Datentransfer ausführen möchte, es also die BUS-Kontrolle wünscht. Alle aktiven Elemente müssen vor einer Belegung einen Belegungswunsch anmelden. Die einzige Ausnahme bildet die CPU, die ohne Anmeldung den BUS belegen kann.

Meldet ein anderes aktives Element einen Belegungswunsch an, so kann die CPU den BUS nicht erneut belegen. Die zur Zeit des Belegungswunsches bestehende Belegung wird regulär beendet.

Die Prioritäten der einzelnen aktiven Elemente sind durch die Leitungsführung des Signals GE vorgegeben:




Wie in obiger Abbildung gezeigt, steigt die Priorität von links nach rechts. Die mit "GP" bezeichneten Funktionsgruppen stellen die "Gewünscht Prioritäten" fest. So darf z.B. das aktive Element 1 nur dann ein "Gewünscht" anmelden, wenn das Element 2 oder ein Element noch höherer Priorität kein "Gewünscht" anmeldet. Ein bereits vorhandener Belegungswunsch wird durch ein "Gewünscht höherer Priorität" unterbrochen.

N Das Signal N ist die zentrale Nullstellung. Sie kommt von der CPU (Taste des Bedienungsfeldes bzw. automatische Nullstellung bei wiederkehrendem Netz).

LØØ...L15 16 Levelleitungen, die decodiert anzeigen, in welcher der möglichen 16 Ebenen der DIETZ 621 arbeitet. Die Levelinformation wird benötigt bei der Anwahl ebenengebundener Interfaces. Bei Ausnutzung der Ebenenbindung können somit die $768_{10}$ Hauptadressen des Peripheriebereichs 16-fach benutzt werden. Die Levelleitungen sind nicht bidirektional, sie gehen nur von der CPU aus.

S00...S15 16 Startleitungen (Interruptleitungen), über die durch ein Startsignal (Interrupt) eine der 16 Ebenen des DIETZ 621 gestartet werden kann.

Diese Leitungen sind nicht bidirektional, sie gehen nur zur CPU.

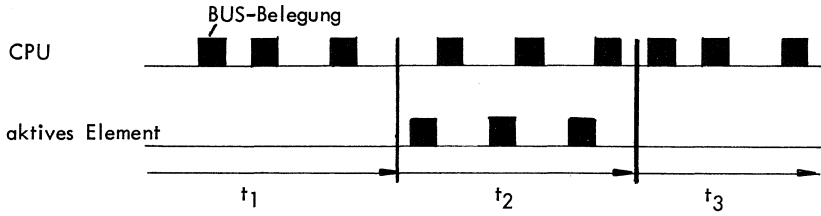
STPX 1 Startleitung, über die ein Parityfehler eines externen Elementes der CPU gemeldet werden kann. Der Interrupt STPX startet die CNP-Ebene.

## DATENTRANSFER

Ein Datentransfer findet grundsätzlich zwischen einem aktiven und einem passiven Element statt, wobei das aktive Element den Transfer einleitet und steuert.

Ist die CPU das aktive Element, handelt es sich um eine programmgesteuerte Datenübertragung. Belegt ein anderes aktives Element den BUS und tauscht unabhängig von dem gerade laufenden Programm Daten mit dem Kernspeicher aus, liegt ein DMA (direkter Kernspeicherzugriff) vor.

Der Ablauf eines einzelnen Belegungsvorgangs ist in beiden Fällen gleich; Unterschiede bestehen lediglich in der Verfügbarkeit der CPU und des BUS.

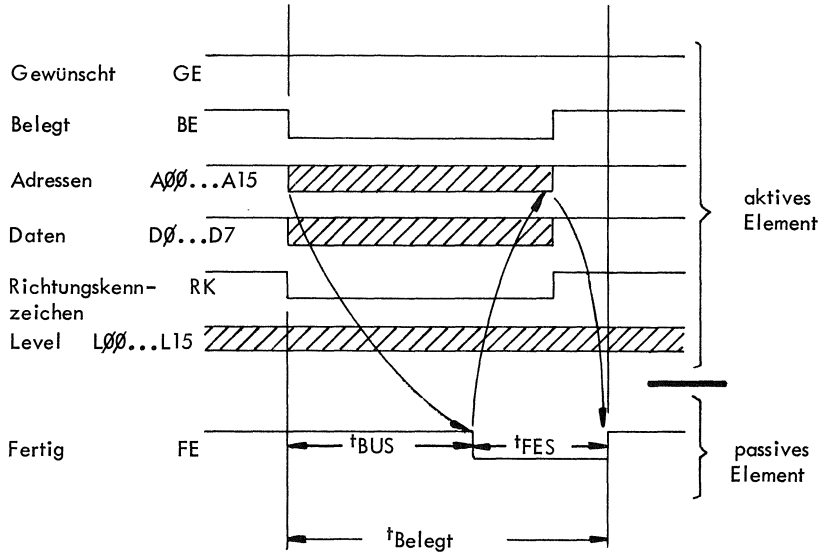



In der Zeit $t_1$ verarbeitet die CPU ein beliebiges Programm, in dessen Ablauf ein zunächst passives Element durch Übergabe von Arbeitsanweisungen von der CPU aus aktiviert wird, angedeutet durch die drei BUS-Belegungen der CPU.

Das aktive Element und die CPU arbeiten in der Zeit $t_2$ unabhängig voneinander und belegen abwechselnd den BUS, um entweder programmgesteuerte Datentransfers (CPU) oder DMA-Zyklen (aktives Element) auszuführen. Nach Ablauf von $t_2$ hat das aktive Element seine Arbeit beendet und verhält sich wieder rein passiv. Im Normalfall teilt das aktive Element der CPU das Ende der Aktiv-Phase durch einen Interrupt mit, der z.B. während $t_3$ vom Programm identifiziert und verarbeitet wird.

Beim Datentransfer wird zwischen "Schreiben" (Senden des aktiven Elements) und "Lesen" (Empfangen) unterschieden.

Schreibzyklus:




Da nur die CPU als aktives Element vorliegt, bleibt das Signal GE im Ruhezustand. Die CPU belegt den BUS durch das Signal BE und schaltet gleichzeitig Adressen, Daten und Richtungskennzeichen RK auf den BUS. Die Ebeneninformation LØØ...L15 wird unabhängig vom Belegungszustand des BUS von der CPU angeboten.

Nach Ablauf der Zeit $t_{BUS}$ übernimmt das durch die Adreß- und Levelinformation angewählte passive Element die angebotenen Daten und quittiert die Übernahme durch Aufschalten des Signals FE.

Wenn das aktive Element das Quittungssignal FE empfängt, werden mit dem BE-Signal die Daten, Adressen und auch das Richtungskennzeichen vom BUS genommen; das passive Element wird nicht mehr angewählt und schaltet seinerseits FE ab. Jetzt erst ist der BUS für eine erneute Belegung frei.

Die Gesamtzeit $t_{Belegt}$ eines solchen Belegungsvorganges ist die Summe der Zeiten $t_{BUS}$ und $t_{FES}$.

In die Zeit $t_{BUS}$ gehen ein:

a) Signallaufzeit des Kabels einschließlich Sende- und Empfangsschaltungen

b) Signallaufzeit der Adreßentschlüßlung des Interfaces

c) Einstellbare Beruhigungszeit zum Abwarten von Einschwingungsvorgängen und Unterdrücken von Störungen

d) Signallaufzeiten der Steuerlogik.

Bei einer BUS-Länge von ca. 150 cm liegt die Zeit $t_{BUS}$ im Bereich von 350 bis 400 ns.

Die Zeit $t_{FES}$ ist die Summe aus:

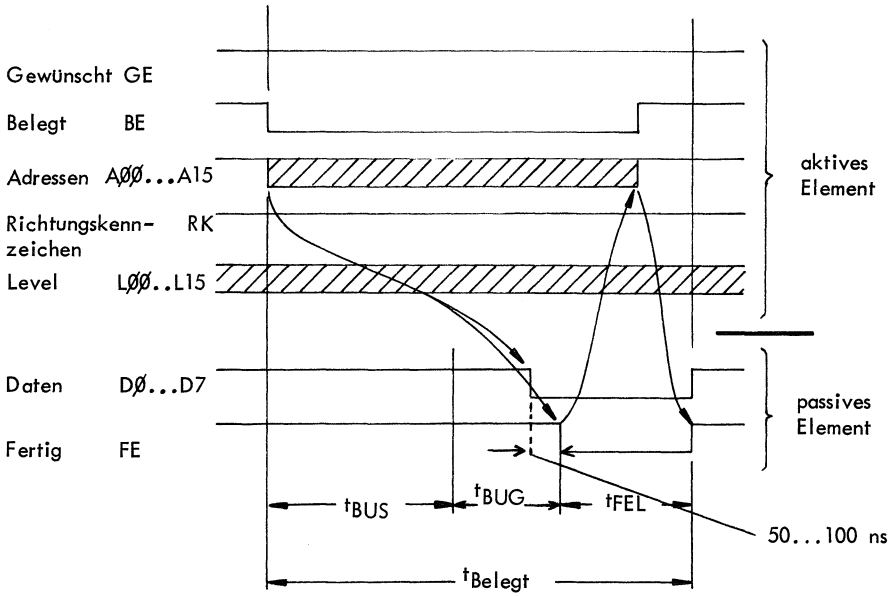
a) Signallaufzeiten von Kabel, Sende- und Empfangsschaltungen und Steuerlogik

b) Signallaufzeit der Adreßentschlüßlung des Interfaces.

Bei gleichen Voraussetzungen wie für $t_{BUS}$ liegt die Zeit $t_{FES}$ im Bereich von 250 bis 300 ns, so daß die Gesamtdauer der Belegung $t_{Belegt}$ = 600...700 ns beträgt.

Bei längeren BUS-Kabeln ist die Belegungsdauer größer; als Richtwert können für je 2 m Kabel ca. 100 ns Zeitzuschlag angenommen werden.

Lesezyklus:



Das aktive Element belegt den BUS und schaltet die Adressen und das Richtungskennzeichen auf. Nach Ablauf der Zeit $t_{BUS}$ leitet das durch die Adreß- und Levelinformation angewählte passive Element einen internen Lesezyklus ein, nach dessen Ablauf die Daten am BUS bereitgestellt werden. Das Quittungssignal FE wird verzögert auf den BUS geschaltet, um die Leseaufforderung erst dann zu quittieren, wenn die Daten sicher anstehen.

Die Zeit vom Aktivieren des passiven Elements bis zum Aufschalten des FE-Signals ist die Zugriffszeit $t_{ZUG}$. Sie beträgt beim Lesen einer Kernspeicherzelle ca. 400 ns.

Wenn das aktive Element das Quittungssignal FE erkannt hat, übernimmt es die angebotenen Daten und schaltet das BE-Signal, das Richtungskennzeichen und die Adressen vom BUS; als Antwort darauf nimmt das passive Element das FE-Signal und die Daten vom BUS; der BUS ist frei für eine erneute Belegung. Das Signal FE steht beim Lesezyklus ca. 100 ns länger an als beim Schreibzyklus, da das aktive Element vor dem Abschalten der Adressen und des Belegt-Signals die Daten übernehmen muß.

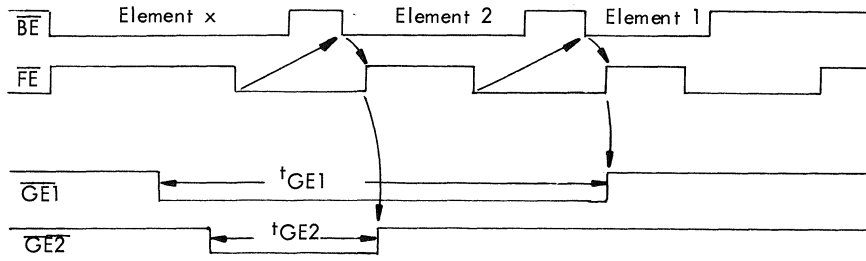
Somit ergibt sich bei einer BUS-Länge von 150 cm eine Gesamt-Belegungsdauer beim Lesevorgang von

$$t_{BELEGT} = t_{BUS} + t_{ZUG} + t_{FEL}$$
$$t_{BELEGT} = (350...400) + (400) + (350...400)$$
$$t_{BELEGT} = 700...1200 \text{ ns}$$

Aus der Darstellung der Belegungsvorgänge ersieht man, daß bei einem fehlenden Quittungssignal (FE-Signal) der BUS nicht wieder freigegeben würde; der BUS und damit das gesamte System wäre blockiert. Um dies zu vermeiden, ist auf dem BUS-Ausgangsbaustein der CPU eine Überwachungsschaltung (Watchdog) realisiert, die die Dauer der BUS-Belegungen laufend überprüft. Sobald auf ein BE-Signal nach spätestens 1 sec kein FE-Signal folgt, erzeugt die Watchdog selbst ein FE-Signal und einen Interrupt für die CNP-Ebene des Rechners. Dadurch wird zum einen der Belegungszustand des BUS beendet und zum anderen eine interpretierbare Fehlermeldung abgegeben.

## PRIORITÄTSSTEUERUNG

Beim Anschluß mehrerer aktiver Elemente an den BUS wird vor einem Belegungsvorgang das Element mit der höchsten Priorität von den "Gewünscht"-Prioritätssteuerungen ausgewählt. Wie im Kapitel "BUS-Signale" bereits dargestellt, werden diese Prioritätssteuerungen mit dem Signal GE (Gewünscht) gesteuert, mit dem ein aktives Element einen Belegungswunsch anmeldet. Der BUS wird von dem aktiven Element mit dem Signal BE belegt und mit FE vom passiven Element quittiert.



Die aktiven Elemente 1 und 2 wollen den BUS belegen, wobei das Element 2 die höhere Priorität hat. Die Signale GE1 und GE2 sind die Gewünscht-Signale dieser Elemente.

Wie aus der Abbildung ersichtlich, kann Element 1 erst dann den BUS belegen, wenn die Belegung von Element 2 beendet ist und wenn kein Belegungswunsch eines Elementes höherer Priorität mehr vorliegt.

Bevor ein aktives Element den BUS belegen kann, müssen demnach folgende Bedingungen erfüllt sein:

a) der BUS ist nicht belegt, d.h. sowohl BE als auch FE stehen nicht an,

b) das Signal GE muß mindestens die Zeit $t_{GE}$ ununterbrochen auf den BUS geschaltet sein.
Diese Bedingung entfällt für die CPU, da sie keinen Belegungswunsch anmeldet und den BUS immer dann belegt, wenn er frei ist.

Die Zeit $t_{GE}$ ist bestimmt durch die Laufzeiten des GE-Signals und beträgt für das von der CPU aus erste aktive Element mindestens 350 ns. Für jedes weitere aktive Element gilt ein Zuschlag von ca. 50 ns.

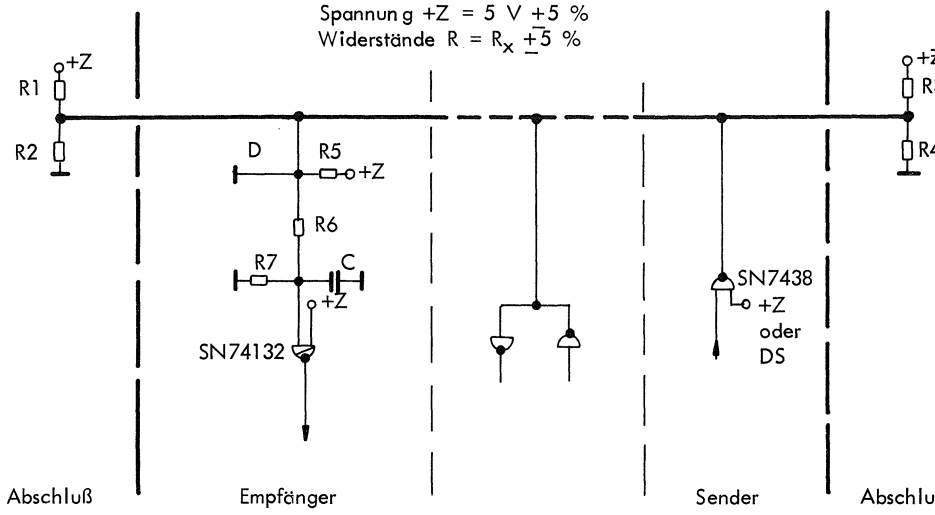
## HARDWARE DES UNIVERSAL-BUS

Der Universal-BUS ist realisiert durch verschiedene Kabeltypen, wobei das Kabel selbst immer gleich ist; lediglich die Anschlußstecker an den Kabelenden sind unterschiedlich; sie sind den zu verbindenden Systemkomponenten angepaßt.

Das Kabel selbst besteht aus zwei Lagen eines 40-adrigen, einseitig abgeschirmten Flachbandkabels des Typs Scotchflex 3380. Die elektrischen Daten dieses Kabels sind:

| | |
|---|---|
| Wellenwiderstand | 75 Ohm (gegen Abschirmung) |
| Signallaufzeit | 5,5 ns/m |

Um eine möglichst schnelle und auch störsichere Signalübertragung auf dem BUS zu erreichen, werden die einzelnen Signalgruppen unterschiedlich behandelt.
Die Dimensionierung der Sender- und Empfängerschaltungen und der Abschlußwiderstände sind der folgenden Abbildung und Tabelle zu entnehmen.

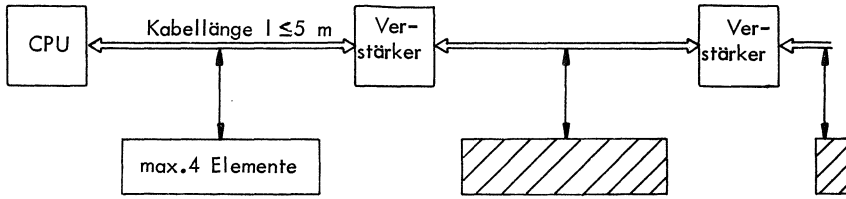



Schematische Darstellung von Sende- und Empfangsschaltungen sowohl für unidirektionale als auch für bidirektionale Signale. Bei den Sendern von unidirektionalen Signalen wird anstelle des SN 7438 oft ein SN 7416 verwendet, da eine Durchschaltung DS des Signals entfällt.

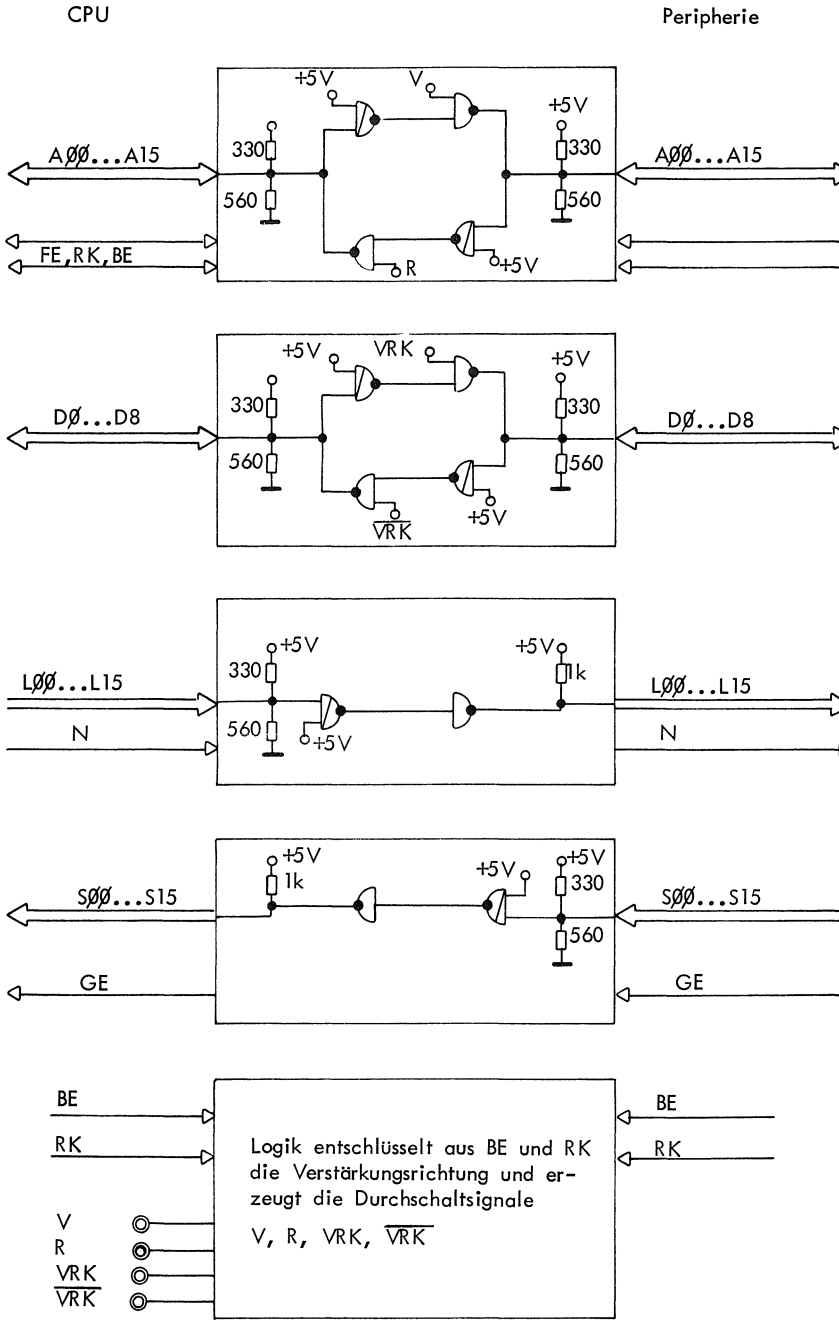
| Signal | Richtung CPU - Per. | R1 (Ohm) | R2 (Ohm) | R3 (Ohm) | R4 (Ohm) | R5 (Ohm) | R6 (Ohm) | R7 (Ohm) | C pF | D |
|---|---|---|---|---|---|---|---|---|---|---|
| Daten Adressen RK, N, BE | ⟷ | 330 | 560 | 330 | 560 | ∞ | 0 | ∞ | - | - |
| FE | ⟷ | 270 | ∞ | 270 | ∞ | ∞ | 680 | 1000 | 100 | Ja |
| G E | ⟵ | 120 | 270 | 4700 | ∞ | ∞ | 680 | 1000 | - | - |
| Starts | ⟵ | 560 | 1500 | 1000 | ∞ | ∞ | 0 | ∞ | - | - |
| Level | ⟶ | 560 | 1500 | 1000 | ∞ | ∞ | 0 | ∞ | - | - |

6 Sender/Empfangsstufen belastet werden, wobei pro Stufe ein TTL Fan In von 1 als Lastfaktor anzusetzen ist.

Bei größeren BUS-Längen oder beim Anschluß von mehr als 6 Elementen wird der Universal-BUS durch BUS-Verstärker in einzelne BUS-Segmente aufgeteilt, die den geforderten Übertragungsbedingungen genügen. Die BUS-Elemente sollten nicht länger als 4 - 5 m sein.




Die BUS-Verstärker bilden den BUS-Abschluß nach und regenerieren die empfangenen Signale.




Blockschaltbild des BUS-Verstärkers

Der BUS-Verstärker bewirkt eine Signalverzögerung von max. 100 ns pro Signal. Das bedeutet für einen Belegungsvorgang eine Verlängerung von insgesamt 400 ns.

Für den Aufbau eines BUS-Systems ergeben sich durch von der Hardware bedingte Gründe folgende Regeln:

a) BUS segmentieren.
   Jedes BUS-Segment hat eine Länge $l \leq 5$ m und wird mit max. 4 zusätzlichen TTL Fan In pro Signal belastet.

b) Elemente bzw. Interfaces, die eine hohe Arbeitsgeschwindigkeit haben, in der Regel aktive Elemente, sollen möglichst rechnernah angeordnet werden, da Verstärker den BUS verlangsamen.

c) Bei der Anordnung mehrerer aktiver Elemente ist auf die Priorität der Elemente zu achten: das der CPU nächste Element hat die niedrigste Priorität.

d) Das physikalische Ende des BUS ist mit einer Widerstandskombination pro Signalleitung abzuschließen (Abschlußkarten vorsehen).

Soll der Universal-BUS des DIETZ 621 über große Entfernungen außerhalb der Systemschränke geführt werden, so ist er über einen Entkopplungsbaustein galvanisch zu trennen. Der so entkoppelte BUS kann nur zum Anschluß passiver Elemente benutzt werden, da alle Signalleitungen mit Ausnahme der Daten unidirektional entkoppelt werden.

# Universal-Interface-Einheit

AUFBAU

Die Universal-Interface-Einheit (UIE) dient zur Aufnahme von Einkarten-Interfaces. Sie beinhaltet standardmäßig

a) zwölf Steckplätze für Einkarten-Interfaces

b) einen Device-Selector

c) entweder einen BUS-Verstärker, eine BUS-Durchschaltung oder eine BUS-Abschlußkarte

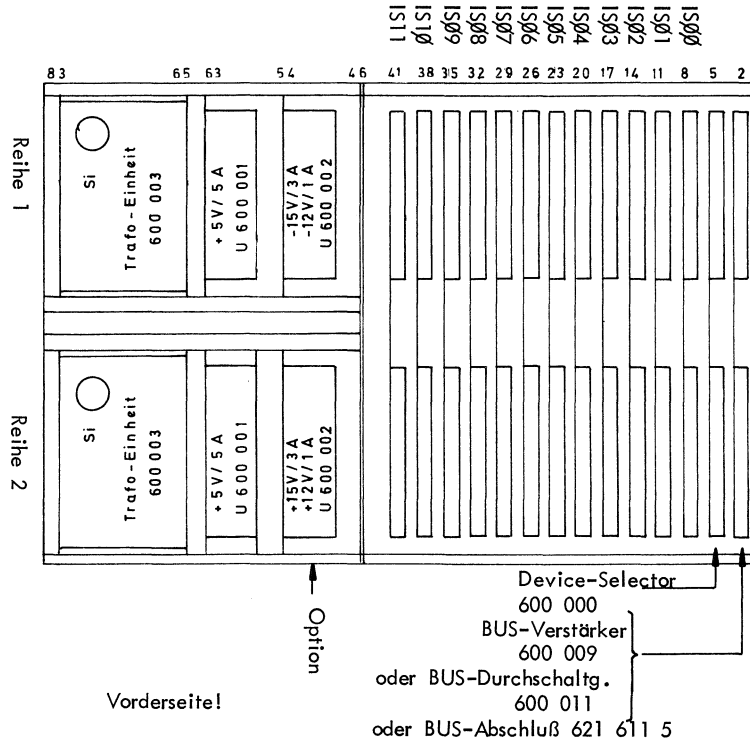
d) eine +5 V-Stromversorgung.

Die Stromversorgung kann bei Bedarf zusätzlich folgende Spannungen liefern:

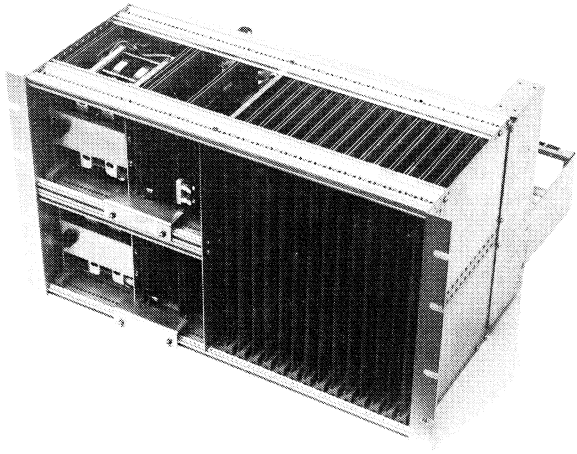
$\pm$15 V/3 A

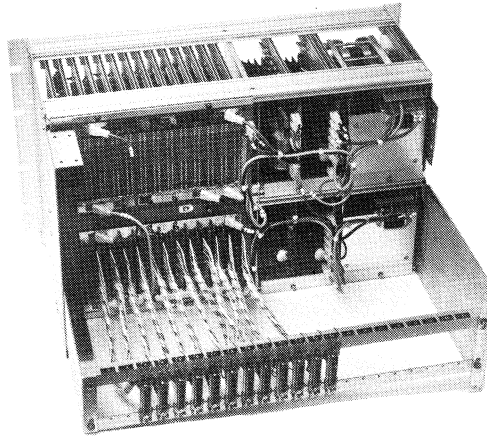
$\pm$12 V/1 A

Die Verbindung zwischen den Interface-Steckplätzen und dem Device-Selector - der sog. Mini-BUS - erfolgt mittels einer gedruckten Leiterplatte, die gleichzeitig die Rückwand der UIE bildet.



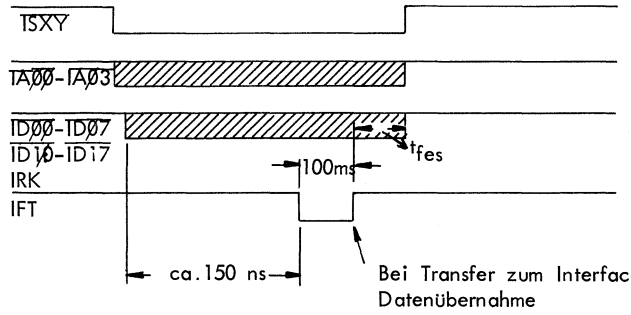
| | |
|---|---|
| Stromversorgung: | +5 V 10 A<br>$\pm$12 V/15 V (Option) 1 A/3 A |
| Netzanschluß: | 220 V $\pm$10 % 50 Hz einphasig<br>Leistungsaufnahme max. 150 VA |
| Größe: | 19"-Einbaurahmen, offen, konvektionsbelüftet<br>6 Einheiten hoch (ca. 270 mm)<br>max. 250 mm tief |
| Gewicht: | ca. 10 kg |





IRMØØ...IRM11 Unidirektionale Signale. Wie die Signale ISØØ...IS11 sind die Rückmeldungen den jeweiligen Steckplätzen zugeordnet (Signal IRMXY des Steckplatzes 8 - IRMØØ, IRMXY des Steckplatzes 11 - IRMØ1 usw. (siehe Bild Universal-Interface-Einheit). Die Rückmeldungen werden auf dem Device-Selector den Interruptleitungen des Universal-BUS zugeordnet.

Zeitdiagramm Mini-BUS:




$t_{fes}$ ≙ 100.. 300 ns (Signallaufzeiten)

MINI-BUS

Der Mini-BUS ist der Datenkanal, an den Einkarten-Interfaces in einer Universal-Interface-Einheit angeschlossen werden. Der Mini-BUS wird durch den DEVICE-Selector aus dem UNIVERSAL-BUS erzeugt.

Im Gegensatz zum Universal-BUS ist der Mini-BUS ein synchroner Datenkanal. Die Datenübernahme erfolgt mit dem Taktimpuls IFT.

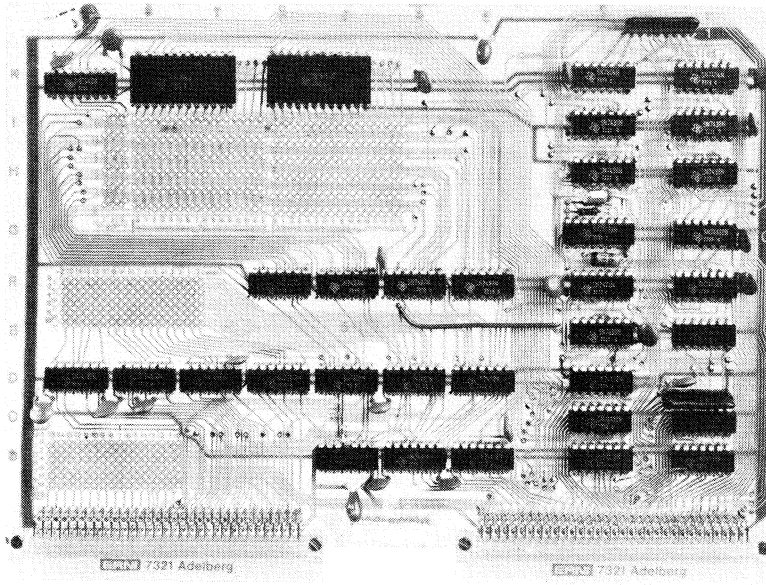
Der Mini-BUS umfaßt folgende Signalleitungen:

(Log 1 ≙ 0 V; Log Ø ≙ +3 V...+5 V)

| | |
|---|---|
| IDP | Bidirektionales Signal. Parity-Bit (bei Standard-Interfaces nicht verwendet) |
| IDØØ...IDØ7 | Bidirektionale Signale Datenbits Ø...7 |
| ID1Ø...ID17 | Bidirektionale Signale Datenbits 8...15 (vorgesehen für 16-bit-Systeme; zur Zeit bei Device-Selector mit IDØØ...IDØ7 zusammengeschaltet; d.h. IDØØ = ID1Ø) |
| IAØØ...IAØ3 | Unidirektionale Signale. Niedrigwertige Adressen des Universal-BUS. Sie dienen zur Erzeugung von Unteradressen auf den Einkarten-Interfaces. |
| ISØØ...IS11 | Unidirektionale Signale. Anwahlsignale für die einzelnen Interface-Plätze. ISØØ wählt Steckplatz 8, ISØ1 Steckplatz 11 usw. (siehe Bild Universal-Interface-Einheit) an.<br>Am jeweiligen Steckplatz heißt das Signal ISXY.<br>Die Signale werden auf dem Device-Selector aus den höherwertigen Adreßbits (AØ4...A15) und gegebenenfalls aus den Levelleitungen (LØØ...L15) entschlüsselt. |
| IRK | Unidirektionales Signal. Richtungskennzeichen gibt die Datenrichtung der Signale IDØØ...IDØ7 bzw. ID1Ø... ID17 an. Bei IRK = Log.1 werden Daten zum Interface, bei IRK = Log.Ø zum Rechner transportiert. |
| IN | Unidirektionales Signal . Nullstellung für Flip-Flops bei Einschalten bzw. Betätigen der Taste RES an der Bedienungskonsole des Computers. |

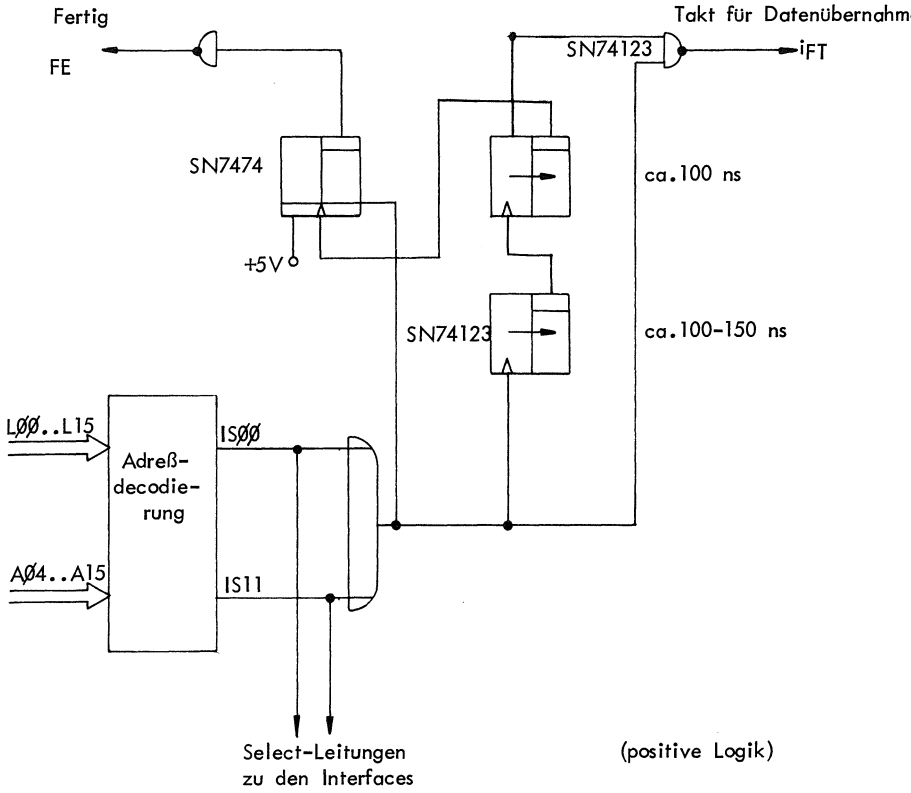
## DEVICE-SELECTOR

Der Datenfluß zwischen dem Computer DIETZ 621 und den Interfaces in einer UIE erfolgt über den Universal-BUS, Device-Selector und Mini-BUS. Auf dem Device-Selector müssen die Adressen der Interfaces mit Lötbrücken programmiert werden. Sie sind durch die Signale AØ4...A15 definiert und eventuell an eine der 16 Hardware-Ebenen gebunden. Jeder Interface-Steckplatz hat eine eigene Select-Leitung (ISØØ... IS11), und bei der Entschlüßlung einer der programmierten Adressen erfährt das entsprechende Interface über diese Leitung, daß es angesprochen werden soll.



Außerdem triggert das Select-Signal eine Zeitstufe an, die nach einer Verzögerung von mindestens 100...150 ns den Interface-Takt (IFT) bildet. Dessen Rückflanke erzeugt das Fertig-Signal, das bewirkt, daß die bis dahin am BUS anstehende Adresse weggeschaltet wird, damit die Select-Information nicht mehr vorhanden ist und hierdurch das "Fertig" wieder zurückgenommen wird.

Der Interface-Takt gelangt über den Mini-BUS auch zu allen angeschlossenen Interfaces und kann dort bei Rechner-Ausgaben als Übernahmetakt benutzt werden.



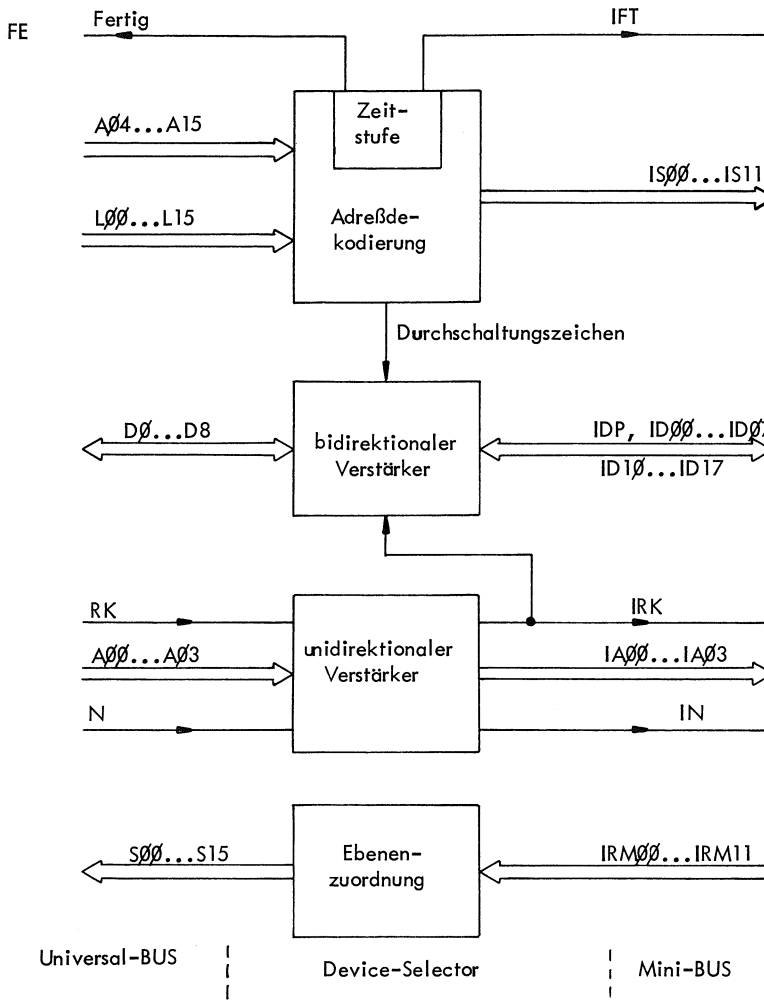

Zeitstufe auf dem Device-Selector

Damit ein Interface, wenn es über die Select-Leitung angewählt wird, erkennen kann, mit welchem seiner Register gearbeitet werden soll, verstärkt der Device-Selector die vier niederwertigen Adreßbits und bietet sie über den Mini-BUS an (IAØØ...IAØ3). Das gleiche geschieht mit dem Richtungskennzeichen (IRK), das hier zur Unterscheidung von Rechner-Ein- und -Ausgaben dient. Auf dem Device-Selector selbst steuert es dementsprechend die Datendurchschaltung.

Jedes Interface kann seinerseits eine Anforderung an den Computer senden. Hierzu geht von jedem Steckplatz eine eigene Leitung (IRMØØ...IRM11) zum Device-Selector, auf dem durch Lötbrücken programmiert werden kann, welche Rechner-Hardware-Ebenen durch die einzelnen IRM-Meldungen gestartet werden sollen.

Eine letzte Aufgabe von Device-Selector und Mini-BUS ist, die Interfaces an die zentrale Nullstellung anzuschließen.

Blockschaltbild Device-Selector (UIE)



## EINKARTEN-INTERFACES

Einkarten-Interfaces (EKI) schließen Bedienungsperipherie, Datenfernübertragungssysteme und Prozeß-Ein/Ausgänge an den Computer DIETZ 621 an. Sie können entweder in den beiden Steckplätzen der Zentraleinheit oder in einer UIE untergebracht werden. Ihr Format ist das einer doppelten Europakarte (233,5 mm x 160 mm), und sie sind mit zwei 64-poligen Steckern ausgerüstet. Stecker 1 in der UIE oben ist die Schnittstelle zum Computer bzw. Mini-BUS. Stecker 2 ist die Schnittstelle zur Peripherie, deren Anschluß rückseitig entweder auf dem zu Stecker 2 gehörenden Gegenstecker oder auf einem verriegelbaren Stecker für Kabelanschluß erfolgt. Dieser verriegelte Stecker ist ein 26-poliger AMP-Stecker, wenn es sich um Prozeßperipherie handelt, ein 25-poliger Cannon-Stecker, wenn es sich um Geräte-Peripherie handelt.

Wesentliche Funktionsgruppen eines EKIs sind (Beispiel für ein Geräte-Interface mit 8 Bit-E/A):

Datenregister: 8-bit-Flip-Flop-Speicher, programmgesteuert einschreib- und/oder abfragbar. Hält Daten für das periphere Gerät bereit (Ausgabe) bzw. übernimmt sie von ihm (Eingabe).

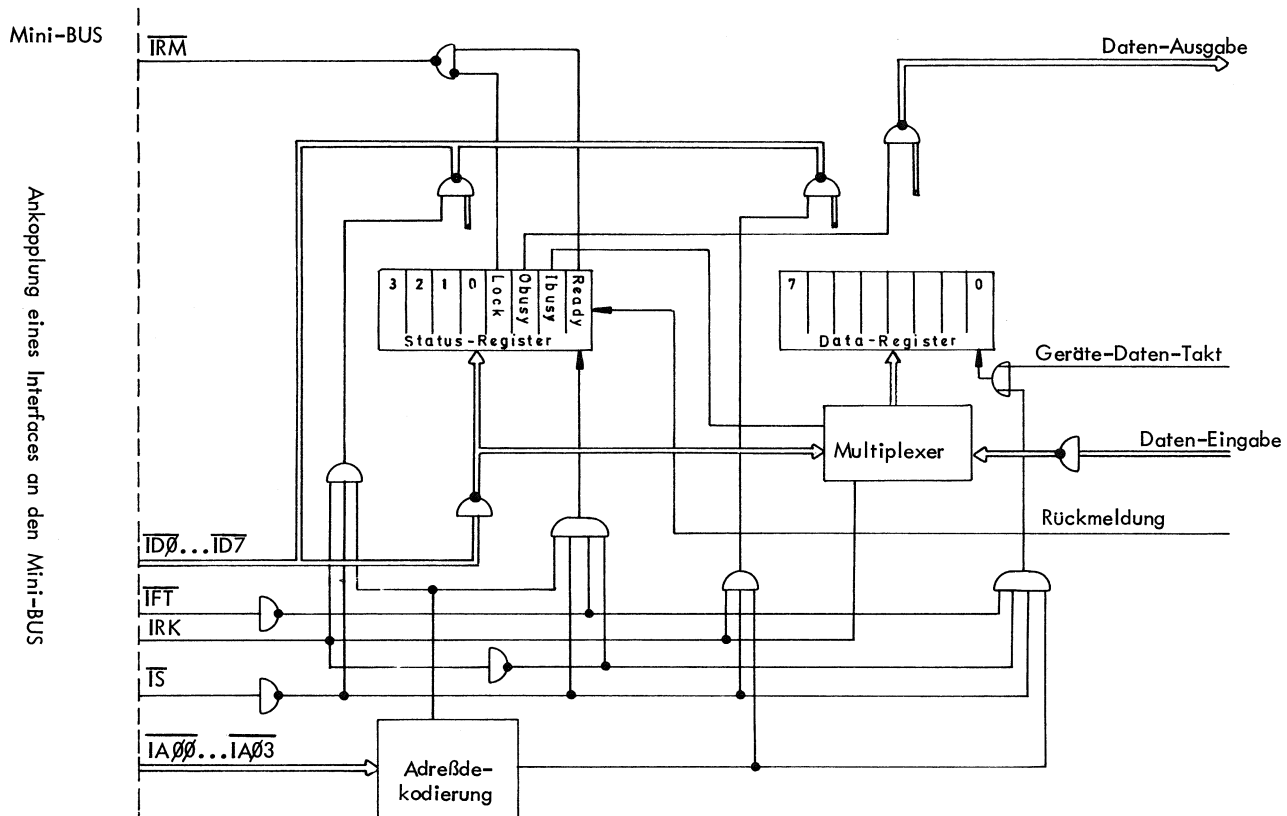
Statusregister: 8-bit-Flip-Flop-Speicher, programmgesteuert einschreib- und abfragbar mit den Funktionen:

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

Eine Datenausgabe zu einem Gerät erfolgt, sobald OBUSY eingeschaltet und READY ausgeschaltet ist. Mit Beendigung der Ausgabe wird READY eingeschaltet und damit, wenn vorher nicht LOCK eingeschaltet wurde, ein Start erzeugt.

Eine Eingabe erfolgt sinngemäß bei eingeschaltetem IBUSY und ausgeschaltetem READY.

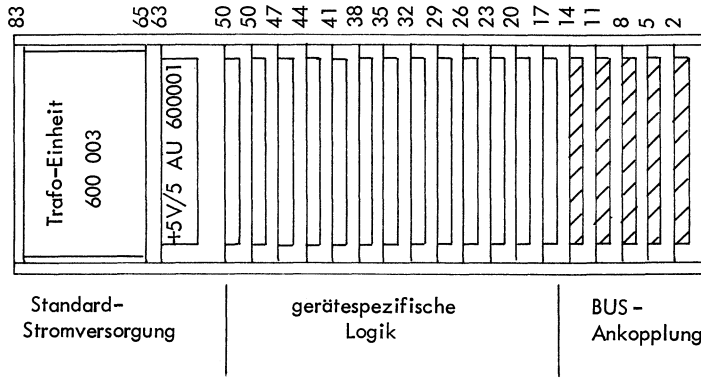
Ankopplung eines Interfaces an den Mini-BUS



# Aktive Elemente

AUFBAU

Zu den aktiven Elementen des DIETZ 621-Systems gehören Controller und Meßsysteme mit Selbststeuerzusätzen. Jedes dieser Elemente ist als eine in sich geschlossene Baugruppe realisiert, die in einem 19"-Einschub der Höhe C untergebracht ist.




Die Baugruppe "aktives Element" ist aufgeteilt in drei Untergruppen:

a) die Standard-Stromversorgung, bestehend aus einer Transformatoreneinheit und einem 5 V/5 A-Regelbaustein

b) die Baugruppe mit gerätespezifischer Logik, deren Aufbau weitestgehend vom anzuschließenden Gerät abhängt

c) die Baugruppe BUS-Ankopplung, die die gerätespezifische Logik an den BUS des DIETZ 621 anpassen und Datentransfers über den BUS steuern soll.

Die Logik-Bausteine sind einfache Europakarten (100 mm x 160 mm) mit 64-poligen Steckern auf den Steckplätzen 14 bis 50, mit 36-poligen Steckern auf den Plätzen 2 bis 11.

## BUS-ANKOPPLUNG

Die BUS-Ankopplung besteht aus 5 gedruckten Schaltungen, die wie folgt angeordnet sind:

| Steckplatz | Baustein | |
|---|---|---|
| 2 | BUS-Verstärker 1 | 605 026 |
| | oder BUS-Abschluß | 605 022 |
| | oder BUS-Durchschaltung | 605 058 |
| 5 | BUS-Verstärker 2 | 605 027 |
| | oder BUS-Abschluß | 605 022 |
| | oder BUS-Durchschaltung | 605 058 |
| 8 | Device-Selector 1 | 605 008 |
| 11 | Device-Selector 2 | 605 009 |
| 14 | Adreßzähler, Bus-Belegung | 605 028 |

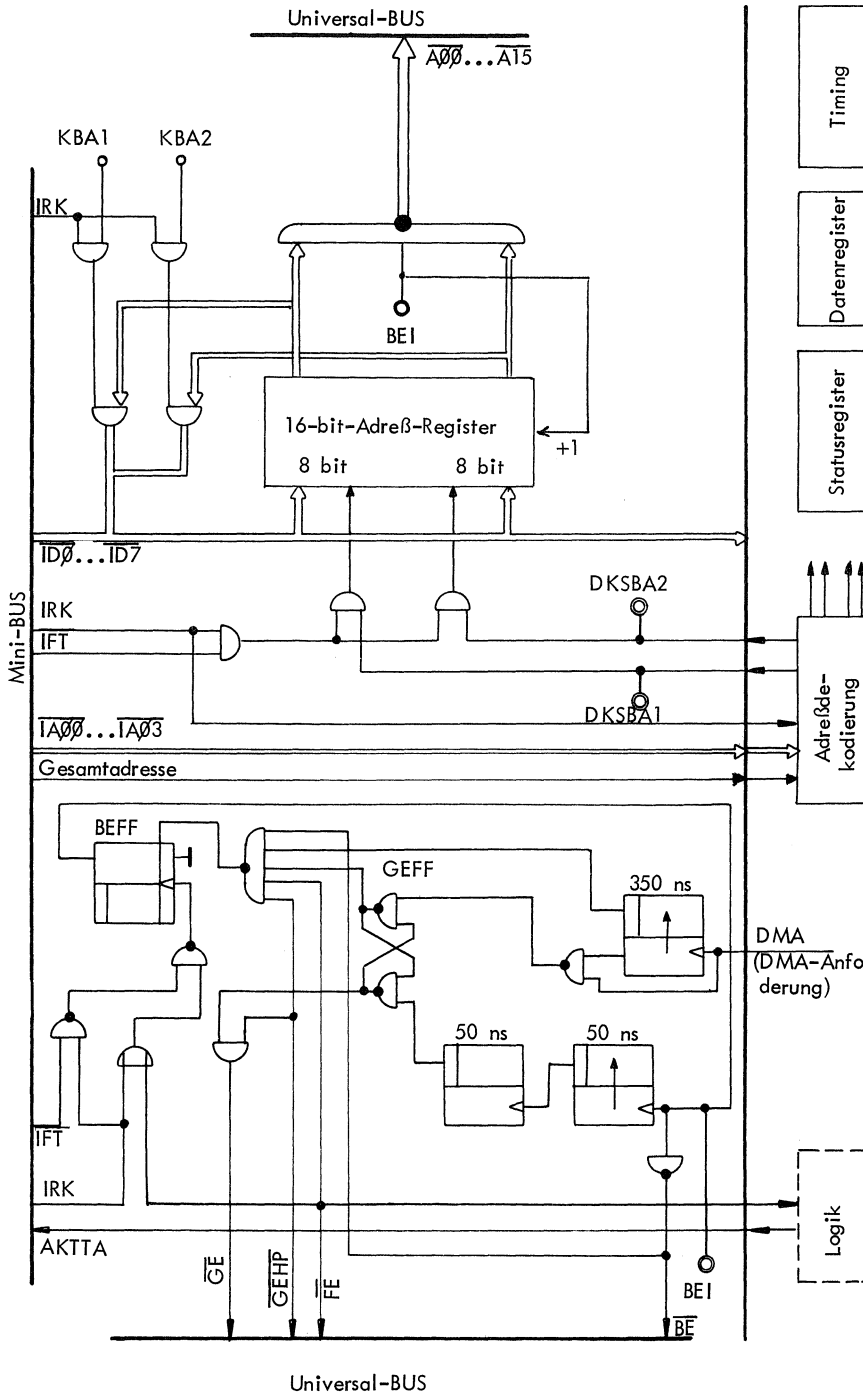
BUS-Verstärker, BUS-Abschlußkarten und BUS-Durchschaltungen entsprechen logisch den in der UIE verwendeten Bausteinen, lediglich die Bauform wurde dem C-Einschub angepaßt.

Die Device-Selectoren sind gegenüber dem Device-Selector für die Einkarten-Interfaces logisch geringfügig erweitert:

Die Durchschaltungskennzeichen für die bidirektionalen Verstärker hängen außer vom Richtungskennzeichen auch davon ab, welches Element im Augenblick aktiv ist, da erst durch beide Informationen die Transferrichtung eindeutig bestimmbar ist (Signale DDE und DDA). Die Zeitstufe des Device-Selectors kann zusätzlich vom aktiven Element beim Lesevorgang angestoßen werden. Sie erzeugt in diesem Fall einen Takt $\overline{IFT}$, schaltet aber nicht das "Fertig-Flip-Flop" ein.

Die gerätespezifische Logik wird somit genau wie ein Einkarten-Interface an den Mini-BUS angeschlossen.

Der Baustein "Adreßzähler, BUS-Belegung" umfaßt die Steuerungslogik für einen Datentransfer ohne Rechnerkontrolle, die Prioritätslogik für das Gewünscht-Signal und ein 16-bit-Register, das bei einem Transfer die Adresse des passiven Elements vorgibt, in der Regel eine Kernspeicherzelle.


Universal-BUS
$\overline{A00}...\overline{A15}$
KBA1
KBA2
IRK
BEI
16-bit-Adreß-Register
8 bit
8 bit
+1
Timing
Datenregister
Statusregister
$\overline{ID0}...\overline{ID7}$
DKSBA2
IRK
IFT
DKSBA1
$\overline{IA00}...\overline{IA03}$
Gesamtadresse
Adreßde-kodierung
Mini-BUS
BEFF
GEFF
350 ns
DMA
(DMA-Anfor-derung)
50 ns
50 ns
IFT
IRK
AKTTA
Logik
$\overline{GE}$
$\overline{GEHP}$
$\overline{FE}$
BEI
$\overline{BE}$
Universal-BUS

Die gerätespezifische Logik enthält u.a. eine Adreßdecodierung, die aus den Adreßbits $\overline{iA\emptyset\emptyset}$ bis $\overline{iA\emptyset 3}$ und der auf dem Device-Selector entschlüsselten Gesamtadresse die Unteradressen der internen Register erkennt. Die Unteradressen DKSBA1 und DKSBA2 sind dem 16-bit-Adreßregister zugeordnet, das von der CPU setz- und abfragbar ist und das bei einem DMA-Zyklus die Kernspeicheradresse ($\overline{A\emptyset\emptyset}...\overline{A15}$) anwählt.

Der eigentliche Belegungsvorgang wird eingeleitet durch eine DMA-Anforderung der gerätespezifischen Logik. Das Signal DMA stößt ein Mono-Flop an, dessen Ausgangsimpuls zum einen das nachgeschaltete Gewünscht-Flip-Flop (GEFF) setzt und damit die DMA-Anforderung speichert und zum anderen zur Zeitüberwachung des GE-Signals dient. Die gespeicherte DMA-Anforderung wird als GE-Signal auf den BUS geschaltet. Ein BUS-GE wird außerdem erzeugt, wenn ein GE höherer Priotität ansteht.

Das Belegt-Flip-Flop (BEFF) wird eingeschaltet, wenn

a) DMA-Anforderung gespeichert,

b) kein Belegt-Signal auf dem BUS ansteht,

c) kein Fertig-Signal auf dem BUS ansteht,

d) kein GEHP (gewünscht höherer Priorität) anliegt und

e) die GE-Überwachungszeit abgelaufen ist.

Das gesetzte Belegt-Flip-Flop (Signal BEI) schaltet alle Signale des aktiven Elements auf den BUS.

Das Gewünscht-Flip-Flop (GEFF) wird mit der verzögerten Vorderflanke des BEI-Signals zurückgesetzt.

Am Ende des Belegungsvorganges wird das Belegt-Flip-Flop auf zweierlei Weise über den Takteingang zurückgesetzt:

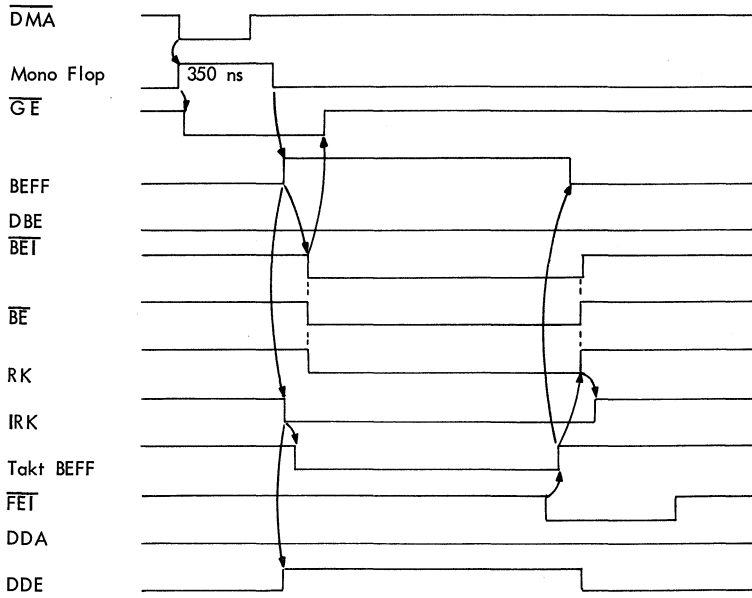
a) bei einem Schreibvorgang mit dem kommenden FE-Signal

b) bei einem Lesevorgang mit der Rückflanke des iFT-Impulses, der von der Zeitstufe des Device-Selectors erzeugt wird. Die Zeitstufe wird dabei angetriggert über das Signal AKTTA, das von der gerätespezifischen Logik während eines Lesevorgangs beim Empfangen des FE-Signals erzeugt wird.

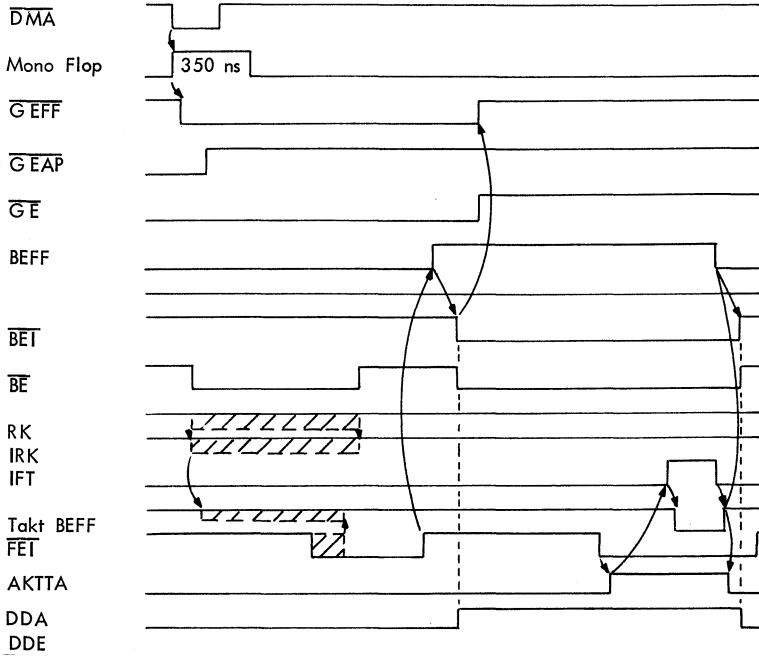
Das Adreßregister wird am Ende eines Belegungsvorganges inkrementiert.

Die folgenden Zeitdiagramme zeigen den Steuerungsablauf bei einem Lese- und einem Schreibvorgang, wobei beim Schreibvorgang vorausgesetzt wurde, daß kein GEHP vorhanden war.

Schreibvorgang (aktiv → passiv)




Lesevorgang (passiv → aktiv)



Die BUS-Belegungssteuerung in der dargestellten Form führt jeweils einen Belegungsvorgang durch und gibt den BUS anschließend wieder frei, so daß ein anderes aktives Element den BUS belegen kann. Werden sehr hohe Transferraten verlangt (DMA-Abstand kleiner als 3 us), kann dafür gesorgt werden, daß ein "Dauer-Belegt" (DBE) erzeugt und die BUS-Kontrolle erst nach Beendigung aller Transfers abgegeben wird.

# Erweiterungen der Zentraleinheit

SPEICHERERWEITERUNG

Im Gehäuse des Computers MINCAL 621 können bis zu 16 Kbyte (bzw. 32 Kbyte) Kernspeicher bzw. bis zu 8 Kbyte reprogrammierbarer Festspeicher untergebracht werden. Wird eine darüberhinausgehende Kapazität benötigt, so sind zusätzliche Speicher-Einheiten erforderlich; sie erlauben eine Erweiterung der Speicherkapazität auf insgesamt 80 Kbyte.

Jede dieser Speichereinheiten enthält:

| | |
|---|---|
| Kernspeicher: | 1 oder 2 Speicher zu je 16 oder je 32 Kbyte<br>Zugriffszeit 300 ns<br>Vollzyklus 650 ns<br>mit Parity-Prüfung/Erzeugung |

Bei Arbeiten mit dem 2. externen Speicher wird vom 1. externen Speicher durch einen Umschaltbefehl umgeschaltet und dann die gleichen Adressen wie beim 1. externen Speicher verwendet (siehe Kapitel "Struktur - Abschnitt Kernspeicher").

| | |
|---|---|
| Festspeicher:<br>(wahlweise statt<br>Kernspeicher) | 1 oder 2 Speicher zu je max. 8 Kbyte<br>in Stufen von 256 byte ausbaufähig<br>reprogrammierbarer MOS-Speicher<br>Zugriffszeit ca. 1 us<br>Adressen: '8ØØØ...'9FFF (1. Speicher)<br>'CØØØ...'DFFF (2. Speicher) |
| Stromversorgung: | eingebaut |
| Netzanschluß: | 220 V ±10 % 50 Hz<br>Leistungsaufnahme ca. 500 VA |
| Größe: | 19"-Einschub, allseitig geschlossen, zwangsbelüftet<br>5 Einheiten hoch (ca. 225 mm)<br>525 mm tief |
| Anschluß: | an Zentraleinheit MINCAL 621 über BUS-Kabel<br>(in unmittelbarer Nähe) |
| Adressen: | '8ØØØ...'BFFF Erweiterung auf 32K ) 1.externer Speicher<br>'8ØØØ...'FFFF Erweiterung auf 48K )<br>'8ØØØ...'BFFF Erweiterung auf 64K ) 2.externer Speicher<br>'8ØØØ...'FFFF Erweiterung auf 80K ) |

## FESTKOMMA MULTIPLIKATION/DIVISION

Der DIETZ 621 hat in der Standard-Ausführung keine Hardware-Multiplikation/Division. Diese Rechnungen sind durch Unterprogramme realisiert.

Zur Steigerung der Geschwindigkeit von arithmetischen Operationen dient das Festkomma-Rechenwerk. Es hat die gleiche Bauform wie Einkarten-Interfaces und wird in eine Universal-Interface-Einheit eingesteckt (siehe Kapitel "Einkarten-Interfaces"). Es besteht aus 2 Steckkarten, die durch einen Verbindungsstecker miteinander verbunden sind.

| | | |
|---|---|---|
| Funktionen: | Multiplikation | Dauer max.6.72 us |
| | Division | " 6.4 us/11.8 us |
| | Zweierkomplement | " 0.64 us |

Wortlänge: Ausgangswerte 16 bit
Ergebnis Multiplikation 32 bit
" Division 32 bit oder 16 bit + 16 bit Rest
" Komplement 16 bit

Zahlendarstellung: Zweier-Komplement

Adresse: '12ØX

Baugröße: 233,5 mm hoch, 160 mm tief (doppelte Europakarte).

Einbauraum: 2 Steckplätze

## GLEITKOMMA-RECHENWERK

Der DIETZ 621 läßt sich zusätzlich mit einer Hardware-Gleitkomma-Arithmetik ausrüsten. Die Laufzeiten aller Programme mit Gleitkomma-Operationen (z.B. BASIC oder BASEX) werden wesentlich verkürzt.

Bei den arithmetischen Operationen des DIETZ 621 -Gleitkomma-Rechenwerkes ist es möglich, nach Ausführung der Operationen ein Normalisieren der Ergebnisse zu verhindern. Auf diese Weise lassen sich auch Festkomma-Operationen mit dem Gleitkomma-Rechenwerk durchführen.

| | | |
|---|---|---|
| Funktionen: | Addition | Dauer max. 6.5 us |
| | Subtraktion | " " 6.5 us |
| | Multiplikation | " " 9.5 us |
| | Division | " " 9.5 us |
| | Bilden des Absolutbetrages | " " 0.1 us |
| | Negation | " " 0.1 us |
| | Umrechnung Gleit- → Festkomma | " " 5.8 us |
| | " Fest- → Gleitkomma | " " 5.8 us |

Wortlänge: 24 bit Mantisse, 8 bit Exponent

Zahlendarstellung: Mantisse und Exponent in Zweier-Komplement

Fehlermeldungen: Überlauf, Unterlauf

Sonstige Meldungen: Ergebnis = Ø, Ergebnis > Ø, Ergebnis < Ø

Adresse: '16ØX

Einbau: in Speichererweiterungs-Einschub.

## BATTERIE-EINHEIT

Das als Register- und Arbeitsspeicher des DIETZ 621 Computers verwendete Halbleiter-RAM kann bis zu einer Kapazität von 1 kbyte über eine bestimmte Zeit des Netzausfalls durch eingebaute Batterien versorgt werden. Ist bei größeren Kapazitäten oder für längere Zeiträume eine Pufferung erforderlich, so ist eine getrennte Batterie-Einheit erforderlich.

### Technische Daten

Pufferungsdauer:

| | | | | |
|---|---|---|---|---|
| bei | 0.25 | kbyte | RAM: | 192 h |
| " | 0.5 | " | " : | 96 h |
| " | 1 | " | " : | 48 h |
| " | 2 | " | " : | 24 h |
| " | 4 | " | " : | 10 h |

Stromversorgung: eingebaut

Netzanschluß: 220 V ±10 % 50 Hz
Leistungsaufnahme ca. 300 VA

Größe: 19"-Einbaurahmen, offen, konvektionsbelüftet
3 Einheiten hoch (ca. 135 mm)
ca. 450 mm tief

Anschluß: an Batteriestecker der Zentraleinheit DIETZ 621 über 2-poliges Kabel.

# Plattenspeicher-Systeme

DIETZ-DISK

Die DIETZdisk ist ein Wechselplattenspeicher mit einem neuartigen Prinzip. In diesem Speichersystem werden die Vorteile von großen Plattensystemen - Zuverlässigkeit, wahlfreier Zugriff, hohe Geschwindigkeit - mit dem niedrigen Preis von Lochstreifen-Peripherie vereint. Wie bei großen Plattensystemen findet keine Berührung zwischen Platte und Lese-Schreib-Kopf statt. Die Positionierung erfolgt nach Servo-Informationen, die sich auf der Speicherkassette befinden. Eine Kassette kann deshalb ohne Schwierigkeiten auf einem Laufwerk beschrieben und auf einem anderen gelesen werden.

Laufwerke:

- DIETZdisk 30 - Einzellaufwerk zum Betrieb einer Plattenkassette
- DIETZdisk 40 - Doppellaufwerk zum Betrieb von zwei Plattenkassetten

Technische Daten:

| | |
|---|---|
| Platten-Drehzahl: | 3000 Upm |
| Positionierzeit: | minimal 100 ms<br>im Mittel 210 ms<br>maximal 320 ms |
| Eingebauter Schreibschutz | |

Kassette:

| | |
|---|---|
| Kapazität: | 262 144 byte |
| Anzahl der Spuren: | 64 |
| Sektoren pro Spur: | 32 |
| Bytes pro Sektor: | 128 (+2 +2 für CRC) |
| Aufzeichnungsdichte: | 3700 bpi |
| Spurdichte: | 50 tpi |
| Maße: | 200 x 200 x 10 mm |

| | |
|---|---|
| Stromversorgung: | Zusätzlicher Einschub 19" (3 Einheiten hoch) |
| Netzanschluß: | 220 V $\pm$10 % 50 Hz |
| Leistungsaufnahme: | ca. 300 VA (Doppellaufwerk) |
| Größe: | 19"-Einschub (4 Einheiten hoch) |
| Anschluß: | Über Einkarten-Interface in UNIVERSAL-INTERFACE-EINHEIT oder in Zentraleinheit und über eingebauten MINI-Controller. Pro MINI-Controller lassen sich bis zu 8 Laufwerke betreiben. |

Datenübertragung zwischen Interface und MINI-Controller unter Programmkontrolle (asynchron).

Funktionen: Lesen eines Sektors
Schreiben eines Sektors
Kopieren eines Sektors.

In Verbindung mit einem Bootstrap-ROM in der Bedienungskonsole des Rechners läßt sich ein Sektor automatisch in den Speicher des Rechners laden.

| | | | |
|---|---|---|---|
| Adressen: | '1BXØ | Bit Ø...4 | Sektor/nur setzbar |
| | | Bit 5...7 | Spur niederwertig |
| | '1BX1 | Bit Ø...2 | Spur höherwertig |
| | | Bit 3...6 | Unit belegt |
| | | Bit 7 | frei |
| | '1BX2 | Statuswort setz- und abfragbar | |
| | | Bit Ø | Ready |
| | | Bit 1 | IBUSY |
| | | Bit 2 | OBUSY |
| | | Bit 3 | LOCK |
| | | Bit 4 | Transfer Complete |
| | | Bit 5 | Write Protect |
| | | Bit 6 | Copy Initiate |
| | '1BX3 | Gerätezustand / nur abfragbar | |
| | | Bit Ø | Unit nicht READY |
| | | Bit 1 | CRC Error |
| | | Bit 2 | Unit Write Protect |
| | '1BX4 | Datenregister /setz- und abfragbar | |

## WECHSELPLATTENSPEICHER

Größere Speicherkapazität und Zugriffsgeschwindigkeit bietet der Wechselplattenspeicher WP 2.4.

Laufwerk: Einzellaufwerk mit 2 beweglichen Köpfen für eine Speicherkassette

| | |
|---|---|
| Plattendrehzahl: | 1500 Upm |
| Positionierzeit: | im Mittel 60 ms |
| Transferrate: | 1.6 MHz entsprechend 200 Kbyte/s |
| Eingebauter Schreibschutz | |

Kassette:

| | |
|---|---|
| Kapazität: | 2.4 Mbyte |
| Anzahl der Spuren: | 2 x 200 |
| Sektoren pro Spur: | 12 |
| Zahl der Bytes pro Sektor: | 512 |

| | |
|---|---|
| Stromversorgung: | Zusätzlicher Einschub (Montage hinter Laufwerk) |
| Netzanschluß: | 220 V $\pm$10 % 50 Hz |
| Leistungsaufnahme: | ca. 500 VA |
| Umgebungstemperatur: | 0...40°C |
| Größe: | 19"-Einschub (5 Höheneinheiten) |
| Anschluß: | Über Controller WPCE 621 direkt an den UNIVERSAL-BUS. Datentransfer im direkten Speicherzugriff (DMA). Pro Controller lassen sich bis zu 4 Laufwerke betreiben. |

Controller:

| | |
|---|---|
| Größe: | 19"-Einschub (3 Höheneinheiten) |
| Stromversorgung: | eingebaut |
| Netzanschluß: | 220 V $\pm$10 % 50 Hz |
| Eingebaute Auto-Load-Funktion: | Bei Tastendruck oder Netzausfall wird automatisch ein Sektor der Platte gelesen und der Rechner gestartet. |

Auf einer der Steckkarten des Controllers befinden sich 5 Schalter, die für Testzwecke und Modifikationen der Betriebsweise gedacht sind. Im Normalfall stehen alle Schalterknebel nach unten. Im einzelnen haben die Schalter (von oben nach unten) folgende Bedeutung:

1) Adreßinkrementierung: Schalter nach oben: verhindert
   Schalter nach unten: zugelassen

2) Auto-LOAD bei Netzausfall bzw. Betätigen der Taste RES der Bedienungskonsole des Rechners: Schalter nach oben: verhindert
   Schalter nach unten: zugelassen

3) LOAD Taste, bei deren Betätigung Auto-LOAD initiiert wird.

4) Schreibschutz Spur Ø: Schalter nach oben: nicht wirksam
   Schalter nach unten: wirksam

5) Schreibsperre Taste, bei deren Betätigung allgemein eine Schreibsperre besteht (Entsprechend Netzeinschalten). Schreibschutz kann durch Taste am Laufwerk aufgehoben werden.

Adreßliste für den Wechselplatten-Controller WPCE 621

| Befehl | Adresse | Datenwort | LD | ST |
|---|---|---|---|---|
| 4 Worte Lesen | '18ØØ | beliebig | x | x |
| KS-Basisadresse niederwertig | '18Ø1 | 'ØØ...'FF | x | x |
| " " höherwertig | '18Ø2 | '4Ø. | x | x |
| Zylinderadresse | '18Ø3 | 'ØØ...'CB | | x |
| Kopfauswahl, Select | '18Ø4 | YYØØØØØX | | x |
| Sektor-Basisadresse | '18Ø5 | 'ØØ...'ØB | | x |
| Sektoranzahl | '18Ø6 | 'ØØ...'ØB | | x |

YY: Angabe der Geräteeinheit:
'ØØ ≙ Gerät 1
'4Ø ≙ Gerät 2
'8Ø ≙ Gerät 3
'CØ ≙ Gerät 4
nur = nicht ≙

X: Kopfauswahl:
X ≙ 1: oberer Kopf
X ≙ 0: unterer Kopf

Sektor-Basisadresse:
Ø ≙ 11; Ø ≙ Sektor 1
11 ≙ Sektor 12

Sektoranzahl:
Ø ≙ 1; 11 ≙ 12

Zylinderadresse (Spur):
Ø ≙ 2Ø3 maximal

| Statuswort (Rechner): | '18Ø7 | | x | x |
|---|---|---|---|---|

| | |
|---|---|
| READY | Bit Ø |
| READ | Bit 1 |
| WRITE | Bit 2 |
| LOCK | Bit 3 |
| RESTORE | Bit 7 |

| | | |
|---|---|---|
| Statuswort (Platte): | '18Ø8 | nur LD |
| Schreibsperre: | Bit Ø | |
| Fehler beim Schreiben (Spannungseinbruch): | Bit 1 | |
| Sektorzahl zu hoch | Bit 2 | |
| Suche nicht beendet (setzt READY) | Bit 3 | |
| Platte nicht bereit | Bit 4 | |
| Zu große Spuradresse | Bit 5 | |
| Übertragungsfehler (Parity oder CRC) | Bit 6 | |

"LD" bzw. "ST" bedeutet:

| | | |
|---|---|---|
| LD: | Load | z.B. LDA; Daten von Controller an Rechner |
| ST: | Store | z.B. STA; Daten von Rechner an Controller |

Die Adresse '18ØX ist nicht niveaugebunden, d.h. sie läßt sich von jeder Ebene aus ansprechen. Die Rückmeldung des Controllers (IRM) erfolgt jedoch auf eine festgelegte Ebene.

# FESTKOPF-PLATTENSPEICHER

Sehr schnellen Zugriff und sichere Funktion auch unter schlechten Umgebungsbedingungen bietet der Festkopf-Magnetplattenspeicher.

Magnetplattenspeicher mit einer festen Platte und festen Schreib-/Lese-Köpfen für jede Spur.

| | |
|---|---|
| Plattendrehzahl: | 3000 Upm |
| Transferrate: | 2 MHz entsprechend 220 Kbyte/s |
| Schreibschutz: | Für jeweils 16 Spuren gemeinsam einschaltbar |
| Kapazität: | 256 Kbyte oder 1024 Kbyte |
| Zugriffszeit: | im Mittel 10 ms |
| Zahl der Spuren: | 64 oder 256 |
| Zahl der Sektoren pro Spur: | 8 |
| Kapazität pro Sektor: | 512 Byte |
| Stromversorgung: | Zusätzlicher Einschub 19" (4 Einheiten hoch) |
| Netzanschluß: | 220 V $\pm$10 % 50 Hz |
| Leistungsaufnahme: | ca. 500 VA |
| Umgebungstemperatur: | 0...50°C |
| Größe: | 19"-Einschub (7 Einheiten hoch)<br>(11 Einheiten mit Stromversorgung) |
| Anschluß: | Über Controller FPCE 621 direkt an dem UNIVERSAL-BUS. Datentransfer im direkten Speicherzugriff (DMA). Pro Controller lassen sich bis zu 4 Plattenspeicher betreiben. |

Controller:

| | |
|---|---|
| Größe: | 19"-Einschub (3 Höheneinheiten) |
| Stromversorgung: | eingebaut |
| Netzanschluß: | 220 V $\pm$10 % 50 Hz |
| Leistungsaufnahme: | ca. 100 VA |

| | |
|---|---|
| Eingebaute Auto-Load-Funktion: | Bei Tastendruck oder Netzausfall wird automatisch ein Sektor der Platte gelesen und der Rechner gestartet.<br><br>Auf einer Steckkarte befindet sich von vorn zugänglich ein Schalter, mit dem sich die Auto-Load-Funktion für Testzwecke ausschalten läßt.<br>Schalterknebel zeigt nach oben: Keine Auto-Load-Funktion<br>Schalterknebel zeigt nach unten: Auto-Load-Funktion. |

Adreßliste für Festkopfplatten-Controller

| | |
|---|---|
| Gesamtadresse: | '19ØX |
| Sektor-Basisadresse und Einheitenauswahl: | x = Ø |
| Sektor-Basisadresse: | Bit Ø. .3 |
| Einheitenauswahl: | Bit 4...5 |
| Spur adresse: | x = 1 |
| Sektoranzahl: | x = 2 (Bit Ø.. 4) |
| KS-Basisadresse 1.Teil (niederwertige Bits): | x = 3 |
| KS-Basisadresse 2.Teil (höherwertige Bits): | x = 4 |
| Statusregister: | x = 5 |
| READY | Bit Ø |
| READ | Bit 1 |
| WRI TE | Bit 2 |
| LOCK | Bit 3 |
| Platte nicht bereit (oder zu hohe Spuradresse): | Bit 4 |
| Schreibversuch in geschützte Zonen | Bit 5 |
| Datenfehler | Bit 6 |
| Daten zu spät an den KS übergeben | Bit 7 |

# Geräte-Interfaces

Geräte-Interfaces dienen zum Anschluß von Ein- und Ausgabegeräten an den DIETZ 621. Die Interfaces sind als Einkarten-Interfaces gebaut und werden in einer Universal-Interface-Einheit oder im Rechnergehäuse (2 Steckplätze) betrieben (siehe Kapitel "Universal-Interface-Einheit).

Der Datenverkehr zwischen Rechner und Peripherie-Gerät erfolgt fast ausschließlich byteweise. Das Interface übernimmt bei einer Ausgabe dieses Byte vom Rechner und speichert es für die Dauer der Ausgabe bzw. übernimmt bei einer Eingabe die Daten in einen Speicher und stellt sie für eine schnelle Übernahme in den Rechner bereit.

Die Steuerung des Interfaces erfolgt durch ein vom Rechner her setz- und abfragbares Statusregister. Typische Signale eines Statusregisters sind (siehe auch Kapitel "Universal-Interface-Einheit"):

- READY ist die Fertigmeldung des Interfaces an den Rechner. Es erzeugt einen Interrupt, der auf dem Device-Selector programmiert wird. Der Interrupt kann durch das LOCK-Flip-Flop verhindert werden.

- IBUSY steuert die Eingabe von Daten. Aktiviert ist das Interface, wenn IBUSY eingeschaltet und READY noch nicht gesetzt ist. Ist die Eingabe erfolgt, wird READY eingeschaltet, und das Interface ist nicht mehr aktiviert.

- OBUSY steuert die Ausgabe von Daten. Sinngemäß ist die Wirkung wie die von IBUSY.

- LOCK verhindert, daß durch READY ein Interrupt erzeugt wird. Der Rechner kann durch Abfragen des Statusregisters feststellen, ob die Eingabe oder die Ausgabe erfolgt ist.

Für die Geräte-Interfaces sind die Adressen '1ØØX...'15FX vorgesehen.

| | |
|---|---|
| X = Ø | Ansprechen des Datenregisters |
| X = 1 | Ansprechen des Statusregisters |

Folgende Geräte-Interfaces sind erhältlich:

- V24-Interface für Geräteanschluß (IV24/TTY)
  8-Kanal-E/A-Schnittstelle
  110...1200 Bd (fest eingestellt), 10 oder 11 Schritte/Zeichen
  mit zusätzlichem Steuerausgang, Schnittstelle nach V24
  (für Standard-Teletype, Mosaikdrucker, Datensichtgerät usw.)

| | | |
|---|---|---|
| Aufbau Statusregister: | Bit Ø | READY |
| | Bit 1 | IBUSY |
| | Bit 2 | OBUSY |
| | Bit 3 | LOCK |
| | Bit 4 | Initiate (steuert Leser des Teletype) |

- Linienstrom-Interface (ILS/TTY)
  5- oder 8-Kanal-E/A-Schnittstelle (fest eingestellt)
  50...200 Bd (einstellbar), 11 Schritte/Zeichen (8-Kanal),
  7.5 Schritte/Zeichen (5-Kanal)
  Halbduplex-Betrieb, Linienstrom 20...40 mA
  (für 5- and 8-Kanal-Fernschreiber mit Linienstrom)

| | | |
|---|---|---|
| Aufbau Statusregister: | Bit Ø | READY |
| | Bit 1 | IBUSY |
| | Bit 2 | OBUSY |
| | Bit 3 | LOCK |

- Streifenleser-/Locher-Interface (ILE/LO)
  8-Kanal-E/A-Schnittstelle (kombiniert)
  für 8 (5)-Kanal-Lochstreifenleser, 125 Z/s, 300 Z/s
  und 8-Kanal-Lochstreifenstanzer, 75 Z/s

| | | |
|---|---|---|
| Aufbau Statusregister: | Bit Ø | READY |
| | Bit 1 | IBUSY |
| | Bit 2 | OBUSY |
| | Bit 3 | LOCK |
| | Bit 4 | VOR (1 = Leser vorwärts, 0 = Leser rückwärts) |
| | Bit 5 | Parity-Fehler (Leser) |
| | Bit 6 | Error (Locher) Streifen gerissen oder zu straff gespannt |
| | Bit 7 | kein Lochstreifenvorrat mehr |

- Kartenleser-Interface (IKLE)
  12-Kanal-Schnittstelle

| | |
|---|---|
| Datenregister 1 (Kanäle 1...8) | X = Ø |
| Datenregister 2 (Kanäle 9...12) | X = 1 |

| | | |
|---|---|---|
| Aufbau Statusregister: | Bit Ø | READY |
| | Bit 1 | IBUSY |
| | Bit 3 | LOCK |
| | Bit 4 | Fehler |
| | Bit 5 | Eingabeschacht leer |
| | Bit 6 | Einziehfehler |

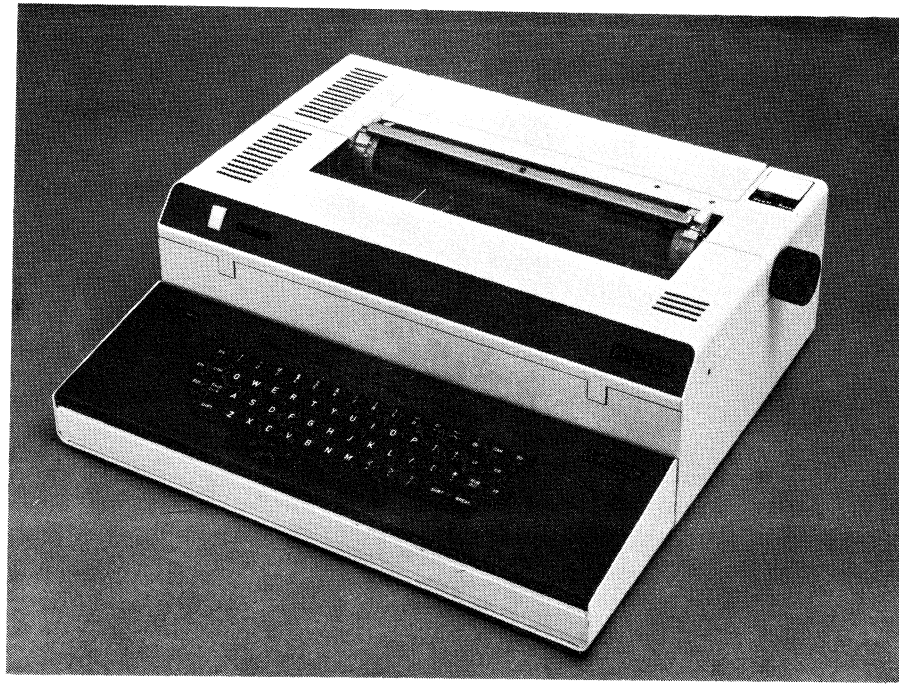
- Ketten-Schnelldrucker-Interface (IMDS)
  gepuffertes 8-bit-Interface

| | | |
|---|---|---|
| Aufbau Statusregister: | Bit Ø | READY |
| | Bit 2 | OBUSY |
| | Bit 3 | LOCK |

# Standard-Peripherals

MOSAIKDRUCKER-TERMINAL

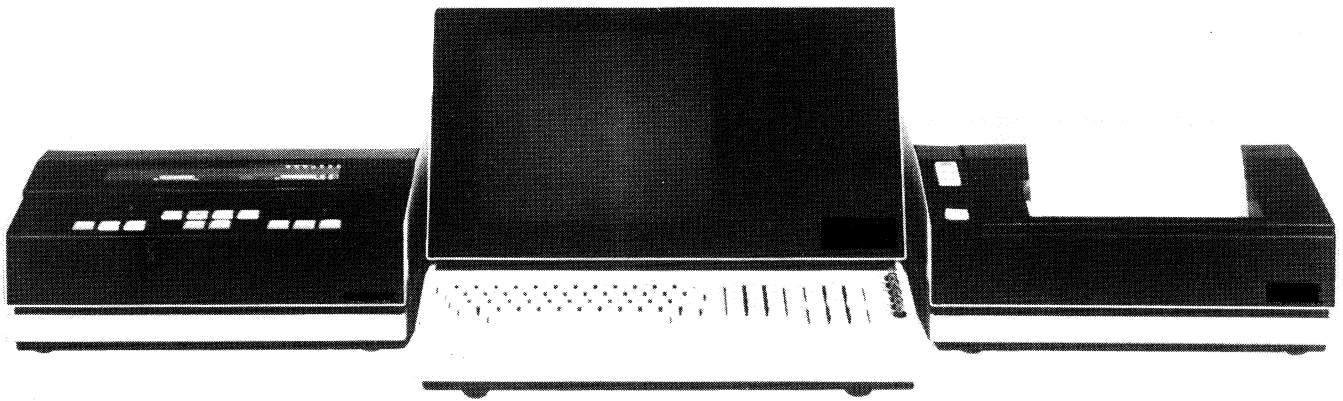
| | |
|---|---|
| Druckgeschwindigkeit: | 50 Zeichen/s |
| Zeilenlänge: | 80 Zeichen |
| Anzahl der Zeichen: | 64 (ASCII) |
| Zeichendichte: | 10 Zeichen/Zoll |
| Zeichenart: | Punktmatrix 8 x 9 |
| Papiergröße: | Variabel bis 244,5 mm |
| Tastenfeld: | Alphanumerisch (ECMA-23C)<br>(wie Teletype) |
| Sonderfunktion: | Seitensteuerung |
| Schnittstellen: | - V24-Schnittstelle, 110...9600 Baud<br>- Parallel-Schnittstelle |
| Abmessungen: | Höhe: 170 mm<br>Breite: 510 mm<br>Tiefe: 310 mm |
| Gewicht: | 20 kg |
| Netzanschluß: | 220 V $\pm$10 % 50 Hz |
| Leistungsbedarf: | ca. 100 VA |



Mosaikdrucker-Terminal

# BILDSCHIRM-TERMINAL BTH 2000

| | |
|---|---|
| Bildschirmgröße | 12" Fernseh-Monitor |
| Bildschirmkapazität | 1998 Zeichen<br>27 Zeilen zu je 74 Zeichen |
| Eingabetastatur | ASCII-Code mit Sonderzeichen |
| Schnittstelle | V24, 1200...9600 Baud |
| Netzanschluß | 220 V 50 Hz 350 W |
| Abmessungen | 560 mm Tiefe, 470 mm Breite, 318 mm Höhe |
| Betriebsarten: | - On-line/Off-line mit eigenem Speicher |
| | - Foreground/Background<br>Vom Computer kommende Zeichen werden dunkelgrün, über die Tastatur eingegebene Zeichen hellgrün angezeigt. |
| | - Tabulator zum Listen, Formular- und Tabellenaufbau |
| | - Adressierbarer Cursor<br>Direkte Adressierung beliebiger Bildschirmpunkte durch den Computer |
| | - Horizontales Rolling-System<br>Einfügen von Zeichen oder Leerstellen ohne Neuschreiben der Restzeile. Die vorhandenen Zeichen werden automatisch nach rechts oder in die nächste Zeile transportiert. |
| | - Vertikale Zeilenverschiebung<br>Einfügung von Zeilen ohne Neuschreiben der anderen. Die bestehenden Zeilen werden automatisch nach unten geschoben. Beim Löschen tritt der umgekehrte Vorgang ein. |
| Optionen: | - Anschluß eines zweiten Bildschirms an dieselbe Tastatur |
| | - TV-Zusatz zum Anschluß eines normalen Fernseh-Monitors |
| | - Hardcopy-Druckeinheit mit Thermodruckwerk, 30 Zeichen/s |
| | - Magnetband-Kassetteneinheit |



Datensichtgerät mit Hardcopy-Druckeinheit und Magnetbandkassetteneinheit

## BILDSCHIRM-TERMINAL BTH 1000

| | |
|---|---|
| Bildschirmgröße: | 12" diagonal (Anzeigefeld 9" x 5") |
| Bildschirmkapazität: | 960 Zeichen<br>12 Zeilen zu je 80 Zeichen |
| Eingabetastatur: | ASCII-Code mit Sonderzeichen |
| Schnittstelle: | V24, 1200...9600 Baud |
| Netzanschluß: | 220 V 50 Hz 350 W |
| Abmessungen: | Tiefe 560 mm<br>Breite 470 mm<br>Höhe 318 mm. |

## MOSAIK-SCHNELLDRUCKER

| | |
|---|---|
| Geschwindigkeit: | 200 Zeilen/min |
| Anzahl der Kolonnen: | 132 |
| Anzahl der Zeichen: | 64 Standard-Zeichen (USASCII) |
| Zeichendichte: | 10 Zeichen/Zoll |
| Zeichenart: | Punktmatrix 5 x 7 |
| Papierarten: | Typ Endlospapier gefaltet, mit perforierten Kanten<br>Breite 4 bis 14 7/8 Zoll, Anzahl der Kopien max. 6 |
| Formularsteuerung: | Standard-8-Kanal-Lochstreifen |
| Schnittstelle: | TTL-Level |
| Abmessungen: | Breite 71,1 cm<br>Tiefe 58,4 cm<br>Höhe 27,9 cm |
| Gewicht: | 68 kg |
| Netzanschluß: | 220 V $\pm$10 % 50 Hz |
| Leistungsaufnahme: | max. 800 W |

## LOCHSTREIFENLESER

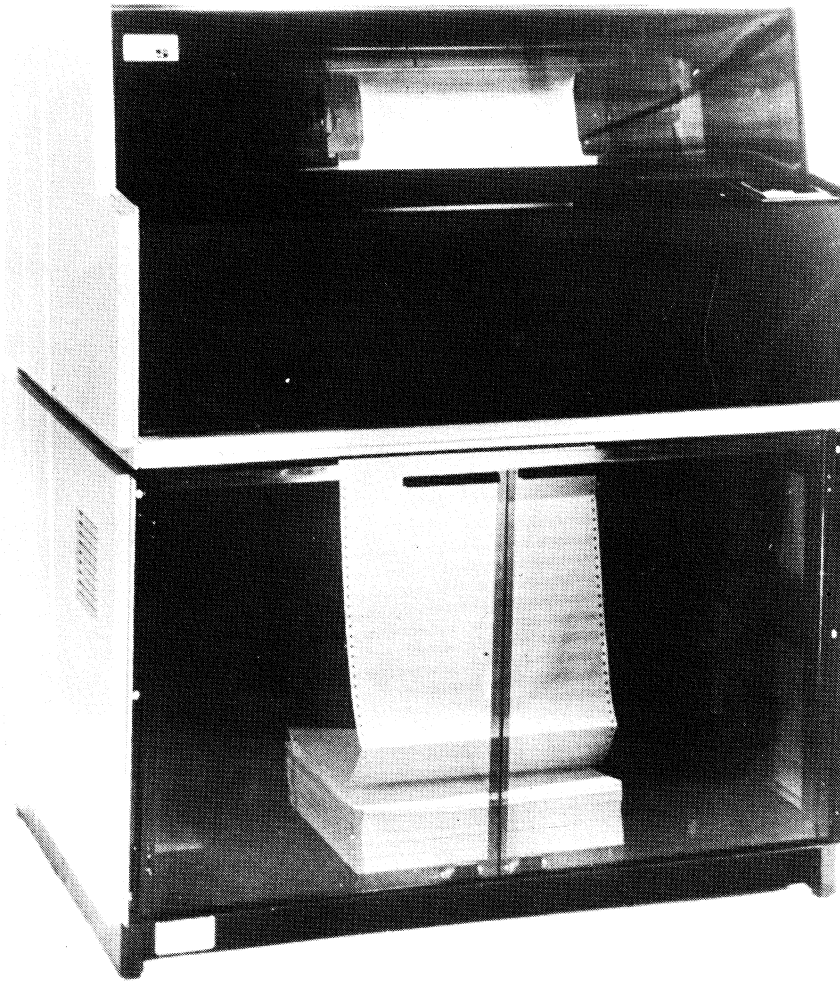
Geschwindigkeit: 125 Zeichen/s / 300 Zeichen/s
Schrittmotor-Antrieb für Vorwärts/Rückwärts-Lesen
Optische Lochstreifenabtastung
1" Lochstreifen, 5 bis 8 Kanäle
mit Spuleinrichtung
19"-Einbaurahmen (5 / 7 Höheneinheiten)
Netzanschluß: 220 V, ±10 %, 50 Hz, 110 / 135 W

## LOCHSTREIFENSTANZER

Geschwindigkeit: 75 Zeichen/s
Schrittmotor-Antrieb für Vorschub
1" Lochstreifen, 0.08...0.11 mm Streifendicke
Streifenarten: Papier, ISO-Norm, ca. 300 m Streifenlänge (120.000 Zeichen)
mit Aufwickelvorrichtung und automatischer Streifenvorratskontrolle
19"-Einbaurahmen (6 Höheneinheiten)
Netzanschluß: 220 V, ±10 %, 50 Hz, 100 W, max. 200 W

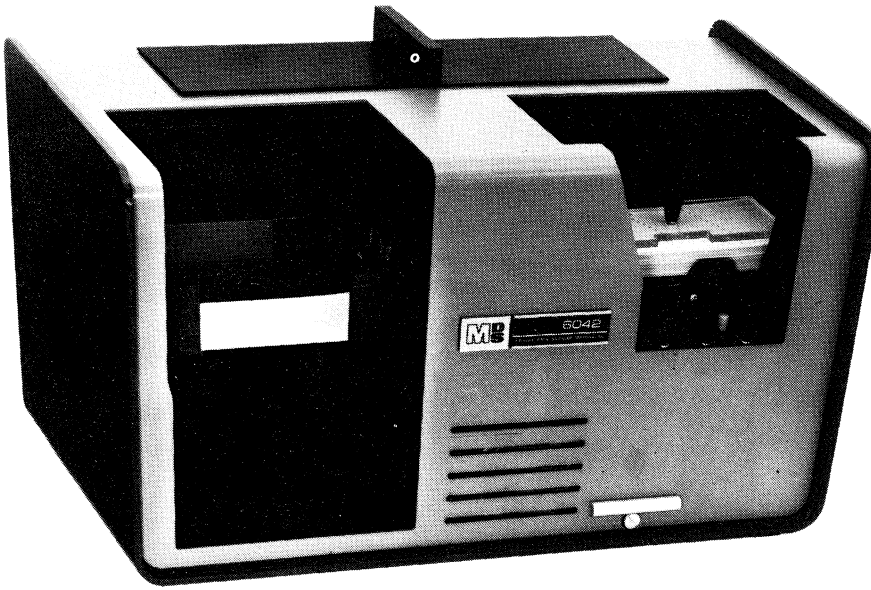
KETTEN-SCHNELLDRUCKER

| | |
|---|---|
| Druckgeschwindigkeit | 300 Zeilen/min |
| Schreibbreite | 132 Zeichen pro Zeile |
| Schnittstelle | TTL-Level |
| Abmessungen | 960 mm Höhe, 800 mm Breite, 600 mm Tiefe |
| Netzanschluß | 220 V, $\pm$10 %, 50 Hz, 5 A |



## LOCHKARTEN-STAPELLESER

| | |
|---|---|
| Lesegeschwindigkeit | 400 Karten/min |
| Lochkarten | 80-spaltige Standard-Lochkarten |
| Leseprinzip | fotoelektrische Abtastung |
| Stapelkapazität | je 500 Karten |
| Abmessungen | 318 mm hoch, 318 mm tief, 585 mm breit |
| Netzanschluß | 220 V, $\pm$10 %, 50 Hz, 400 VA |



8-KANAL-FERNSCHREIBER

als Ein/Ausgabe-Gerät mit

- Tastatur und Druckwerk für 64 Zeichen
- Geschwindigkeit 10 Zeichen/s, 110 Baud
- Schreibbreite 72 Zeichen/Zeile
- eingebautem Lochstreifenleser/stanzer für 1" Standard-Lochstreifen
- Leser-Einzelschritt-Betrieb
- V24-Schnittstelle
- zusätzlicher Funktionstaste für Rechnerstart
- Netzanschluß 220 V, 50 Hz, ca. 280 VA
  bei Anlauf kurzzeitig 500 VA
- Abmessungen 560 x 840 x 470 mm
- Wartung alle 750 Stunden, spätestens jedoch alle 6 Monate
  (eingebauter Betriebsstundenzähler).

Optionen:

- automatische Motor-Ein/Ausschaltung
- Stachelradwalze für 8 1/2" randperforierte Endlos-Formulare
- geräuschgedämpftes Tischgehäuse


8-Kanal-Fernschreiber in geräuschgedämpftem Tischgehäuse

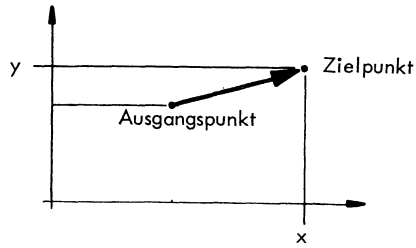
# Graphische Ausgabe

An den DIETZ 621 können Inkrementalplotter, XY-Schreiber oder Speicheroszillographen angeschlossen werden. Für diese Geräte sind Programmoduln erhältlich, die folgende Funktionen realisieren:

```
HOME
PLOT (X,Y,Z)
SYMB (h, s)
```

HOME bewirkt, daß die Ausgangsstellung (Koordinaten-Nullpunkt) eingenommen bzw. der Speicherbildschirm gelöscht wird.

PLOT läßt den Schreibstift bzw. den Strahl vom jeweiligen Ausgangspunkt zum Zielpunkt mit den Koordinaten (X, Y) wandern. Ist Z gleich Ø, so geschieht dies nichtschreibend; ist Z gleich 1, so wird zwischen Ausgangs- und Zielpunkt linear interpoliert und geschrieben:




SYMB ist ein Befehl zur graphischen Ausgabe von alphanumerischem Text (große Buchstaben, Ziffern, einige Sonderzeichen). Der Parameter h gibt die Schrifthöhe an, der Parameter s den Schriftzug in Form eines Textstrings oder einer String-Variablen. Der Schriftzug beginnt am jeweiligen Ausgangspunkt und verläuft parallel zur X-Achse.

Für den Anschluß der Geräte stehen folgende Interfaces zur Verfügung:

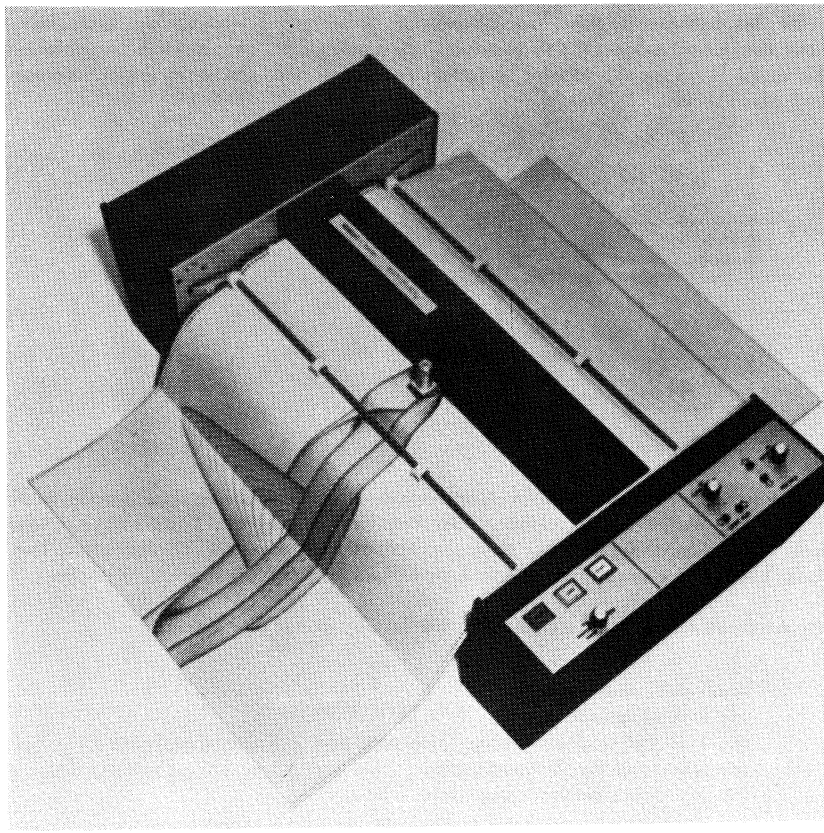
- Analog-XY-Interface (IAXY)
  mit 2 Analogausgängen 0...1 V $R_i$ = 700 Ohm
  mit 2 10-bit-Digital-Analog-Umsetzern und 2 Kaskadenregistern
  mit Steuerung für Z-Koordinaten
  (für XY-Schreiber und graphische Displays)

- Digital-Plotter-Interface (IPLOT)
  mit Impulsausgängen für XY-Koordinaten und Federsteuerung
  (für Digital-Plotter).

## DIGITAL-PLOTTER

Digital-Inkremental-Plotter

| | |
|---|---|
| Geschwindigkeit | 300 Schritte/s |
| Schrittgröße | 0.1 oder 0.25 mm |
| Papierformat | DIN A4<br>DIN A3, 2-gefaltetes Endlospapier<br>DIN A2, 2-gefaltetes Endlospapier |



## ALPHANUMERISCH-GRAPHISCHES DATENSICHTGERÄT

| | |
|---|---|
| Bildschirmgröße | 11" Speicherröhre |
| Alphanumerische Darstellung | 35 Zeilen mit je 72 Zeichen<br>63 ASCII-Zeichen<br>5 x 7 Segment-Darstellung<br>Cursor-Adressierung |
| Graphische Darstellung | Vektor-Betriebsart<br>1024 (10 bit) adressierbare Punkte |
| mit Tastatur und Standfuß | |
| Schnittstelle | V24, 1200 Baud |
| Netzanschluß | 220 V, 50 Hz |
| Option | Hardcopy-Einheit |

# Magnetband-Systeme

Als Massenspeicher zur Archivierung großer Datenmengen steht für den DIETZ 621 eine Reihe von Magnetbandeinheiten verschiedener Geschwindigkeiten, Bitdichten und Spulengrößen zur Verfügung.

LAUFWERKE

- MBE 7840 - 9-Spur-Magnetbandlaufwerk

| | |
|---|---|
| Schreibdichte: | 800 cpi |
| Bandgeschwindigkeit: | 12,5...25 ips |
| Spulendurchmesser: | 7" (7200" Bandlänge) |
| Betriebsart: | Read after Write |
| Aufzeichnung: | IBM-kompatibel |
| Baugröße: | 19"-Einschub (5 Einheiten hoch) |
| Stromversorgung: | eingebaut |
| Netzanschluß: | 220 V $\pm$10 % 50 Hz |
| Leistungsaufnahme: | ca. 300 VA |
| Temperaturbereich: | 0...50°C |

- MBE 7970 B/C - 9-Spur-Magnetbandlaufwerk

| | |
|---|---|
| Schreibdichte: | 800 cpi |
| Bandgeschwindigkeit: | 12,5...25 ips |
| Spulendurchmesser: | 10.5" (28800" Bandlänge) |
| Betriebsart: | Read after Write |
| Aufzeichnung: | IBM-kompatibel |
| Baugröße: | 19"-Einschub (14 Einheiten hoch) |
| Stromversorgung: | eingebaut |
| Netzanschluß: | 220 V $\pm$10 % 50 Hz |
| Leistungsaufnahme: | ca. 400 VA |
| Temperaturbereich: | 0...50°C |

- MBE 7970 E - 9-Spur-Magnetbandlaufwerk

| | |
|---|---|
| Schreibdichte: | 1600 cpi |
| Bandgeschwindigkeit: | 12.5...25 ips |
| Spulendurchmesser: | 10.5" (28800" Bandlänge) |
| Betriebsart: | Read after Write |
| Aufzeichnung: | IBM-kompatibel |
| Baugröße: | 19"-Einschub (14 Einheiten hoch) |
| Stromversorgung: | eingebaut |
| Leistungsaufnahme: | ca. 400 VA |
| Temperaturbereich: | 0...50°C |

Anschluß der Magnetbandlaufwerke über:

- Einkarten-Interface
  eingebaut in UNIVERSAL-INTERFACE-EINHEIT

| | |
|---|---|
| Datentransfer: | vom Programm gesteuert |
| Betriebsarten: | Lesen, Schreiben, Rewind, Backspace und Filemark-Schreiben |
| Adressen: | '1Ø2X..'1Ø5X |
| Zahl der Laufwerke pro Interface: | 1...4 |
| Stromaufnahme: | +5 V ca. 600 mA |

- Controller
  zum direkten Anschluß an den UNIVERSAL-BUS

| | |
|---|---|
| Datentransfer: | im direkten Speicherzugriff (DMA) |
| Betriebsarten: | Lesen, Schreiben, schneller Vorlauf, schneller Rücklauf, Schreiben File-Mark, Backspace |
| Adressen: | '1Ø2X..'1Ø5X |
| Zahl der Laufwerke pro Controller: | 1...4 |
| Baugröße: | 19"-Einschub (3 Einheiten hoch) |
| Netzanschluß: | 220 V $\pm$10 % 50 Hz |
| Leistungsaufnahme: | ca. 100 VA |

Auf einer Karte des Controllers sind von vorn zwei Schalter zugänglich, die für Testzwecke eingebaut sind:

| | |
|---|---|
| Oberer Schalter: | Schalthebel nach oben: Inkrementieren des Adreßzählers verhindert |
| | Schalthebel nach unten: Inkrementieren des Adreßzählers zugelassen |
| Unterer Schalter: | Schalthebel nach oben: Inkrementieren des Blocklängenzählers verhindert |
| | Schalthebel nach unten: Inkrementieren des Blocklängenzählers zugelassen |

Das Band-System beschreibt das Band kompatibel zu den Bändern der Serie 360 von IBM.

Folgende Befehle sind möglich:

Lesen vom Band
Schreiben auf das Band
Zurückspulen bis zum Anfang (Rewind)
Zurücksetzen um einen Block (Backspace)
Schreiben einer Filemark
Schneller Vorlauf (nur bei Verwendung eines Controllers)

SYSTEM MIT EINKARTEN-INTERFACE

Der Aufruf des Band-Systems erfolgt durch einen Unterprogramm-Sprung in einer beliebigen Ebene. Die Parameter werden in folgenden Registern übergeben:

| | |
|---|---|
| Akku ⓪ = Betriebsart | Ø = Lesen, 1 = Schreiben |
| | 2 = Rewind, 3 = Backspace |
| | 4 = Filemark-Schreiben |
| Reg 3, 4 | KS-Adresse |
| Reg 5, 6 | Blocklänge bzw. Blockzahl bei Backspace |
| Reg 7 | frei |
| Reg 8, 9 | Rückkehradresse im CSA-Aufruf |
| Reg 'ØA, 'ØB | Rückkehradresse im Fehlerfall |

Ein erkannter Fehler wird im Akku @ übergeben.

| Fehler-Nr. | Fehlerart |
|---|---|
| 1 | nicht On-Line |
| 2 | nicht Ready |
| 3 | Parity-Fehler |
| 4 | File Mark gelesen |
| 5 | Blocklänge zu klein |
| 6 | Blocklänge zu groß |
| 7 | Begin of Tape (BOT) |
| 8 | Band leer |
| 9 | KS-Adresse < '4000 |
| 10 | Länge < 16 Byte |
| +16 | End of Tape (EOT) |

Der Aufruf des Systems erfolgt durch einen Unterprogrammsprung in einer beliebigen Ebene.

| | |
|---|---|
| KS-Bereich: | relativ |
| Länge: | ca. 1 K |
| Aufruf: | CSA, '3E, TOS; |
| Parameter: | Reg @ ...'6 |
| | <@> = Unit (Ø...3) |
| benutzte Register: | @ ...'17 |

| Programmbefehl | Einsprung | KS-Basis-Adr | Länge | Anzahl |
|---|---|---|---|---|
| READ | 'XØØØ | Reg 3, 4 | Reg 5,6 | - |
| WRITE | 'XØØ2 | Reg 3, 4 | Reg 5,6 | - |
| WRITE FILEMARK | 'XØØ4 | - | - | - |
| REWIND | 'XØØ6 | - | - | - |
| BACKSPACE | 'XØØ8 | - | - | Reg 3, 4 |
| FORWARD | 'XØØA | - | - | Reg 3, 4 |
| BACKSPACE to FILEMARK | 'XØØ8 | - | - | <3,4> = Ø [1] |
| FORWARD to FILEMARK | 'XØØA | - | - | <3,4> = Ø [2] |

1) Nach Ausführung steht Kopf vor der File Mark

2) Nach Ausführung steht Kopf hinter der File Mark

Ein erkannter Fehler wird im Akku @ übergeben.

| Fehler-Nr. | Fehlerart |
|---|---|
| 1 | nicht On-Line |
| 2 | nicht Ready |
| 3 | --- |
| 4 | Begin of Tape |
| 5 | Parity |
| 6 | angegebene Blocklänge zu klein |
| 7 | Schreibversuch trotz Schreibsperre |
| 8 | keine Betriebsart erkannt |
| Fehler-Nr. | Warnung |
| 9 | angegebene Blocklänge zu groß |
| 10 | Filemark gelesen |
| 13 | End of Tape |
| >13 | Fehler + End of Tape |

Benutzte Register: 'ØØ...'6Ø.

# Digitale Ein- und Ausgänge

VORBEMERKUNG

Alle digitalen Ein- und Ausgänge sind als EINKARTEN-Interfaces gebaut, die in eine Universal-Interface-Einheit eingesteckt werden.

Grundsätzlich gibt es alle Ein- und Ausgänge in folgenden Versionen:

TTL-Schnittstelle
HTL-Schnittstelle
HTL-Schnittstelle potentialgetrennt über Fotokoppler
Reed-Relais-Schnittstelle (potentialgetrennt)

Nur bei den digitalen Ausgängen sind TTL- und HTL-Schnittstelle identisch.

Die TTL-Schnittstelle wird verwendet, wenn schnelle Elektronik (in TTL) bedient wird und keine Störungen von außen in das System gelangen.

Die HTL-Schnittstelle dient zum Anschluß an Logik mit höherem Signalpegel (12...30 V) und ist unempfindlich gegen statische und dynamische Störungen.

Potentialtrennung wird immer dann verwendet, wenn Erdschleifen nicht mit Sicherheit vermieden werden können oder wenn Gefahr besteht, daß andere Störungen durch die Peripherie in das System gelangen können.

Bei schnellen Signalen verwendet man eine Potentialtrennung durch Fotokoppler, bei langsamen Signalen eine durch Reed-Relais.

Die Versorgung der potentialgetrennten Elektronik erfolgt durch Zusatzstromversorgungen in der UIE oder durch externe Stromversorgungen.

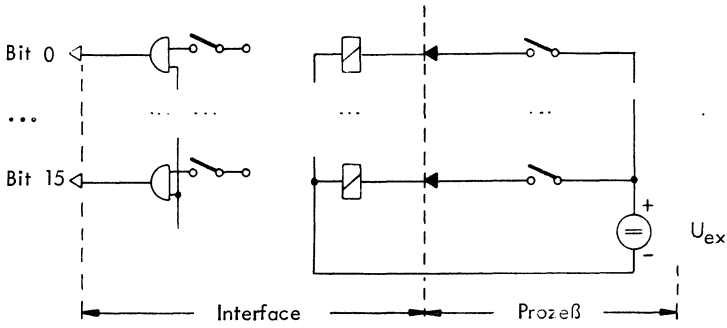
Bei allen Ein- und Ausgängen sind Log.1 $\hat{=}$ 0 V und Log.Ø $\hat{=}$ externe Spannung oder Log.1 $\hat{=}$ geschlossener Kontakt und Log.Ø $\hat{=}$ geöffneter Kontakt.

## STATISCHE EINGÄNGE

Je 16 Eingänge sind auf einem Baustein zusammengefaßt. Jeder Eingang besteht aus einem Eingangsverstärker und einem Schalter, der das jeweilige Eingangssignal auf den Datenkanal des Universal-BUS durchschaltet.

Jeweils 8 Eingänge werden mit einer Adresse gleichzeitig durchgeschaltet.

Beispiel: Statischer digitaler Eingang mit Reed-Relais.




Adressen der statischen Eingänge: '3ØØX...'33FX

| | |
|---|---|
| X = 1 | Durchschalten der Eingänge 1 bis 8 |
| X = 2 | " " " 9 bis 16 |

Folgende Bausteine sind verfügbar:

- 16-bit-Digitaleingang statisch/TTL (PSSE 16/5)
  Prozeß-Interface zur statischen Abfrage von 16 digitalen Eingangssignalen
  TTL-Schnittstelle (5 V)

- 16-bit-Digitaleingang statisch/HTL (PSSE 16/12.60)
  Prozeß-Interface zur statischen Abfrage von 16 digitalen Eingangssignalen
  HTL-Schnittstelle (12...60 V)

- 16-bit-Digitaleingang statisch/potentialfrei (PSSE 16/FK)
  Prozeß-Interface zur statischen Abfrage von 16 digitalen Eingangssignalen
  HTL-Schnittstelle (12...60 V)
  mit Potentialtrennung über Fotokoppler

- 16-bit-Digitaleingang statisch/Relais (PSSE 16/R)
  Prozeß-Interface zur statischen Abfrage von 16 digitalen Eingangssignalen
  über Relais ( ≥ 12 V, 15 mA)
  16 Spulenanschlüsse + 1 gemeinsame Rückleitung

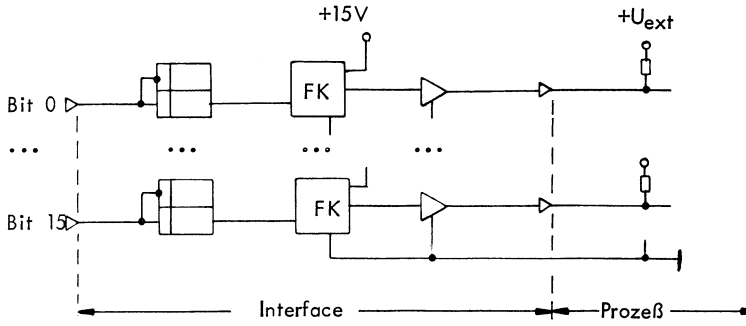
- 32-bit-Digitaleingang statisch/TTL (PSSE 32/5)
  Prozeß-Interface zur statischen Abfrage von 32 digitalen Eingangssignalen
  TTL-Schnittstelle (5 V)

## SPEICHERNDE AUSGÄNGE

Je 16 Ausgänge sind auf einem Einkarten-Interface zusammengefaßt. Jeder Ausgang besteht aus einem Speicher (Flip-Flop) und dem jeweiligen Treiber.

Je 8 Ausgänge werden gleichzeitig vom Programm mit einer Adresse angesprochen. Jeder Ausgang ist einem Datenbit zugeordnet. Bei Datenbit = 1 wird der Ausgang gesetzt, bei Datenbit = Ø rückgesetzt. Ein Abfragen der eingeschriebenen Information ist nicht möglich.

Beispiel: Digitaler Ausgang mit Potentialtrennung über Fotokoppler.




Adressen der digitalen Ausgänge: '34ØX...'37FX.

X = 1 Ansprechen der Ausgänge 1... 8
X = 2 " " " 9...16

Folgende Bausteine sind verfügbar:

- 16-bit-Digitalausgang/TTL (PSSA 16)
  Prozeß-Interface zur Speicherung und Ausgabe von 16 Ausgangssignalen TTL-Schnittstelle (5...30 V, max. 80 mA)

- 16-bit-Digitalausgang/potentialfrei (PSSA 16 FK)
  Prozeß-Interface zur Speicherung und Ausgabe von 16 Ausgangssignalen TTL-Schnittstelle (5...30 V, max. 80 mA)
  mit Potentialtrennung über Fotokoppler

- 16-bit-Digitalausgang/Relais (PSKA 16 R)
  Prozeß-Interface zur Speicherung und Ausgabe von 16 Ausgangssignalen
  16 Kontaktausgänge von Reed-Relais
  1 Arbeitskontakt je bit mit gemeinsamer Rückleitung
  max. 110 V 0.5 A 10 W bei ohmscher Last.

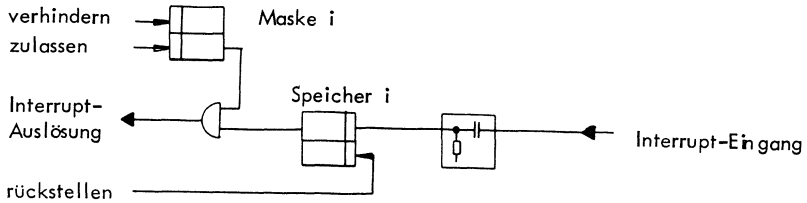
- 32-bit-Digitalausgang/TTL (PSSA 32)
  Prozeß-Interface zur Speicherung und Ausgabe von 32 Ausgangssignalen TTL-Schnittstelle (5...30 V, max. 80 mA)

## DYNAMISCHE EINGÄNGE

Über dynamische Eingänge stellt der Computer fest, daß eine Zustandsänderung (Signaländerung) eingetreten ist. Solche Änderungen verlangen im allgemeinen eine möglichst unverzögerte Reaktion des Rechners. Dies wird erreicht, indem bei einer Signaländerung ein Interrupt erzeugt wird. Man spricht deshalb auch von Interrupt-Eingängen.

Auf einem Baustein sind 8 dynamische Eingänge zusammengefaßt. Jeder Eingang besteht aus einem differenzierten Eingang und einem Speicher. Die Speicher eines Bausteins lösen im gesetzten Zustand den gleichen Interrupt aus (zu programmieren auf dem Device-Selector). Die Speicher sind vom Programm her abfragbar und rücksetzbar (Rücksetzen mit Datenbit = 1). Jedem Eingang ist ein Masken-Flip-Flop zugeordnet, das in gesetztem Zustand einen Interrupt zuläßt. Diese Maske ist vom Programm her setz- und rücksetzbar.

Beispiel: Dynamischer Eingang.



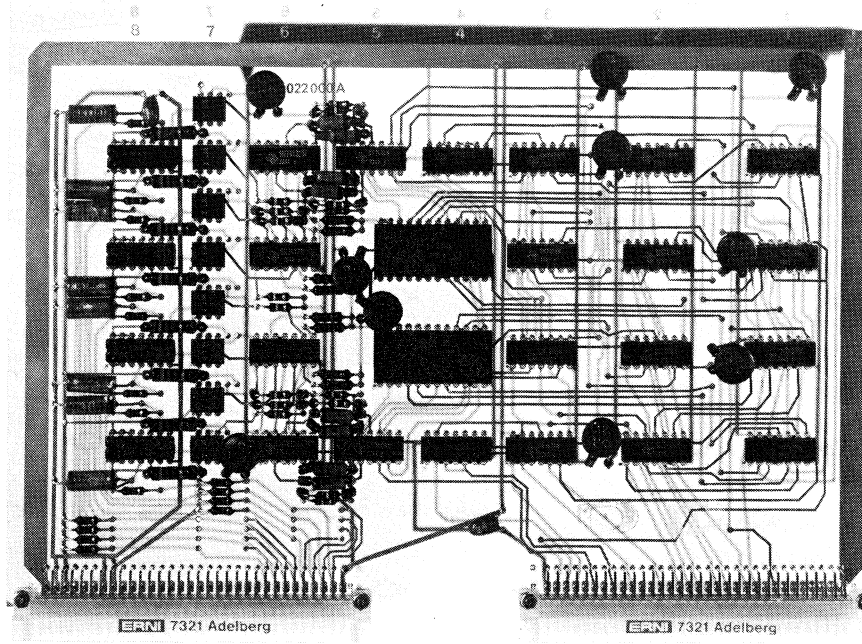

Adressen der dynamischen Eingänge: '38ØX... 38FX

| | |
|---|---|
| X = 1 | Abfragen der Interrupt-Speicher |
| X = 1 | Rücksetzen der Interrupt-Speicher (jeweiliges Datenbit = 1) |
| X = 4 | Setzen der Maske (Datenbit = 1 ≙ Zulassen Interrupt; Datenbit = Ø ≙ Verhindern Interrupt). |

Das Masken-Register ist nicht abfragbar.

Folgende Bausteine sind verfügbar:

- 8-bit-Digitaleingang dynamisch/TTL (PDSE 8/5)
  Prozeß-Interface zur Speicherung und Abfrage von 8 digitalen Eingangssignalen
  mit Differenziereingang, 8-bit-Speicher und Interrupt-Auslösung (verriegelbar)
  TTL-Schnittstelle (5 V)




8-bit dynamischer Digitaleingang (PDSE/FK)

- 8-bit-Digitaleingang dynamisch/HTL (PDSE/12.60)
  Prozeß-Interface zur Speicherung und Abfrage von 8 digitalen Eingangssignalen
  mit Differenziereingang, 8-bit-Speicher und Interrupt-Auslösung (verriegelbar)
  HTL-Schnittstelle (12...60 V)

- 8-bit-Digitaleingang dynamisch/potentialfrei (PDSE/FK)
  Prozeß-Interface zur Speicherung und Abfrage von 8 digitalen Eingangssignalen
  mit Differenziereingang, 8-bit-Speicher und Interrupt-Auslösung (verriegelbar)
  HTL-Schnittstelle (12...60 V)
  mit Potentialtrennung über Fotokoppler

- 8-bit-Digitaleingang dynamisch/Relais (PDSE/R)
  Prozeß-Interface zur Speicherung und Abfrage von 8 digitalen Eingangssignalen über Relais ( $\geq$ 12 V, 15 mA) mit Differenziereingang, 8-bit-Speicher und Interrupt-Auslösung
  (verriegelbar)

- 16-bit-Digitaleingang dynamisch/TTL (PDSE 16/5)
  Prozeß-Interface zur Speicherung und Abfrage von 16 digitalen Eingangssignalen
  mit Differenziereingang, 16-bit-Speicher und Interrupt-Auslösung (verriegelbar)
  TTL-Schnittstelle (5 V)

## ZÄHLER

Der Zähler dient zum Zählen schneller Impulse. Die Impulse werden in einen 16-bit-Zähler eingezählt. Vom Rechner ist der Stand des Zählers abfragbar und rücksetzbar. Bei einem Überlauf wird ein Carry-Flip-Flop gesetzt, das einen Interrupt auslöst, es sei denn, dies ist durch das Setzen eines ARM-Flip-Flops verhindert. Sind also mehr als 65536 Impulse zu zählen, so müssen die Interrupt-Signale vom Programm gezählt werden.

Es stehen 3 Eingangsschaltungen zur Verfügung:

| | | |
|---|---|---|
| TTL-Eingang | Zählfrequenz | 20 MHz |
| HTL-Eingang | " | 1 MHz |
| HTL-Eingang potentialfrei | " | 30 kHz |

Adressen der Zähler: '39ØX...'39FX

| | |
|---|---|
| X = 1 | Abfragen Zähler niederwertiges Byte |
| X = 2 | Abfragen Zähler höherwertiges Byte |
| X = 1 | Setzen Zähler niederwertiges Byte |
| X = 2 | Setzen Zähler höherwertiges Byte |
| X = 4 | Setzen Statusregister |
| X = 4 | Abfragen Statusregister |

Aufbau des Statusregisters:

Bit Ø = Carry (Interrupt)
Bit 1 = IBUSY (Aktivieren Zähler)
Bit 2 = Ø
Bit 3 = ARM

Folgende Bausteine sind erhältlich:

- 16-bit-Zähler (PIZE 16/5)
  zum Zählen von Eingangsimpulsen
  max. Zählfrequenz 20 MHz
  steuer-, abfrag- und setzbar
  TTL-Schnittstelle (5 V)

- 16-bit-Zähler (PIZE 16/15)
  zum Zählen von Eingangsimpulsen
  max. Zählfrequenz 1 MHz
  steuer-, abfrag- und setzbar
  HTL-Schnittstelle (15 V)

- 16-bit-Zähler (PIZE 16/15 FK)
  zum Zählen von Eingangsimpulsen
  max. Zählfrequenz 30 kHz
  steuer-, abfrag- und setzbar
  HTL-Schnittestelle (potentialfrei).

## STEUERZÄHLER

Der Steuerzähler dient zur Ausgabe von Signalen definierter Dauer. Hierzu wird ein 16-bit-Zähler vom Rechner gesetzt und von einem eingebauten Quarzoszillator weitergezählt, bis der Zähler auf Ø steht. Von einem Start durch den Rechner bis zum Erreichen des Zustands Ø steht am Ausgang des Steuerzählers ein Signal an. Bei Erreichen der Ø wird ein Interrupt an den Rechner ausgelöst, es sei denn, dies ist durch Setzen eines ARM-Flip-Flops verhindert.

Der Oszillator kann mit einem 10 MHz (Auflösung 100 ns) oder einem 1 MHz (Auflösung 1 us) bestückt werden. Der Baustein ist mit einem TTL-Ausgang und einem potentialgetrennten HTL-Ausgang lieferbar.

Die Zahl der Impulse, die die Dauer des Ausgangssignals bestimmen, müssen als Zweier-Komplement eingeschrieben werden.

Adressen des Steuerzählers: '39ØX...'39FX.

| | |
|---|---|
| X = 1 | Setzen Zähler niederwertiges Byte |
| X = 2 | Setzen Zähler höherwertiges Byte |
| X = 1 | Abfragen Zähler niederwertiges Byte |
| X = 2 | Abfragen Zähler höherwertiges Byte |
| X = 4 | Setzen Statusregister |
| X = 4 | Abfragen Statusregister |

Aufbau des Statusregisters:

Bit Ø = Carry (Interrupt)
Bit 1 = Ø
Bit 2 = OBUSY (Auslösung Zähler)
Bit 3 = ARM

Folgende Bausteine sind erhältlich:

- 16-bit-Steuerzähler (PISA 16)
  zur Ausgabe eines Steuersignals definierter Dauer
  mit 10 MHz- oder 1 MHz-Quarz, Auflösung 0.1 us oder 1 us
  vom Programm setz- und steuerbar, auch als Zähler benutzbar
  TTL-Schnittstelle (5...30 V)

- 16-bit-Steuerzähler (PISA 16/FK)
  zur Ausgabe eines Steuersignals definierter Dauer
  mit 10 MHz- oder 1 MHz-Quarz, Auflösung 0.1 us oder 1 us
  vom Programm setz- und steuerbar, auch als Zähler benutzbar
  HTL-Schnittstelle (5...30 V), potentialfrei

## WATCHDOG

Für Zwecke der Selbstüberwachung eines Rechner-Systems dient der Watchdog-Ausgang. Dieses Einkarten-Interface enthält eine Verzögerungsschaltung (1...60 sec einstellbar), die regelmäßig vom Programm angestoßen werden muß. Unterbleibt dies länger als die Verzögerungszeit, so wird der Watchdog-Ausgang wirksam, und ein äußerer Alarm kann ausgelöst werden.

Mit einem Schalter auf dem Baustein kann der Watchdog-Ausgang für die Inbetriebnahmephase unwirksam gemacht werden.

Als Ausgang steht ein Relais-Kontakt (Öffner + Schließer) zur Verfügung.

Die Ansprechadresse der Watchdog ist '3FDX.

- Watchdog-Ausgang (PW DOG)
  zum Überwachen von Prozeß-Systemen
  Relais-Ausgang 2 A/500 V/100 W

# Analoge Ein- und Ausgänge

VORBEMERKUNG

Die analogen Ein- und Ausgänge sind unterteilt in

- Einkarten-Interface: mittelschneller ADU
  DAU-Spannungsausgang
  DAI-Stromausgang
  2fach-DAI-Stromversorgung
- Aaalog-Meßsysteme: mittelschnelles ADU-System
  schnelles ADU-System
  integrierendes ADU-System

ANALOGE EINGÄNGE

Zur Erfassung eines oder mehrerer Meßsignale muß das Gleichspannungssignal von einem Analog-Digital-Umsetzer (ADU) in einen Digitalwert verwandelt werden, der vom Rechner verarbeitet werden kann.

Abhängig von den Forderungen: Genauigkeit, Meßgeschwindigkeit und Störsicherheit ist zu wählen zwischen:

- ADU (integrierend) mit hoher Auflösung, mit hoher Störunterdrückung für Gleichtakt- und Gegentaktstörungen, aber mit geringer Meßgeschwindigkeit

- ADU (Stufenverschlüßlung) geringere Auflösung, hohe Meßgeschwindigkeit, sehr geringe Unterdrückung von Gegentaktstörungen und beim Einkarten-ADU auch sehr geringe Unterdrückung von Gleichtaktstörungen.

Beim integrierenden ADU wird das Meßsignal über die Zeit einer Periode der Netzfrequenz (20 ms) integriert, so daß diese Frequenz, die meist die Hauptgegentaktstörung darstellt, ausgefiltert wird. Gleichtaktstörungen werden durch einen potentialfreien Eingang unterdrückt. Zur Meßstellenumschaltung wird ein Relais-Multiplexer benutzt.

Bei einem ADU, der nach dem Prinzip der Stufenverschlüßlung arbeitet, muß das Meßsignal für die gesamte Dauer der Umsetzung anliegen. Beim ADU-System wird die Meßsignalübernahmezeit auf ca. 1/8 der Umsetzzeit verkürzt, wenn ein "Sample and Hold"-Verstärker eingebaut wird. Hochfrequente Störungen werden während der Übernahmezeit durch einen Kondensator integriert. Niederfrequente Gegentaktstörungen werden nicht unterdrückt. Die Auswirkung von Gleichtaktstörungen wird durch potentialfreie Eingänge

verringert (nicht beim Einkarten-ADU). Zur schnellen Meßstellenumschaltung wird ein Halbleiter-Multiplexer benutzt. Das ADU-System kann mit einer DMA-Steuerung für selbstgesteuerte Messungen im Cycle-Stealing-Betrieb ausgestattet werden. Mit dieser Steuerung ist der ADU in der Lage, selbständig bis zu 256 aufeinanderfolgende Messungen durchzuführen. Dies ist in drei Arbeitsweisen möglich:

- Mehrkanal-Messung
  Nach jeder Messung wird der Meßstellen-Adreß-Zähler inkrementiert

- Einkanal-Messung
  Die gewünschte Anzahl von Messungen wird an einer Meßstelle durchgeführt.

- Zweikanal-Messung
  Die gewünschte Anzahl von Messungen wird abwechselnd zwischen zwei direkt benachbarten Meßstellen durchgeführt.

Folgende Analog-Eingänge sind verfügbar:

MITTELSCHNELLER EINKARTEN-AD-UMSETZER (ADE)

- Einkarten-Interface
  Analogeingang 12 bit/+10 V, ±5 V, ±10 V (ADE 12.010/ADE 12.505/ ADE 12.1010)
  Prozeß-Interface mit Analog-Digital-Umsetzer 12 bit
  Konversionszeit ca. 27 us
  Eingangsspannung +10 V, +5 V, ±10 V, nicht potentialfrei
  Eingangswiderstand 100 MOhm
  max. Fehler ±2 LSB

Adresse des ADE 12: '3AØX...'3AFX

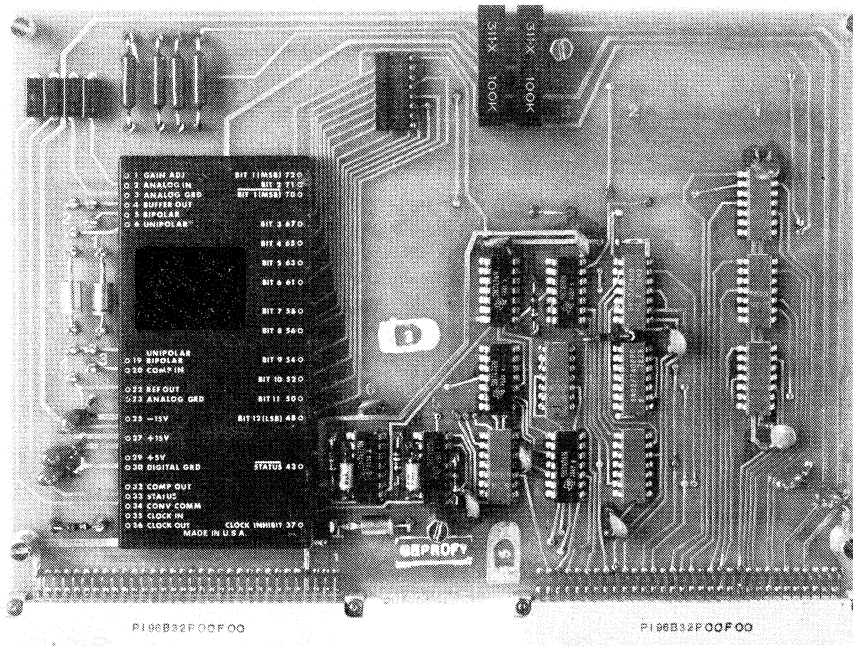
| | |
|---|---|
| X = 1 | Datendurchschaltung niederwertige Bits |
| X = 2 | Datendurchschaltung höherwertige Bits |
| X = 4 | Statusregister laden und abfragen |

Bitzuordnung der Daten:

| | höherwertig | | | | | | | | niederwertig | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Bit | 7 | 6 | 5 | 4 | 3 | 2 | 1 | Ø | 7 | 6 | 5 | 4 |
| Null | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø | Ø |
| Endwert | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| | MSB | | | | | | | | LSB | | | |

Aufbau des Statusregisters:

| | |
|---|---|
| Bit Ø | Ready |
| " 1 | IBUSY |
| " 3 | LOCK |




Analog-Eingang ADE 12.010

## EINKARTEN-RELAIS-MULTIPLEXER (MURE 16)

- Einkarten-Interface
  16 Eingänge, dreipolig
  HG-Reedrelais
  Schaltfrequenz 200 $Hz_{max}$

# MITTELSCHNELLES ANALOG-MESSYSTEM ADM 621

| | |
|---|---|
| Auflösung: | 12 bit (Stufenverschlüßler) |
| Konversionszeit: | ca. 33 us |
| Eingangswiderstand: | 100 MOhm |
| Meßbereich: | 0...+10 V, 0 ...+1 V (Option), -5...+ 5 V (Option), -0.5...+0.5 V (Option), -10...+10 V (Option), -1 ...+1 V (Option), 0...+5 V )Option), -2,5...+2,5 V (Option) } zusätzl. 1 LSB |
| max. Fehler: | ±2 LSB |
| Potentialtrennung: | zwischen Meßkreis und Logik durch Fotokoppler (Option) |
| Betriebsarten: | programmgesteuert, oder<br>selbstgesteuert (Option):<br>für Messung mehrerer Meßwerte bzw. Meßkanäle und Ablage der Digitalwerte im Kernspeicher (DMA)<br>vom Programm vorwählbar: Speicher-Basisadresse, Meßkanal-Basisadresse, Blocklänge, Einkanal-Messung/inkrementierende Mehrkanal-Messung/Zweikanal-Messung |
| Meßfrequenz: | ca. 30 kHz im selbstgesteuerten Betrieb mit MOS-FET-Multiplexer<br>ca. 200 Hz mit Relais-Multiplexer |
| Meßkanäle: | 1. MOS-FET-Multiplexer (Option)<br>1...4 Bausteine mit 8 oder 16 Eingängen, einpolig<br>(max. 64 Kanäle)<br><br>2. Relais-Multiplexer (Option)<br>1...4 Bausteine mit 8 Eingängen, zweipolig<br>HG-Reedrelais<br>Die Anzahl der Eingänge ist durch den unten beschriebenen Multiplexer-Einschub MU2 auf bis zu 244 Eingänge erweiterbar. |
| Sample-and-Hold: | Option<br>Eingangswiderstand $10^6$ MOhm |
| Stromversorgung: | eingebaut |
| Größe: | 19"-Einbaurahmen, offen (abgeschirmt), konvektionsbelüftet<br>Höhe 3 Einheiten (ca. 135 mm); Tiefe ca. 230 mm |
| Meßanschlüsse: | über rückseitige Steckverbindungn |
| Netzanschluß: | 220 V ±10 % 50 Hz ca. 70 VA |

## MULTIPLEXER MU2

| | |
|---|---|
| Meßkanäle: | 1...24 Bausteine mit 8 Eingängen<br>(max. 192 Eingänge)<br>zweipolig |
| Relais: | HG-Reedrelais<br>max. $10^9$ Schaltsignale |
| Schaltfrequenz: | max. 200 Hz |
| Stromversorgung: | eingebaut |
| Größe: | 19"-Einbaurahmen, konvektionsbelüftet<br>Höhe 3 Einheiten (ca. 135 mm); Tiefe ca. 420 mm |
| Meßanschlüsse: | über rückseitige Steckverbindungen |
| Netzanschluß: | 220 V $\pm$ 10 % ca. 10 VA |

# SCHNELLES ANALOG-MESSYSTEM ADS 621

| | |
|---|---|
| Auflösung: | 10 bit (Stufenverschlüßler)<br>8 bit ( " ) |
| Konversionszeit: | ca. 4 us |
| Meßbereich: | 0...+10 V<br>-5...+ 5 V (Option) |
| Eingangswiderstand: | 2.5 kOhm (ohne Sample-and-Hold)<br>max. Fehler 10 bit +3.5 LSB<br>8 bit ∓1.5 LSB |
| Potentialtrennung: | zwischen Meßkreis und Logik durch Fotokoppler (Option) |
| Betriebsart: | selbstgesteuert<br>für Messungen mehrerer Meßwerte bzw. Meßkanäle und Ablage im Kernspeicher<br>vom Programm vorwählbar: Speicher-Basisadresse, Meßkanal-Basisadresse, Blocklänge, Einkanal/inkrementierende Mehrkanal-Messung/Zweikanal-Messung |
| Meßfrequenz: | 235 kHz (10 bit)<br>250 kHz ( 8 bit) } im selbstgesteuerten Betrieb |
| Meßkanäle: | MOS-FET-Multiplexer (Option)<br>1...4 Bausteine mit 8 Eingängen<br>(max. 32 Kanäle) |
| Sampe-and-Hold: | Option<br>Eingangswiderstand $10^5$ MOhm |
| Stromversorgung: | eingebaut |
| Größe: | 19"-Einbaurahmen, offen (abgeschirmt), konvektionsbelüftet<br>Höhe 3 Einheiten (ca. 135 mm), Tiefe ca. 250 mm |
| Meßanschlüsse: | über rückseitige Steckverbindungen |
| Netzanschluß: | 220 V ∓10 % 50 Hz, ca. 70 VA |

Adressen von ADM 621 und ADS 621: '3FØX, '3F2X...'3F9X

Beim ADM 621 ohne Selbststeuerung:

| | |
|---|---|
| X = 1 | Datendurchschaltung niederwertige Bits |
| X = 2 | Datendurchschaltung höherwertige Bits |
| X = 3 | Meßstellenadresse im ADM-Einschub |
| X = 4 | Statusregister |
| X = 5 | Meßstellenadresse im MU2-Einschub |

Beim ADM 621 und beim ADS 621 mit Selbststeuerung:

| | |
|---|---|
| X = Ø | Meßstellenadresse im AD-Einschub |
| X = 1 | Kernspeicher-Basisadresse niederwertige Bits |
| X = 2 | Kernspeicher-Basisadresse höherwertige B-its |
| X = 3 | Blocklängenzähler |
| X = 4 | Statusregister |
| X = 5 | Meßstellenadresse im MU2-Einschub |

Die Bitzuordnung der Daten und der Aufbau des Statusregisters beim ADM ohne Selbststeuerung entsprechen der Festlegung, wie sie oben beim ADE aufgezeigt wurde. Beim ADM mit Selbststeuerung und beim ADS ist das Statusregister wie folgt aufgebaut:

| | |
|---|---|
| Bit Ø | READY |
| " 1 | IBUSY |
| " 3 | LOCK |
| " 4 | Externe Starterzeugung |
| " 5 | Mehrkanal-Messung |
| " 6 | Zweikanal-Messung |

Bit 5 = Ø, Bit 6 = Ø: Einkanal-Messung

Die Bitzuordnung der Multiplexer-Kanäle:

1) Im AD-Einschub:

| Bit 5 4 3 2 1 Ø | |
|---|---|
| Ø Ø Ø Ø Ø Ø | MU Kan.1 |
| Ø Ø Ø Ø Ø 1 | MU Kan.2 |
| . | . |
| . | . |
| . | . |
| 1 1 1 1 1 1 | MU Kan.64 |

2) Im MU2-Einschub:

| Bit 7 6 5 4 3 2 1 Ø | |
|---|---|
| Ø Ø Ø Ø Ø Ø Ø Ø | MU Kan.0 |
| Ø Ø Ø Ø Ø Ø Ø 1 | MU Kan.1 |
| ⋮ | |
| 1 Ø 1 1 1 1 1 1 | MU Kan.191 |

Der Ausgangs-Code von ADE, ADM und ADS:

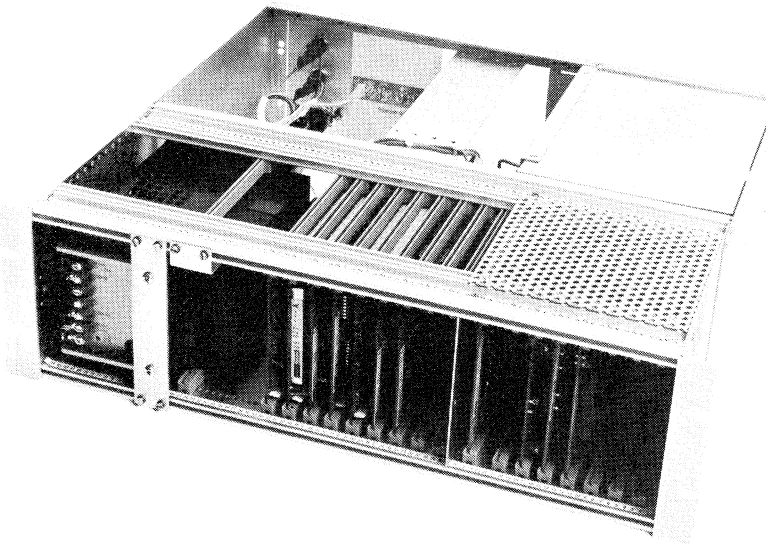
| | Digitalausgang | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| ADU-Bit | MSB | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | LSB |
| Rechner-Bit | ID7 | .6 | .5 | .4 | .3 | .2 | .1 | .Ø | .7 | .6 | .5 | .4 |

Unipolar (Binär)

| | |
|---|---|
| Endwert | 1 1 1 1 1 1 1 1 1 1 1 1 |
| halber Endwert | 1 0 0 0 0 0 0 0 0 0 0 0 |
| kleinste digitale Einheit | 0 0 0 0 0 0 0 0 0 0 0 1 |
| Null | 0 0 0 0 0 0 0 0 0 0 0 0 |

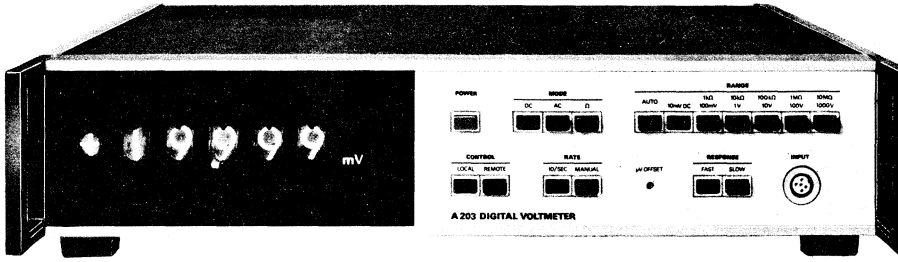
Bipolar (Offset Binär)

| | |
|---|---|
| positiver Endwert | 1 1 1 1 1 1 1 1 1 1 1 1 |
| 1/2 positive kleinste digitale Einheit | 1 0 0 0 0 0 0 0 0 0 0 0 |
| 1/2 negative kleinste digitale Einheit | 0 1 1 1 1 1 1 1 1 1 1 1 |
| negativer Endwert | 0 0 0 0 0 0 0 0 0 0 0 0 |



Analog-Meßsystem ADS 621

# INTEGRIERENDES MESSYSTEM ADI/200 - ADA 203
# ADI/210 - ADA 213




| | | |
|---|---|---|
| Typ: | ADI/200<br>ADA/203 | ADI/210<br>ADA/213 |
| Auflösung: | 19.999 | 119.999 |
| | Dual-Ramp-Integrierendes Verfahren | |
| Meßbereiche: | 10 mV= bis 1000 V=<br>100 mV~ bis 1000 V~<br>1 kOhm bis 10 MOhm } nur bei ADA/203 | 100 mV= bis 1000 V=<br>1 V~ bis 1000 V~<br>1 kOhm bis 10 MOhm } nur bei ADA/213 |
| Meßfolge: | 10 Messungen/s | 10 (25/100) Messungen/s |
| Meßkanäle: | Relais-Multiplexer, 3- bis 8-polig<br>bis 100 Kanäle pro Einschub<br>(max. 1000 Kanäle) | |
| Rechneranschluß: | über zwei Einkarten-Interfaces (Einbau in UIE) | |
| Betriebsart: | programmgesteuert | |
| Größe: | 19"-Einbaurahmen<br>ADU: 2 Höheneinheiten<br>je Multiplexer-Einschub: 4 Höheneinheiten | |
| Netzanschluß: | 220 V $\pm$10 % 50 Hz | |

Adresse des ADI: '3F1X

| | |
|---|---|
| X = 1<br>X = 2<br>X = 3 | Meßergebnis |
| X = 4 | Statusregister |
| X = 5 | Steuerregister |

Bitzuordnung des Meßergebnisses:

| Signal | Adresse | | Bit |
|---|---|---|---|
| | Haupt | Einer | |
| $10^0$1 | '3F1 | 1 | Ø |
| 2 | | 1 | 1 |
| 4 | | 1 | 2 |
| 8 | | 1 | 3 |
| $10^1$1 | | 1 | 4 |
| 2 | | 1 | 5 |
| 4 | | 1 | 6 |
| 8 | | 1 | 7 |
| $10^2$1 | | 2 | Ø |
| 2 | | 2 | 1 |
| 4 | | 2 | 2 |
| 8 | | 2 | 3 |
| $10^3$1 | | 2 | 4 |
| 2 | | 2 | 5 |
| 4 | | 2 | 6 |
| 8 | | 2 | 7 |
| $10^4$1 | | 3 | Ø |
| 2 | | 3 | 1 |
| 4 | | 3 | 2 |
| 8 | | 3 | 3 |
| $10^5$1 | | 3 | 4 |
| Polarity | | 3 | 7 |

Bitzuordnung des Statusregisters:

| Statusregister | Adresse | Bit | Bemerkung |
|---|---|---|---|
| READY | '3F14 | Ø | Rückmeldung DVM startet Niv. |
| IBUSY | '3F14 | 1 | startet das DVM |
| ARM | '3F14 | 3 | Startverriegelung |
| RATE Commands | '3F14 | 4 | Meßfolge |
| | '3F14 | 5 | " |

Bitzuordnung der Steuerregister:

| Steuerregister | Adresse | Bit | Bemerkung |
|---|---|---|---|
| Range Code 1 | '3F15 | ø | } Meßbereichsvorwahl |
| Range Code 2 | | 1 | |
| Range Code 4 | | 2 | |
| Data is changed | | 3 | Meßbereich wurde geändert |
| Slow/Fast | | 4 | langsam/schnell |
| Remote | | 5 | Fernbedienung |
| Mode-Commands | | 6 | } Art der Messung (=, ~ , Ohm) |
| | | 7 | |

Codierung der Meßbereiche:

| Meßbereich | | | Range-Code 4 Bit 2 | 2 Bit 1 | 1 Bit 0 |
|---|---|---|---|---|---|
| 1000 V= | 1000 V ~ | 10 MOhm | 0 | 0 | 0 |
| 100 V= | 100 V ~ | 1 MOhm | 0 | 0 | 1 |
| 10 V= | 10 V ~ | 100 kOhm | 0 | 1 | 0 |
| 1 V= | 1 V ~ | 10 kOhm | 0 | 1 | 1 |
| 100 mV= | 100 mV ~ | 1 kOhm | 1 | 0 | 0 |
| 10 mV= | - | - | 1 | 0 | 1 |

Codierung der Meßfolge:

| Meßfolge | Bit 5 | Bit 4 |
|---|---|---|
| 10/sec | 1 | 1 |
| 25/sec | 0 | 1 |
| 100/sec | 1 | 0 |

Codierung der Art der Messung:

| Art der Messung | Bit 7 | Bit 6 |
|---|---|---|
| DC | 1 | 1 |
| AC | 0 | 1 |
| Ohm | 1 | 0 |

Das Ansprechen und die Adressierung des Relais-Multiplexers erfolgen wie bei digitalen Ausgängen (Adressen: '34ØX...37FX).

Bitzuordnung der Meßstellen:

| Meßstelle 3-stellig, BCD | | Adresse | Bit |
|---|---|---|---|
| $10^0$ | 1 | '3␣␣1 | 0 |
| | 2 | | 1 |
| | 4 | | 2 |
| | 8 | | 3 |
| $10^1$ | 1 | | 4 |
| | 2 | | 5 |
| | 4 | | 6 |
| | 8 | '3␣␣1 | 7 |
| $10^2$ | 1 | '3␣␣2 | 0 |
| | 2 | | 1 |
| | 4 | | 2 |
| | 8 | '3␣␣2 | 3 |

| Meßstelle | $10^2$ | $10^1$ | $10^0$ |
|---|---|---|---|
| 1 | Ø | Ø | Ø |
| usw. | | | |

## ANALOGE AUSGÄNGE

Zur Ausgabe von Analogwerten durch den Rechner ist eine Umwandlung von Digitalwerten in einem Digital-Analog-Umsetzer (DAU) erforderlich. Der Analogwert kann als Spannungswert oder als Stromwert ausgegeben werden. Es sind daher drei Typen von Analogausgängen verfügbar:

- Analogausgang 10 bit/10V (DAU 1010)
  Prozeß-Interface mit 10-bit-Register, Digital-Analog-Umsetzer und Verstärker
  Ausgangsspannung 0...10 V, niederohmig

- Analogausgang 10 bit/20 mA (DAI 1020)
  Prozeß-Interface mit 10-bit-Register, Digital-Analog-Umsetzer und Verstärker
  Ausgangsstrom 0...20 mA, max. Bürde 400 Ohm

- Analogausgang zweimal 8 bit/20 mA (2-DAI-820)
  Prozeß-Interface mit 2 Systemen, je ein 8-bit-Register mit Digital-Analog-Umsetzer und Verstärker
  Ausgangsstrom 0...20 mA, max. Bürde 400 Ohm

Adressen des DAU und des DAI: '3BØX...'3BFX

X = 1 Daten niederwertige Bits, beim 2-ADI-820 Daten des Systems 1
X = 2 Daten höherwertige Bits nach Ausgabe des Analogwertes, beim 2-ADI-820 Daten des Systems 2

Die niederwertigen Bits sind kaskadiert und werden gleichzeitig mit den höherwertigen Bits gewandelt.

Bitzuordnung der Daten:

| | höherwertig | niederwertig |
|---|---|---|
| | 7 6 5 4 3 2 1 Ø | 7 6 |
| z.B. Endwert | 1 1 1 1 1 1 1 1 | 1 1 |
| | MSB | LSB |

# Datenfernübertragung

Bei der Datenübertragung von einem Computer zu einem anderen verwendet man bei größeren Entfernungen Telefonleitungen. Allerdings ist das nur mit Modems (Modulator/ Demodulator) möglich (bei galvanisch durchverbundener Leitung bis max. 30 km auch Gleichstromübertragungselemente, z.B. GDN).

Auf der Geräteseite sind diese Einrichtungen mit der genormten Schnittstelle V24 ausgerüstet.

Die erhältlichen Datenübertragungs-Interfaces bedienen folgende V24-Signale:

| | | | |
|---|---|---|---|
| E1 | Schutzerde | Stift-Nr. | 1 |
| E2 | Betriebserde | | 7 |
| D1 | Sendedaten | | 2 |
| D2 | Empfangsdaten | | 3 |
| S1 | Übertragungsleitung anschalten | | 20 |
| S2 | Sendeteil einschalten | | 4 |
| S3 | Empfangsteil ausschalten | | 18 |
| S4 | Hohe Übertragungsgeschwindigkeit einschalten | | 23 |
| M1 | Betriebsbereitschaft | | 6 |
| M2 | Sendebereitschaft | | 5 |
| M3 | Ankommender Ruf | | 22 |
| M5 | Empfangssignalpegel | | 8 |
| T2 | Sendeschrittakt | | 15 |
| T4 | Empfangsschrittakt | | 17 |

Die Datenübertragung erfolgt seriell entweder asynchron oder synchron.

Bei der asynchronen Übertragung besteht jedes Zeichen aus

1 Startschritt
8 Datenschritten
1 oder 2 Stopschritten

d.h. die Synchronisation erfolgt erneut bei jedem Zeichen. Dieses Verfahren ist nur bei Übertragungseinrichtungen möglich, die frequenztransparent sind, d.h. nicht genau auf eine Frequenz abgestimmt sind. Dies trifft auf Modems bis 1200 bit/s und auf Gleichstromübertragungseinrichtungen zu.

Bei der synchronen Übertragung wird durch eine Synchronisierphase die Synchronität von Sender und Empfänger hergestellt. Jedes Zeichen besteht aus 8 Datenbits. Eine Nachsynchronisation erfolgt bei Bitwechseln und durch eingefügte Synchronisierzeichen.

Bei der synchronen Datenübertragung wird der Übertragungstakt entweder vom jeweiligen Sender (Eigentakt) oder vom Modem erzeugt (Modemtakt).

Es stehen für die Datenübertragung folgende Interfaces zur Verfügung:

- DÜ-Interface Asynchron (IV24/DAS)
  8-Kanal-E/A-Schnittstelle nach V24
  für asynchronen Betrieb
  110...9600 Bd (fest eingestellt), 10 oder 11 Schritte/Zeichen
  (für asynchrone Datenübertragung)

- DÜ-Interface Synchron (IV24/DSM)
  8-Kanal-E/A-Schnittstelle nach V24
  für bitsynchronen Betrieb
  600...9600 Bd (fest eingestellt)
  (für synchrone Datenübertragung mit Synchronisation durch Modem)

- DÜ-Interface Synchron E (IV24/DSE)
  8-Kanal-E/A-Schnittstelle nach V24
  für bitsynchronen Betrieb
  600...9600 Bd (fest eingestellt)
  (für synchrone Datenübertragung mit Eigen-Synchronisation)

Bei der Datenübertragung unterscheidet man zwei Prinzipien:

- Der eine Rechner ist übergeordnet und ruft den untergeordneten Rechner mit einer Sendeaufforderung (Polling) oder einer Empfangsaufforderung (Selecting).

- Die Rechner sind gleichberechtigt. Der jeweils aktive Rechner baut die Übertragung auf (Contention).

Das Polling-Selecting-Prinzip ermöglicht auch einen Party-Line-Betrieb, d.h. an einer Leitung befinden sich mehrere untergeordnete Rechner, die vom Master adressiert werden.

Für den DIETZ 621 bestehen Programme, die mit Hilfe der o.g. Interfaces die bei IBM und Siemens gebräuchlichen Prozeduren sowohl für asynchrone als auch für synchrone Datenübertragung im Polling-Selecting- und im Contention-Prinzip bedienen können. Für Polling-Selecting-Betrieb bestehen auch Programme, mit denen der DIETZ 621 untergeordnete Rechner auch im Party-Line-Betrieb bedient.

# Programmier- Hinweise

## VORBEMERKUNG

Dieser Abschnitt enthält einige Hinweise für die Programmierung des DIETZ 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 Rechnern 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=* STA ,@ ,'ØØFØ     }   "    "  F
            JPX ,   ,     ,'ØØ       Sprung zum Anfang
```

ANF liegt auf Adresse '4ØØØ. 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 DIETZ 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 die CNP-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.

Da auch Netzausfallschutz, Speicher-Parity und BUS-Kontrolle auf die CNP-Ebene wirken, ist der Uhr-Start zu identifizieren. Dies geschieht durch einen GL-Befehl, der in der CNP-Ebene abläuft, bei Uhr-Start Bit 7 = 1 ins Arbeitsregister überträgt und den Uhr-Start selbsttätig zurückstellt:

```
GL  ,@
BOC,@ ,'8Ø, CLCK
```

Es wird nach CLCK verzweigt, wenn es sich um einen Uhr-Start handelte.

## PROZESS-EIN/AUSGABEN

Für die Prozeßperipherie des DIETZ 621 d.h. alle Ein/Ausgabeschaltungen, die sich nicht auf Geräte, wie z.B. Fernschreiber, Leser, Locher usw. beziehen, sind z.B. 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          }     Eingabe von EXT (= '2F78)
                    LDA,REG7,'EXT3,IXR }     + 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 DIETZ 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 256 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:

'1gg0 = 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 (Beispiel für Teletype):

Bit 0 = 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 (angebauter Leser) zugleich mit IBUSY vom Programm eingeschaltet werden.

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

Ein/Ausgabe im Multiprogramming:

Diese auf die Struktur des DIETZ 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 (angebauter 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. ie ein schneller Streifenleser und -locher bilden die Standard-Peripherie eines DIETZ 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 Loch-streifengeräte: | Geräte-Nummer: | '01 |
| | BUS-Adresse Datenregister: | '1010 |
| | BUS-Adresse Statusregister: | '1011 |
| | Locher: | Ausgabe programmieren |
| | Leser: | Eingabe programmieren mit INITIATE [1] |

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.

Die Interfaces hierzu befinden sich auf Platz 1 (Teletype) und 2 (Locher/Leser) in der Zentraleinheit. Sie können auf Wunsch durch Einkarten-Interfaces für andere periphere Schnittstellen ersetzt werden. Auf Wunsch kann der durch READY des Interfaces 2 (Geräte-Nr. '01) ausgelöste Start auf eine andere Ebene gelegt werden. Im übrigen bleiben die obengenannten Adreßvereinbarungen usw. bestehen.

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

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

# ASSEMBLER

VORBEMERKUNG

Der Assembler MINCASS 600 ist ein Programm zur Übersetzung von symbolischen Programmen in die Maschinensprache des Computers DIETZ 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 drei Assembler-Versionen verfügbar, die sich durch zusätzliche Funktionen voneinander unterscheiden:

Assembler MINCASS 600
für Systeme ab 4 kbyte Kernspeicher und Teletype

Editor-Assembler MINCASS 600 E
für Systeme ab 8 kbyte Kernspeicher mit Teletype
(schnelle Lochstreifenausrüstung empfohlen)

Makro-Assembler MINCASS 600 M
für Systeme ab 16 kbyte Kernspeicher mit Teletype und
schneller Lochstreifenausrüstung

Funktionen, die nur vom Makro-Assembler MINCASS 600 M ausgeführt werden, sind im folgenden mit (M) gekennzeichnet; mit (E) solche Funktionen, die zum MINCASS 600 E und MINCASS 600 M gehören.

Der Assembler benutzt die ersten 256 Bytes (Adresse 'ØØØØ...'ØØFF) des RAMs.

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} | * & | {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, oder auch eine Hexa-Zahl (wobei für z = 256 ein Name mit Nullwert bzw. die Hexa-Zahl 'ØØ stehen muß. Als Trennzeichen steht ( * ) oder (&) hinter der Anzahl.

# MINCAL 600 PROGRAM FORM

DIETZ Computer SYSTEME

PROGRAM POTENZEN VON 2 DATE 4/4/73 NAME MAYER 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
                        0      ,'4000;
/:
/** PROGRAMM ZUR BERECHNUNG DER POTENZEN VON ZWEI;
/:
/** MIT EXPONENTEN VON 0 BIS 47
/:
/*** REGISTER BELEGUNG;
/:
VAR  :             Q      ,'40;
EXP  :             Q      ,'42;
CTR  :             Q      ,'43;
TELE :             Q      ,0;
/:
/  PROGRAMM-ANFANG;
/:
ANF  :          2<=LOC    ,EXP    ,0;
                2=2LOC    ,VAR    ,1;
AA   :             INEC,CTR    ,8       ,A2;
/ WAGENRUECKLAUF;
                2=*WTC    ,TELE,'6400;
                   LOC    ,CTR    ,0;
/ LINKSSCHIEBEN + ANSGABE;
A2   :          2<=SLO    ,VAR;
                6  $WORD,TELE,VAR;
/ ENDE ?;
                   INEC,EXP    ,A5      ,AA;
                   HLT
                   JPA ,       ,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; in jedem 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 muß 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.

Nach der Anweisung kann ein Kommentar stehen. Er wird durch einen Schrägstrich (/) - anstelle des Semikolons - eingeleitet und durch ein Semikolon (;) abgeschlossen.

## 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 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 Make 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 oder 4 Hexa-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.
Hexa-Strings bestehen aus 2, 4, 6, ... Hexa-Zeichen mit Apostroph davor.
Achtung: Das Byte mit der niedrigsten Adresse steht ganz rechts!

Text-Zeichen: Textzeichen oder -Strings bestehen aus einem oder mehreren druckbaren ASCII-Zeichen; davor und danach 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"

## ZEICHENVORRAT

Es sind alle druckbaren ASCII-Zeichen mit folgenden Ausnahmen erlaubt:

Semikolon (;) nur für Anweisungs-Ende
Schrägstrich (/) nur für Kommentar-Anfang
Anführungszeichen (") nur für Text-Anfang und -Ende
Linkspfeil (←) bzw. Hochpfeil (↑) machen davorliegendes Zeichen bzw. Anweisung ungültig
Doppelkreuz (#) außer am Programmanfang verboten
Leerschritte (␣) werden - außer in Text-Strings und Kommentaren - überlesen.

Steueranweisungen

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

## Beginn Programm

O Ursprung Programm

Definiert die Adresse der nächstfolgenden speicherbelegenden Anweisung. Zu Beginn jedes Programms sollte eine O-Anweisung stehen (sonst Beginn mit 'ØØØØ).

Spezifikation: 4-stellige Hexa-Zahl

Beispiel: O, '4F12

Z Ende Programm

Schließt das symbolische Programm ab.

Wertzuweisung

Bewirkt Zuweisung eines Wertes zu einem Namen. Belegt keinen Speicherplatz.

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

Die Q-Definition kann im Programm grundsätzlich an beliebiger Stelle stehen. Ausnahme: Wird der Name in einer weiteren Q-Anweisung oder als Name für eine Anzahl (DO-Befehl) oder für den Akkumulator benutzt, so ist er an beliebiger Stelle vor seiner Benutzung zu definieren.

```
Beispiele:  A    :      Q,  12
            ZH15 :  2* Q,  -32768
            REG3 :      Q,  'F3
            ADR  :  2* Q,  'ØFA6
            TX1  :      Q,  "A"
            TX2  : 12* Q,  "ALPHABET␣!!!"
            NAM1 :      Q,  NAM2
            XYØ  :      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.

(M) 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,  32767
```

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 (Leerschritte und Zeilentrennung werden überlesen).

Beispiele:

```
   H,   'FF
2* H,   'ØB3
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 DIETZ 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, IP, 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 ,'4ØA2,'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
JPA  ,      ,'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.

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.

## (M) MAKRO-INSTRUKTIONEN

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

Der Benutzer kann Makro-Anweisungen auf zweierlei Art gebrauchen:

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

Selbstdefinierte Makros zu vom Benutzer erstellten Unterprogrammen.

### (M) Standard-Makros

Die Standard-Makros sind in der Befehlsliste des MINCASS 600 M Makro-Assemblers vermerkt; ihre Namen und Funktionen sind im Kapitel LIBRARY ausführlich beschrieben. Sie werden im Programm wie normale Anweisungen geschrieben.

Die Verwendung von Standard-Makrobefehlen setzt voraus, daß zur Ausführungszeit außer dem Objektprogramm auch die benötigten Teile (Packs) der LIBRARY im Kernspeicher enthalten sind. Dies kann auf zweierlei Weise sichergestellt werden:

a) Automatisches Hinzufügen der LIBRARY:

Hierbei merkt sich der Assembler während der Umwandlung die benutzten Standard-Makros und die zugehörigen Packs der LIBRARY. Nach der Umwandlung werden in einem Zusatzlauf die Teile der LIBRARY hinzugefügt, die vom Objektprogramm benötigt werden. Sie stehen dann auf dem Maschinencode-Streifen bzw. später im Kernspeicher unmittelbar hinter dem umgewandelten Programm; ihre Lage ist daher von dessen Länge und ihre Zusammensetzung von den darin benutzten Makros bestimmt. Nicht benötigte Packs der LIBRARY werden nicht übernommen.

b) Vorbestimmte Lage der LIBRARY:

Der Benutzer kann sich dafür entscheiden, die einzelnen Packs der LIBRARY in vorbestimmte Plätze des Kernspeichers zu legen. In diesem Falle hat er dafür zu sorgen, daß sie sich zur Ausführungszeit dort befinden, z.B. durch getrenntes Einlesen; die LIBRARY wird nicht dem Maschinencode-Streifen des umgewandelten Programms hinzugefügt.

Im symbolischen Programm ist die Lage der benötigten Packs anzugeben, und zwar zu Anfang z.B. in folgender Weise:

```
#
M;
WRID,'8ØEØ;
ARD ,'8DØØ;
#
...                (folgt eigentliches symbolisches Programm)
```

Dies bedeutet, daß das LIBRARY-Pack WRID (beginnend bei Adresse '8ØEØ) und das Pack ARD (beginnend bei '8DØØ) verwendet werden sollen.

Voraussetzung ist, daß die Zuordnung der Packs zu den Makro-Befehlen und ihre gegenseitige Abhängigkeit vom Benutzer beachtet werden. Alle benutzten der 7 LIBRARY-Packs sind anzugeben, und zwar in der Reihenfolge WRTH, WRID, WRF, WRG, ARD, ARF, ARG.

(M) Benutzer-Makros

Der Benutzer kann Unterprogrammen, die getrennt zu erstellen, zu testen und in vorbestimmte Speicherplätze einzulesen sind, eigene Namen geben und diese als Makro-Instruktionen in seinen symbolischen Programmen verwenden.

Benutzer-Makros haben im Programm die Gestalt

XYml

mit folgender Bedeutung:

| | | |
|---|---|---|
| X | Buchstabe | 2 Zeichen vorgeschrieben, kennzeichnen Unterprogramm |
| Y | Buchstabe oder Ziffer | |
| m | Adressierungsart | bei Bedarf |
| l | Länge des übergebenen Parameters | |

Für XY sind alle Kombinationen verboten, die für Maschinenbefehle oder Standard-Makros verwendet werden, für das erste Zeichen (X) in jedem Falle die Buchstaben W, R und K.

Für m ist bei Bedarf einer der Buchstaben C, X, R, L oder A einzusetzen (entsprechend Konstanten-, indirekter, Register-, relativer oder absoluter Adressierung).

Für l ist einer der Buchstaben D, F oder G einzusetzen, wenn statt eines Bytes 2, 3 oder 4 aufeinanderfolgende Bytes als Parameter übergeben werden sollen.

Benutzer-Makros können 3 Typen von Hauptprogramm aufrufen:

a) Einfaches Unterprogramm: Es wird kein Parameter übergeben.
Schreibweise im Programm z.B.: M3;
Erzeugt wird folgende Instruktion:
CSA, (Rückkehradresse),(Anfangsadresse von M3);

b) Unterprogramm LD-Typ: Vor Aufruf des Programms wird ein Parameter ins Unterprogramm übergeben. Schreibweise im Programm z.B.: DMX,@ , , X;
Erzeugt wird diese Befehlsfolge:
LDX, (Übergaberegister), , X;
CSA, (Rückkehradresse),(Anfangsadresse von DM);

c) Unterprogramm ST-Typ: Nach Ablauf des Unterprogramms wird ein Parameter ins Hauptprogramm übergeben. Schreibweise im Programm z.B.:
H9A,@ ,ADDR;
Erzeugt wird diese Befehlsfolge:
CSA, (Rückkehradresse),(Anfangsadresse von H9);
STA, (Übergaberegister),ADDR

Während also beim Typ a) nur ein zweistelliger Makro-Name erlaubt ist, ist bei den Typen b) und c) der Name um die Adressierungsart (m) zu ergänzen und ein Hinweis für die Operandenadresse zu geben; dadurch wird bestimmt, woher der Parameter geholt bzw. wohin er gebracht wird. Die Regeln entsprechen genau denen für BUS-bezogene Befehle.

Aus formalen Gründen ist bei Makros vom LD- bzw. ST-Typ als Spezifikation das Zeichen @ einzusetzen. Die Angabe einer Anzahl vor Benutzer-Makros ist verboten.

Ist ein Parameter von 2, 3 oder 4 Byte Länge zu übergeben, so wird an den Makro-Namen als 4. Zeichen (I) der Buchstabe D, F oder G angehängt; Beispiel:

DMRG,@ , OPD;
Erzeugt die Befehlsfolge:
4 = & LDR, (Übergaberegister),OPD;
CSA, (Rückkehradresse),(Anfangsadresse von DM);

Zu Beginn des Programms sind die Namen der Benutzer-Makros als Anweisungen mit dem Rückkehradreß-Register und der Anfangsadresse des zugehörigen Unterprogramms sowie (für Fälle b und c) dem Typ (LD oder ST) und der Adresse des Übergaberegisters anzugeben; alle Adreßangaben sind dabei Hexa-Zahlen. Dies geschieht z.B. in folgender Form:

```
#
M;
M3 ,'5AØØ;'1E
DM,'588Ø,'1C,'1Ø,LD;
H9 ,'59ØØ,'2A,'1F,ST;
#
...
```

(folgt eigentliches symbolisches Programm)

Typ

Adresse Übergaberegister

Rückkehradreß-Register

Anfangsadresse Unterprogramm

Unter M können außerdem die im vorigen Abschnitt beschriebenen Pakete der LIBRARY erscheinen.

## 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:

> Ein Linkspfeil (←) macht das vorangehende Zeichen ungültig, mehrere aufeinanderfolgende Linkspfeile (← ←...) die entsprechende Anzahl vorangehender Zeichen. Danach sind die richtigen Zeichen einzugeben.
>
> Ein Hochpfeil (↑) macht die gesamte Anweisung bis zum vorangehenden Semikolon ungültig. Danach ist die Anweisung neu einzugeben.

Bei Benutzung der einfachsten Assembler-Version MINCASS 600 werden fehlerhafte symbolische Lochstreifen mit dieser Methode korrigiert, indem man sie off-line (im Local-Betrieb) auf dem Teletype dupliziert und ändert.

## 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, und zur Fehlererkennung |

Nach Start des Rechners meldet sich der Assembler auf dem Teletype mit # und Klingelzeichen. 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 | |
| Ausgabe: | OSB | Langsamer Locher (Teletype) | } symbolisches Programm |
| | OFB | Schneller Locher | |
| EXC Eingabe: | ISB | Langsamer Leser (Teletype) | } symbolisches Programm |
| | IFB | Schneller Leser | |
| Ausgabe: | OSH | Langsamer Locher (Teletype) | } Maschinencode-Programm |
| | OFH | Schneller Locher | |

Beispiele:
```
ASS,IKB,OSB
ASS,ISB,OFB
EXC,ISB,OSH
EXC,IFB,OFH
```

Nach Vorwahl der Betriebsart ist "Wagenrücklauf" einzugeben, und der Lauf beginnt. Das symbolische Programm, gleichgültig ob über die Tastatur eingegeben oder als Lochstreifen eingelesen, 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
```

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 SAV zusätzlich ein:

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

Zusätzlich kann mit PRO das Drucken eines Protokolls auf dem Teletype vorgewählt werden, z.B.:

```
ASS,ISB,PRO
ASS,IFB, OFB,PRO
EXC,IFB,OFH,PRO
```

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

| | | |
|---|---|---|
| Fehlercode | (2 Ziffern oder Leerschritt) | |
| Leerschritt | | |
| Anweisungs-Nummer | (4 Ziffern) | |
| Leerschritt | | |
| (Basis-) Adresse | (4 Hexa-Ziffern) | |
| Leerschritt | | |
| Marke | (4 Zeichen) | } falls vorhanden |
| . | | |
| . | | |
| Anzahl | (4 Zeichen) | } falls vorhanden |
| Zusatzzeichen | ( >,< , = oder Leerschritt) | |
| Trennzeichen | (* oder &) | |
| Befehl | (4 Zeichen) | |
| , | | |
| Spezifikation | (4 Zeichen) | } oder längere Spezifikation |
| , | | |
| 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; nur bei EXC-Lauf). | |

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

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

## (E) EDITOR-BETRIEB

Die Assembler MINCASS 600 E und M erlauben darüberhinaus die Korrektur fehlerhafter symbolischer Programme während eines Assembler-Laufs. Korrekturen werden vor dem Lauf eingegeben und in zwei Pufferbereiche A (für Korrekturvorschriften) und B (für neue symbolische Anweisungen) eingegeben. Danach wird der Lauf (ASS, evtl. auch EXC) ausgeführt; die vorgegebenen Korrekturen werden dabei automatisch ausgeführt.

Zur Eingabe der Korrekturen wird durch Eintippen von

COR,kk  oder COR

(danach Wagenrücklauf) die zugehörige Betriebsart vorgewählt. kk ist eine zweistellige Dezimalzahl; sie gibt die Größe des Pufferbereichs A an, d.h. die maximale Anzahl der folgenden Korrekturvorschriften. Diese werden dann in folgender Weise eingegeben:

| | | |
|---|---|---|
| m D | Anweisung m löschen | |
| m-n D | Anweisung m bis n löschen | |
| m A | Anweisung m ändern | danach jeweils Eingabe der neuen Anweisung(en) in üblicher symbolischer Form. Anweisungen durch Semikolon (;) getrennt. Nach letzter Anweisung Doppelkreuz (#) statt Semikolon eingeben. |
| m-n A | Anweisung m bis n ändern | |
| m I | Nach m neue Anweisung(en) einfügen | |

m, n sind bis zu 4-stellige Dezimalzahlen; sie entsprechen den Anweisungsnummern des zu korrigierenden Lochstreifens bzw. auf dem zugehörigen Protokoll.

Wird kein kk eingegeben, gilt die in einer früheren Anweisung gemachte Größe des Pufferbereichs und der Korrekturvorschriften.

Außerdem gibt es folgende Kommandos für die Überprüfung bzw. Änderung des Korrekturpuffers:

| | |
|---|---|
| L | Listen aller eingegebenen Korrekturen |
| m L | Listen der Korrektur zu Anweisung m |
| m-n L | Listen der Korrekturen zu Anweisungen m bis n |
| C | Löschen aller eingegebenen Korrekturen |
| m C | Löschen der Korrektur zu Anweisung m |
| m-n C | Löschen der Korrekturen zu Anweisungen m bis n |

Dies bezieht sich sowohl auf Korrekturvorschriften als auch auf neue Anweisungen, die mit COR in den Puffer eingegeben worden sind.

Nach D, A, I, L und C ist Wagenrücklauf zu betätigen. Durch Eingabe von ## wird die Betriebsart beendet.

Ein danach ausgeführter Assembler-Lauf berücksichtigt automatisch die eingegebenen Korrekturen und führt zu einem berichtigten Lochstreifen bzw. Protokoll.

Sind sehr viele Korrekturen nötig, so kann der hierfür benötigte Speicherraum dadurch geschaffen werden, daß die Markenliste nicht aufgebaut wird. Man führt einen reinen Korrekturlauf (anstelle eines Assemblerlaufs) durch, indem man die Eingabe ASS wegläßt und durch Eingabe von

ISB, OSB oder
IFB, OFB

diese Betriebsart vorwählt (mit Wagenrücklauf danach).

Beispiele für Korrekturvorschriften und -anweisungen:

| | |
|---|---|
| 2-1Ø D | Anweisung 2 bis 10 löschen |
| 12 A | Anweisung 12 ändern: |
| H126: CSA,'1Ø,UP18 | (neue Anweisung) |
| 335 I | hinter Anweisung 335 einfügen: |
| XAB : LDC, @ ,"T" | (neue Anweisung) |
| 4ØØ D | Anweisung 4ØØ löschen |
| 1Ø2Ø-1Ø21 A | Anweisung 1Ø2Ø und 1Ø21 ändern: |
| NOP | (neue Anweisung) |
| NOP | (neue Anweisung) |

WEITERE FUNKTIONEN

(E) Die dem Assembler zur Verfügung stehende Speichergröße kann durch Eingabe von

ADR, (Endadresse = 4-stellige Hexa-Zahl)

bestimmt werden. Dies ist zu Beginn durchzuführen; andernfalls wird '5FFF (Ende 8 kbyte-Speicher) als Endadresse genommen.

(M) Der Protokollausschrieb erfolgt in Abschnitten von DIN A4-Blattgröße. Auf jedem Blatt steht zu Anfang die Blatt-Nummer sowie der Programmname. Dieser kann bis zu 48 Zeichen haben und vom Benutzer durch

COM, (Programmname)

eingegeben werden.

(M) Nach EXC-Lauf eines Programms, in dem Standard-Makros vorkommen, fordert der Rechner durch Ausdrucken von LIB auf, einen Maschinencode-Streifen mit der LIBRARY in den (schnellen) Leser einzulegen. Nachdem dies geschehen ist, wird mit Wagenrücklauf bestätigt und die LIBRARY (bzw. Teile davon) dupliziert.

## FEHLERLISTE

| Fehlercode | |
|---|---|
| Ø1 | Parity-Fehler |
| Ø2 | Name mehrfach definiert |
| Ø3 | Name beginnt nicht mit Buchstabe |
| Ø4 | Speicherüberlauf |
| Ø5 | Befehl nicht zulässig |
| Ø6 | DO-Befehl falsch bzw. falsche Anzahl |
| 1Ø | Name nicht definiert |
| 11 | Operand hat falsche Länge |
| 12 | Adreßabstand für relative Adressierung zu groß |
| 13 | Konstante (Eingabe als Operand) zu lang |
| 14 | Hexa-Zahl enthält unerlaubtes Zeichen |
| 15 | Hexa-Zahl endet zu früh |
| 16 | Additive Q-Definition länger als 32 byte |
| 17 | Textstring als Operand zu lang |
| 18 | Textstring in Definition zu lang |
| 19 | F- oder G-Format falsch |
| 21 | Komma fehlt nach Befehl |
| 22 | Semikolon oder Schrägstrich fehlt. |
| 23 | Anweisung endet zu früh |
| 24 | Leerspalte ist nicht leer |
| 25 | Angabe in einer Spalte fehlt |
| 28 | O-Anweisung setzt Adreßzähler zurück (Warnung) |
| (M) 4Ø | Makro enthält unerlaubte Anzahl-Angabe |
| (M) 41 | Makro enthält andere Spezifikation als @ |
| (M) 42 | Formatangabe unzulässig |

## ASSEMBLER-DATEN

MINCASS 600 bei 4k: max. 65 Namen; bei 8k: max. 880 Namen in Markenliste

MINCASS 600 E bei 8k: max. 610 Namen; bei 16k: max. 1425 Namen in Markenliste

MINCASS 600 M bei 16k: max. 815 Namen in Markenliste

Zugrunde gelegt sind Namen mit 2-byte-Werten (z.B. Adressen), die 5 Byte in Markenliste einnehmen. Namen mit n-byte-Werten erfordern 4+n bytes, wodurch sich die o.a. Anzahl von Namen erniedrigt.

Beim MINCASS 600 E/M sind außerdem die Korrekturpuffer in Abzug zu bringen (9 byte je Korrekturvorschrift sowie 1 byte je Zeichen für neue Anweisungen).

PROGRAMM-BEISPIEL

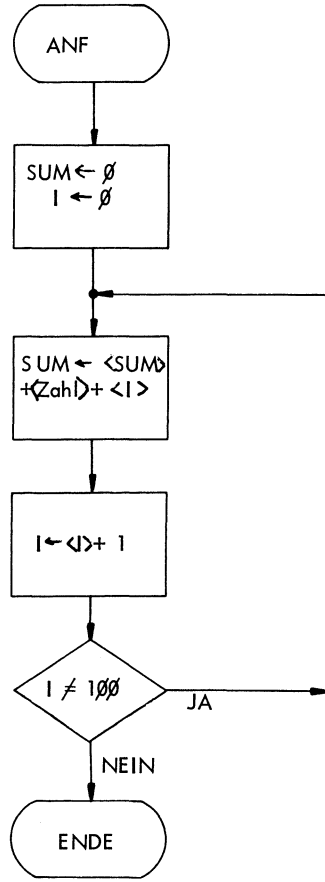
Aufgabenstellung:

In den Zellen mit den symbolischen Adressen ZAHL bis ZAHL + 99 stehen 100 Werte, deren Summe in der Zelle mit der symbolischen Adresse SUM abgelegt werden soll.

Ein Überlauf soll berücksichtigt werden.

Das Programm beginnt mit der symbolischen Adresse ANF und wird durch einen Sprung nach Zelle ENDE verlassen.

Blockdiagramm:



Programm:

```
0000                    O    ,'6000 /
                SUMMATION MIT UEBERLAUF;
0001            I    :    Q    ,     3;
0002            SUM  :    Q    ,     4;
0003            ZAHL:  2* Q    ,'4000;
0004             /
                ;
0005  6000    ANF  :    6<&LDC ,I    ,     0;        19 06 81 03 00
0006  6005    L1   :       LDA ,'07 ,ZAHL  ,I    ; 8F 07 00 40 03
0007  600A              2=*ADR ,SUM ,     7;        1E 95 04 07
0008  600E                 INEC,I   ,   100,L1    ; 5B 03 64 F4
0009  6012    END  :       HLT ;                    02
0010  6013                 JPL ,    ,ANF   ;        F8 EC
0011                       Z   ;
```

symbolischer Befehl — Maschinencode

Kernspeicher-Adresse des 1. Befehls-Bytes

Zeilennummer des Protokolls

Erläuterungen:

Zu Zeile Ø: Die O-Zuweisung legt die Programm-Anfangsadresse fest.

Zu Zeile 1 - 2: Den symbolischen Namen I und SUM wird der Wert 3 bzw. 4 zugewiesen.

Zu Zeile 3: Dem Namen ZAHL wird der 2 Byte große Wert 'ØØØØ zugewiesen.

Zu Zeile 5: Das Register I (= Register 'Ø3) und die 5 folgenden Register (Register 'Ø4...'Ø8) werden mit Ø geladen.

Zu Zeile 6: Die Adresse ZAHL (= 'ØØØØ) wird mit dem Register I (= 'Ø3) indiziert: ZAHL + <I> .
Der Inhalt der sich daraus ergebenden Adresse wird in das Register 'Ø7 geladen.

Zu Zeile 7: Der 2 Byte große Inhalt der Register 'Ø7 und 'Ø8 wird zum 2 Byte-Inhalt der Register SUM und SUM + 1 addiert (Register 'Ø8 hat immer Nullinhalt).

Zu Zeile 8: Der Inhalt des Registers I wird um 1 erhöht und auf 1ØØ (= '64) geprüft. Wenn er nicht 1ØØ ist, verzweigt das Programm zur Marke L1 (= Adresse '6ØØ5).

# LIBRARY

VORBEMERKUNG

Die Bibliothek (LIBRARY) des DIETZ 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:

| | |
|---|---|
| WRTH | (Ein/Ausgabe von Text und Hexa-Zahlen) |
| WRID | (Ein/Ausgabe von 1- und 2-byte-Ganzzahlen) |
| WRF | (Ein/Ausgabe von 3-byte-Gleitkommazahlen) |
| WRG | (Ein/Ausgabe von 4-byte-Gleitkommazahlen) |
| ARD | (Doppelbyte-Arithmetik) |
| ARF | (3-byte-Gleitkomma-Arithmetik) |
| ARG | (4-byte-Gleitkomma-Arithmetik) |

WRID bedingt das Vorhandensein von ARD, WRF das von ARF, WRG das von ARG.

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 64 Bytes (Register-Adressen ØØ...3F) zur Verfügung stehen; dieser Bereich, einschließlich dem Akkumulator @ , kann durch die Unterprogramme verändert werden.

Die vollständige LIBRARY ist ca. 6.8 kbyte lang.

PAKET WRTH (Länge 'Ø12Ø)

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 |
| KTm | Eingabe Text über Tastatur |
| KHm | Eingabe Hexa über Tastatur |

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 1F3Ø 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.

Achtung: Die ein- oder auszugebenden Zeichen benutzen den Pool. Die Übergabe erfolgt im Intern-Format ab Register '1Ø (bei WHm, WTm, KTm, RTm) bzw. ab Register '18 (KHm, RHm). Das bedeutet, daß durch die Übergabe alle Register '1Ø - '1Ø + N-1 (bzw. '18 + N-1) belegt werden. Sind nur die Register bis '3F für die LIBRARY reserviert, so lassen sich maximal '28 d.i. 40 Bytes durch einen Aufruf übertragen.

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
```

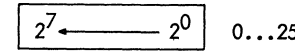
Die Ausgabe der Zeichen erfolgt im ASCII-(ISO-7-) Code mit geradzahliger Parität. Bei der Eingabe wird auf diese Parität geprüft; fehlerhafte Zeichen werden zwar abgelegt, jedoch wird dann ein Register-Byte '08 auf FF gesetzt; der Benutzer kann dieses Byte nach Ablauf des Makrobefehls im Hauptprogramm abfragen. Im Normalfall hat das Register '08 Nullinhalt.

Achtung: Bei der Übergabe von Hexa-Konstanten werden die Bytes zeitlich in aufsteigender Adreßreihenfolge behandelt (d.h. von rechts nach links!).

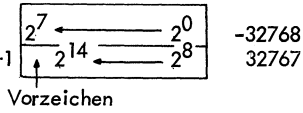
## PAKET ARD (Länge '00C0)

Mit diesem Unterprogramm-Paket können Doppelbyte-Ganzzahlen arithmetisch behandelt sowie ein- und ausgegeben werden, einschließlich der Umwandlungen von Binär- in Dezimalzahlen und umgekehrt.

Einbyte-Ganzzahlen sind stets positiv:




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

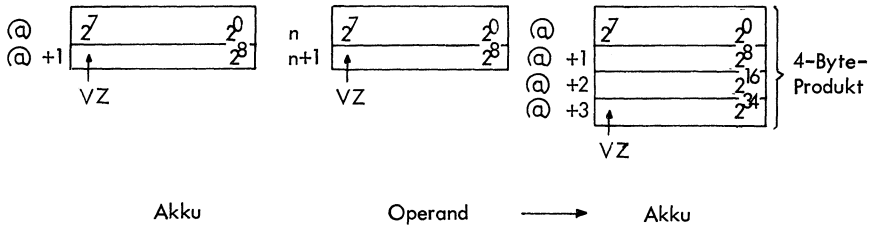



Folgende arithmetischen Befehle sind für Doppelbyte-Ganzzahlen vorgesehen:

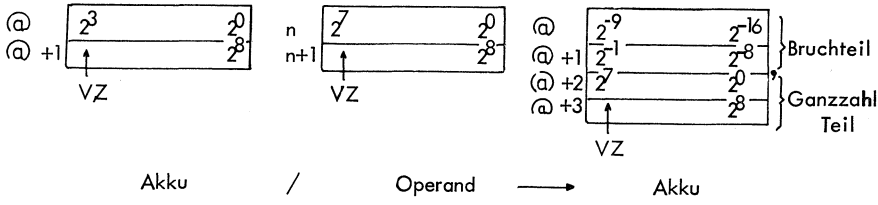
| | |
|---|---|
| MPmD | Multiplizieren Doppelbyte |
| DVmD | Dividieren " |

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 , @ , -25000 | -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 |

Anmerkung:
Es werden in diesem Paket die Register bis einschließlich '1F belegt.

PAKET WRID (Länge '0300)

Zu diesem Paket gehören 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 Periphergerät 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 (Ausnahme: Return, Line-feed, Rubout).

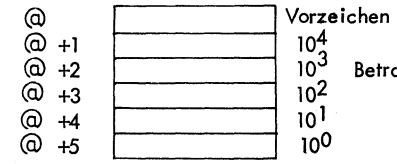
Hinsichtlich Paritäts-Erzeugung und -Prüfung des ASCII-Codes gelten die Bemerkungen des vorigen Abschnitts; das Fehler-Register hat die Adresse '10.

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. 'ØD oder 'ØØ (BCD)

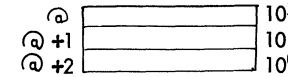
Beispiele für Doppelbyte-Ein/Ausgabe- und Konversionsbefehle:

| | |
|---|---|
| 6ß RDAD ,DEV,ADDR | 6-Zeichen-Zahl von DEV nach ADDR (binär) |
| 9ß WDCD,'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 WRID-Pakets sind schließlich noch Ein/Ausgabe- und Konversionsbefehle für Einbyte-Ganzzahlen:

| | | |
|---|---|---|
| RI m | Lesen Einbyte-Ganzzahl | |
| WIm | Schreiben Einbyte-Ganzzahl | |
| RAm | 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:

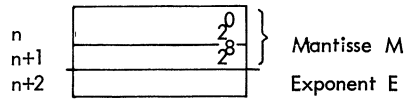
| | |
|---|---|
| 2ßRIA ,DEV,ADDR | 2-stellige Ganzzahl von DEV nach ADDR |
| 6ßWIC ,'F3,125 | ␣ ␣ ␣ 1 2 5 auf 'F3 ausgeben |
| RAX , @ , , IND | siehe Doppelbyte-Beispiele |
| WAR, @ ,REG7 | |
| RBA , @ , '2FØ8 | |
| WBL , @ ,VAR, IXR | |

Anmerkung: Es werden in diesem Paket die Register bis einschließlich '23 belegt.

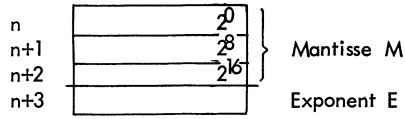
PAKETE ARF UND ARG (Länge '024Ø, '025Ø ohne Funktionen)
(Länge der Funktionen '074Ø)

Diese Unterprogramm-Pakete geben dem Benutzer die Möglichkeit, mit Gleitkomma-Zahlen zu rechnen. 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 " " |

| | | (Gleitkomma G-Typ) | |
|---|---|---|---|
| POmG | Potenzieren | " | " |
| SQ | Wurzel | " | " |
| SI | Sinus | " | " |
| CO | Cosinus | " | " |
| TA | Tangens | " | " |
| AT | Arcus Tangens | " | " |
| LO | Logarithmus | " | " |
| EX | Exponent | " | " |
| RN | Random 0...1 | " | " |
| IT | Integer | " | " |
| SG | Signum | " | " |
| AB | Absolut | " | " |

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
```

```
SEITE 0000

0000                    0    ,'4000 ;
0001          /
          BEISPIEL FUER ERWEITERTE LIBRARY;
0002          /
          X=ABS(SIN(M*0.1))**.3/5.2;
0003          /
          ;
0004 4000 ANF :    2=*WTC ,0    ,'0A0D ;        1E 81 10 0D 0A FD 0E 59
                                                40 00 10 02
0005 400C        8     $KERG,0    ,@      ;     FD 28 75 43 00 10 08 00
                                                1F 04 E4 02
0006 4018               MPAG,@    ,CON1   ;     1F 04 8D 09 4D 40 FD 1E
                                                A3 45
0007 4022               SI   ;                  FD 30 03 48
0008 4026               AB   ;                  FD 0A F3 47
0009 402A               POAG,@    ,CON2   ;     1F 04 8D 09 51 40 FD 2C
                                                E9 49
0010 4034               DVAG,@    ,CON3   ;     1F 04 8D 09 55 40 FD 1E
                                                05 46
0011 403E       15.6 $WERG,0    ,@      ;     1F 04 84 02 FD 2A 93 40
                                                00 10 0F 06
0012 404A               JPA ,     ,ANF    ;     FC 00 40
0013 404D CON1:         G   ,.1;                66 66 66 E6
0014 4051 CON2:         G   ,.3;                CC CC 4C E8
0015 4055 CON3:         G   ,5.2;               33 33 53 EC
0016                    Z   ;
```

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) |

Anmerkung: Es werden bei AD, SB, MP und DV die Register bis einschließlich '1F belegt. Die Funktionen belegen die Register bis einschließlich '37.

PAKET WRF UND WRG (Länge '0F0, '0520)

Diese Pakete erlauben die Ein/Ausgabe von Gleitkomma-Zahlen mit folgenden Makrobefehlen:

| | | | | |
|---|---|---|---|---|
| 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 $ als Trennzeichen, also:

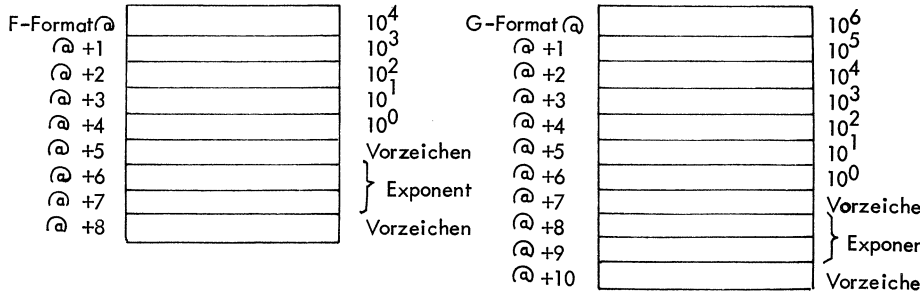
w . d $

Beispiele hierfür:

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

Hinzu kommen als Konversionsbefehle:

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




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 vorigen Abschnitt.

Anmerkung: Diese Pakete belegen die Register bis einschließlich '33.

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.

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
```

WRTH

| | |
|---|---|
| WHm | Schreiben Hexa |
| WTm | Schreiben Text |
| KHm | Eingabe Hexa |
| RHm | Lesen Hexa |
| KTm | Eingabe Text |
| RTm | Lesen Text |

WRID

| | |
|---|---|
| WIm | Schreiben Ein-Byte-Ganzzahl |
| WAm | Konversion Ein-Byte Binär → ASCII |
| WBm | Konversion Ein-Byte Binär → BCD |
| WDmD | Schreiben Doppelbyte - Ganzzahl |
| WAmD | Konversion Doppelbyte Binär → ASCII |
| WBmD | Konversion Doppelbyte Binär → BCD |
| KIm | Eingabe Ein-Byte - Ganzzahl |
| RIm | Lesen Ein-Byte-Ganzzahl |
| RAm | Konversion Ein-Byte-Ganzzahl ASCII → Binär |
| RBm | Konversion Ein-Byte-Ganzzahl BCD → Binär |
| KDmD | Eingabe Doppelbyte-Ganzzahl |
| RDmD | Lesen Doppelbyte-Ganzzahl |
| RAmD | Konversion Doppelbyte ASCII → Binär |
| RBmD | Konversion Doppelbyte BCD → Binär |

WRF

| | | |
|---|---|---|
| WEmF | Schreiben E-Format | F-Format |
| WFmF | Schreiben F-Format | " |
| WAmF | Konversion Binär → ASCII | " |
| WBmF | Konversion Binär → BCD | " |
| KEmF | Eingabe E-Format | " |
| KFmF | Eingabe F-Format | " |
| REmF | Lesen E-Format | " |
| RFmF | Lesen F-Format | " |
| RAmF | Konversion ASCII → Binär | " |
| RBmF | Konversion BCD → Binär | " |

WRG

| | | |
|---|---|---|
| WEmG | Schreiben E-Format | G-Format |
| WFmG | Schreiben F-Format | " |
| WAmG | Konversion Binär → ASCII | " |
| WBmG | Konversion Binär → BCD | " |
| KEmG | Eingabe E-Format | " |
| KFmG | Eingabe F-Format | " |
| REmG | Lesen E-Format | " |
| RFmG | Lesen F-Format | " |
| RAmG | Konversion ASCII → Binär | " |
| RBmG | Konversion BCD → Binär | " |

ARD

| | |
|---|---|
| MPmD | Funktion A* B Doppelbyte |
| DVmD | Funktion A/B Doppelbyte |

ARF

| | |
|---|---|
| MPmF | Funktion A* B F-Typ |
| DVmF | Funktion A/B F-Typ |
| SBmF | Funktion A-B F-Typ |
| ADmF | Funktion A+B F-Typ |

ARG

| | |
|---|---|
| MPmG | Funktion @ x B G-Typ |
| DVmG | Funktion @ /B G-Typ |
| SBmG | Funktion @ -B G-Typ |
| ADmG | Funktion @ +B G-Typ |
| RN | Funktion Random zwischen Ø und 1 |
| AT | Arctan (@) |
| SG | SIGN (@) |
| AB | Absolut (@) |
| SQ | Wurzel (@) |
| TA | TAN (@) |
| CO | COS (@) |
| SI | SIN (@) |
| POmG | @ ↑ B |
| EX | EXP (@) |
| LO | LOGE (@) |
| IT | INT (@) |

# MONITOR

## VORBEMERKUNG

MONITOR ist ein Programm zum Testen von Programmen, die im Kernspeicher des DIETZ 621 abgelegt sind.

Zunächst wird das Monitor-Programm in einen freien Kernspeicherbereich geladen, das N-Register der Ebene Ø über die Bedienungskonsole auf die Anfangsadresse dieses Bereichs gesetzt und der Rechner gestartet.

Der Monitor meldet sich dann mit MON und verlangt mit LEV: zunächst die Angabe der Ebene, auf der der Monitor laufen soll. Der Benutzer muß durch eine vierstellige Eingabe ØØØl die Ebene l als Hexazahl angeben, auf der der Monitor und das zu testende Programm laufen soll.

Durch Betätigen von RUBOUT kann die Eingabe wiederholt werden (dies gilt auch für alle folgenden Kommandos).

Als nächstes verlangt der Monitor mit BUF: die Anfangsadresse des Insert-Puffer-Bereichs.

Der Benutzer kann durch eine 4-stellige Hexa-Adresse nnnn die Anfangsadresse eines Bereichs (Insert-Puffers) angeben, in den später einzufügende Maschinencode-Bytes abgelegt werden sollen. Wird statt nnnn ein # eingegeben, so wird der Insert-Puffer unmittelbar an den Monitor angehängt.

Danach gibt der Monitor ein Klingelzeichen und * aus zum Zeichen, daß die Eingabe eines Steuerbefehls erwartet wird (dies gilt auch für alle folgenden Eingaben).

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

## STEUERKOMMANDOS

Das zu testende Programm wird mit

aaaa S (cr)

gestartet, wobei die Startadresse aaaa wie bei allen Adreßangaben als 4-stellige Hexazahl einzugeben ist. (cr) bedeutet Wagenrücklauf. Die Startadresse muß dem Basis-Byte eines Instruktionsstrings entsprechen. Veränderungen des Benutzerprogramms durch eingebaute Monitor-Halts oder Inserts sind zu beachten.

Ist der angesprochene Befehl durch einen vorher eingebauten Monitor-Halt überdeckt, so gibt der Monitor die Fehlermeldung

HK (Halt - Kollision)

und führt den Start nicht aus.

Läuft das Programm später auf einen vorher eingebauten Halt, so druckt der Monitor dessen Adresse aus und erwartet einen neuen Steuerbefehl. Das zu testende Programm kann dann durch eines der folgenden Kommandos wieder angestoßen werden:

N (cr): nächste Instruktion ausführen, dann wieder anhalten

G (cr): weiterlaufen bis zum nächsten Monitor-Halt

Alle eingebauten Halts (8 byte lang), die durch einen N-Schritt berührt werden, werden ausgebaut. Ist der durch einen N-Schritt auszuführende Befehl ein bedingter Sprung, so wird vor Ausführung des Befehls sowohl hinter dem Befehl als auch am Sprungziel ein Halt eingebaut. Daher müssen bei N-Halts 2 der 5 möglichen Halts reserviert werden, d.h. N wird nicht ausgeführt, wenn mehr als 3 andere Halts schon eingebaut sind. Dies wird durch Drucken eines Fragezeichens angezeigt. Die durch N eingebauten Halts werden zu Beginn des nächsten N-Schritts wieder ausgebaut.

Durch das Kommando

E (cr)

wird der Monitorbetrieb beendet. E baut alle Halts und Inserts aus und setzt die nötigen Merkerzellen im Monitor so, daß durch einen Start über die Rechner-Konsole der Monitorbetrieb wieder aufgenommen werden kann.

War der Monitor auf einer höheren Ebene als Ø gelaufen, so wird sich nach E (cr) der Monitor noch einmal auf der Startebene Ø melden, so daß dort noch einmal E (cr) gegeben werden muß.

## 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 Instruktionsstrings 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.

Als Halt wird vom Monitor im Benutzerprogramm ein 8 byte langer Befehlsstring eingebaut:

```
'Ø2 = & STA,@ ,BPP      Rückkehr-Register für Benutzer retten;
      CSA,@ ,HASU       UP-Sprung nach Monitor;
```

Veränderungen des Benutzerprogramms durch eingebaute Halts sind zu beachten.

Ein Halt einzubauen, ist in 3 Fällen verboten:

a) Der einzubauende Halt würde sich mit einem schon eingebauten Halt überschneiden. Fehlermeldung: HK (Halt - Kollision)

b) Es sind schon 5 Halts eingebaut. Fehlermeldung: HU (Halt - Überlauf)

c) Halt-Adresse < '4ØØØ. Fehlermeldung: HV (Halt verboten).

Es können bis zu 5 Haltbefehle eingebaut werden.

Zu Beginn eines G-Schrittes wird der Halt, auf dem weitergestartet wird, zunächst ausgebaut, um die ursprünglichen Benutzer-Befehle ausführen zu können; bei Erreichen des nächsten Monitor-Halts wird dieser Halt dann wieder eingebaut. Daraus ergibt sich, daß in Programmschleifen immer mindestens 2 Halts vorzusehen sind.

Durch den Befehl

H (cr) (ohne Adreßangabe)

wird eine Liste der Adressen aller eingebauten Halts abgerufen.

Haltbefehle kann man einzeln mit

aaaa D (cr)

wieder eliminieren. Ist der durch aaaa spezifizierte Halt nicht vorhanden, so gibt der Monitor die Fehlermeldung HN (Halt nicht vorhanden) aus.

Durch D (cr) werden sämtliche vorgesehenen Halts wieder ausgebaut.

## ABFRAGEN, ÄNDERN, EINFÜGEN

Nach Eingabe von

aaaa L (cr)

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

aaaa-bbbb L (cr)

der Inhalt sämtlicher Adressen von aaaa bis bbbb einschließlich. Je Byte wird eine 2-stellige Hexazahl gedruckt; ein Leerschritt trennt sie vom nächsten Byte. Zu Beginn jeder Zeile wird die Adresse des nächsten Bytes angegeben.

Durch Betätigen der Taste WRU kann der Bediener das Ausdrucken des nächsten Byte-Inhalts anfordern. Das kann beliebig wiederholt werden, wobei jeweils die nächsthöhere Adresse abgefragt wird. Beendet wird der L-Zustand durch Eingabe von # . Jedes andere eingegebene Zeichen wird überlesen.

Die Adressen aaaa sind, auch wenn sie sich auf den Pool beziehen, immer absolute, nicht niveau-gebundene Adressen. Dies gilt auch für die folgenden Kommandos.

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 aaaa bis bbbb durch einen String (xx yy ...). Einzugeben sind je Byte 2 Hexa-Ziffern. Die einzelnen Bytes können durch Leerschritt oder CR, LF getrennt sein oder unmittelbar hintereinander eingegeben werden. Wird der String vorzeitig durch # beendet, so bekommen die restlichen Bytes Nullinhalt. Eingabe von RUBOUT löscht bereits angefangene Bytes. Hat man eine Zahl eingegeben, die keine Hexazahl sein kann, so gibt der Monitor ein Fragezeichen (?) aus und erwartet eine neue Eingabe des letzten Bytes. Die vorhergehenden Byte-Inhalte werden davon nicht berührt.

Schließlich besteht die Möglichkeit, durch

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

im Programm (beginnend bei Adresse aaaa) einen Byte-String (xx yy ...) einzufügen; er wird in dem eingangs erwähnten Pufferbereich abgelegt. Für die Eingabe von Byte-Strings gilt das bei A beschriebene Vorgehen.

Durch das Kommando I wird an der Stelle aaaa ein 3 byte langer Befehl (JPA,,Puffer) eingebaut. Alle durch diese 3 Bytes berührten Befehle werden ausgebaut und im Insert-Puffer vor den einzubauenden Insert-String gelegt. Am Ende des Insert-Strings steht ein JPA auf dem nächsten Befehl hinter den ausgebauten Befehlen.

Die Anzahl der Bytes, die pro Insert im Puffer gebraucht werden, berechnet sich folgendermaßen: Anzahl der eingefügten Bytes + Anzahl der ausgebauten Bytes + 9.
Der Insert-Bereich wird folgendermaßen aufgebaut:

| | |
|---|---|
| I1-Adresse<br>Anzahl ausgebaute Bytes<br>Differenz der Adressen I1 - I2<br>ausgebaute Befehle<br>eingebaute Befehle<br>Rücksprung | 1. Einfügung |
| I2-Adresse<br>...<br>... | 2. Einfügung |

Warnung:

a) Relativ adressierte Befehle dürfen durch I nicht aus dem Benutzerprogramm in den Puffer verlegt werden.

b) Veränderungen des Benutzerprogramms durch eventuell eingebaute Halts sind zu beachten.

Es sind beliebig viele Einfügungen möglich; auf das Puffer-Ende wird nicht geprüft.

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 Kommands bereit:

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

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

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

# DBOS

## VORBEMERKUNG

Die Grundfunktionen des DIETZsystems 621 werden durch DBOS (Disk Based Operating System) gewährleistet. Es bildet das Grund-Betriebssystem, das bei allen Hardware- und Software-Konfigurationen vorhanden ist und auf dem die erweiterten Betriebssysteme aufbauen.

DBOS hat folgende Funktionen:

- Benutzer-/System-Dialog über Konsolgerät
- Dateiverwaltung über Konsolgerät
- Dateizugriff (sektorweise) vom Programm
- Behandlung von Systemfehlern

Die DBOS-Funktionen sind bei den DIETZsystemen 621 C und 621 D identisch.

## SYSTEM-DIALOG

Unter DBOS kann der Benutzer im Dialog Programme aus Platten-Dateien in den Kernspeicher laden, zur Ausführung bringen und wieder auf die Platte zurückschreiben.

Die Kommandos hierfür lauten:

| | |
|---|---|
| GET,u,f,(a) | Programm laden |
| RUN,u,f,(a) | Programm laden und zur Ausführung bringen |
| PUT,u,f,(a) | Programm zurückschreiben |

Dabei bedeuten:

u = Plattenspeicher-Nr. (unit)

f = Dateiname (max. 6 Zeichen)

a = Kernspeicher-Basisadresse
(falls nicht angegeben, wird nächste freie Adresse benutzt).

Falls mehrere Programmsegmente geladen werden sollen, so geschieht dies durch eine Folge von GET-Kommandos. PUT erlaubt das Rückschreiben modifizierter Programme. Mit RUN wird ein Programm geladen und die Kontrolle an dieses übergeben; bei dessen Ende wird in DBOS zurückgesprungen, und das System ist wieder im Dialog-Betrieb.

## DATEI-VERWALTUNG

DBOS erlaubt die Reservierung und Behandlung von Dateien auf den Plattenspeichern. Dateien sind unter einem Dateinamen f zugreifbar, bestehen aus einer ganzen Zahl L von Sektoren zu je 128 byte und können vollständige Programme, Programm-Moduln oder Daten enthalten.

Dateien werden im Dialog-Betrieb des DBOS reserviert, gelöscht usw. Hierzu dienen die Kommandos:

| | |
|---|---|
| CREA ,u,f ,L | Eröffnen einer Datei von L Sektoren Länge unter dem Namen f |
| KILL ,u,f | Löschen der Datei f |
| ALTR ,u,f1,f2 | Ändern des Datei-Namens f1 in f2 |
| LENG,u,f ,L | Kürzen der Datei-Länge auf L Sektoren |
| PROT ,u,f ,p | Eingabe eines Schutzzeichens (Schreibschutz) |
| LIST ,u, (f) | Listen der eröffneten Datei-Namen und der zugehörigen Parameter (bzw. der Parameter einer bestimmten Datei f) |

Die Funktionen CREA, KILL, ALTR und LENG können auch als Befehle vom Benutzerprogramm aus gegeben werden.

## DATEI-ZUGRIFF VOM PROGRAMM

Zu Platten-Dateien kann vom Benutzer-Programm sektorweise lesend oder schreibend zugegriffen werden. Vorher ist jedoch die Datei mit

OPEN (u,f,w)

zu öffnen; dabei wird ihr eine Arbeits-Nummer w zugewiesen, mit der im folgenden gearbeitet wird. Der eigentliche Zugriff erfolgt durch die Befehle

| | |
|---|---|
| GFB(w,a,s,l) | Lesen von l Sektoren ab Sektor s in Kernspeicher-Feld a |
| PFB (w,a,s,l) | Schreiben von l Sektoren ab Sektor s aus Kernspeicher-Feld a |

Durch den Befehl

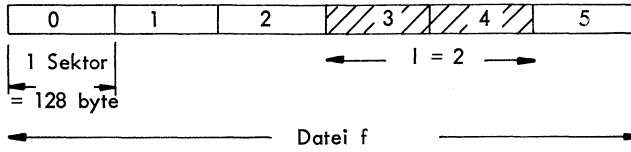
CLSE (w)

wird die Datei wieder geschlossen und die Nummer w freigemacht.

Die Arbeits-Nummer w läuft von 0 bis 7; d.h. es können bis zu 8 Dateien gleichzeitig bearbeitet werden.

Bei Multiprogramming-Systemen unter MPOS können insgesamt 32 Dateien gleichzeitig eröffnet sein und bearbeitet werden. Der Lese-/Schreib-Zugriff über GFB bzw. PFB erfolgt dabei nacheinander von den verschiedenen Ebenen aus in ihrer zeitlichen bzw. prioritären Reihenfolge.

Beispiel: Datei mit 6 Sektoren Länge. Zugriff auf Sektoren 3 und 4 (s = 3, l = 2):




SYSTEMFEHLER

D BOS umfaßt Routinen zur Feststellung und Behandlung von Systemfehlern, insbesondere

- Netzausfall und -wiederkehr
- Kernspeicher-Parity
- BUS-Fehler
- Plattenzugriffs-Fehler,

die entweder zu einer Rückkehr in den DBOS-Dialogbetrieb mit Ausgabe entsprechender Fehlermeldungen auf dem Konsolgerät führen oder vom Benutzer vorgesehene, auf seine speziellen Bedürfnisse zugeschnittene Routinen ansprechen.

LISTE DER DBOS-FUNKTIONEN

| Name | Kommando | Programmbefehl | Bedeutung |
|---|---|---|---|
| CREA | x | x | Datei eröffnen |
| KILL | x | x | Datei löschen |
| ALTR | x | x | Dateinamen ändern |
| LENG | x | x | Dateilänge kürzen |
| PROT | x | | Dateischutz eingeben |
| LIST | x | | Dateiparameter listen |
| GET | x | | Programm laden |
| RUN | x | | Programm laden und starten |
| PUT | x | | Programm ablegen |
| OPEN | | x | Datei öffnen |
| CLSE | | x | Datei schließen |
| GFB | | x | Block lesen |
| PFB | | x | Block schreiben |

# DFMS

VORBEMERKUNG

DFMS (Disk File Management System) ist ein komfortables Zugriffs- und Verwaltungssystem für Platten-Dateien. Es schließt alle Funktionen des Grund-Betriebssystems DBOS ein, enthält jedoch eine Reihe von zusätzlichen Funktionen, insbesondere:

- Eröffnen von vorübergehenden Dateien (temporary files)
- Datei-Schutzfunktionen
- Strukturierung von Dateien in Sätze (records)
- Satz-Zugriff über Satz-Nummern oder Satz-Schlüssel (keys)
- Exklusiver Zugriff zu Sätzen
- Lesen und Schreiben von Datei-Inhalten mit sequentiellem oder Random-Zugriff.

Die Funktionen des DFMS sind bei den DIETZsystemen 621 C und 621 D identisch; jedoch kommt vor allem letzteres wegen seiner größeren Plattenspeicher-Kapazität für das DFMS in Betracht.

DFMS ist vor allem in Verbindung mit C-BASIC und FORTRAN IV zur Behandlung kommerziell-administrativer und technisch-wissenschaftlicher Datenbestände auf Plattenspeichern geeignet.

DATEI-STRUKTUREN

DFMS erlaubt den Aufbau von Dateien, die

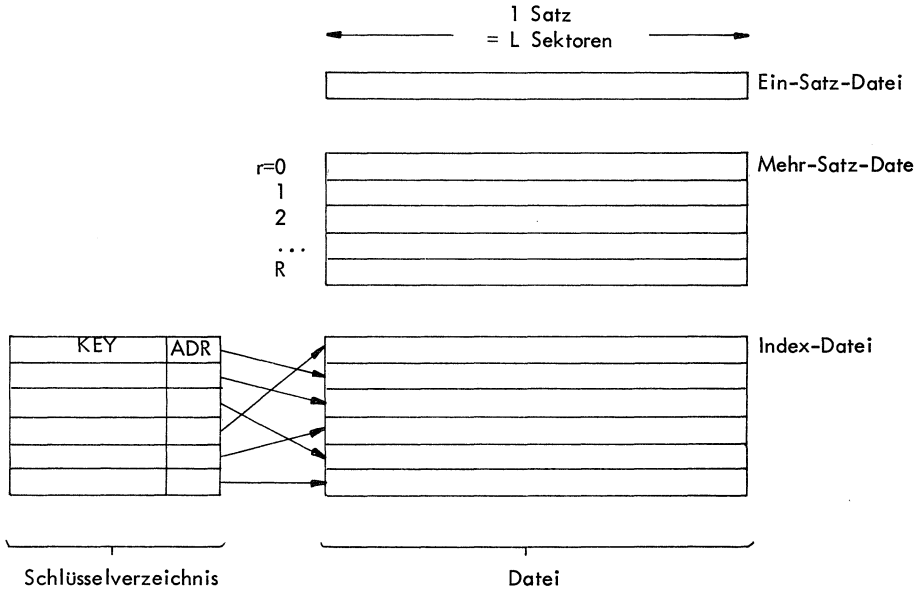
- aus nur einem Satz
- aus mehreren Sätzen gleicher Länge
  (mit Zugriff über die Satz-Nummer)
- aus mehreren Sätzen gleicher Länge sowie einem Schlüsselverzeichnis
  (Index-Datei mit Zugriff über keys)

bestehen.

Art und Größe der Datei werden bei der Eröffnung definiert. Jeder Satz hat die Größe eines oder einer ganzen Zahl von Sektoren von je 128 byte.

Jede Datei hat einen Namen f, der max. 6 Zeichen lang ist; unter Angabe des Namens und der Plattenspeicher-Nummer u kann zur Datei zugegriffen werden. Die Auswahl des Satzes erfolgt entweder über die Satz-Nummer r oder auch - im Falle der Index-Datei -

über einen Schlüssel k. Das Schlüsselverzeichnis wird durch einen gesonderten Befehl eröffnet, in dem insbesondere die Länge der Schlüssel (bis zu 60 Zeichen) angegeben ist.




DATEI-VERWALTUNG IM KOMMANDO-BETRIEB

Der Benutzer kann wie unter DBOS im Dialog Dateien eröffnen, löschen, umbenennen, das Dateiverzeichnis ausdrucken, Programme holen und ablegen, indem er eines der Kommandos

| | |
|---|---|
| CREA | LIST |
| KILL | GET |
| ALTR | RUN |
| LENG | PUT |
| PROT | |

benutzt. CREA beinhaltet zusätzlich die Angabe der Satz-Anzahl R.

Hinzu kommt das Kommando

CRIN

für das Eröffnen des Schlüsselverzeichnisses einer Index-Datei.

Im Kommando-Betrieb eröffnete Dateien haben stets dauernden Charakter (permanent files); sie sind explizit durch KILL zu löschen.

## DATEI-VERWALTUNG DURCH PROGRAMMBEFEHLE

Permanente Dateien können außer durch Dialog-Kommandos auch im Programm verwaltet werden; dazu dienen die Befehle

| | |
|---|---|
| CREA(u,f ,L,R) | Eröffnen einer Datei mit Namen f auf Plattenspeicher u mit R Sätzen zu je L Sektoren |
| KILL (u,f) | Löschen der Datei f auf Plattenspeicher u |
| ALTR (u,f1,f2) | Ändern des Datei-Namens f1 in f2 |
| CRIN(u,f ,K) | Eröffnen eines Schlüsselverzeichnisses zur Datei f mit der Schlüssel-Länge K |

Daneben kann das Programm Dateien auch vorübergehend eröffnen ("temporary" oder "scratch files"); dies geschieht durch den Befehl

| | |
|---|---|
| OPNW(u,w,L,R) | Vorübergehendes Eröffnen einer Datei mit der Arbeits-Nummer w |

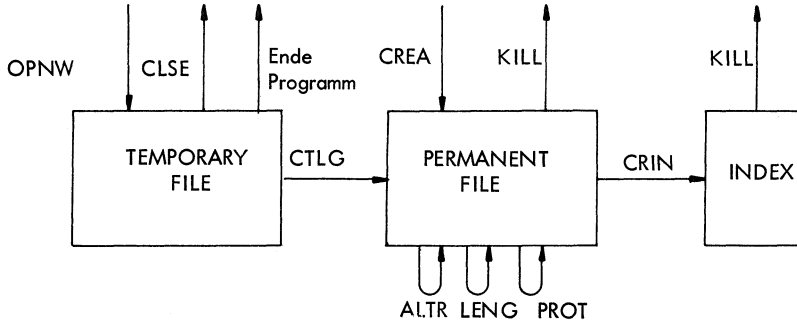
Eine so eröffnete Datei wird entweder durch den Befehl

| | |
|---|---|
| CLSE(w) | Datei w schließen |

gelöscht oder dadurch, daß das Programm beendet wird und die Kontrolle wieder von DBOS übernommen wird. Durch den Befehl

| | |
|---|---|
| CTLG(w,f) | Eintragen unter Name f in Datei-Verzeichnis |

wird jedoch eine vorübergehend eröffnete Datei zu einer permanenten Datei.




DATEI-ZUGRIFF VOM PROGRAMM

Bevor mit einer permanenten Datei gearbeitet werden kann, ist sie zu öffnen durch

OPEN(u,f,w)

wobei ihr eine Arbeits-Nummer w zugeteilt wird, auf die alle folgenden Befehle Bezug nehmen. Entsprechendes gilt für vorübergehend eröffnete Dateien und den zugehörigen Befehl

OPNW(u,w,L,R)

Durch den Befehl

CLSE(w)

wird eine Datei wieder geschlossen, die Arbeits-Nummer w freigegeben und die Datei, falls sie vorübergehend eröffnet war, wieder gelöscht.

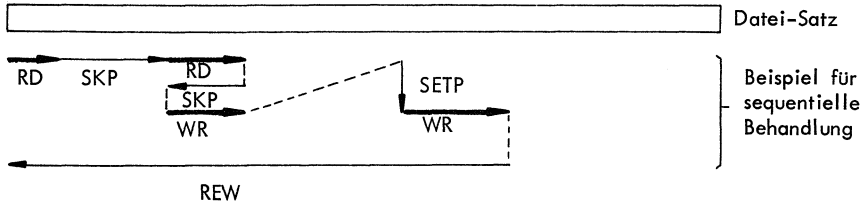
Eine geöffnete Datei kann mit den Befehlen

| | |
|---|---|
| GFB(w,a,s,l) | Lesen |
| PFB (w,a,s,l) | Schreiben |

sektorweise gelesen bzw. beschrieben werden, wie unter DBOS beschrieben.

Stattdessen gibt es jedoch auch die Möglichkeit der sequentiellen Beschreibung des Datei-Inhaltes. Hierzu dienen die Befehle

| | |
|---|---|
| RD (w,a,n) | Lesen n Zahlenwerte in Speicherfeld a |
| WR (w,a,n) | Schreiben n Zahlenwerte in Speicherfeld a |
| RDS (w,a,n,c) | Lesen n Zeichen bzw. bis Zeichen "c" |
| WRS (w,a,n,c) | Schreiben n Zeichen bzw. bis Zeichen "c" |
| SKP (w,n,z) | Überspringen z mal n Zahlenwerte (vor- oder rückwärts) |
| SKPS(w,n,c,z) | Überspringen z mal n Z ichen bzw. z mal über Zeichen "c" (vor- oder rückwärts) |
| SETP (w,n,z) | Setzen Pointer auf absolute Position (z mal n byte) im Satz |
| BKSP (w) | Setzen Pointer auf Satzanfang |




A lle vorgenannten Befehle beziehen sich auf den Zugriff innerhalb eines Satzes. Nach OPEN bzw. OPNW steht der Datei-Zeiger (Pointer) am Anfang des ersten Satzes der Datei. Zur Auswahl eines beliebigen Satzes dient der Befehl

| | |
|---|---|
| SREC(w,r) | Satz r selektieren |

Alle künftigen Befehle beziehen sich dann auf diesen Satz, wobei am Satzanfang begonnen wird. Stattdessen kann auch mit

| | |
|---|---|
| NREC(w) | Springen zum nächsten Satzanfang |

die Behandlung des folgenden Satzes eingeleitet werden.

Im Falle von Index-Dateien wird der gewünschte Satz über

| | |
|---|---|
| SKEY(w,k) | Satz zu Schlüssel k selektieren |

angesprochen und der Pointer auf dessen Anfang gestellt.

Zur Behandlung des Satzschlüssel-Verzeichnisses dienen folgende Befehle:

| | |
|---|---|
| ENTK (w,k) | Schlüssel k eintragen |
| ALTK (w,k1,k2) | Schlüssel k1 in k2 umbenennen |
| DELK (w,k) | Schlüssel k löschen |
| GTFK (w,a) | Ersten Schlüssel aus Verzeichnis lesen |
| GTNK (w,a) | Nächsten Schlüssel aus Verzeichnis lesen |

In Multiprogramming-Systemen kann einem Programm über die Befehle

| | |
|---|---|
| EREC (w,r) | Exklusiver Zugriff zu Satz r |
| EKEY(w,k) | " " " " gemäß Schlüssel k |

der alleinige Zugriff zu einem Satz erteilt werden; diese Befehle verhindern den Zugriff von jeder anderen Programmebene aus, entsprechen im übrigen jedoch den Befehlen SREC bzw. SKEY. Zur gleichen Zeit kann eine Programmebene nur zu einem Satz den ausschließlichen Zugriff haben; die Exklusivität endet mit Anwahl eines anderen Satzes oder mit Schließen der Datei über CLSE.

FEHLERMELDUNGEN

Fehler bei der Dateibehandlung im Programm, z.B.

- Eröffnung doppelt definierter Dateinamen
- Zugriff zu nicht existierenden oder geschützten Dateien
- Zugriff zu exklusiv beanspruchten oder nicht vorhandenen Sätzen
- Überschreiten der Satz- bzw. Dateilänge

werden im DFMS festgestellt und abfragbar gemacht, so daß das Benutzer-Programm in geeigneter Weise darauf reagieren kann.

LISTE DER DFMS-FUNKTIONEN

| Name | Kommando | Programmbefehl | Bedeutung | |
|---|---|---|---|---|
| CREA | x | x | (permanente) Datei eröffnen | |
| CRIN | x | x | Schlüsselverzeichnis eröffnen | |
| KILL | x | x | Datei löschen | |
| ALTR | x | x | Dateinamen ändern | |
| LENG | x | x | Dateilänge kürzen | |
| PROT | x | | Dateischutz eingeben | |
| LIST | x | | Dateiparameter listen | |
| GET | x | | Programm laden | |
| RUN | x | | Programm laden und starten | |
| PUT | x | | Programm ablegen | |
| OPNW | | x | Datei vorübergehend eröffnen | |
| OPEN | | x | Datei öffnen | |
| CLSE | | x | Datei schließen | |
| CTLG | | x | Datei permanent machen | |
| SREC | | x | Satz nach Nummer anwählen | |
| EREC | | x | " " " exklusiv anwählen | |
| SKEY | | x | Satz nach Schlüssel anwählen | |
| EKEY | | x | " " " exklusiv anwählen | |
| GFB | | x | Block lesen ) | Random |
| PFB | | x | Block schreiben ) | Random |
| RD | | x | Lesen Zahl | Sequentiell |
| WR | | x | Schreiben Zahl | Sequentiell |
| RDS | | x | Lesen Zeichen | Sequentiell |
| WRS | | x | Schreiben Zeichen | Sequentiell |
| SKP | | x | Springen über Zahl | Sequentiell |
| SKPS | | x | Springen über Zeichen | Sequentiell |
| SETP | | x | Pointer versetzen | Sequentiell |
| BKSP | | x | Zurück zu Satzanfang | Sequentiell |
| NREC | | x | Vor zu nächsten Satzbeginn | Sequentiell |
| ENTK | | x | Key eintragen | |
| ALTK | | x | Key ändern | |
| DELK | | x | Key löschen | |
| GTFK | | x | Ersten Key lesen | |
| GTNK | | x | Nächsten Key lesen | |

# MPOS

VORBEMERKUNG

Bei Multiprogramming-Anwendungen des DIETZsystems werden

- die Programmauftrags-Verwaltung und
- die Verwaltung gemeinsamer Betriebsmittel (Ressourcen)

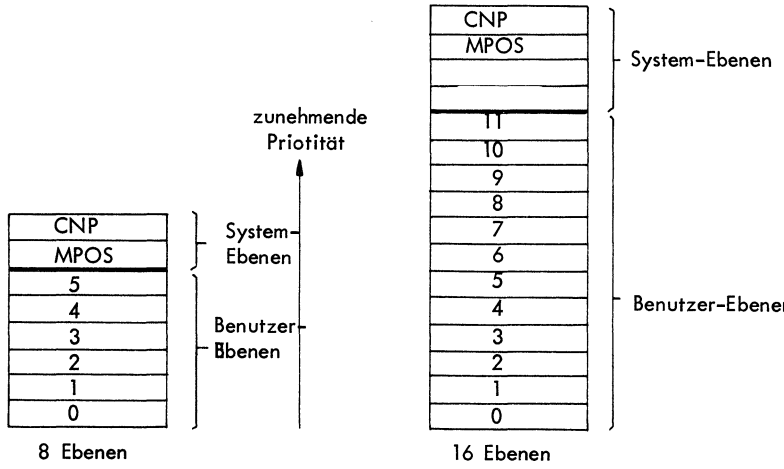
durch MPOS (Multi Program Operating System) geregelt. Dieses Betriebssystem, das stark durch die Programmebenen-Struktur des DIETZ 621 unterstützt wird, ist bei Multiprogramming-Betrieb von BASIC und C-BASIC implementiert und ist zugleich ein Bestandteil des für MARS 600 und BASEX verwendeten Echtzeit-Betriebssystems RTOS.

EBENEN-STRUKTUR

Multiprogramming bedeutet gleichzeitiger Ablauf mehrerer Programme, die gemeinsame Ressourcen (CPU, Kernspeicher, Plattenspeicher, Peripherie) benutzen.

Beim DIETZsystem 621 belegt jedes dieser Programme eine "Ebene" mit einem eigenen Register-Satz (von 128 byte Größe), in dem der Programm-Kontext enthalten ist. Die Ebenen und damit die Programme haben unterschiedliche Priorität, so daß im Regelfall ein Programm auf der Ebene I die Programme auf den Ebenen 0...(I-1) unterbricht, sobald es läuft, d.h. die CPU benutzt.

Es wird unterschieden zwischen Benutzer-Ebenen, auf denen Anwender-Programme laufen können, und System-Ebenen, die dem Betriebssystem vorbehalten sind. Je nach System-Konfiguration sind 6 oder 12 Benutzer-Ebenen verfügbar:



Benutzer-Ebene 0 ist dadurch gekennzeichnet, daß ein über DBOS (sowie bei Netzwiederkehr mit automatischem Restart) gerufenes Programm zunächst in dieser Ebene läuft (von der aus dann alle anderen Benutzer-Ebenen aktiviert werden können).

Zwei bzw. vier Ebenen sind dem Benutzer nicht zugänglich; eine dieser System-Ebenen ist für Funktionen des MPOS reserviert, die andere (CNP) für die Erkennung von Systemfehlern (die in DBOS ausgewertet werden) sowie (im Falle des RTOS) für die Echtzeit-Uhr.

## PROGRAMM-AUFTRÄGE

Zur Synchronisation der Abläufe in einem Multiprogramming-System besteht unter MPOS die Möglichkeit, von der laufenden Ebene aus ein Programm in einer anderen Ebene l, beginnend an der Stelle n, zu aktivieren. Die Befehle hierzu lauten z.B.

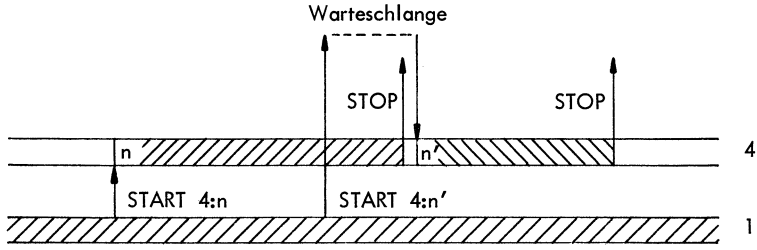
| | |
|---|---|
| in MARS 600: | XP,l,n |
| in BASIC/BASEX: | START l:n |

Läuft kein anderes Programm in der Zielebene l, so wird es umgehend aktiviert; andernfalls wird es in eine Warteschlange eingereiht, die vom MPOS gesteuert abgearbeitet wird.

Jedes Programm meldet sich beim MPOS ab, sobald es beendet ist; der zugehörige Befehl lautet

| | |
|---|---|
| in MARS 600: | EJ |
| in BASIC/BASEX: | STOP |

Beispiel für Programm-Aufträge von Ebene 1 an Ebene 4:



RESSOURCE-VERWALTUNG

Eine zweite Funktion des MPOS ist die Verwaltung von Ressourcen, die von mehreren Programmebenen benutzt werden.

Die Plattenspeicher des DIETZsystems 621 sind gemeinsame Betriebsmittel. Sie werden in der Weise verwaltet, daß jeweils ein Plattenzugriff entsprechend dem jeweiligen Programmbefehl abgewickelt wird, bevor eine andere Ebene zur Platte zugreifen kann. Liegen Zugriffs-Befehle von mehreren Ebenen vor, so werden sie in absteigender Reihenfolge der Prioritäten nacheinander abgearbeitet.

MPOS regelt außerdem den exklusiven Zugriff zu einem Satz einer Datei, der von jeder Ebene aus einmal gleichzeitig möglich ist.

In ähnlicher Weise wie der Platten-Zugriff werden das unter RTOS ansprechbare Analog-Meßsystem und ähnliche, mit direktem Speicherzugriff arbeitende Systeme verwaltet.

Peripherie-Geräte, wie Drucker, Bildschirm-Terminals usw. sind beim DIETZsystem 621 von allen Benutzer-Ebenen aus anzusprechen. Für den Fall, daß Konflikte (gleichzeitiger Zugriff mehrerer Ebenen auf ein Peripheral) zu befürchten sind, kann das betreffende Periphergerät mit der Geräte-Nummer d vom Benutzer-Programm zur alleinigen Benutzung angefordert und später wieder freigegeben werden; dazu dienen die Programmbefehle

| | |
|---|---|
| PREQ(d) | Ressource d anfordern und belegen, wenn frei |
| PREL (d) | Ressource d freigeben |
| RRQW (d) | Ressource d anfordern, belegen wenn frei bzw.warten bis frei |

Diese Befehle können zur Verwaltung nicht nur von Periphergeräten, sondern auch beliebiger anderer Betriebsmittel im Sinne von Semaphoren verwendet werden, z.B. von Programmen, Speicherbereichen usw.

# RTOS

VORBEMERKUNG

RTOS (Real Time Operating System) ist ein Echtzeit-Betriebssystem, das Prozeßanwendungen des DIETZsystems 621 unterstützt und in Verbindung mit den Programmiersprachen MARS 600 und BASEX arbeitet.

Es bietet - neben denen des MPOS - folgende Funktionen:

- Laufende Führung der Echtzeit
- Verwaltung von Zeitaufträgen
- Verwaltung von Interrupts
- Behandlung der Prozeßperipherie.

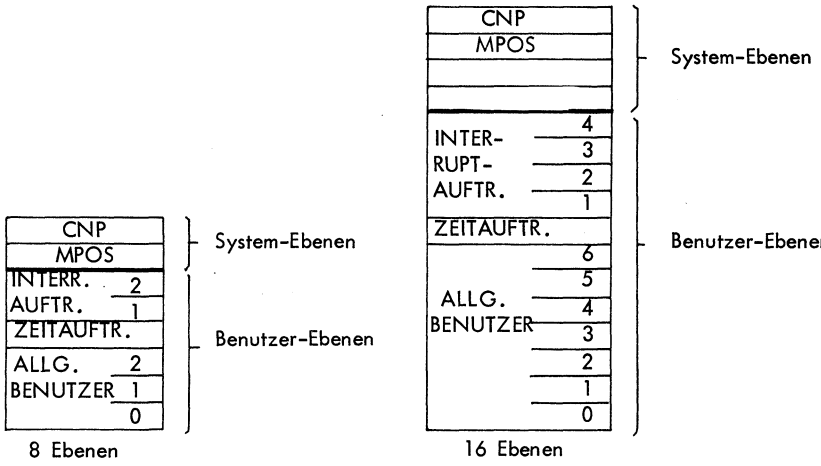
MARS 600 und BASEX benutzen zwei Versionen des RTOS, die geringfügige Unterschiede aufweisen, in den Grundzügen jedoch identisch sind. Die einzelnen Programmbefehle sind den Sprachbeschreibungen zu entnehmen; im folgenden werden BASEX-Beispiele herangezogen.

EBENEN-AUFTEILUNG

Unter RTOS werden die Programmebenen des DIETZsystems 621 bestimmten Funktionen zugeordnet:

- Systemebenen: wie MPOS
- Benutzer-Ebenen: 2 oder 4 Interrupt-Auftragsebenen
  1 Zeitauftrags-Ebene
  3 oder 7 allgemeine Benutzer-Ebenen

Je nach Systemkonfiguration ergibt sich folgende Aufteilung:



ECHTZEIT-UHR

Unter RTOS wird laufend die Echtzeit geführt. Sie wird von der quarzgesteuerten Uhr der CPU abgeleitet und ist mit einer Auflösung von 1, 10, 100 oder 1000 ms verfügbar (konfigurationsabhängig). Zu Beginn der Programmausführung läuft die Echtzeit-Uhr von Null an.

ZEITAUFTRAGS-VERWALTUNG

Das System kann von beliebigen Benutzer-Ebenen aus veranlaßt werden, zu vorgegebenen Zeiten bestimmte, programmierte Funktionen (Zeitauftrags-Routinen) durchzuführen.

Ein Zeitauftrag lautet beispielsweise:

```
1ØØ AFTER 25Ø : LET OUTB(2)=1        (Nach 250 ms Bit-Ausgang 2 setzen).
```

Zeitauftrags-Routinen laufen in einer eigens dafür reservierten Benutzer-Ebene mit relativ hoher Priorität ab.

Es können gleichzeitig bis zu 40 Zeitaufträge geführt werden. Werden zwei oder mehr Zeitaufträge zur gleichen Zeit aktuell, so werden sie nacheinander abgearbeitet.

INTERRUPT-VERWALTUNG

Interrupts im Sinne des RTOS sind statistisch auftretende Signale aus dem Prozeß, die auf Interrupt-Eingänge wirken und auf die das System in vorbestimmter Weise zu reagieren hat; die Reaktion besteht aus einer programmierten Interrupt-Routine.

Das System kann von beliebigen Benutzer-Ebenen aus beauftragt werden, einen eintreffenden Interrupt mit einer bestimmten Reaktion zu beantworten. Beispiele für Interrupt-Aufträge:

```
2ØØ ON INT 6 : LET A = INA(3)        (Bei Auftreten von Interrupt 6: Analog-
                                      eingang 3 nach A).

35Ø ON INT 1 : GOTO 5ØØ              (Bei Auftreten von Interrupt 1:
...
5ØØ CALL HLTC(Ø)                     Zähleingang Ø schließen und
51Ø LET Z12 = INC(Ø)                 Zählerinhalt nach Z12).
```

Interrupt-Routinen laufen in 2 bzw. 4 hierfür reservierten Benutzer-Ebenen mit hoher Priorität. Je nach ihrer Nummer sind die Interrupt-Eingänge einer dieser Ebenen zugeordnet und haben damit untereinander verschiedene Priorität bei der Ausführung der zugehörigen Routinen.

Interrupt-Eingänge können einzeln maskiert und wieder geöffnet werden:

```
DISAB 1,2        (Interrupt 1 und 2 verhindern)
ENAB 2           (Interrupt 2 zulassen)
```

Auch bei maskiertem Eingang werden eintreffende Interrupts gespeichert; sie werden bei Wiederzulassung sofort wirksam. Bei Beginn der Programm-Ausführung sind alle Interrupt-Eingänge maskiert; erwünschte Interrupts sind durch ENAB zuzulassen.

Treffen mehrere Interrupts, die zur gleichen Ebene gehören, gleichzeitig oder während des Laufs einer Interrupt-Routine auf einer Ebene ein, so werden sie nacheinander in der Reihenfolge steigender Nummern abgearbeitet. Dagegen unterbrechen Interrupts, die zu einer höheren Ebene gehören, Interrupt-Routinen in niedrigen Ebenen (sowie eine eventuell laufende Zeitauftrags-Routine und Programme in den allgemeinen Benutzer-Ebenen).

## PROZESSPERIPHERIE

Das RTOS unterstützt die gesamte Prozeßperipherie des DIETZsystems 621; jedoch werden im allgemeinen nur die I/O-Treiber in das RTOS aufgenommen, welche die jeweilige Systemkonfiguration benötigt.

Außer den oben erwähnten Interrupt-Eingängen sind u.a. folgende Prozeßanschlüsse ansprechbar:

| | | |
|---|---|---|
| - Statische digitale Eingänge: | INB(x) | Bitweise Abfrage |
| | INW(x) | Wortweise Abfrage binär |
| | IND(x) | " " BCD |
| - Speichernde digitale Ausgänge: | OUTB(x) | Bitweise Ausgabe |
| | OUTW(x) | Wortweise Ausgabe binär |
| | OUTD(x) | " " BCD |
| - Einkanal-Analogeingänge: | INA(x) | |
| - Einkanal-Analogausgänge: | OUTA(x) | |

| | | |
|---|---|---|
| Analog-Meßsysteme: | CALL ADCS (a,q,k) | Einkanal-Messung |
| | CALL ADCD (a,q,k) | Doppelkanal-Messung |
| | CALL ADCM(a,q,k) | Mehrkanal-Messung |
| - Zähleingänge: | OUTC(x) | Setzen |
| | INC (x) | Abfragen |
| | CALL ACTC (x) | Öffnen |
| | CALL HLTC (x) | Schließen |
| - Zeitausgänge: | OUTT(x) | Setzen |
| | CALL ACTT (x) | Aktivieren |

Bemerkung: x = Nummer des Ein-/Ausgangs
a = Speicherfeld
q = Anzahl Meßwerte
k = (Basis-) Meßkanal

# MARS 600

MARS 600 (Makro-Assembler für Realtime-Systeme) bietet die Möglichkeit, DIETZ 621-Systeme für Echtzeit- und Prozeßbetrieb zu programmieren.

MARS 600 ist eine Erweiterung der Assembler-Sprache des DIETZ 621; alle Elemente des Assemblers MINCASS 600 M sind darin enthalten (siehe Abschnitt ASSEMBLER). MARS 600-Programme stützen sich auf das Echtzeit-Betriebssystem RTOS.

Im folgenden sind alle Makrobefehle aufgeführt, die für MARS 600 spezifisch sind.

## PROGRAMMBEGINN UND -ENDE

Diese Befehle dienen zum Initialisieren bestimmter Funktionen des Betriebssystems, zum Abschluß von Auftragsprogrammen und zur Beendigung des Gesamtprogramms:

| | |
|---|---|
| BGN | Initialisiert Teilfunktionen des Betriebssystems (z.B. System-Uhr) |
| EJ | Beendet ein durch XP, XD, XT oder XI ausgelöstes Auftragsprogramm; die betreffende Ebene meldet sich beim Betriebssystem ab und wird für ein anderes Programm frei. |
| END | Ende des Gesamtprogramms (mit Rückgabe der Kontrolle an DBOS). |

## PROGRAMM-VERWALTUNG

Von jeder Benutzer-Ebene aus können Programme gestartet werden, die in anderen Benutzer-Ebenen (außer in den Zeit- und Interrupt-Ebenen) laufen. Sofern die beauftragte Ebene frei ist (und in keiner anderen höheren Ebene ein Programm aktiv ist), wird das Auftragsprogramm unverzüglich begonnen; andernfalls wird es in eine Warteschlange von Programmen eingereiht, die nacheinander abgearbeitet werden.

| | |
|---|---|
| XPm,@,l<br>2*A,addr | In Ebene l wird ein Programm gestartet, das an der Stelle addr beginnt |

## ZEITVERWALTUNG

Die in RTOS laufend geführte Systemzeit, die von der quarzgesteuerten Rechner-Uhr gesteuert wird, erlaubt die Erteilung von Programmaufträgen, die mit einer angebbaren Verzögerung oder zu einem bestimmten Zeitpunkt ausgeführt werden. Außerdem kann die Systemzeit abgefragt und neu gesetzt werden.

| | |
|---|---|
| XDm, @, h<br>2*A, addr | Realtivzeit-Auftrag. Löst nach einer Zeitverzögerung t ein Programm auf der Zeitebene aus, das an der Stelle addr beginnt. Die Zeit t (ms) steht in @ (bei h = 0) bzw. in @ und @+1 (bei h = 1). |
| XTm, @, h<br>2*A, addr | Absolutzeit-Auftrag. Löst zum Zeitpunkt T der System-Uhr ein Programm auf der Zeitebene aus, das an der Stelle addr beginnt. Die Zeit T steht in @ und hat die Bedeutung Sekunden (h = 0), Minuten (h = 1) oder Stunden (h = 2). |
| SEC | Liefert den Sekunden-Teil der Systemzeit (in @). |
| MIN | Liefert den Minuten-Teil der Systemzeit (in @). |
| HOUR | Liefert den Stunden-Teil der Systemzeit (in @). |
| TIME | Setzt den Sekunden-, Minuten- und Stunden-Teil der Systemzeit auf den Inhalt von @, @+1 und @+2. |

INTERRUPT-VERWALTUNG

Auf externe Interrupts reagiert das System durch Auftragsprogramme, die durch einen Makrobefehl der Interrupt-Nummer i zugewiesen werden und die bei Eintreffen des Interrupts auf einer der Interrupt-Ebenen ablaufen:

| | | |
|---|---|---|
| i = Ø... 31 | Interrupt-Ebene | 1 |
| i = 32... 63 | " " | 2 |
| i = 64... 95 | " " | 3 |
| i = 96...123 | " " | 4 |

↑ zunehmende Priorität

Gleichzeitig eintreffende Interrupts, die zu einer Ebene gehören, werden in der Reihenfolge zunehmender Nummern i nacheinander abgewickelt.

Weitere Makrobefehle dienen zum Maskieren und Öffnen der Interrupt-Eingänge sowie zum Löschen gespeicherter Interrupts. Zu Programmbeginn sind alle Interrupts maskiert.

| | |
|---|---|
| XIm, @, i<br>2*A, addr | Weist dem Interrupt i ein Auftragsprogramm zu, das bei addr beginnt. |
| EIm, @, i | Öffnet den Interrupt-Eingang i. |
| DIm, @, i | Schließt (maskiert) den Interrupt-Eingang i. |
| CIm, @, i | Löscht einen eventuell gespeicherten Interrupt am Eingang i.<br>(Auch bei maskiertem Eingang wird ein eintreffender Interrupt gespeichert. CI ist im Normalfall nicht nötig, da bei Beginn eines Interrupt-Auftragsprogramms der Interrupt-Speicher automatisch gelöscht wird). |

## DIGITALE EIN/AUSGÄNGE

Die folgenden Befehle dienen zur Behandlung von statischen digitalen Eingängen und speichernden digitalen Ausgängen. Jeweils 16 davon befinden sich auf einem Einkarten-Interface, das bei wortweiser (16-bit-weiser) Behandlung durch seine Nummer x (x = 0, 1, 2...) gekennzeichnet ist. Jedes Bit der Interfaces kann auch einzeln behandelt werden; seine Nummer y ist anzugeben (y = 0...15 für x = 0; y = 16...31 für x = 1, usw.).

| | |
|---|---|
| IWm ,@,x | Wortweise Abfrage von Eingang x nach @, @+1 |
| IBm ,@,y | Bitweise Abfrage von Eingang y nach @ (Bit 0) |
| TBm ,@,y<br>2*A,addr | Testen von Bit-Eingang y. Falls Ergebnis = 1, Sprung nach der Stelle addr; sonst Fortsetzung des Programms mit dem nächsten Befehl. |
| OWm,@,x | Wortweise Ausgabe von @, @+1 nach Ausgang x. |
| OBm ,@,y | Bitweise Ausgabe von @(Bit 0) nach Ausgang y |
| ABm ,@,y | Bit-Ausgang y auf 1 Setzen. |
| CBm ,@,y | Bit-Ausgang y nullstellen. |

## ZÄHLEINGÄNGE UND ZEITAUSGÄNGE

Mit den folgenden Befehlen werden 16-bit-Zähleingänge und 16-bit-Zeitausgänge (Timer) behandelt, die sich auf Einkarten-Interfaces befinden und die durch eine Nummer x (x = 0, 1, 2...) gekennzeichnet sind.

| | |
|---|---|
| ICm ,@ ,x | Wortweise Abfrage von Zähler x nach @, @+1 |
| OCm, @ ,x | Wortweise Ausgabe von @, @+1 nach Zähler x |
| CCm, @ ,x | Zähler x nullsetzen |
| ACm, @ ,x | Zähleingang x öffnen |
| HCm, @ ,x | Zähleingang x schließen |
| OTm, @ ,x | Wortweise Ausgabe von @, @+1 nach Zähler x |
| ATm ,@ ,x | Timer x auslösen |

## WATCHDOG

Die Watchdog enthält eine Zeitschaltung, die vom Programm laufend angestoßen werden soll. Unterbleibt der Anstoß für eine längere als die eingestellte Zeit, z.B. durch einen Fehler, so wird ein Alarmkontakt betätigt.

| | |
|---|---|
| AW | Anstoßen Watchdog |

## ANALOGE EIN/AUSGÄNGE

Die folgenden Befehle dienen zur Abfrage von Einkanal-Analogeingängen (12 bit) und Einkanal-Analysenausgängen (10 bit), die sich auf Einkarten-Interfaces befinden und - getrennt - durch Nummer x (x = 0, 1, 2...) gekennzeichnet sind. Ein weiterer Makrobefehl steuert das mit bis zu 64 Meßkanälen ausgerüstete Analog-Meßsystem (12 bit).

| | |
|---|---|
| IAm ,@,x | Abfrage Analogeingang x nach @ , @+1 (rechtsbündig) |
| OAm,@,x | Ausgabe von @ , @+1 (rechtsbündig) nach Analogausgang x |

```
GAm,@,h
2*A,addr
D,k
D,q
```

Auslösung des Analog-Meßsystems. Es werden q Meßwerte des Kanals k bzw. ab Kanal k in ein Speicherfeld übertragen, das an der Stelle addr beginnt.

Es bedeutet h = 0: Einkanal-Messung (nur k)
h = 1: Zweikanal-Messung (k,k+1,k,k+1,...)
h = 2: Mehrkanal-Messung (k,k+1,k+2,...)

Enthält das Meßsystem keinen Selbststeuer-(DMA-) Zusatz, so sind für h statt 0/1/2 die Konstanten 3/4/5 anzugeben.

Bei Zweikanal-Messung sind für k nur gerade Zahlen (0, 2, 4, ...) erlaubt.

Im Speicherfeld sind 2 Bytes je Meßwert zu reservieren; die Werte werden dort rechtsbündig abgelegt.

SYSTEMFEHLER

Bei Systemfehlern (sich ankündigender Netzausfall, BUS-Fehler, Kernspeicher-Parity-Fehler) wird die höchste Ebene des Systems aktiviert. Der Benutzer kann angeben, welche Fehlerbehandlungsprogramme in einem solchen Fall ausgeführt werden sollen. Dies geschieht durch Makrobefehle, die vom Benutzer-Programm einmal durchlaufen werden sollen und die den Fehlern bestimmte Sprungadressen zuweisen.

```
  POWF      Bei Netzausfall: Fehlerprogramm ab Stelle addr
2*A,addr

  BUSF      Bei BUS-Fehler: Fehlerprogramm ab Stelle addr
2*A,addr

  MEMF      Bei Speicher-Fehler: Fehlerprogramm ab Stelle addr
2*A,addr
```

HINWEISE

```
@        bedeutet Akkumulator (Register-Adresse '02)
@ +1       "          "   +1 (  "       "      '03)
@ +2       "          "   +2 (  "       "      '04)
```

m bezeichnet die Adressierungsart (C, X, R, L oder A). Beispiele für IAm,@ ,x:

```
IAC,@ ,3          x = 3          (Konstante)
IAX,@,    ,'5F    x = << '5F>>   (Indexregister '5F)
IAR ,@ ,REG       x = < REG >    (Register REG)
IAA,@,ADR         x = < ADR >    (Absolute Adresse ADR)
```

MARS 600 benutzt die ersten 64 Register jeder Benutzer-Ebene ('ØØ...'3F). Die restlichen 64 Register ('4Ø...7F) stehen dem Benutzer zur freien Verfügung. Außerdem werden der Akkumulator @ ('Ø2) sowie ggfs. die folgenden Register-Plätze ('Ø3, 'Ø4, ...) bei den MARS-Makrobefehlen häufig zur Übergabe von Daten benutzt.

| Makro | Bedeutung |
|---|---|
| BGN | Begin |
| EJ | End Job |
| END | End |
| XD | Execute with Delay |
| XT | Execute at Time |
| SEC | Seconds |
| MIN | Minutes |
| HOUR | Hours |
| TIME | Time |
| XI | Execute on Interrupt |
| EI | Enable Interrupt |
| DI | Disable Interrupt |
| CI | Clear Interrupt |
| XP | Execute Program |
| IW | In Word |
| IB | In Bit |
| TB | Test Bit |
| OW | Out Word |
| OB | Out Bit |
| AB | Activate Bit |
| CB | Clear Bit |
| IC | In Counter |
| OC | Out Counter |
| CC | Clear Counter |
| AC | Activate Counter |
| HC | Halt Counter |
| OT | Out Timer |
| AT | Activate Timer |
| AW | Activate Watchdog |
| IA | In Analog |
| OA | Out Analog |
| GA | Get Analog |
| POWF | Power Failure |
| BUSF | Bus Failure |
| MEMF | Memory Failure |

```
SEITE 0000

0000                    0    ,'4000 ;
0001 4000               PGN ;                          FD 3A 9C 40
0002 4004               XIC ,@    ,      0;            81 09 00 FD 3A 51 49
0003 400B          2* A    ,INT0  ;                    3C 40
0004 400D               XIC ,@    ,      1;            81 09 01 FD 3A 51 49
0005 4014          2* A    ,INT1  ;                    49 40
0006 4016               XIC ,@    ,      7;            81 09 07 FD 3A 51 49
0007 401D          2* A    ,ENDE  ;                    38 40
0008 401F               EIC ,@    ,      0;            81 09 00 FD 3A 74 49
0009 4026               EIC ,@    ,      1;            81 09 01 FD 3A 74 49
0010 402D               EIC ,@    ,      7;            81 09 07 FD 3A 74 49
0011 4034               FJ  ;                          FD 3A B1 48
0012 4038 ENDE:         END ;                          FD 3A EC 48
0013        ;
0014 403C INT0:         XPC ,@    ,      0;            81 05 00 FD 3A CA 48
0015 4043          2* A    ,LEV0  ;                    56 40
0016 4045               EJ  ;                          FD 3A B1 48
0017        ;
0018 4049 INT1:         XPC ,@    ,      1;            81 05 01 FD 3A CA 48
0019 4050          2* A    ,LEV1  ;                    83 40
0020 4052               EJ  ;                          FD 3A B1 48
0021        ;
0022 4056 LEV0:         LDC ,'41 ,      0;            81 41 00
0023 4059 A     :       LDCD,@    ,   1000;            1E 80 E8 03
0024 405D               XDC ,@    ,      1;            81 09 01 FD 3A C4 49
0025 4064          2* A    ,OUT0  ;                    6A 40
0026 4066               EJ  ;                          FD 3A B1 48
0027        ;
0028 406A OUT0:    2<*LDC ,@    ,      0;            1A 80 00
0029 406D               OWC ,@    ,      0;            81 09 00 FD 3A E0 4A
0030 4074               ABF ,@    ,'41   ;             85 09 41 FD 3A 15 4B
0031 407B               INEC,'41 ,'0F   ,A    ; 5B 41 0F DB
0032 407F               EJ  ;                          FD 3A B1 48
0033        ;
0034 4083 LEV1:         LDC ,'40 ,      0;            81 40 00
0035 4086 B     :       IWC ,@    ,      0;            81 09 00 FD 3A 6C 4B
0036 408D               OWC ,@    ,      0;            81 09 00 FD 3A E0 4A
0037 4094               INFC,'40 ,    255,B   ; 5B 40 FF EF
0038 4098               EJ  ;                          FD 3A B1 48
0039                    Z   ;
```

# BASIC

BASIC ist eine international gebräuchliche, am Dartmouth-College entwickelte Programmiersprache für mathematische und technisch-wissenschaftliche Aufgaben. Sie ist besonders leicht erlernbar und bietet dem Benutzer die Möglichkeit, im Dialogbetrieb zu arbeiten.

Im folgenden ist BASIC so beschrieben, wie es in den DIETZ 621-Systemen implementiert ist.

## BASIC-STATEMENTS

### REM

m REM Kommentar

Dient zur Einfügung von Bemerkungen; wird bei Ausführung übergangen.
Beispiel:

```
30 REM   AUSGANG 2 NULLSETZEN
```

### LET

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

Weist einer oder mehreren Variablen den Wert eines Ausdrucks zu.
Beispiele:

```
100 LET A=5.02
705 LET X222=X221 AND (INP(2) OR INP(17))
855 LET N=N+INA(1),N=N+1
```

### INPUT

m INPUT Variable 1, Variable 2, ...
m INPUT "Text", Varaibale 1, Variable 2, ...

Bewirkt Eingabe von Daten über ein Peripheriegerät, das mit PRINT DEV spezifiziert werden kann. Die Daten werden den Variablen nacheinander zugewiesen. Ein zwischen Anführungszeichen stehender Text wird vorher auf dasselbe Gerät ausgegeben.
Beispiele:

```
1020 INPUT X,Y,GAMM
1050 INPUT A$
1220 INPUT "KONTROLLWERT ",KWT
```

PRINT

m PRINT "Text"
m PRINT Ausdruck 1, Ausdruck 2, ...
m PRINT "Text", Ausdruck 1, Ausdruck 2, ...
m PRINT DEV (Gerät) ...
m PRINT FMT (Format) ...
m PRINT TAB (Spalten) ...
m PRINT

Bewirkt Ausgabe von Daten auf einem Periphergerät, das mit DEV spezifiziert werden kann. Zwischen Anführungszeichen stehende Texte werden direkt ausgegeben. Bei Ausdrucken wird der Wert berechnet und dieser ausgegeben, wobei das Format über FMT spezifizierbar ist. TAB läßt so viel Leerspalten, wie der dahinter in Klammern stehende Ausdruck ergibt. PRINT ohne weitere Angaben bedeutet "Leerzeile".
Beispiele:

```
730 PRINT "ERGEBNIS"
740 PRINT A,B,C,A*B+C
755 PRINT "ERGEBNIS";A*B+C,T%$
770 PRINT FMT(F5.2);A,B,FMT(E12.3);D
777 PRINT TAB(30*SIN(X)+30.5);"*"
780 PRINT
805 PRINT DEV(6);"MESSPROGRAMM"
```

DATA

m DATA Konstante 1, Konstante 2, ...

Eröffnet eine Liste von Konstanten, die durch READ Variablen zugewiesen werden. Die Konstanten aller DATA-Anweisungen gehören zu einem Block in der Reihenfolge der Auflistung bzw. der Zeilen-Nummern.
Beispiele siehe READ.

READ

m READ Variable 1, Variable 2

Weist den Variablen Konstanten aus dem DATA-Block zu.
Beispiele:

```
50 DATA 100,5.24,-.00003
105 DATA "AB",3.1416,32768,4711,1
260 READ H,V1,V2
275 READ TEXT$
395 READ PI,P215,EDC,EINS
```

Zugewiesen wird:

1ØØ → H, 5.24 → V1, -0.0003 → V2
AB → TEXT
3.1416 → PI, 32768 → P215, 4711 → EDC, 1 → EINS

RES

m RES
m RES n

Setzt den DATA-Zeiger auf den Beginn des ersten DATA-Feldes zurück bzw. auf das DATA-Feld mit der Nummer n. Die nächste READ-Anweisung liest wieder von dort.
**Beispiele:**

```
400 RES
450 RES 260
```

GOTO

m GOTO n
m GOTO Ausdruck OF n1, n2

Setzt das Programm bei der Anweisung mit der Nummer n fort. In der Form des berechneten GOTO wird das Programm an der Stelle n1, n2,... fortgesetzt, je nachdem, ob der Wert des Ausdrucks gleich 1, 2, ... ist. Ist der Wert des Ausdrucks kleiner als 1 oder größer als die Anzahl der Nummern, wird die darauf folgende Anweisung ausgeführt.
**Beispiele:**

```
30 GOTO 100
120 GOTO 2+INT(7) OF 100,20,155
```

GOSUB

m GOSUB n
m GOSUB Ausdruck OF n1, n2, ...

Ruft ein Unterprogramm auf, das mit der Anweisung n beginnt. Nach Rückkehr aus dem Unterprogramm (durch RETURN) wird die Anweisung mit der auf m folgenden Nummer ausgeführt. Das berechnete GOSUB entspricht dem berechneten GOTO.
**Beispiele:**

```
205 GOSUB A OF 310,320,330,355
600 GOSUB 1335
```

RETURN

m RETURN

Bewirkt den Rücksprung aus einem Unterprogramm.
Beispiel:

```
380 RETURN
```

IF

m IF Bedingung THEN n

Verzweigt zur Anweisung n, wenn die Bedingung erfüllt ist; andernfalls wird die Anweisung mit der auf m folgenden Nummer ausgeführt. Als Bedingung kann eine Gleichung oder eine Ungleichung eingesetzt werden, aber auch ein Boole'scher Ausdruck (erfüllt, wenn Wert = 1) oder ein arithmetischer Ausdruck (erfüllt, wenn Wert ungleich 0).
Beispiele:

```
20 IF X/5>=N THEN 60
80 IF A OR P OR INP(2) THEN 35
150 IF X12-1 THEN 125
252 IF INW(6)=0 THEN 310
300 IF 3*INA(1)+612<1024 THEN 330
```

FOR

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

Eröffnet eine Programmschleife; die zu Anfang auf den Wert des Ausdrucks 1 gesetzte Laufvariable wird durch die Anweisung NEXT so oft um 1 bzw. den Wert des Ausdrucks 3 erhöht, bis der Wert des Ausdrucks 2 erreicht oder überschritten ist; danach wird die Schleife beendet.
Beispiele siehe NEXT.

NEXT

m NEXT Laufvariable

Schließt eine Programmschleife formal ab. Nach dem letzten Durchlauf wird die Anweisung mit der auf m folgenden Nummer ausgeführt.
Beispiele:

```
150 FOR I=1 TO 4
155 LET Z(I)=INW(2+I)
160 NEXT I
```

DIM

m DIM Feld 1, Feld 2, ...

Reserviert im Speicher ein- und zweidimensionale Felder von Zahlenvariablen, Einzelvariablen und Overlay-Felder für Programmoduln.
Beispiel:

```
410 DIM B(39),ARRY(5,10),A(134)
```

CHAR

m CHAR String 1, String 2, ...

Reserviert im Speicher Stringvariablen.
Beispiel:

```
30 CHAR TX$(12),TOTO$(32)
```

DEF

m DEF FN... (Argument) = Ausdruck

Definiert eine Funktion. Nach FN steht Name. Das Argument ist eine Variable, die im Ausdruck rechts vom Gleichheitszeichen wieder vorkommen muß. FN... wird im Augenblick des späteren Aufrufs der durch den jeweiligen Wert der Variablen bestimmte Ausdruck zugewiesen.
Beispiel:

```
175 DEF FNDEGR(X)=1.42*SIN(X)*EXP(-.5*X)
```

CALL

m CALL Name
m CALL Name (Parameter)

Ruft eine Systemprozedur auf, die mit der Übergabe von bis zu 4 Parametern verbunden sein kann.
Beispiel:

```
200 CALL LDST(A(2),B$,4)
```

END

m END

Beendet den Ausführungsbetrieb und läßt das System in den Bedienungsbetrieb zurückkehren.
Beispiel:

```
9000 END
```

WEITERE SPRACHELEMENTE VON BASIC

| | |
|---|---|
| Namen: | Buchstaben, u.U. gefolgt von bis zu 3 Buchstaben oder Ziffern. |
| Zahlenvariablen: | Gekennzeichnet durch Namen. Können einfach oder doppelt indiziert sein. Interne Darstellung als Gleitkomma-Zahlen (3 byte Mantisse, 1 byte Exponent; Genauigkeit der Mantisse $10^{-7}$, Exponent $10^{\pm 40}$). |
| Stringvariablen: | Gekennzeichnet durch Namen mit Suffix $. Bestehen aus 2 oder mehr Zeichen. In Teilstrings zerlegbar, z.B. A$ (p,l). |
| Zahlenkonstanten: | In beliebigem Format anzugeben. |
| Stringkonstanten: | ASCII-Zeichen in Anführungszeichen eingeschlossen, z.B. "ABC". |

Operatoren:

| | |
|---|---|
| + | Addition |
| - | Subtraktion |
| * | Multiplikation |
| / | Division |
| ↑ | Potenzierung |
| NOT | Negation |
| AND | Konjunktion |
| OR | Disjunktion |
| MIN | Minimum |
| MAX | Maximum |
| < | kleiner |
| < = | kleiner oder gleich |
| = | gleich |
| # | ungleich |
| > = | größer oder gleich |
| > | größer |
| & | Stringverkettung |

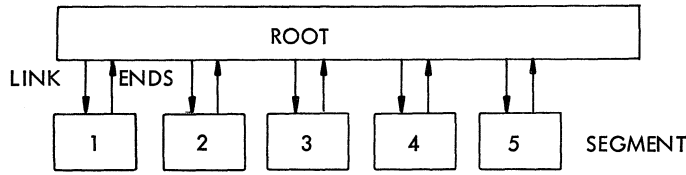
Funktionen:

| | |
|---|---|
| ABS | Absolutwert |
| INT | Ganzzahl-Teil |
| SGN | Vorzeichen |
| SQR | Quadratwurzel |
| SIN | Sinus |
| COS | Cosinus |
| TAN | Tangens |
| ATN | Arcustangens |
| LOG | Natürlicher Logarithmus |
| EXP | Exponentialfunktion |
| RND | Zufallsfunktion |

Klammerung: mit runden Klammern beliebig zulässig

PROGRAMM-SEGMENTIERUNG

Bei plattenunterstützten Systemen ist es möglich, BASIC-Programme zu segmentieren. Diese bestehen aus einem residenten Teil (ROOT), der nacheinander verschiedene Programmteile (SEGMENTE) von der Platte in einen Overlay-Bereich lädt und zur Ausführung bringt.




Zu beachten ist, daß nur ein Overlay-Bereich existiert, d.h. sich zur gleichen Zeit nur 1 Segment im Speicher befinden kann; außerdem können Segmente keine weiteren Segmente rufen. Dies gilt auch für Multiprogramming-Anwendungen von BASIC unter MPOS, wo segmentierte Programme zweckmäßig nur in einer Ebene laufen und die Programme aller anderen Ebenen resident zu halten sind.

Die zugehörigen Statements lauten:

LINK

m LINK Name

Lädt das Segment mit dem angegebenen Namen und bringt es, beginnend mit m, zur Ausführung.
Beispiel:

```
650 LINK ADAM
```

ENDS

m ENDS

Bewirkt die Rückkehr in die Root und zur Ausführung des auf LINK folgenden Statements.

Zur Generierung bzw. Verkettung eines segmentierten Programms dienen später beschriebene Kommandos.

## DATEI-ZUGRIFFS-BEFEHLE

Die unter dem Basis-Betriebssystem bzw. dem Datei-Verwaltungssystem zulässigen Zugriffsbefehle sind in Abschnitt DBOS undDFMS beschrieben. Sie werden mit CALL... (...) aufgerufen.

## SYSTEMPROZEDUREN IN BASIC

BASIC enthält in der Grundversion folgende Systemprozeduren:

| | | |
|---|---|---|
| String-/Zahl-Umspeicherung: | CALL LDST (a,b$,l) | Der String b$ wird mit dem Inhalt des Zahlenfeldes a geladen; es werden l Bytes übertragen. |
| | CALL STST (a,b$,l) | Der Inhalt des Strings b$ wird im Zahlenfeld a gespeichert; es werden l Bytes übertragen. |

## MULTIPROGRAMMING-BASIC

BASIC kann unter MPOS in mehreren Benutzer-Ebenen laufen. In dieser Multiprogramming-Version von BASIC sind folgende Systemvariablen bzw. Systemprozeduren zusätzliche vorgesehen:

| | | |
|---|---|---|
| Ebenen-Bindung: | LEV | Liefert die Nummer der laufenden Ebene. (Verwendbar z.B. als ebenen-abhängiger Index). |
| Ressource-Verwaltung: | CALL PREQ (d) | Fordert das Peripheral d für die laufende Ebene an und belegt es, wenn frei. |
| | CALL PREL (d) | Gibt das Peripheral d frei. |

KOMMANDOS

| | |
|---|---|
| LIST | Gesamtes Programm listen |
| LIST m, | Programm ab Anweisung m listen |
| LIST m,n | Programm von Anweisung m bis n listen |
| LIST m | Anweisung m listen |
| SCRATCH | Gesamtes Programm löschen |
| DELETE m, | Programm ab Anweisung m löschen |
| DELETE m,n | Programm von Anweisung m bis n löschen |
| DELETE m | Anweisung m löschen |
| RUN | Programm starten (bei der niedrigsten Anweisungsnummer) |
| RUN m, | Programm ab Anweisung m starten |
| RUN m,n | Programm starten bei Anweisung m (stoppt bei Anweisung n) |
| RUN m | Programm starten bei Anweisung m |

Kommandos für die Erzeugung segmentierter Programme:

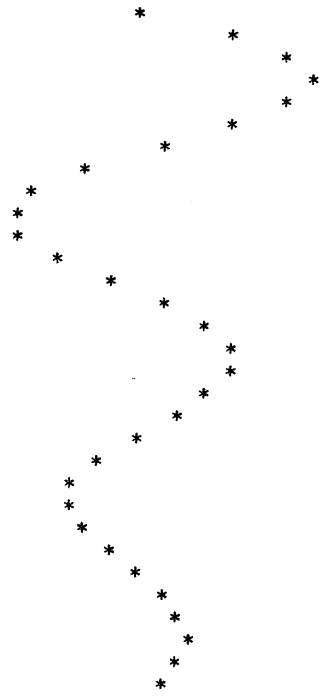
| | |
|---|---|
| LOAD Name | Root bzw. Segment laden |
| SAVE Name | " " " ablegen |
| INIT | Segmentiertes Programm verketten (initialisieren) |
| ROOT? Name<br>SEGMENT? Name 1<br>SEGMENT? Name 2<br>...<br>SEGMENT? # | Dialog |

KORREKTUREN

| | |
|---|---|
| ← | Eliminiert vorangehendes Zeichen |
| ← ← | Eliminiert 2 vorangehende Zeichen |
| ... | ... |
| RUBOUT | Eliminiert alle vorangehenden Zeichen der Zeile |

BASIC-Beispiel 1

```
LIST
100 REM    PROGRAMMBEISPIEL 2 MC6
110 DEF FNF(X)=SIN(X)*EXP(-.1*X)
115 FOR I=0 TO 15 STEP .5
120 PRINT TAB(30.5+15*FNF(I));"*"
140 NEXT I
150 END

*READY

RUN
```



```
*READY
```

BASIC-Beispiel 2

```
LIST
1 REM     PROGRAMMBEISPIEL 11 MC6
10 REM     "EWIGER KALENDER"
20 PRINT "WELCHES DATUM ";
30 INPUT K,M,C
35 LET K1=K,M1=M,C1=C
40 LET C=C/100,D=INT(.1+100*(C-INT(C))),C=INT(C)
50 LET M=M-2
60 IF M>0 THEN 100
70 LET M=M+12,D=D-1
80 IF D>0 THEN 100
90 LET D=99,C=C-1
100 LET X=INT((26*M-2)/10)+K+D+INT(C/4)+INT(D/4)-2*C
105 IF X<6 THEN 110
106 LET X=X-7
107 GOTO 105
110 PRINT "AM";K1;".";M1;".";C1;"IST";
120 GOTO X+1 OF 140,150,160,170,180,190,130
130 PRINT " SAMSTAG"
135 GOTO 20
140 PRINT " SONNTAG"
145 GOTO 20
150 PRINT " MONTAG"
155 GOTO 20
160 PRINT " DIENSTAG"
165 GOTO 20
170 PRINT " MITTWOCH"
175 GOTO 20
180 PRINT " DONNERSTAG"
185 GOTO 20
190 PRINT " FREITAG"
200 GOTO 20
210 END

*READY

RUN
WELCHES DATUM ?1,4,1973
AM1 .4 .1973 IST SONNTAG
WELCHES DATUM ?29,4,1973
AM29 .4 .1973 IST SONNTAG
WELCHES DATUM ?30,4,1973
AM30 .4 .1973 IST MONTAG
WELCHES DATUM ?1,5,1973
AM1 .5 .1973 IST DIENSTAG
WELCHES DATUM ?STOP

*READY
```

# BASEX

BASEX ist eine am Physikalischen Institut der Universität Freiburg entwickelte Programmiersprache für Echtzeit-Anwendungen. Sie enthält außer dem kompletten Sprachumfang von BASIC weitere Sprachelemente für Echtzeit-Verarbeitung im Multiprogramming sowie für Prozeß-Ein/Ausgaben.

Im folgenden ist BASEX so beschrieben, wie es in den DIETZ 621-Systemen implementiert ist. BASEX wird vom Echtzeit-Betriebssystem RTOS unterstützt, das Bestandteil jedes BASEX-Systems ist.

STATEMENTS AUS BASIC

| | |
|---|---|
| REM | Kommentar |
| LET | Wert-Zuweisung |
| INPUT | Eingabe in Periphergerät |
| PRINT | Ausgabe auf Periphergerät |
| DATA | Konstanten-Liste |
| READ | Lesen aus Konstanten-Liste |
| RES | Versetzen Zeiger Konstanten-Liste |
| GOTO | Sprung |
| GOSUB | Unterprogramm-Sprung |
| RETURN | Rücksprung aus Unterprogramm |
| IF | Bedingte Verzweigung |
| FOR | Schleifen-Beginn |
| NEXT | Schleifen-Abschluß |
| DIM | Reservierung dimensionierter Zahlen-Variablen |
| CHAR | Reservierung von String-Variablen |
| DEF | Definition Funktion |
| CALL | Aufruf Systemprozedur |
| END | Ende Programm |

Einzelheiten siehe Abschnitt BASIC.

ZUSÄTZLICHE STATEMENTS IN BASEX

ON INT

m ON INT Interrupt: Auftragsanweisung

Ordnet einem durch seine Nummer bzw. den Wert eines entsprechenden Ausdrucks angegebenen Interrupt-Eingang eine Routine zu, die aus der Auftragsanweisung besteht bzw. (bei GOTO), mit ihr beginnt und die bei Auftreten des Interrupts ausgeführt wird.
Beispiele:

```
510 ON INT 3: LET WX=INW(6)
520 ON INT N: GOTO 550
```

ENAB

m ENAB Interrupt 1, Interrupt 2, ...

Läßt die angegebenen Interrupts zu.
Beispiel:

```
655 ENAB 3,8
```

DISAB

m DISAB Interrupt 1, Interrupt 2, ...

Verhindert das Wirksamwerden der angegebenen Interrupts.
Beispiel:

```
585 DISAB 8
```

AFTER

m AFTER Zeit: Auftragsanweisung

Gibt dem System den Auftrag, nach der als Festwert oder in Form eines Ausdrucks angegebenen Zeit (in ms) eine Routine auszuführen, die aus der Auftragsanweisung besteht bzw. (bei GOTO) mit ihr beginnt.
Beispiele:

```
805 AFTER 335: LET OUTB(12)=0
825 AFTER 12000-MSEC: GOTO 120
```

WAIT

m WAIT Ausdruck

Läßt das Programm anhalten, bis der Wert des Ausdrucks ungleich Null geworden ist. Soll nur in Ebene Ø verwendet werden.
Beispiel:

```
300 WAIT INB(5)
```

EQUI

m EQUI Name = % Hexa-String F27C %

Ordnet einem Namen, der als Prozeßvariable vom Eingabe-Typ (oder als Systemprozedur in Verbindung mit PUT) im Programm auftritt, ein im Maschinencode als Hexa-String formuliertes Programm zu.
Beispiel:

```
155 EQUI INXY=%19043000300 4F27C%
156 PRINT INXY
```

EQUO

m EQUO Name = % Hexa-String F27C %

Ordnet einem Namen, der als Prozeßvariable vom Ausgabe-Typ (oder als Systemprozedur in Verbindung mit PUT) im Programm auftritt, ein in Maschinencode als Hexa-String formuliertes Programm zu.
Beispiel:

```
170 EQUO CLR?=%FC0134F27C%
171 LET CLR?=1
```

PUT

m PUT Name

Ruft eine parameterlose Systemprozedur auf, die mit EQUO definiert ist.
Beispiel:

```
615 PUT CLR?
```

START

m START Ausdruck: n

Startet in der Ebene, die dem Wert des Ausdrucks entspricht, ein Programm, das mit der Zeilen-Nummer n beginnt.
Beispiel:

```
?00 START 2:605
```

STOP

m STOP

Beendet das Programm in der jeweiligen Ebene.
Beispiel:

```
630 STOP
```

## SYSTEMVARIABLEN UND SYSTEMPROZEDUREN

Systemvariablen sind im BASEX-System vorprogrammierte Funktionen mit Variablen-Charakter, die unter ihren Namen (mit oder ohne Index) dem Benutzer-Programm zur Verfügung stehen. Sie liefern entweder einen Zahlenwert innerhalb eines Ausdrucks, oder ihnen wird ein Wert zugewiesen.

Systemprozeduren sind ebenfalls im BASEX-System enthaltene Funktionen, die unter ihren Namen mit CALL aufgerufen werden, ggfs. mit Übergabe von bis zu 4 Parametern. Sie dienen u.a. zur Initialisierung von Ein/Ausgabe-Vorgängen.

| | | |
|---|---|---|
| Zeitvariablen: | MSEC | Liefert die Absolutzeit in ms |
| | SEC | " " " " s |
| | MIN | " " " " min |
| | HOUR | " " " " h |
| Ebenen-Bindung: | LEV | Liefert die Nummer der laufenden Ebene |
| Digitale Eingänge: | INW(x) | Wortweise binäre Eingabe (16 bit) |
| | IND(x) | Wortweise BCD-Eingabe " |
| | INB(x) | Bitweise Eingabe |
| Digitale Ausgänge: | OUTW(x) | Wortweise binäre Ausgabe (16 bit) |
| | OUTD(x) | Wortweise BDE-Ausgabe " |
| | OUTB(x) | Bitweise Ausgabe |
| Zähleingänge: | INC(x) | Abfragen Zählerinhalt (16 bit) |
| | OUTC(x) | Setzen Zählerinhalt " |
| | CALL ACTC(x) | Zähleingang öffnen |
| | CALL HLTC(x) | Zähleingang schließen |
| Zeitausgänge: | OUTT(x) | Setzen Timer-Inhalt (16 bit) |
| | CALL ACTT(x) | Starten Timer |
| Analog-Eingänge: | INA(x) | Eingabe Analogwert (12 bit) |
| Analog-Ausgänge: | OUTA(x) | Ausgabe Analogwert (10 bit) |
| Analog-Meßsystem: | CALL ADCS(a,k,q) | Einkanal-Messung |
| | CALL ADCD(a,k,q) | Zweikanal-Messung |
| | CALL ADCM(a,k,q) | Mehrkanal-Messung |

Bemerkung:

x = Nummer des jeweiligen Ein/Ausgangs
a = Feldname für Puffer-Bereich
k = (Basis-) Meßkanal
q = Anzahl Messungen

PROGRAMM-SEGMENTIERUNG

(siehe Abschnitt BASIC).

WEITERE SPRACHELEMENTE VON BASEX

(siehe Abschnitt BASIC).

KOMMANDOS UND KORREKTUREN

(siehe Abschnitt BASIC).

Außerdem gibt es das Kommando TIME:

| | |
|---|---|
| TIME (cr) | Die Systemzeit in Stunden, Minuten und Sekunden wird ausgedruckt. |
| TIME h:m:s(cr) | Die Systemzeit wird auf die eingegebene Stunde, Minute und Sekunde gesetzt. |

Beispiele für Interrupt-Programmierung:

```
0600 REM AUFTRAG FUER INTERRUPT 0
0605 ON INT 0: LET OUTB(6)=0
     ...
0630 REM AUFTRAG FUER INTERRUPT 8
0645 ON INT 8: CALL ADCM(ARRY,1,10)
     ...
0650 REM INTERRUPTS 3 UND 8 ZULASSEN
0655 ENAB 3,8
     ...
0680 REM INTERRUPT 8 VERHINDERN
0685 DISAB 8
```

Der Analogeingang 1 ist wiederholt in einem zeitlichen Abstand zu messen, der dem Inhalt der Variablen DELT entspricht, und zwar so lange, bis der Bit-Eingang 14 nicht mehr erregt ist; anschließend wird der Mittelwert gebildet:

```
0835 LET N=0,W=0
0840 AFTER DELT:GOTO 850
     ...
0850 IF INB(14)=0 THEN 870
0855 LET W=W+INA(1),N=N+1
0860 AFTER DELT:GOTO 850
0865 STOP
0870 LET W=W/N
0875 STOP
```

Um 12000 ms Absolutzeit soll der Zähleingang 2 geöffnet, 1000 ms später wieder geschlossen und der Inhalt abgefragt werden. Hierbei wird in der Zeitroutine der Auftrag für eine zweite gegeben:

```
0825 AFTER 12000-MSEC:GOTO 120
     ...
0120 CALL ACTC(2)
0125 AFTER 1000: GOTO 140
0130 STOP
0140 CALL HLTC(2)
0145 LET X=INC(2)
0147 STOP
```

# C-BASIC

Mit C-BASIC (Commercial BASIC) steht für das DIETZsystem 621 eine erweiterte Version der Programmiersprache BASIC zur Verfügung, die zur Lösung kommerziell-administrativer Aufgaben geeignet ist und vor allem bei interaktiven, dialogfähigen Systemen ihre Anwendung findet.

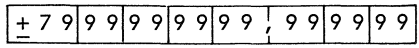
C-BASIC enthält den vollen Sprachumfang von BASIC, arbeitet im Einbenutzer-Betrieb oder im Multiprogramming unter MPOS und benutzt das Basis-Betriebssystem DBOS sowie das Dateiverwaltungssystem DFMS.

Im folgenden sind nur die Elemente von C-BASIC beschrieben, die über den Umfang von BASIC hinausgehen.

## KOMMERZIELLE ZAHL

C-BASIC sieht als dritten Datentyp - neben Strings und einfachen Zahlen - die kommerzielle Zahl (X-Zahl) vor.

Die X-Zahl kann 16-stellige, vorzeichenbehaftete Dezimalwerte darstellen mit fester Lage des Dezimalpunktes. Intern belegt sie 8 Bytes.




Der Zahlenbereich beträgt somit:

minimal -7 999 999 999.99 9999
maximal +7 999 999 999.99 9999

Die wesentlichen Operationen, die in C-BASIC mit der X-Zahl möglich sind, umfassen:

- Wertzuweisung
- Addition
- Subtraktion
- Multiplikation
- Division
- Rundung in angegebener Stelle
- Konversion X-Zahl in einfache Zahl
- " einfache Zahl in X-Zahl
- " X-Zahl in String
- " String in X-Zahl
- Vergleich

FORMAT-MASKEN

In C-BASIC wird die Ein-/Ausgabe von X-Zahlen über Format-Masken gesteuert. Die Masken geben das Format an, in dem eine X-Zahl auf einem Periphergerät (Bildschirm, Drucker usw.) präsentiert wird.

Die Format-Maske besteht aus einer Folge von Steuerzeichen, welche die Art der externen Darstellung angeben. Folgende Zeichen sind vorgesehen:

| | |
|---|---|
| 9 | Ziffernstelle |
| Z | Ziffernstelle mit Unterdrückung führender Nullen |
| * | Ziffernstelle mit Schutzstern bei führenden Nullen |
| - | Vorzeichen-Stelle (- wenn negativ, Leerstelle wenn positiv) |
| + | Vorzeichen-Stelle (- wenn negativ, + wenn positiv) |
| , | Komma (Dezimalpunkt) |
| ␣ | Leerstelle vor/hinter der Zahl zur Feldbegrenzung bzw. innerhalb der Zahl zur Tausender-Trennung |

Beispiele von Format-Masken für die Zahl 1234.567890:

```
Maske:          999999,999999
Darstellung:    001234,567890

Maske:            -9999,99
Darstellung:      ␣1234,56

Maske:          ZZZ999,99+␣
Darstellung:    ␣␣1234,56+␣

Maske:          ***999,99␣
Darstellung:    **1234,56␣

Maske:      +␣ZZZ␣ZZ9,99␣␣␣
Darstellung:␣␣␣+1␣234,56␣␣␣

Maske:        ZZZ␣ZZ9
Darstellung:  ␣␣1␣234
```

BILDSCHIRM-BEFEHLE

Zur Erleichterung der Bildschirm-Ein-/Ausgabe, die vor allem bei dialogfähigen Systemen von Bedeutung ist, enthält C-BASIC eine Reihe von Steuerbefehlen für alphanumerische Bildschirm-Terminals.

Diese Befehle umfassen u.a.:

- Positionierung des Cursors in Ausgangsstellung (erste Spalte der ersten Zeile)
- Positionierung des Cursors auf eine vorgegebene Spalte einer angegebenen Zeile
- Zeile löschen
- Zeile einfügen
- Löschen des Bildschirm-Inhaltes
- Löschen des Foreground-Inhaltes
- Umschalten auf Foreground
- Umschalten auf Background
- Auslösen der Übertragung (Bildschirm-Inhalt senden)

# FORTRAN IV

FORTRAN IV ist eine problemorientierte Programmiersprache, die u.a. für die Lösung von mathematischen Aufgaben und für technisch-wissenschaftliche Anwendungen geeignet ist.

Der Sprachumfang von FORTRAN IV ist im folgenden so beschrieben, wie er für das DIETZsystem 621 implementiert ist.

Er entspricht den einschlägigen ISO- bzw. ASA-Normen und ist mit IBM 1130/1800 FORTRAN kompatibel.

## GRUNDZÜGE

Das FORTRAN-Quellprogramm besteht aus einer Folge von Textzeilen, die

- Kommentare
- Anweisungen (Statements) mit beliebig vielen Folgezeilen
- END-Zeilen

sein können. Vor jeder Anweisung kann eine Anweisungs-Nummer n stehen. Das Quellprogramm wird von einem Übersetzer (Compiler) in ein Objektprogramm (Maschinencode) übersetzt, das vom Computer direkt ausgeführt werden kann. Der Compiler führt syntaktische und lexigraphische Analysen durch, bei denen u.a. formale Fehler erkannt und dem Benutzer gemeldet werden.

Das Quellprogramm wird in den Programmeinheiten übersetzt, die jeweils durch eine END-Zeile abgeschlossen sind. Die Objektprogramme sind verschiebbar und können durch den LINKING LOADER zu einem Gesamt-Programm verkettet oder mit anderen Benutzer- oder Bibliotheksprogrammen verbunden werden.

Ein FORTRAN-Programm kann aus folgenden Programmeinheiten bestehen:

- Hauptprogramm
- Subprogrammen (externe Funktionen, Unterprogramme)
- Datenspezifikations-Programmen

Für die Korrektur fehlerhafter FORTRAN-Quellprogramme steht ein TEXT-EDITOR zur Verfügung.

| | |
|---|---|
| Arithmetische Ergibtanweisung: | Variable = arithmetischer Ausdruck<br>Feldelement = arithmetischer Ausdruck |
| Boole'sche Ergibtanweisung: | Variable = Boole'scher Ausdruck<br>Feldelement = Boole'scher Ausdruck |
| STOP-Anweisung: | STOP<br>STOP Oktalzahl |
| PAUSE-Anweisung: | PAUSE<br>PAUSE Oktalzahl |
| Unbedingte Sprunganweisung: | GOTO |
| Aufruf-Anweisung: | CALL Subroutine-Name<br>CALL Subroutine-Name (Liste der aktuellen Parameter) |
| Rücksprung-Anweisung: | RETURN |
| Arithmetische Wennanweisung: | IF (arithmetischer Ausdruck) n1, n2, n3 |
| Boole'sche Wennanweisung: | IF (Boole'scher Ausdruck) ausführbare Anweisung |
| Berechnete Sprunganweisung: | GOTO (n1, n2, ...), Variable |
| Gesetzte Sprunganweisung: | GOTO Variable, (n1, n2, ...) |
| Laufanweisung: | DO n Laufvariable = Anfangswert, Endwert<br>DO n Laufvariable = Anfangswert, Endwert, Schrittweite |
| Leeranweisung: | CONTINUE |
| Eingabe-Anweisung: | READ (Gerät)<br>READ (Gerät) Eingabeliste<br>READ (Gerät, Format)<br>READ (Gerät, Format) Eingabeliste |
| Ausgabe-Anweisung: | WRITE (Gerät) Ausgabeliste<br>WRITE (Gerät, Format) Ausgabeliste |
| Rückspulanweisung: | REWIND Gerät |
| Rücksetzanweisung: | BACKSPACE Gerät |
| Dateiabschluß-Anweisung: | ENDFILE Gerät |
| Formatanweisung: | FORMAT Formatspezifikation |

| | |
|---|---|
| Feldanweisung: | DIMENSION Liste der Felderklärungen |
| Bereichsanweisung: | COMMON Bereichsliste<br>COMMON/Bereichsname/Bereichsliste |
| Äquivalenzanweisung: | EQUIVALENCE (Äquivalenzliste 1), (Äquivalenzliste 2), ... |
| Typenanweisungen: | INTEGER Namensliste<br>REAL Namensliste<br>DOUBLE PRECISION Namensliste<br>COMPLEX Namensliste<br>LOGICAL Namensliste |
| EXTERNAL-Anweisung: | EXTERNAL Liste der Unterprogramm-Namen |
| Anfangswert-Anweisung: | DATA Namensliste/Konstantenliste, ... |
| Formelfunktionen: | Funktionsname (Liste der formalen Parameter) = arithmetischer Ausdruck<br>Funktionsname (Liste der formalen Parameter) = Boole'scher Ausdruck |
| Funktionsanweisung: | FUNCTION Funktionsname (Liste der formalen Parameter)<br>Typangabe FUNCTION Funktionsname (Liste der formalen Parameter) |
| Unterprogramm-Anweisung: | SUBROUTINE Unterprogramm-Name<br>SUBROUTINE Unterprogramm-Name (Liste der formalen Parameter) |
| Datenspezifikations-Anweisung: | BLOCK DATA |
| Sprungzielzuweisung: | ASSIGN Anweisungs-Nummer TO einfache Variable |

| | |
|---|---|
| Z eichenvorrat: | A\|B\|C\|D\|E\|F\|G\|H\|I\|J\|K\|L\|M\|N\|O\|P\|Q\|R\|S\|T\|U\|V\|W\|X\|Y\|Z\|<br>0\|1\|2\|3\|4\|5\|6\|7\|8\|9\|<br>=\|+\|-\|*\|/\|(\|)\|,\|.\|$\|␣\| |
| Anweisungsnummer: | max. 5 Ziffern |
| O peratoren: | +\|-\|*\|/\|**<br>.LT.\|.LE.\|.EQ.\|.NE.\|.GT.\|.GE.<br>.NOT.\|.AND.\|.OR. |
| Datentypen: | Integer (ganze Zahl) 16 bit<br>Real (reelle Zahl) 24 bit Mantisse, 8 bit Exponent<br>Double (erhöht genaue Zahl) 56 bit " , 8 bit "<br>Complex (komplexe Zahl)<br>Literal<br>Boole'sche Zahl |
| Konstanten: | ganze Zahl Beispiel: 56<br>reelle Zahl " : -6.75 \| -3E+116<br>erhöht genaue Zahl " : -Ø.475669231D13<br>komplexe Zahl " : (-6E4,.9576)<br>Oktalzahl " : 765Ø7<br>Hollerith-Konstante " : 1ØHERGEBNIS=<br>Boole'sche Zahl " : .TRUE.\|.FALSE. |
| Name: | Buchstabe, gefolgt von bis zu 5 Buchstaben oder Ziffern.<br>Bezeichnet eine Variable, ein Zahlenfeld, ein Feldelement, einen COMMON-Bereich, eine Funktion oder ein Unterprogramm. |
| Dateneinheiten: | einfache Variable Name<br>Zahlen-Feld Name<br>Feldelement Name (Indexliste aus max. 3 Indizes)<br>COMMON-Bereich<br>Externe Dateneinheit (z.B. Datei, Satz) |
| A usdrücke: | arithmetische Ausdrucke<br>Boole'sche Ausdrucke |
| Formatumwandlungs-schlüssel: | Fw.d F-Format<br>Ew.d E-Format<br>Gw.d G-Format<br>Dw.d D-Format<br>Iw I-Format<br>Lw L-Format<br>Aw A-Format<br>nH Literal H-Format<br>nX Leerstellen<br>/ neue Zeile |

| Standardfunktionen: | | | Parameter: | | Ergebnis |
|---|---|---|---|---|---|
| | EXP | Exp.-Funktion | | R | R |
| | DEXP | " | | D | D |
| | CEXP | " | | C | C |
| | ALOG | Nat.Logarithmus | | R | R |
| | DLOG | " | | D | D |
| | CLOG | " | | C | C |
| | ALOG10 | 10er-Logarithmus | | R | R |
| | DLOG10 | " | | D | D |
| | SIN | Sinus | | R | R |
| | DSIN | " | | D | D |
| | CSIN | " | | C | C |
| | COS | Cosinus | | R | R |
| | DCOS | " | | D | D |
| | CCOS | " | | C | C |
| | TANH | Tangens Hyp. | | R | R |
| | SQRT | Quadratwurzel | | R | R |
| | DSQRT | " | | D | D |
| | CSQRT | " | | C | C |
| | ATAN | Arcus Tangens | | R | R |
| | DATAN | " | | D | D |
| | ATAN2 | Arcus Tangens Quotient | | R | R |
| | DATAN2 | " | | D | D |
| | ABS | Absolutwert | | R | R |
| | IABS | " | | I | I |
| | DABS | " | | D | D |
| | CABS | " | | C | R |
| | AINT | Ganzzahl-Teil | | R | R |
| | INT | " | | R | I |
| | IDINT | " | | D | I |
| | AMOD | Modulo-Fkt. | | R | R |
| | MOD | " | | I | I |
| | DMOD | " | | D | D |
| | AMAX0 | Maximalwert | | I | R |
| | AMAX1 | " | | R | R |
| | MAX0 | " | | I | I |
| | MAX1 | " | | R | I |
| | DMAX1 | " | | D | D |

| | | Parameter: | Ergebnis: |
|---|---|---|---|
| AMIN0 | Minimalwert | I | R |
| AMIN1 | " | R | R |
| MIN0 | " | I | I |
| MIN1 | " | R | I |
| DMIN1 | " | D | D |
| FLOAT | Konversion | I | R |
| IFIX | Konversion | R | I |
| SIGN | Vorzeichen | R | R |
| ISIGN | " | I | I |
| DSIGN | " | D | D |
| DIM | Min.-Diff. | R | R |
| IDIM | " | I | I |
| SNGL | Konversion | D | R |
| DBLE | Konversion | R | D |
| REAL | Realteil | C | R |
| AIMAG | Imaginärteil | C | R |
| CMRLX | Konversion | R | C |
| CONJG | kompl. Konjugation | C | C |

Erklärung: I = Integer
R = Real
D = Double
C = Complex

# UTILITIES

Für das DIETZsystem 621 stehen benutzer-orientierte Dienstprogramme (Utilities) zur Verfügung.

Sie werden bei Bedarf unter DBOS von der Platte abgerufen und führen Funktionen aus, die der Benutzer von Zeit zu Zeit braucht, z.B. um sein System zu reorganisieren, Dateien zu kopieren usw. Utilities haben im Betriebssystem den Rang von Benutzerprogrammen.

## PACK

Das Dienstprogramm PACK dient zur Reorganisation eines Plattenspeichers, dessen Nummer u anzugeben ist.

PACK verdichtet den Plattenspeicher-Inhalt, indem es alle Dateien des Speichers lückenlos aneinanderreiht.

Der Raum, den mit KILL gelöschte Dateien physisch einnehmen, wird dabei vom Inhalt der dahinterstehenden Dateien aufgefüllt.

Auf diese Weise entsteht ein zusammenhängender freier Raum auf dem Plattenspeicher.

Aufruf: RUN, PACK
Dialog: UNIT? u

## COPY

Mit dem Dienstprogramm COPY können Datei-Inhalte von einer Datei in eine andere übertragen werden bzw. von anderen Geräten eingelesen oder auf ihnen ausgegeben werden.

Quell-Datei (source) und Ziel-Datei (destination) sind dabei anzugeben.

Die File-Namen und die Unit-Nummern beim Plattenspeicher sind anzugeben; die Ziel-Datei ist ggfs. vorher zu eröffnen.

Aufruf: RUN, COPY
Dialog:
FROM DEV-TYP:DSK
SOURCE UNIT? u1
SOURCE FILE? f1
TO DEV-TYP:DSK
DESTINATION UNIT? u2
DESTINATION FILE? f2

# MINCTEST 600

MINCTEST 600 ist ein Diagnostik-Programm zur Überprüfung der Funktionen der Zentraleinheit DIETZ 621.

Es besteht aus zwei unabhängigen Teilen:

- Einzelbefehlstest
- Endprüfung.

EINZELBEFEHLSTEST

Dieser Programmteil überprüft die einzelnen Befehlsabläufe der Zentraleinheit. Es besteht aus insgesamt 81 Einzeltests, die systematisch aufeinander aufbauen und mit denen jeder einzelne Befehl, jede Adressierungsart usw. getestet werden.

Das Programm arbeitet im Dialog mit dem Benutzer, der die gewünschten Tests einleitet und dann der Erfolg bzw. Mißerfolg des Tests in Klartext gemeldet werden.

Voraussetzung für den Test ist ein funktionstüchtiger Hardware-Bootstrap in der Zentraleinheit, ein Konsolgerät (mit Streifenleser oder getrenntem schnellem Lochstreifen-Leser).

ENDPRÜFUNG

Dies ist ein zusammenhängendes Programm, das nacheinander alle wichtigen Teile der Zentraleinheit, wie Kernspeicher, RAM, Ebenen-Logik, BUS-Schnittstelle usw. überprüft und den erfolgten Test meldet.

Danach gibt das Programm automatisch die Rechner-Konfiguration aus (z.B. Speichergrößen, Ebenen-Zahl und -Belegung, usw.).

Schließlich geht das Programm in einen Dauertest unter worst-case-Bedingungen über.

# DIETZsystem 621

VORBEMERKUNG

Mit dem DIETZsystem 621 steht ein standardisiertes Computer-System für universellen Einsatz auf der Basis des DIETZ 621 zur V rfügung. Es bildet ein abgeschlossenes Hardware-/Software-System mit Konfigurations-Möglichkeiten für nahezu jeden Anwendungsfall.

SYSTEM-EIGENSCHAFTEN

Zwei Versionen des DIETZsystems 621 sind verfügbar; sie unterscheiden sich durch Art und Umfang des Systemspeichers und umfassen in der Grundausführung:

- DIETZsystem 621 C:
  - Zentraleinheit mit 1 Kbyte RAM + 32 Kbyte Kernspeicher, Netzausfallschutz, Echtzeituhr, Bedienungskonsole
  - DIETZdisk 256 Kbyte (System)
  - DIETZdisk 256 Kbyte (Benutzer)
  - Konsoldrucker 50 Z/s 80 Z/Zl mit Tastatur
  - Systemschrank

- DIETZsystem 621 D:
  - Zentraleinheit mit 1 Kbyte RAM + 32 Kbyte Kernspeicher, Netzausfallschutz, Echtzeituhr, Bedienungskonsole
  - Wechselplattenspeicher 2.4 Mbyte (System)
  - DIETZdisk 256 Kbyte (Benutzer)
  - Konsoldrucker 50 Z/s 80 Z/Zl mit Tastatur
  - Systemschrank

Die Grundsysteme können modular um Periphergeräte, Prozeßanschlüsse und Datenfernübertragungs-Anschlüsse erweitert werden.

Die zum DIETZsystem gehörende Basis-Software erlaubt das Programmieren in folgenden Sprachen:

- Assembler MINCASS 600
- MARS 600 für Echtzeit-Anwendungen
- BASIC für technisch-wissenschaftliche Anwendungen
- BASEX für Echtzeit-Anwendungen
- C-BASIC für kommerzielle Anwendungen
- FORTRAN IV für technisch-wissenschaftliche Anwendungen.

Beide Versionen (621 C und D) werden vom Basis-Betriebssystem DBOS unterstützt. In Multiprogramming-Anwendungen wird das Betriebssystem MPOS verwendet. Es erlaubt den gleichzeitigen Betrieb von 6 oder 12 unabhängigen Benutzerprogrammen, die in den hierarchisch gegliederten Hardware-Programmebenen des Computers laufen.

Im Falle von Echtzeit-Anwendungen, z.B. beim Einsatz als Prozeßrechner, wird das Echtzeit-Betriebssystem RTOS verwendet, das die Zeitverwaltung, Interrupt-Verwaltung und Prozeß-Ein/Ausgabe steuert und zusätzlich das MPOS beinhaltet.

Ein weiterer Bestandteil der Software ist das Dateiverwaltungs-System DFMS, mit dem die Programm- und Datenbestände sowohl der System- als auch der Benutzer-Platte verwaltet werden.

Dadurch, daß beide Systemversionen plattenorientiert sind, bieten sie dem Benutzer ein hohes Maß an Bedienungs- und Programmierungskomfort. Dieser wird noch erhöht durch die Möglichkeit, das Benutzer-Plattenlaufwerk zur Ein- und Ausgabe von Programmen und Daten auf DIETZdisk-Kassetten zu verwenden, mit den diesem Medium eigenen Vorzügen hinsichtlich einfacher Handhabung, robuster Ausführung, großer Zuverlässigkeit und schnellem Zugriff.

Typische Einsatzgebiete des DIETZsystems 621 sind u.a.:

- Prozeßrechner-Aufgaben in Industrie, Medizin, Forschung und Lehre
- Rechensystem für technisch-wissenschaftliche Anwendungen
- Interaktives Computersystem für kommerziell-administrativen Einsatz
- Intelligentes Terminal, Front-End-Prozessor und Datenkonzentrator in Datenfernverarbeitungs-Systemen.

## DIE GRUNDAUSFÜHRUNG

Die Grundausführung besteht körperlich aus einem 19"-Systemschrank (einfache Breite), in dem sich Zentraleinheit und Plattensystem befinden und in dem Einbauraum für weitere Einheiten vorhanden ist. Getrennt davon steht der Konsoldrucker, über den der System-Dialog geführt wird; er steht aber auch für die Verwendung im Benutzer-Programm zur Verfügung.

Der System-Plattenspeicher enthält

- das Betriebssystem DBOS
- weitere Betriebssysteme, soweit implementiert
- Dateien mit systemgebundenen Programmen
- Dateien des Benutzers.

Zwischen 621 C und D besteht hier kein Unterschied, abgesehen von Kapazität, technischer Ausführung und Zugriffsgeschwindigkeit. Beim 621 C wird das DBOS durch einen Hardware-Bootstrap von der Platte geladen, während dies beim 621 D nach Einschalten des Netzes automatisch geschieht.

Die Benutzer-Platte enthält Dateien gleicher Struktur wie der Systemspeicher. Sie dient zum Einlesen neuer Programme und zur Archivierung von Benutzerprogrammen (d.h. als Ersatz für die früher übliche Lochstreifen-Peripherie). Außerdem kann der Benutzer auf dieser Platte Daten abspeichern oder von ihr lesen.

In der Grundausführung sind 8 Hardware-Ebenen mit je 128 Registern vorgesehen; unter MPOS bzw. RTOS sind 6 davon als Benutzer-Ebenen (Ø...5) verwendbar; die restlichen zwei sind System-Ebenen.

## HARDWARE-ERWEITERUNGEN

Die Grundausführung des DIETZsystems 621 C und D ist systematisch erweiterbar, so daß für jeden Bedarfsfall das passende Gesamtsystem konfiguriert werden kann.

Folgende Erweiterungen sind möglich:

| | |
|---|---|
| - Halbleiter-Speicher (RAM): | - Erweiterung auf 2 Kbyte = 16 Ebenen (12 Benutzer-, 4 System-Ebenen) |
| - Kernspeicher: | - Erweiterung auf 48 Kbyte (in Speichererweiterungs-Einschub SPE-621) |
| - Plattenspeicher (nur 621 D): | - Erweiterung auf 4.8, 7.2 oder 9.6 Mbyte (erfordert zweiten Systemschrank) |
| - Prozessoren: | - Festkomma-Rechenwerk (MP/DV)<br>- Gleitkomma-Prozessor (in Speichererweiterungs-Einschub SPE-621) |
| - Periphergeräte: | - Bildschirm-Terminal BTH 1000<br>- Bildschirm-Terminal BTH 2000<br>- weitere Konsoldrucker PH 50 mit/ohne Tastatur<br>- 8-Kanal-Fernschreiber ASR 33/V24 und /LS<br>- Schnelldrucker TAL 2200<br>- Lochkarten-Stapelleser MDS 6042<br>- 8-Kanal-Streifenleser LE 125<br>- 8-Kanal-Streifenlocher LO 75<br>- Alphanum./graph. Bildschirmgerät TEK 4010<br>- Speicheroszillograph TEK 611<br>- Digitalplotter DP-10, -1, -3<br>- Magnetband-Laufwerke (1. ...4 anschließbar, erforderlich Controller, in eigenem Systemschrank) |

- Prozeßanschlüsse:
  - Interrupt-Eingänge
  - Statische digitale Eingänge
  - Speichernde digitale Ausgänge
  - Zähleingänge
  - Zeitausgänge
  - Watchdog-Ausgang
  - Einkanal-Analogeingänge
  - Einkanal-Analogausgänge
  - Mittelschnelles Analog-Meßsystem
  - Schnelles Analog-Meßsystem
  - Integrierendes Meßsystem

  } in zweitem Systemschrank

- DFÜ-Anschlüsse:
  - Datenfernübertragungs-Interface (V24/asynchron und synchron)

Im Systemschrank der Grundausführung ist Einbauraum für die Speicher-Erweiterung (SPE-621) enthalten; ferner für eine Universal-Interface-Einheit (UIE-621), die 12 Plätze für Einkarten-Interfaces bereithält.

Bei Bedarf müssen weitere UIEs für Einkarten-Interfaces vorgesehen werden, die in einem zweiten Systemschrank untergebracht werden.

## PROGRAMMIERSPRACHEN

Für das DIETZsystem 621 C und D stehen 6 Programmiersprachen zur Auswahl, die je nach Bedarf zur Anwendung kommen. Dabei ist zu beachten, daß

- das System entweder unter DBOS im System-Bedienungsbetrieb oder unter Kontrolle eines Benutzerprogramms läuft (wobei BASIC, BASEX und C-BASIC einen zusätzlichen "Command-Mode" kennen);

- das Benutzerprogramm einschließlich aller (ggfs. im Multiprogramming parallel laufenden) Teile einen einheitlichen Programm-Kontext (Job) bildet;

- jedes Benutzerprogramm aus einer der verfügbaren Sprachen hervorgehen muß (was den Einbau von Maschinencode-Prozeduren in BASIC, BASEX, C-BASIC und FORTRAN nicht ausschließt; im übrigen umfaßt MARS 600 alle Assembler-Befehle).

Bei der Wahl der geeigneten Programmiersprachen ist im übrigen auf folgende Umstände zu achten:

- Assembler: Alle System-Ressourcen sind ansprechbar, jedoch unter Beachtung entsprechender Vorsichtsmaßnahmen hinsichtlich Konfliktfällen und, z.B. bei der Prozeßperipherie, nur mit Kenntnis der absoluten Adressen sowie in Einzelbefehls-Schritten. Multiprogramming ist möglich nach vom Benutzer festgelegten Regeln.

- MARS 600: Alle System-Ressourcen sind verfügbar, größtenteils über Makros. Multiprogramming ist so vorgesehen, wie durch RTOS/MPOS geregelt. Rückgriff auf RTOS-Funktionen ist möglich.

- BASIC und C-BASIC: Alle System-Ressourcen sind verfügbar, jedoch nicht die Prozeßperipherie (kein Rückgriff auf RTOS-Funktionen).

  Unterschieden wird:

  - Single-User-Betrieb (kein Multiprogramming)

  - Multiprogramming-Betrieb unter MPOS. BASIC-Programme auf allen Benutzer-Ebenen möglich (ausgelöst vom Hauptprogramm in Ebene Ø). Segmentierte Programme laufen nur auf einer Ebene.

- BASEX: Alle System-Ressourcen sind verfügbar. Multiprogramming ist so vorgesehen, wie durch RTOS/MPOS geregelt. Rückgriff auf RTOS-Funktionen möglich.

- FORTRAN IV: Alle System-Ressourcen sind verfügbar, jedoch nicht die Prozeßperipherie (außer durch vom Benutzer eingebaute Maschinencode-Programme).

  Single-User-Betrieb (kein Multiprogramming).

BETRIEBSSYSTEME

Das plattenorientierte Basis-Betriebssystem DBOS des DIETZsystems 621 unterstützt den Benutzer-/System-Dialog und erlaubt in einer einfachen Form die Verwaltung der Plattendateien und den Zugriff zu ihnen.

Darüberhinaus sind folgende Betriebssysteme je nach Anwendungsfall und verwendeter Programmiersprache zusätzlich implementierbar:

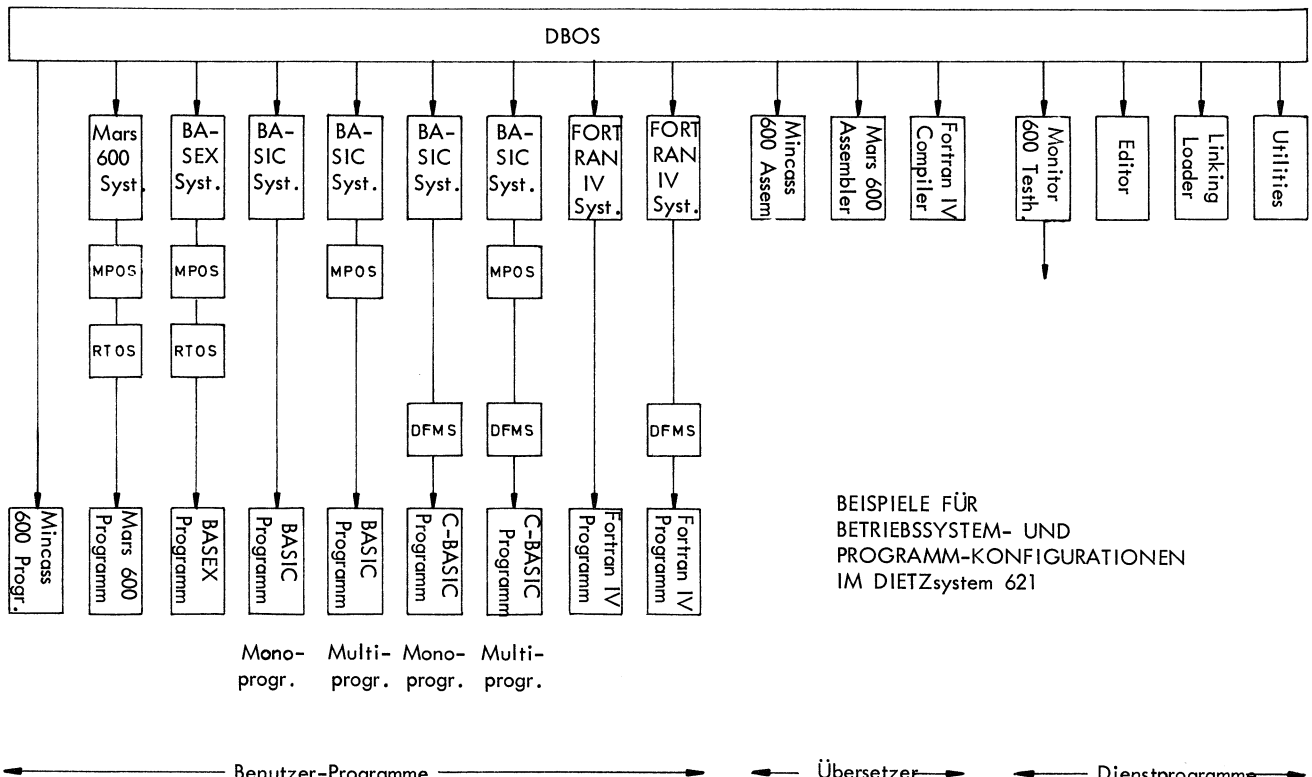
- DFMS: Komfortables Datei-Verwaltungs- und Zugriffs-System (Ein- und Mehrsatz-Dateien, Index-Dateien; sequentieller und Random-Zugriff; Datei-Schutzfunktionen).

- MPOS: Multiprogramming-Betriebssystem (Programm-Auftragsv erwaltung, Ressource-Verwaltung)

- RTOS: Echtzeit-Betriebssystem (Führung der Systemzeit; Zeitauftrags- und Interrupt-Verwaltung; Behandlung der Prozeßperipherie).

Die folgende Tabelle enthält die möglichen Betriebssystem-Konfigurationen (x = Standard; o = Option).

| PROGRAMMIER-SPRACHE | | BETRIEBSSYSTEM | | | |
|---|---|---|---|---|---|
| | | DBOS | DFMS | MPOS | RTOS |
| ASSEMBLER | | x | o | | |
| MARS 600 | | x | o | x | x |
| BASIC | Single-User | x | o | | |
| | Multipro-gramming | x | o | x | |
| BASEX | | x | o | x | x |
| C-BASIC | Single User | x | x | | |
| | Multipro-gramming | x | x | x | |
| FORTRAN IV | | x | o | | |



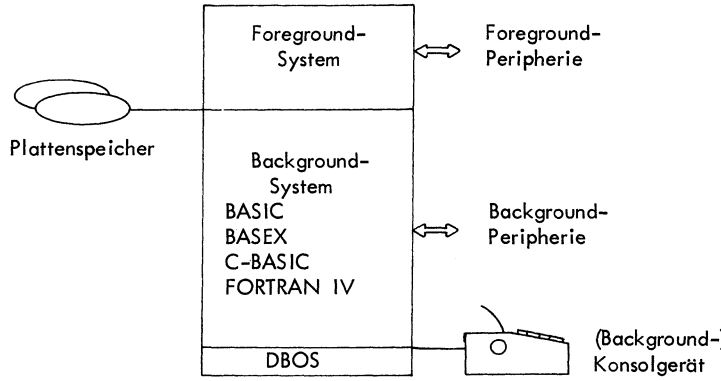

BEISPIELE FÜR
BETRIEBSSYSTEM- UND
PROGRAMM-KONFIGURATIONEN
IM DIETZsystem 621

FOREGROUND-BACKGROUND-BETRIEB

Mit den DIETZsystemen 621 C und D ist eine bestimmte Form des Foreground-/Background-Betriebs zulässig und leicht implementierbar.

Unter Verwendung einer entsprechenden Hardware-Konfiguration, welche ausreichende Speicherkapazität und Ebenen-Zahl gewährleistet, ist es möglich,

- im "Hintergrund" BASIC-, BASEX-, C-BASIC- oder FORTRAN IV-Programme in beliebiger Folge zu erstellen, zu compilieren, zu edieren und zur Ausführung zu bringen, während
- im "Vordergrund" ein in Assembler geschriebenes, ausgetestetes Benutzer-Programm oder ein DIETZ-Anwendersystem (z.B. eine Terminal-Emulation) läuft, wobei hierfür eine oder mehrere getrennte Ebenen mit hoher Priorität reserviert sind.




Foreground- und Background-Bystem benutzen getrennte Peripherie-Einheiten; die Plattenspeicher sind jedoch Ressourcen, die beiden Systemen gemeinsam sind. Das Konsolgerät ist dem Background-System zugeordnet.

| V-Bez | T-Bez | Baugruppe | Einbau Ort | Einbau Größe | Soft-ware | Bem. |
|---|---|---|---|---|---|---|
| S621C | | DIETZsystem 621C/Grundausführung mit:<br>- Zentraleinheit 32KB + 1 KB RAM (8 Ebenen)<br>- DIETZdisk-Doppellaufwerk 2 x 256 KB<br>- Mosaikdrucker-Terminal 50 Z/s 80 Z/ZI mit Tastatur<br>- 19"-Systemschrank | | S1 | S | |
| S621D | | DIETZsystem 621D/Grundausführung mit:<br>- Zentraleinheit 32 KB + 1 KB RAM (8 Ebenen)<br>- DIETZdisk-Einfachlaufwerk 256 KB<br>- Wechselplattenspeicher 2.4 MB mit Controller<br>- Mosaikdrucker-Terminal 50 Z/s 80 Z/ZI mit Tastatur<br>- 19"-Systemschrank | | S1 | S | |
| S621C1 | | DIETZsystem 621C wie oben, jedoch:<br>- zus.Bildschirm-Terminal BTH 1000 als Konsole<br>- Mosaikdrucker ohne Tastatur | | S1 | S | 1) |
| S621D1 | | DIETZsystem 621D wie oben, jedoch:<br>- zus.Bildschirm-Terminal BTH 1000 als Konsole<br>- Mosaikdrucker ohne Tastatur | | S1 | S | 1) |
| S621C2 | | DIETZsystem 621C wie oben, jedoch:<br>- zus.Bildschirm-Terminal BTH 2000 als Konsole<br>- Mosaikdrucker ohne Tastatur | | S1 | S | 1) |
| S621D2 | | DIETZsystem 621D wie oben, jedoch:<br>- zus.Bildschirm-Terminal BTH 2000 als Konsole<br>- Mosaikdrucker ohne Tastatur | | S1 | S | 1) |
| RL-E16 | | Erweiterung auf 2KB RAM/16 Ebenen | MC | | | |
| SPE-621 | | Speichereinheit | G1 | E5 | | 2) |
| KS-E48 | | Kernspeicher-Erweiterung auf 48KB | SPE | M1 | S | |
| GKE-621 | | Gleitkomma-Prozessor | SPE | M1 | S | |
| FKP-621 | | Festkomma-Prozessor | UIE | B2 | S | |
| EWP4.8 | | Plattenspeicher-Erweiterung auf 4.8 MB | SYS | E6 | S | 3) |
| EWP7.2 | | " " " 7.2 MB | SYS | E10 | S | |
| EWP9.6 | | " " " 9.6 MB | SYS | E14 | S | |
| UIE-621/S | | Universal-Interface-Einheit mit Stromversorgung, Device-Selector, BUS-Anschluß und Anschlußsteckern; für 12 Einkarten-Interfaces | SG oder SYS | E6 | | 4) |
| SYS-E A | | 19"-Systemschrank (Erweiterung) | | S1 | | 5) |

Bemerkungen:

1) Bildschirm statt Mosaikdrucker-Terminal als Dialog-Konsole, + Mosaikdrucker als Konsol-Hardcopy
2) für KS-E48 und GKE-621
3) nur bei 621D
4) 1 x im Grund-Systemschrank, Rest in Erweiterungs-Schränken. Enthält 12 Plätze "B"
5) ein- oder mehrfach nötig

| V-Bez | T-Bez | Baugruppe | Einbau Ort | Einbau Größe | Soft ware | Bem |
|---|---|---|---|---|---|---|
| PH50 | | Mosaikdrucker-Terminal 50 Z/s 80 Z/Zl mit Tastatur | | | S | |
| IV24/PH50 | 602 000 | + Interface | UIE | B1 | | |
| PH50 RO | | Mosaikdrucker 50 Z/s 80 Z/Zl ohne Tastatur | | | S | |
| IV24/PH50 | 602 000 | + Interface | UIE | B1 | | |
| BTH2000 | | Bildschirm-Terminal 27 Zl 74 Z/Zl F/B | | | S | |
| IV24/DIS | 602 000 | + Interface | UIE | B1 | | |
| BTH1000 | | Bildschirm-Terminal 12 Zl 80 Z/Zl | | | S | |
| IV24/DIS | 602 000 | + Interface | UIE | B1 | | |
| LE 125 | | 8-Kanal-Streifenleser 125 Z/s mit Spuler | | E5 | T | |
| ILE/LO | 602 005 | + Interface | UIE | B1 | | |
| LO 75 | | 8-Kanal-Streifenlocher 75 Z/s | | E6 | T | |
| ILE/LO | 602 005 | + Interface | UIE | B1 | | |
| MDS6042 | | Lochkarten-Stapelleser 400 K/min 80 Sp/K | | | T | |
| IKLE | 601 045 | + Interface | UIE | B1 | | |
| TAL2200 | | Mosaik-Schnelldrucker 200 Zl/min 132 Z/Zl | | | T | |
| ITAL2200 | 602 017 | + Interface | UIE | B1 | | |
| TEK4010 | | Alphanum./graphisches Bildschirm-Terminal | | | T | |
| IV24 | 602 000 | + Interface | UIE | B1 | | |
| TEK611 | | Graphisches Speicher-Display | | | T | |
| IAXY | 601 043 | + Interface | UIE | B1 | | |
| XYS-A4 | | XY-Schreiber DIN A4 | | | T | |
| IAXY | 601 043 | + Interface | UIE | B1 | | |
| XYS-A3 | | XY-Schreiber DIN A3 | | | T | |
| IAXY | 601 043 | + Interface | UIE | B1 | | |
| DP-10 | | Inkremental-Plotter DIN A4 | | | T | |
| IPLOT | 601 044 | + Interface | UIE | B1 | | |
| DP-1 | | Inkremental-Plotter DIN A3 | | | T | |
| IPLOT | 601 044 | + Interface | UIE | B1 | | |
| DP-2 | | Inkremental-Plotter DIN A2 | | | T | |
| IPLOT | 601 044 | + Interface | UIE | B1 | | |

KONFIGURATIONSLISTE MAGNETBAND-SYSTEME

| V-Bez | T-Bez | Baugruppe | Einbau Ort | Einbau Größe | Soft-ware | Bem |
|---|---|---|---|---|---|---|
| MBE7840-9 | | 9-Spur-Laufwerk 800 cpi 7" Spulen | SYS | E5 | T | 1) |
| MBE-621/800P | | + Controller | SYS | E3 | | |
| MBE-621/800 | | 9-Spur-Laufwerk 800 cpi 10.5" Spulen (Master) | SYS | | T | 1) |
| | | + Controller | SYS | E3 | | |
| | | 9-Spur-Laufwerk 800 cpi 10.5" Spulen (Slave) | SYS | | T | 1)2) |
| MBE-621/1600 | | 9-Spur-Laufwerk 1600 cpi 10.5" Spulen (Master) | SYS | | T | 1) |
| | | + Controller | SYS | E3 | | |
| | | 9-Spur-Laufwerk 1600 cpi 10.5" Spulen (Slave) | SYS | | T | 1)2) |
| | | | | | | |

Bemerkungen: 1) Einbau in eigenem Systemschrank empfohlen
2) bis zu 3 Slaves

KONFIGURATIONSLISTE DFÜ-SCHNITTSTELLEN

| V-Bez | T-Bez | Baugruppe | Einbau Ort | Einbau Größe | Soft-ware | Bem |
|---|---|---|---|---|---|---|
| IV24/DAS | 602 003 | V24-DFÜ-Interface asynchron | UIE | B1 | | 1) |
| IV24/DSM | 602 004 | " " " bi tsynchron/Modemtakt | UIE | B1 | | 1) |
| IV24/DSE | 602 005 | " " " " /Eigentakt | UIE | B2 | | 1) |
| | | | | | | |

Bemerkung: 1) Software (DFÜ-Prozeduren) gegen Mehrpreis

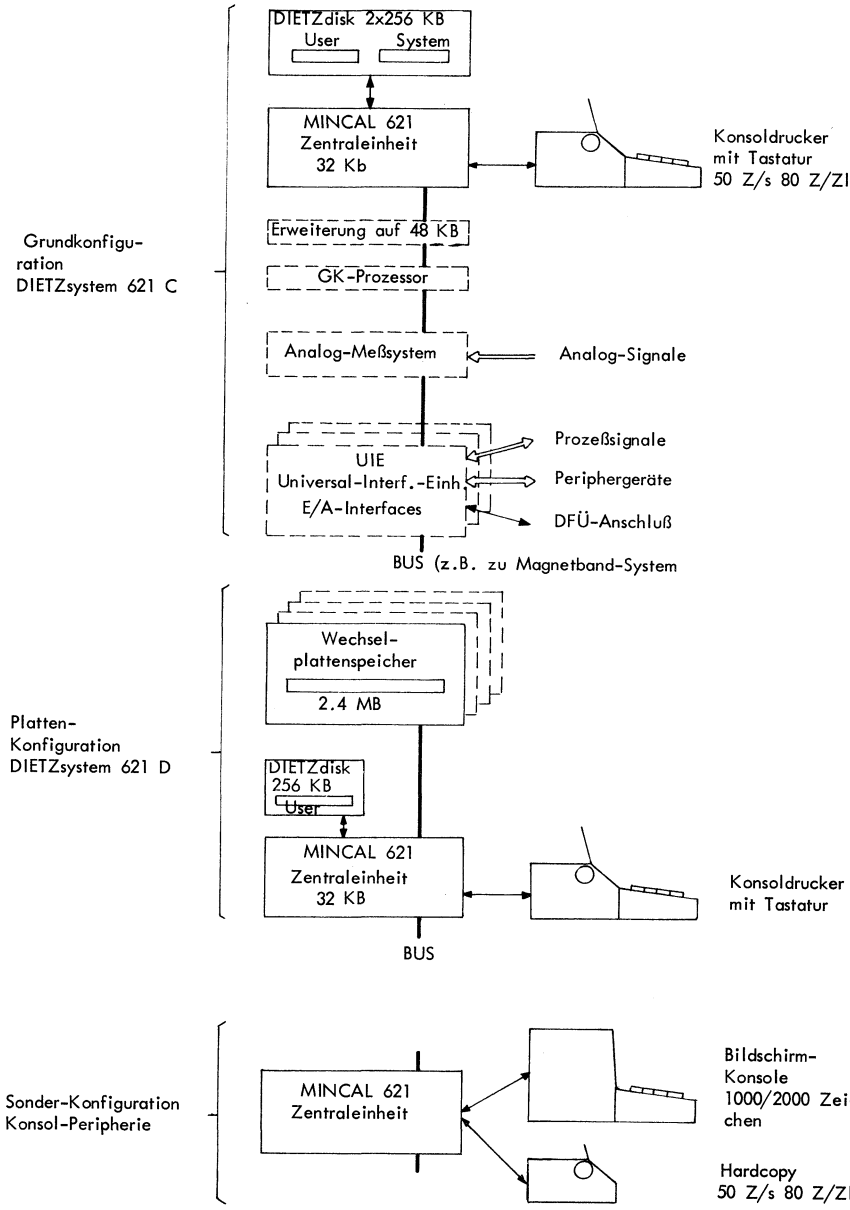
ERLÄUTERUNGEN

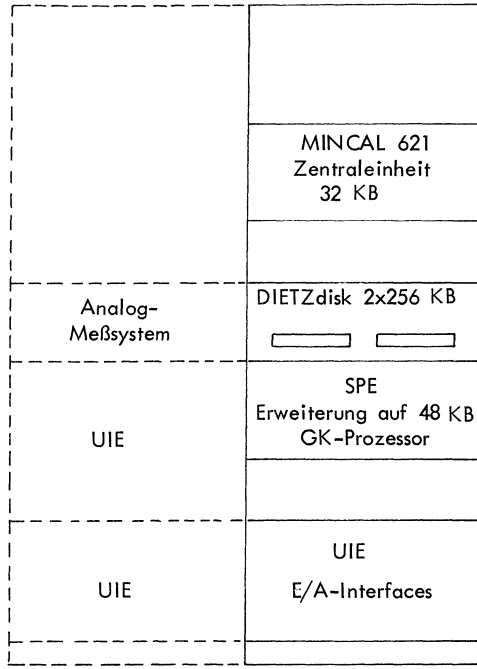
| | | |
|---|---|---|
| Spalte Einbauart: | MC | Zentraleinheit MC-621 |
| | SPE | Speichereinheit SPE-621 |
| | UIE | Universal-Interface-Einheit UIE 621 |
| | ADM | Analog-Meßsystem ADA-621 |
| | MUI | Meßstellen-Umschalter MUI-3 |
| | G | Grund-Schrank 621 C/D |
| | SYS | Erweiterungsschrank SYS-E |
| Spalte Einbaugröße: | S1 | 19"-Systemschrank (einfache Breite) |
| | E... | Einschub (... Einheiten hoch) |
| | M1 | Speicherkarte (1 Platz) |
| | B... | Baustein in UIE (... Plätze breit) |
| Software: | S | Standard-Software |
| | T | von Peripheral-Treiber unterstützt |
| | R | von RTOS unterstützt |

| V-Bez | T-Bez | Baugruppe | Einbau Ort | Größe | Soft-ware | Bem |
|---|---|---|---|---|---|---|
| PDSE8/5 | 601 020 | 8-bit-Interrupt-Eingang TTL | UIE | B1 | R | |
| PDSE8/12.60 | 601 026 | " " " HTL | UIE | B1 | R | |
| PDSE8/FK | 601 027 | " " " Fotokoppler | UIE | B1 | R | |
| PDSE8/R | 601 024 | " " " Relais | UIE | B1 | R | |
| PDSE8/FL | 601 080 | " " " Fotokoppler 2-pol/bit | UIE | B1 | R | 1) |
| PDSE16/5 | 601 088 | 16-bit-Interrupt-Eingang TTL | UIE | B1 | R | 2) |
| PSSE16/5 | 601 010 | 16-bit-Digitaleingang TTL | UIE | B1 | R | |
| PSSE16/12.60 | 601 016 | " " HTL | UIE | B1 | R | |
| PSSE16/FK | 601 017 | " " Fotokoppler | UIE | B1 | R | |
| PSSE16/R | 601 014 | " " Relais | UIE | B1 | R | |
| PSSE16/FL | 601 082 | " " Fotokoppler 2-pol/bit | UIE | B1 | R | 1) |
| PSSE32/5 | 601 085 | 32-bit-Digitaleingang TTL | UIE | B1 | R | 3) |
| PSSA16 | 601 030 | 16-bit-Digitalausgang TTL | UIE | B1 | R | 4) |
| PSSA16/FK | 601 033 | " " Fotokoppler | UIE | B1 | R | 4) |
| PSSA16/R | 601 036 | " " Relais | UIE | B1 | R | |
| PSSA32 | 601 086 | 32-bit-Digitalausgang TTL | UIE | B1 | R | 3) |
| PIZE16 | 601 060 | 16-bit-Zähleingang | UIE | B1 | R | |
| PISA16 | 601 061 | 16-bit-Zeit-Steuerausgang 1 MHz TTL | UIE | B1 | R | 5) |
| PWDOG | 601 070 | Watchdog-Ausgang | UIE | B1 | R | |
| ADE12.010 | 601 000 | Einkanal-Analogeingang 12 bit 0...10 V | UIE | B2 | R | |
| ADE12.505 | 601 001 | " " " -5...5 V | UIE | B2 | R | |
| ADE12.1010 | 601 002 | " " " -10...10 V | UIE | B2 | R | |
| MUE16R | 601 087 | 16fach-Relais-Multiplexer 3-polig | UIE | B1 | R | |
| DAU1010 | 601 041 | Analog-Ausgang 10 bit 0...10 V | UIE | B1 | R | |
| DAI1020 | 601 042 | " " " 0...20 mA | UIE | B1 | R | |
| ADM621S | | Mittelschnelles Analog-Meßsystem 12 bit | SYS | E3 | R | |
| MUM/E16 | | 16-Kanal-Multiplexer (MOS) | ADM | | R | 6) |
| ADM-S&H | | Sample-&Hold-Zusatz | ADM | | R | |
| ADI200 | | Integrierendes Meßsystem 19.999 DC | SYS | E2 | | |
| ADI210 | | " " 119.999 DC | SYS | E2 | | |
| ADA203 | | " " 19.999 DC/AC/R | SYS | E2 | | |
| ADA213 | | " " 119.999 DC/AC/R | SYS | E2 | | |
| PADI | 601 009 | + Interface zu ADI/ADA | UIE | B1 | | |
| MUI-3 | | Meßstellenumschalter | SYS | E4 | | |
| PIMUI | 601 047 | + Interface zu MUI | UIE | B1 | | |
| MUI-3/E10 | | 10-Kanal-Multiplexer (Relais) | MUI | | | 7) |

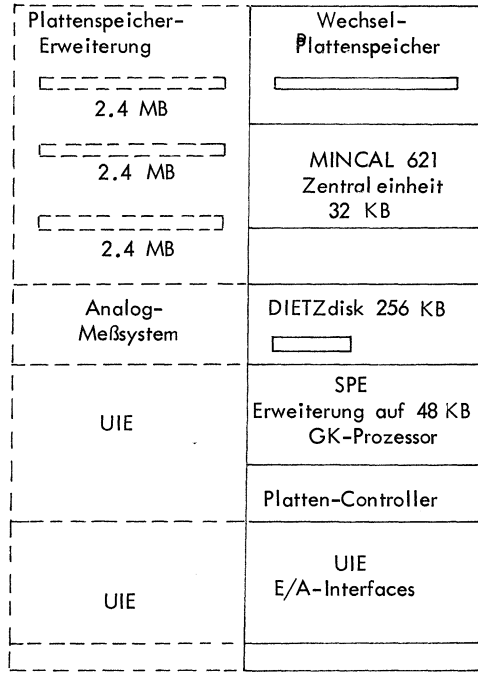
Bemerkungen:

1) 2-poliger, passiver Eingang/bit (5 V) "1" wenn 0V
2) unter RTOS nicht gemischt mit 8-bit-Interrupt-Eingängen
3) unter RTOS nicht gemischt mit 16-bit-Ein/Ausgängen
4) Open-Collector-Ausführung. Andere Version auf Anfrage
5) andere Versionen (10 MHz; Fotokoppler) auf Anfrage
6) max. 4 je ADM
7) max. 10 je MUI


Grundkonfigu-
ration
DIETZsystem 621 C
DIETZdisk 2x256 KB
User
System
MINCAL 621
Zentraleinheit
32 Kb
Konsoldrucker
mit Tastatur
50 Z/s 80 Z/Zl
Erweiterung auf 48 KB
GK-Prozessor
Analog-Meßsystem
Analog-Signale
UIE
Universal-Interf.-Einh.
E/A-Interfaces
Prozeßsignale
Periphergeräte
DFÜ-Anschluß
BUS (z.B. zu Magnetband-System
Platten-
Konfiguration
DIETZsystem 621 D
Wechsel-
plattenspeicher
2.4 MB
DIETZdisk
256 KB
User
MINCAL 621
Zentraleinheit
32 KB
Konsoldrucker
mit Tastatur
BUS
Sonder-Konfiguration
Konsol-Peripherie
MINCAL 621
Zentraleinheit
Bildschirm-
Konsole
1000/2000 Zei-
chen
Hardcopy
50 Z/s 80 Z/Zl




Aufbau
DIETZsystem
621 C




Aufbau
DIETZsystem
621 D

# Prozeßterminal-System 6150

Häufig besteht die Aufgabe, mehrere voneinander unabhängige Prozeßrechner zu installieren, die eine gewisse räumliche Nachbarschaft haben. Ein Beispiel ist die Automatisierung von Geräten, Labors und Versuchsanordnungen im Fachbereich einer Hochschule. Jedes System soll Zugriff zu Programmen und Dateien in einem Großraumspeicher haben und gelegentlich leistungsfähige Periphergeräte benutzen. Hier liegt der Gedanke nahe, die aufwendige Speicher- und Geräteperipherie an einer zentralen Stelle verfügbar zu halten.

Das Prozeßterminal-System 6150 ist die Konsequenz aus diesen Überlegungen. Es umfaßt

- bis zu 16 Prozeßterminals 6155 mit DIETZ 621
- ein Poolsystem 1621, das sternförmig mit den Terminals verbunden ist.

Die Prozeßterminals verhalten sich wie Stand-alone-Einheiten des DIETZsystems 621 der Benutzer arbeitet im Dialog über Konsol-Fernschreiber an den Terminals, und diese führen alle Prozeß- und Verarbeitungsprogramme aus.

Einige Aufgaben jedoch werden vom Poolsystem übernommen:

- Es enthält einen Plattenspeicher (Kapazität 2.4...9.6 Mbyte), der die Programme der verschiedenen Benutzer enthält und in dem Dateien aufgebaut sind, die von den Benutzern wie üblich definiert und angesprochen werden.

- Es übernimmt die Ausführung größerer Programme, aktiviert durch die Terminals

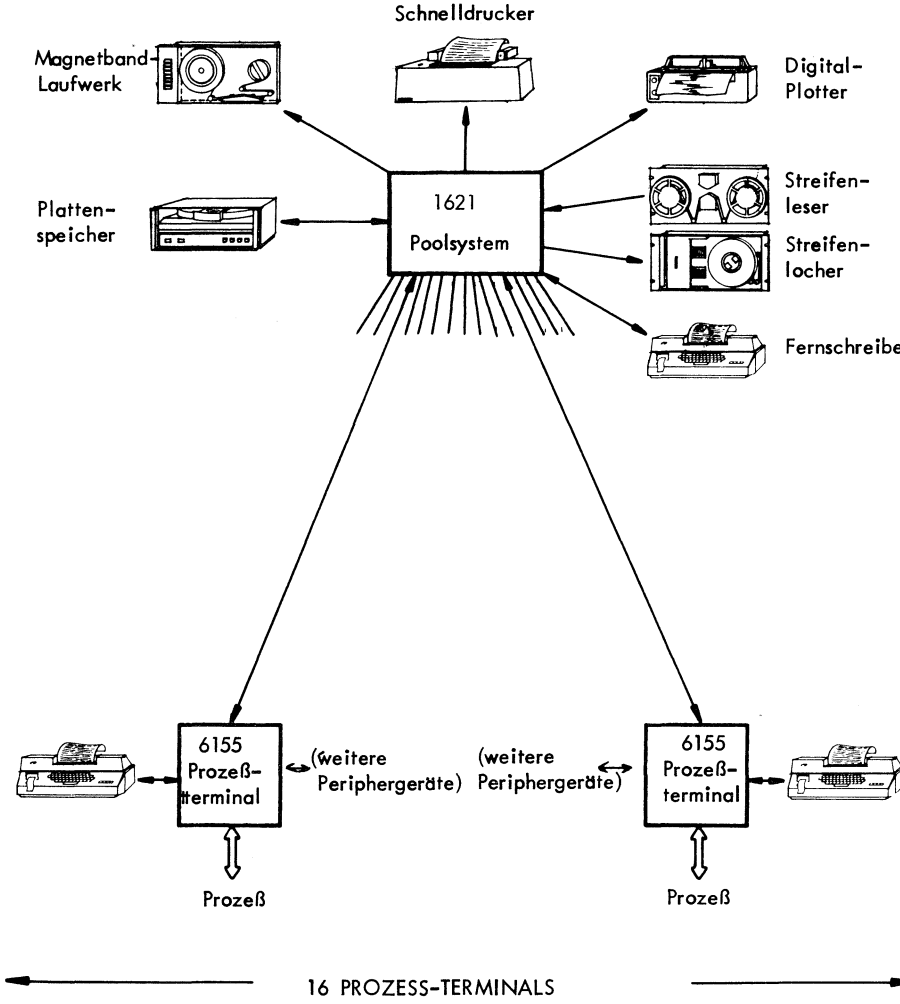
- Es hält leistungsfähige Periphergeräte bereit, die von den Terminals nach Bedarf angesprochen werden können, z.B.:

  - einen Zeilendrucker (300...350 Zeichen/min; 132 Z/Zeile)
  - ein 9-Spur-Magnetbandgerät (10.5" Spulen)
  - einen Digital-Plotter für graphische Ausgaben.

Das Poolsystem, das einen Computer vom Typ MINCAL 1600 enthält, führt keine Real-time-Verarbeitung von BASEX-Programmen aus; jedoch kann mit Hilfe der Konsol-Peripherie - Konsol-Fernschreiber sowie Streifenleser und -locher - zusätzlich Batch-Verarbeitung in Assembler, FORTRAN oder BASIC durchgeführt werden.

Daten, Programme und Kommandos werden zwischen Poolsystem und Terminals über Zweidraht-Leitungen mit einer Übertragungsrate von 2400 Bd ausgetauscht. Die maximale Entfernung beträgt je nach Kabeltyp bis zu 300 m; darüberhinaus muß eine spezielle Leitungsanpassung vorgenommen werden.

Die Prozeßterminals werden automatisch vom Poolsystem mit dem Betriebssystem geladen, sobald sie eingeschaltet werden; hierfür ist ein spezieller Festprogrammspeicher im Terminal vorgesehen.




6150

PROZESSTERMINAL-SYSTEM

# Computer-Fibel

Dies soll eine kleine Hilfe für alle die Benutzer des DIETZ 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 oder mehr diskrete 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 IC-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".

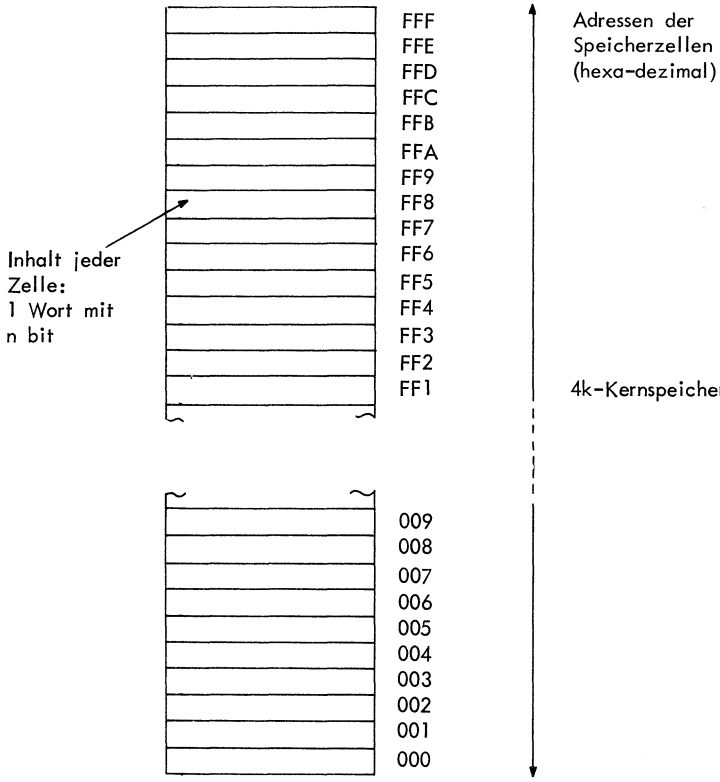
## REGISTER UND SPEICHER

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 | FFE |
| letzte Adresse | 1 1 1 1 | 1 1 1 1 | 1 1 1 1 | FFF |




Der RAM-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

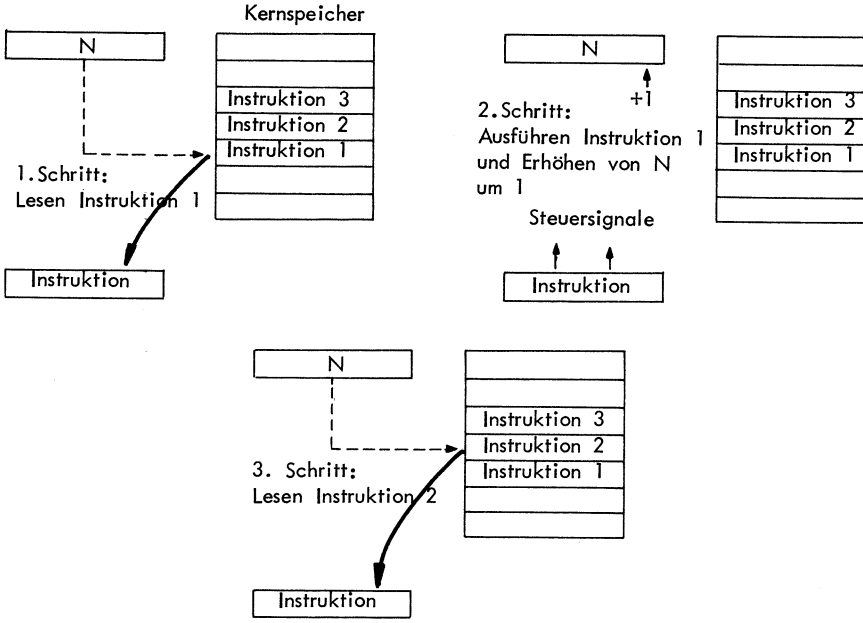
Speichern und Wiederauffinden von Daten ist zwar für einen Computer wesentlich, aber er kann noch mehr: Mit diesen Daten rechnen, sie manipulieren, ausgeben oder von außen aufnehmen. Aber all dies muß ihm genau vorgeschrieben werden. Dann führt er die gegebenen Anweisungen blitzschnell und sklavisch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

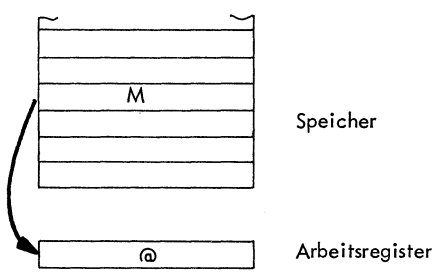
Um die Instruktionen aus dem Speicher zu lesen, benötigt der Computer eine Adresse. Diese Adresse muß natürlich "mitlaufen" und immer die Instruktion adressieren, die gerade ausgeführt werden soll. Dieses "Zählen" der Adressen übernimmt der INSTRUKTIONSZÄHLER (oder auch N-Register).

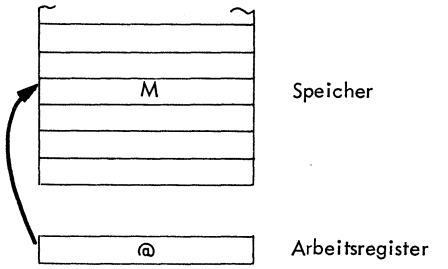
Die aus dem Speicher gelesenen Instruktionen werden in einem anderen Register gespeichert. Dieses Register erzeugt Steuersignale für das RECHENWERK (das eigentlich ausführende Organ des Computers) und bestimmt, mit welchen Daten gearbeitet werden soll.



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).




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

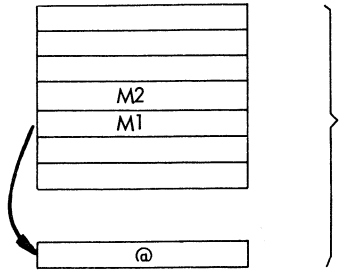
Bemerkenswert bei diesen Transportvorgängen ist, daß beim Datenempfang der alte Inhalt zerstört oder überschrieben wird, beim Senden aber erhalten bleibt. Beim Transport < M> → @ haben anschließend M und @ den gleichen Inhalt, nämlich den ursprünglich nur in M gespeicherten.

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:

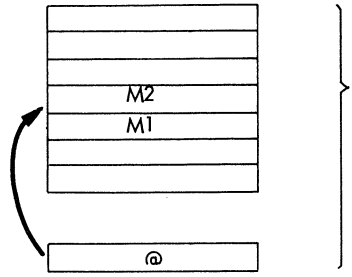
```
LDA,@ ,M1
STA,@ ,M2
```

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:




1. Operation
Laden aus Adresse M1
< M1> → @




2. Operation
Speichern in Adresse M2
< @ > → M2

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

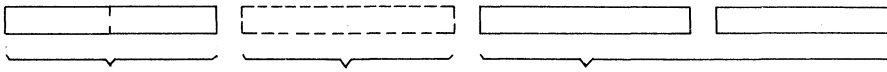
- Was ist zu tun? (Laden, Speichern)
  Diese Angabe nennt man den BEFEHL

- 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 DIETZ 621 sieht eine Instruktion so aus:

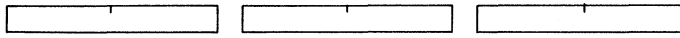



1. Byte: enthält den Befehl

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):

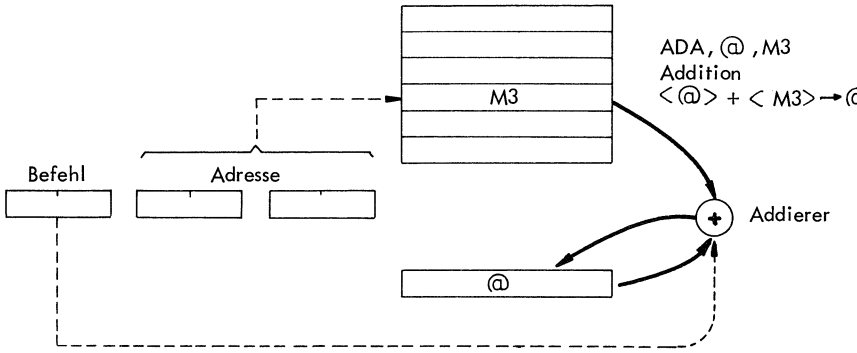



| | | | | | | |
|---|---|---|---|---|---|---|
| 8 | C | E | 2 | 0 | 1 | (LDA,@ ,M1) |
| E | C | E | 3 | 0 | 1 | (STA,@ ,M2) |

## 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:

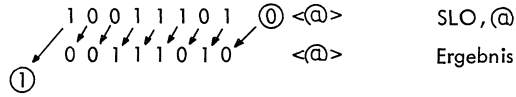
```
0 1 0 0 1 1     <(@)>          EOA, (@) ,Mn
0 1 1 1 0 1     <Mn>
-----------
0 0 1 1 1 0     <(@)>  Ergebnis
```

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.

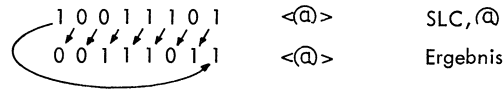
Außer den Befehlen, die einen Operanden mit dem Akkumulator verknüpfen, gibt es auch Befehle, die nur den Inhalt des Arbeitsregisters auf eine bestimmte Weise verändern. Hierzu gehören die Schiebebefehle. Der Inhalt des Akkus läßt sich rechts oder links verschieben, und das offen und geschlossen. Was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

Schiften links offen




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

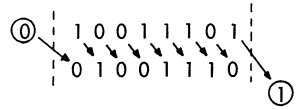
Schiften links geschlossen (Rotieren)




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.

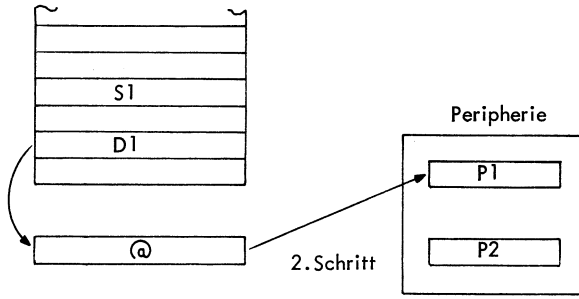
Eine typische Befehlsfolge für einen Ausgabevorgang sieht so aus:

| | |
|---|---|
| 1) Laden Datenwort aus Speicher | LDA, @ ,D1 |
| 2) Ausgabe Datenwort an Peripherie | STA, @ ,P1 |

D1 = Kernspeicher-Adresse und

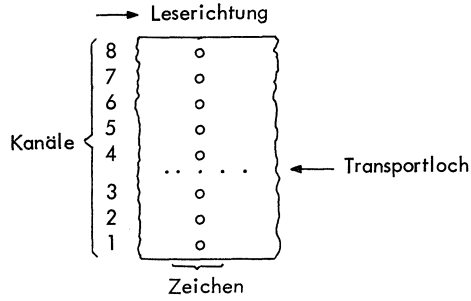
P1 = Externe Adresse.

1. Schritt



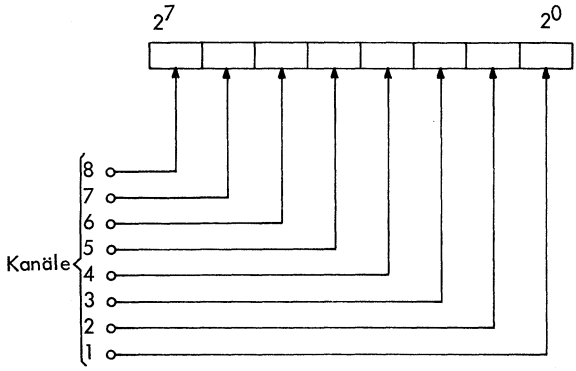
Normalerweise verstehen die Peripherie-Geräte nicht den Binärcode, mit dem der Computer rechnet. Sie haben ihren eigenen, z.B. den ASCII-Code. Dieser Code kommt auch in den Lochstreifen vor, die der Computer liest oder stanzt.

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<br>Bit | 8 7 6 5 4 3 2 1<br>$2^7 \leftarrow \quad \rightarrow 2^0$ | hexa-dez.<br>Darstellung | ASCII-Code-<br>Bedeutung |
|---|---|---|---|
| | 0 0 1 1 0 0 0 0 | '30 | Ziffer Ø |
| | 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) |
| 'ØF | 0 0 0 0 1 1 1 1 | = Maske |
| 'Ø5 | 0 0 0 0 0 1 0 1 | = Ziffer 5 (Binär) |

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

| | | |
|---|---|---|
| 'Ø7 | 0 0 0 0 0 1 1 1 | = 7 (Binär) |
| 'BØ | 1 0 1 1 0 0 0 0 | = Ergänzung |
| 'B7 | 1 0 1 1 0 1 1 1 | = Ziffer 7 (ASCII) |

Bemerkung:

Um bei der Schreibweise von Zahlen die dezimale von der hexa-dezimalen Darstellung unterscheiden zu können, wird eine hexa-dezimale Zahl 1Ø als '1Ø gekennzeichnet ('1Ø = 16!).
Ebenfalls wird die Null (Ø) durchgestrichen, um sie von dem Buchstaben O unterscheiden zu können.

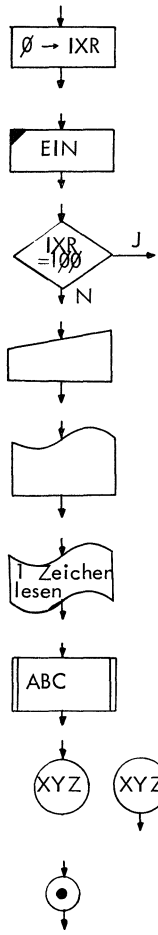
## WIR SCHREIBEN EIN PROGRAMM

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.

Diese Aufgabenstellung ist noch sehr leicht zu überschauen, und man könnte sie daher sofort mit den bekannten symbolischen Befehlen programmieren.

Trotzdem wollen wir uns bereits bei dieser Aufgabe wie echte Programmierer verhalten; denn bei denen kommt zuerst das Blockdiagramm, um stets die Übersicht zu behalten.

In Blockdiagrammen verwenden wir graphische Symbole für einzelne Aufgaben. 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

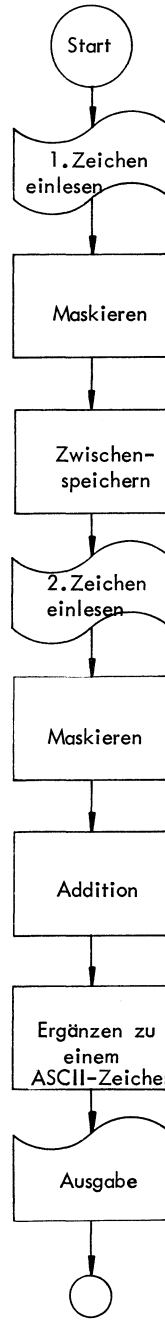
Lesen oder Stanzen eines Lochstreifens (im Zweifelsfall hineinschreiben)

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

Unser Blockdiagramm des Lösungsweges sieht also folgendermaßen aus:



Nachdem wir durch das Blockdiagramm den genauen Ablauf des Programms festgelegt haben, können wir nun die symbolischen Befehle schreiben.

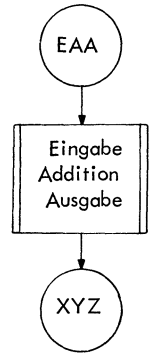
```
EAA:    LDA ,@ ,EXT     (Eingabe 1. Zeichen)
        ANA,@ ,MASK     (Maske)
        STA ,@ ,ZWS     (Zwischenspeichern)
        LDA ,@ ,EXT     (Eingabe 2. Zeichen)
        ANA,@ ,MASK     (Maske)
        ADA ,@ ,ZWS     (Addition)
        ORA ,@ ,ASCI    (Ergänzung)
        STA ,@ ,EXT     (Ausgabe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

MASK:   H  ,'ØF

ASCI:   H  ,'BØ

ZWS:    V

EXT:    2*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 Linksmarke. Das geschieht mit den Symbolen V (VARIABLE) und H (Hexa-dezimaler FESTWERT). Mit Q wird dem symbolischen Gerätenamen EXT die Adresse ØØF1 zugewiesen.

## 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.

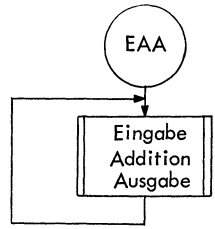



Dann muß an die Stelle der drei Pünktchen die Instruktion: JPA, , XYZ gesetzt werden. Damit wird der Instruktionszähler auf die Anfangsadresse des Programmteils XYZ gesetzt.

Oder man kann auch nach EAA zurückverzweigen:

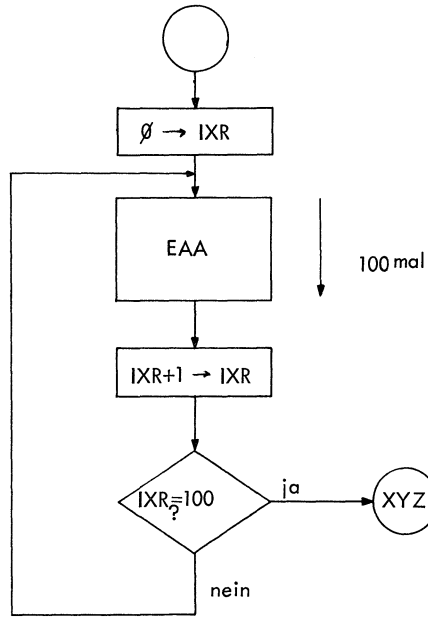
```
JPA, , EAA
```

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 100. 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:

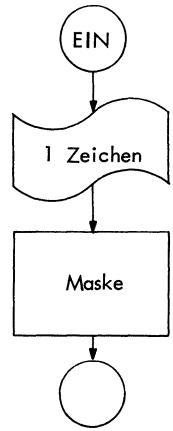
```
JPX, , ,RET
```

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

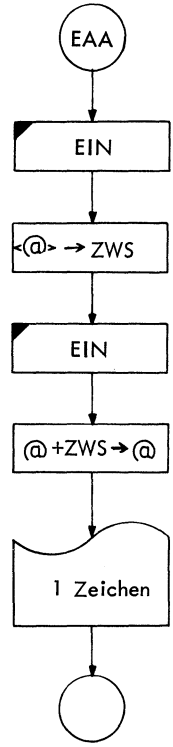



1 Zeichen lesen

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.

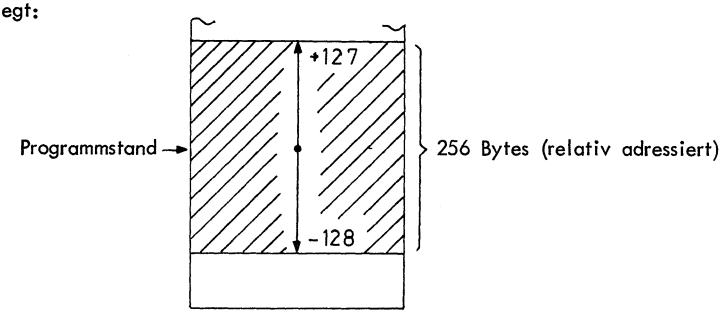
Außerdem haben wir beim Rücksprung aus dem Unterprogramm die INDIREKTE Adressierung kennengelernt. Hierbei ist die effektive Adresse gleich dem Inhalt der programmierten Adresse:

| | |
|---|---|
| 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 |

# Tabellen

vorn

| Leiterseite | | Bauelementseite |
|---|---|---|
| 2A | | 2C |
| +Z | 1 | +Z |
| BE | 2 | GE |
| FE | 3 | |
| | 4 | |
| | 5 | |
| | 6 | |
| RK | 7 | |
| | 8 | |
| | 9 | |
| A15 | 10 | LØØ |
| A14 | 11 | LØ1 |
| A13 | 12 | LØ2 |
| A12 | 13 | LØ3 |
| A11 | 14 | LØ4 |
| AlØ | 15 | LØ5 |
| AØ9 | 16 | LØ6 |
| AØ8 | 17 | LØ7 |
| AØ7 | 18 | LØ8 |
| AØ6 | 19 | LØ9 |
| AØ5 | 20 | L1Ø |
| AØ4 | 21 | L11 |
| AØ3 | 22 | L12 |
| AØ2 | 23 | L13 |
| AØ1 | 24 | L14 |
| AØØ | 25 | L15 |
| | 26 | |
| | 27 | |
| | 28 | |
| | 29 | |
| | 30 | |
| | 31 | |
| ⊥ | 32 | ⊥ |

2Ø2

hinten

| Leiterseite | | Bauelementseite |
|---|---|---|
| 1A | | 1C |
| +Z | 1 | +Z |
| | 2 | |
| | 3 | |
| | 4 | |
| | 5 | |
| | 6 | |
| | 7 | |
| | 8 | |
| D8 | 9 | |
| D7 | 10 | SØØ |
| D6 | 11 | SØ1 |
| D5 | 12 | SØ2 |
| D4 | 13 | SØ3 |
| D3 | 14 | SØ4 |
| D2 | 15 | SØ5 |
| D1 | 16 | SØ6 |
| DØ | 17 | SØ7 |
| | 18 | SØ8 |
| N | 19 | SØ9 |
| | 20 | S1Ø |
| STPX | 21 | S11 |
| STWD | 22 | S12 |
| | 23 | S13 |
| | 24 | S14 |
| | 25 | S15 |
| | 26 | |
| | 27 | |
| | 28 | |
| | 29 | |
| | 30 | |
| | 31 | |
| ⊥ | 32 | ⊥ |

1Ø2

Bemerkung: Alle Signale (außer RK) mit negativer Logik (0 V wenn angewählt).

Steckerbelegung: UNIVERSAL-BUS-Anschluß (Zentraleinheit DIETZ 621)

oben

| Leiterseite 1A | | Bauelementseite 1C |
|---|---|---|
| +Z | 1 | DØ |
| IDP | 2 | D1 |
| IDØ7 | 3 | D2 |
| ID17 | 4 | D3 |
| IDØ6 | 5 | D4 |
| ID16 | 6 | D5 |
| IDØ5 | 7 | D6 |
| ID15 | 8 | D7 |
| IDØ4 | 9 | D8 |
| ID14 | 10 | AØØ |
| IDØ3 | 11 | AØ1 |
| ID13 | 12 | AØ2 |
| IDØ2 | 13 | AØ3 |
| ID12 | 14 | AØ4 |
| IDØ1 | 15 | AØ5 |
| ID11 | 16 | AØ6 |
| IDØØ | 17 | AØ7 |
| ID1Ø | 18 | AØ8 |
| IRK | 19 | AØ9 |
| IN | 20 | A1Ø |
| IFT | 21 | A11 |
| IAØØ | 22 | A12 |
| IAØ1 | 23 | A13 |
| IAØ2 | 24 | A14 |
| IAØ3 | 25 | A15 |
| ISØØ | 26 | RK |
| IRMØØ | 27 | N |
| ISØ1 | 28 | STPX |
| IRMØ1 | 29 | BE |
| ISØ2 | 30 | GE |
| IRMØ2 | 31 | FE |
| ⊥ | 32 | LØØ |

Mini-BUS — Universal-BUS

unten

| Leiterseite 2A | | Bauelementseite 2C |
|---|---|---|
| +Z | 1 | LØ1 |
| ISØ3 | 2 | LØ2 |
| IRMØ3 | 3 | LØ3 |
| ISØ4 | 4 | LØ4 |
| IRMØ4 | 5 | LØ5 |
| ISØ5 | 6 | LØ6 |
| IRMØ5 | 7 | LØ7 |
| ISØ6 | 8 | LØ8 |
| IRMØ6 | 9 | LØ9 |
| ISØ7 | 10 | L1Ø |
| IRMØ7 | 11 | L11 |
| ISØ8 | 12 | L12 |
| IRMØ8 | 13 | L13 |
| ISØ9 | 14 | L14 |
| IRMØ9 | 15 | L15 |
| IS1Ø | 16 | SØØ |
| IRM1Ø | 17 | SØ1 |
| IS11 | 18 | SØ2 |
| IRM11 | 19 | SØ3 |
| | 20 | SØ4 |
| | 21 | SØ5 |
| | 22 | SØ6 |
| | 23 | SØ7 |
| | 24 | SØ8 |
| | 25 | SØ9 |
| | 26 | S1Ø |
| | 27 | S11 |
| | 28 | S12 |
| | 29 | S13 |
| | 30 | S14 |
| | 31 | S15 |
| ⊥ | 32 | ⊥ |

Mini-BUS — Universal-BUS

Bemerkung: Alle Signale (außer RK, IRK) mit negativer Logik (0V wenn angewählt)

Steckerbelegung: DEVICE-SELECTOR (Universal-Interface-Einschub)

oben

| Leiterseite | | Bauelementseite |
|---|---|---|
| 1A | | 1C |
| +Z | 1 | +Z |
| | 2 | |
| +12 V | 3 | +12 V |
| | 4 | |
| -12 V | 5 | -12 V |
| | 6 | |
| IDP | 7 | IDP |
| IDØ7 | 8 | ID17 |
| ⊥ | 9 | ⊥ |
| IDØ6 | 10 | ID16 |
| IDØ5 | 11 | ID15 |
| ⊥ | 12 | ⊥ |
| IDØ4 | 13 | ID14 |
| IDØ3 | 14 | ID13 |
| ⊥ | 15 | ⊥ |
| IDØ2 | 16 | ID12 |
| IDØ1 | 17 | ID11 |
| IDØØ | 18 | ID1Ø |
| ⊥ | 19 | ⊥ |
| IAØØ | 20 | IAØØ |
| IAØ1 | 21 | IAØ1 |
| ⊥ | 22 | ⊥ |
| IAØ2 | 23 | IAØ2 |
| IAØ3 | 24 | IAØ3 |
| IRK | 25 | IRK |
| ⊥ | 26 | ⊥ |
| IN | 27 | IN |
| ⊥ | 28 | ⊥ |
| IFT | 29 | IFT |
| ISXY | 30 | ISXY |
| IRMXY | 31 | IRMXY |
| ⊥ | 32 | ⊥ |

Mini-BUS

unten

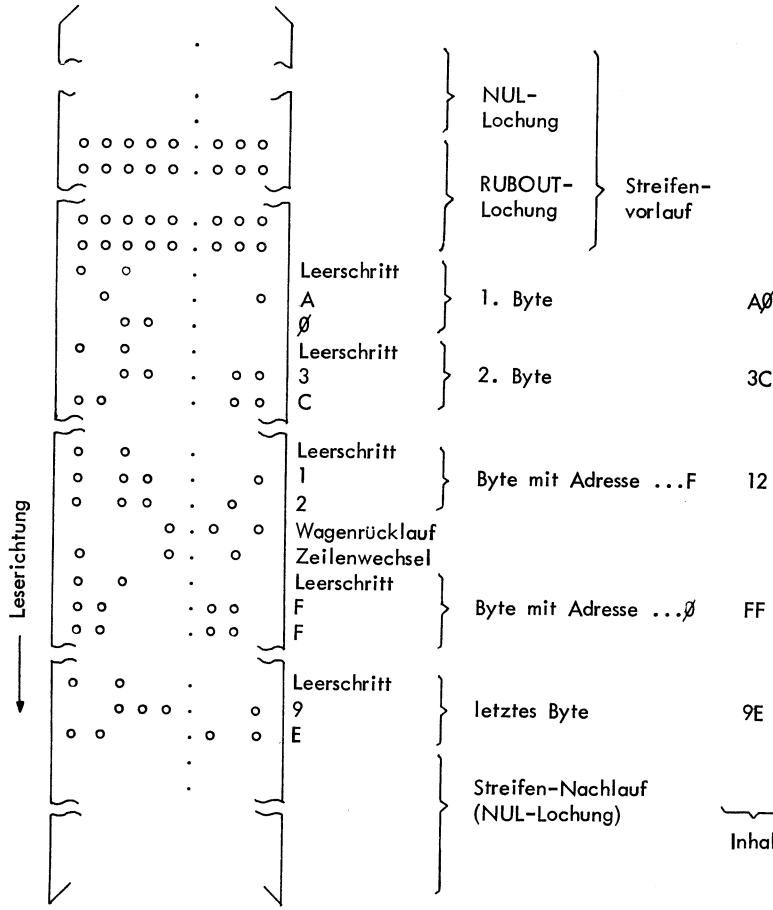
| Leiterseite | | Bauelementseite |
|---|---|---|
| 2A | | 2C |
| +Z | 1 | +Z |
| | 2 | |
| | 3 | |
| | 4 | |
| | 5 | |
| | 6 | |
| | 7 | |
| | 8 | |
| | 9 | |
| | 10 | |
| | 11 | |
| | 12 | |
| | 13 | |
| | 14 | |
| | 15 | |
| | 16 | |
| | 17 | |
| | 18 | |
| | 19 | |
| | 20 | |
| | 21 | |
| | 22 | |
| | 23 | |
| | 24 | |
| | 25 | |
| V | 26 | V |
| U ex 1 | 27 | U ex 1 |
| U ex 2 | 28 | U ex 2 |
| U ex 3 | 29 | U ex 3 |
| U ex 4 | 30 | U ex 4 |
| U ex 5 | 31 | U ex 5 |
| ⊥ | 32 | ⊥ |

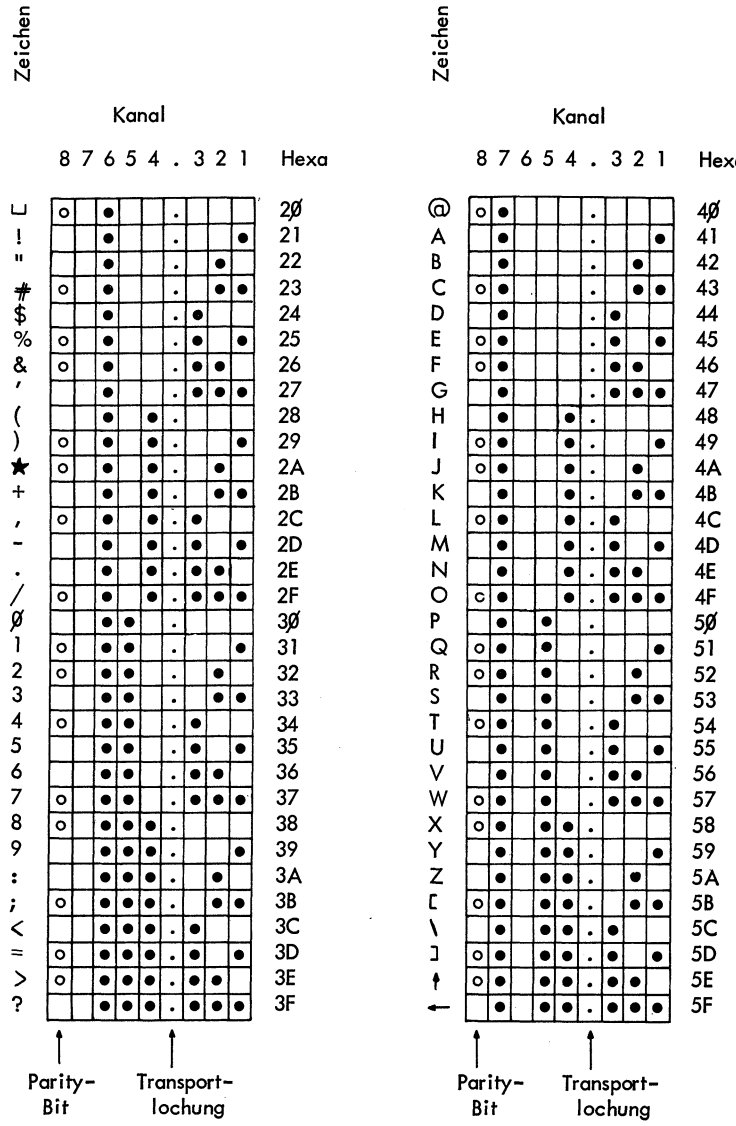
für gerätespezifische Signale

Bemerkung: Alle Signale (außer IRK) mit negativer Logik (0 V wenn angewählt)

U ex 1 ... U ex 5 sind gedruckte Verbindungen, die für alle Steckplätze einheitlich für externe Spannungsversorgung verschaltet werden können.
V ist eine gedruckte Verbindung von A- und B-Seite

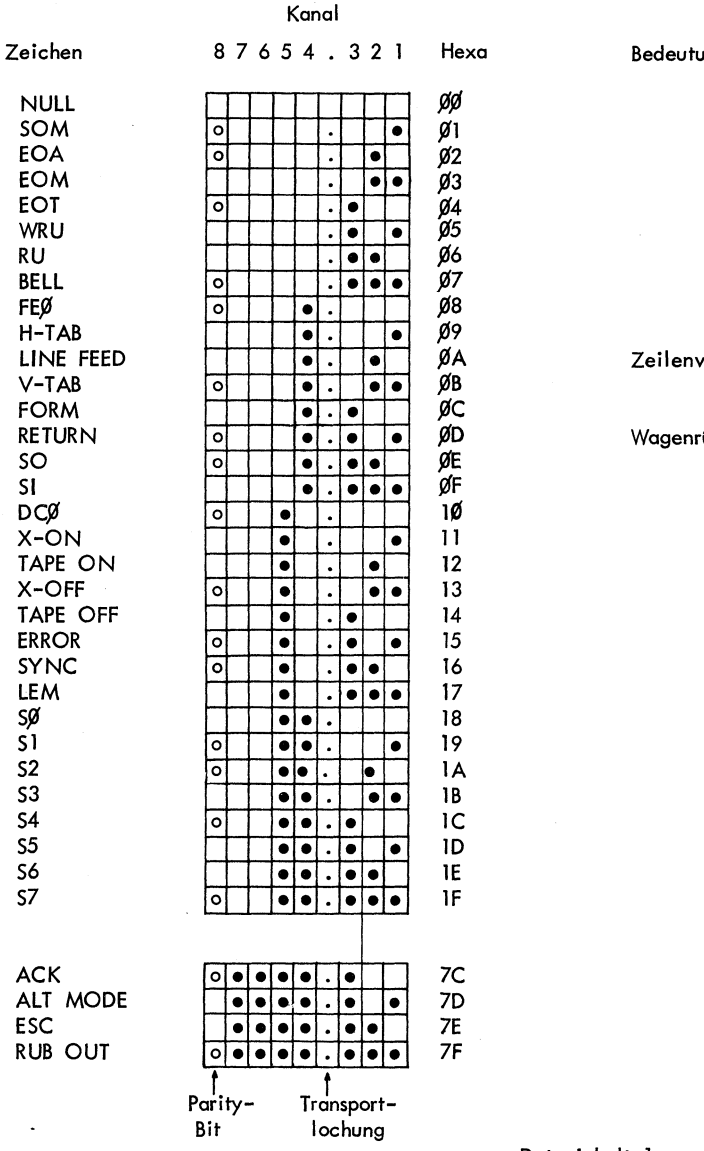
Steckerbelegung: EINKARTEN-INTERFACE (Mini-BUS).


NUL-
Lochung
RUBOUT-
Lochung
Streifen-
vorlauf
Leerschritt
A
Ø
1. Byte
AØ
Leerschritt
3
C
2. Byte
3C
Leerschritt
1
2
Byte mit Adresse ...F
12
Wagenrücklauf
Zeilenwechsel
Leerschritt
F
F
Byte mit Adresse ...Ø
FF
Leerschritt
9
E
letztes Byte
9E
Streifen-Nachlauf
(NUL-Lochung)
Inhalt
Leserichtung
Kanal
8 7 6 5 4 . 3 2 1
Paritätsbit -
Transportloch-




⊔ (Zeichen 2Ø)
bedeutet Leerschritt

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


Kanal
Zeichen
8 7 6 5 4 . 3 2 1
Hexa
Bedeutung
NULL ØØ
SOM Ø1
EOA Ø2
EOM Ø3
EOT Ø4
WRU Ø5
RU Ø6
BELL Ø7
FEØ Ø8
H-TAB Ø9
LINE FEED ØA Zeilenvorschub
V-TAB ØB
FORM ØC
RETURN ØD Wagenrücklauf
SO ØE
SI ØF
DCØ 1Ø
X-ON 11
TAPE ON 12
X-OFF 13
TAPE OFF 14
ERROR 15
SYNC 16
LEM 17
SØ 18
S1 19
S2 1A
S3 1B
S4 1C
S5 1D
S6 1E
S7 1F
ACK 7C
ALT MODE 7D
ESC 7E
RUB OUT 7F
Parity-Bit
Transport-lochung
o = Dateninhalt 1
● = Lochung im Streifen
● = Stromschritt (MARK)

| $2^n$ | n | $2^{-n}$ |
|---|---|---|
| 1 | 0 | 1,0 |
| 2 | 1 | 0,5 |
| 4 | 2 | 0,25 |
| 8 | 3 | 0,125 |
| 16 | 4 | 0,062 5 |
| 32 | 5 | 0,031 25 |
| 64 | 6 | 0,015 625 |
| 128 | 7 | 0,007 812 5 |
| 256 | 8 | 0,003 906 25 |
| 512 | 9 | 0,001 953 125 |
| 1 024 | 10 | 0,000 976 562 5 |
| 2 048 | 11 | 0,000 488 281 25 |
| 4 096 | 12 | 0,000 244 140 625 |
| 8 192 | 13 | 0,000 122 070 312 5 |
| 16 384 | 14 | 0,000 061 035 156 25 |
| 32 768 | 15 | 0,000 030 517 578 125 |
| 65 536 | 16 | 0,000 015 258 789 062 5 |
| 131 072 | 17 | 0,000 007 629 394 531 25 |
| 262 144 | 18 | 0,000 003 814 697 265 625 |
| 524 288 | 19 | 0,000 001 907 348 632 812 5 |
| 1 048 576 | 20 | 0,000 000 953 674 316 406 25 |
| 2 097 152 | 21 | 0,000 000 476 837 158 203 125 |
| 4 194 304 | 22 | 0,000 000 238 418 579 101 562 5 |
| 8 388 608 | 23 | 0,000 000 119 209 289 550 781 25 |
| 16 777 216 | 24 | 0,000 000 059 604 644 775 390 625 |
| 33 554 432 | 25 | 0,000 000 029 802 322 387 695 312 5 |
| 67 108 864 | 26 | 0,000 000 014 901 161 193 847 656 25 |
| 134 217 728 | 27 | 0,000 000 007 450 580 596 923 828 125 |
| 268 435 456 | 28 | 0,000 000 003 725 290 298 461 914 062 5 |
| 536 870 912 | 29 | 0,000 000 001 862 645 149 230 957 031 25 |
| 1 073 741 824 | 30 | 0,000 000 000 931 322 574 615 478 515 625 |
| 2 147 483 648 | 31 | 0,000 000 000 465 661 287 307 739 257 812 5 |
| 4 294 967 296 | 32 | 0,000 000 000 232 830 643 653 869 628 906 25 |
| 8 589 934 592 | 33 | 0,000 000 000 116 415 321 826 934 814 453 125 |
| 17 179 869 184 | 34 | 0,000 000 000 058 207 660 913 467 407 226 562 5 |
| 34 359 738 368 | 35 | 0,000 000 000 029 103 830 456 733 703 613 281 25 |
| 68 719 476 736 | 36 | 0,000 000 000 014 551 915 228 366 851 806 640 625 |

NOTIZEN

| Symbol. Befehle | BEFEHLSBYTE hexa | bintär | String 1 | 2 |
|---|---|---|---|---|
| NOP | ØØ | 00000000 | | |
| SEL,1 | Ø1 | 00000001 | | 1 |
| HLT | Ø2 | 00000010 | | |
| HSL,1 | Ø3 | 00000011 | | 1 |
| ECL | Ø4 | 00000100 | | |
| | | | | |
| | | | | |
| DCL | Ø8 | 00001000 | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| 2 & | 1Ø | 00010000 | | |
| z & | 11 | 00010001 | | z |
| 2 * | 12 | 00010010 | | |
| z * | 13 | 00010011 | | z |
| 2>& | 14 | 00010100 | | |
| z>& | 15 | 00010101 | | z |
| 2>* | 16 | 00010110 | | |
| z>* | 17 | 00010111 | | z |
| 2<& | 18 | 00011000 | | |
| z<& | 19 | 00011001 | | z |
| 2<* | 1A | 00011010 | | |
| z<* | 1B | 00011011 | | z |
| 2=& | 1C | 00011100 | | |
| z=& | 1D | 00011101 | | z |
| 2=* | 1E | 00011110 | | |
| z=* | 1F | 00011111 | | z |
| | | | | |
| | | | | |
| GS ,@ | 22 | 00100010 | | |
| s | 23 | 00100011 | | s |
| GL ,@ | 24 | 00100100 | | |
| s | 25 | 00100101 | | s |
| | | | | |
| | | | | |
| SRO,@ | 28 | 00101000 | | |
| s | 29 | 00101001 | | s |
| SRC,@ | 2A | 00101010 | | |
| s | 2B | 00101011 | | s |
| SLO,@ | 2C | 00101100 | | |
| s | 2D | 00101101 | | s |
| SLC,@ | 2E | 00101110 | | |
| s | 2F | 00101111 | | s |

| Symbol. Befehle | BEFEHLSBYTE hexa | bintär | String 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|---|
| BZ ,@ | 4Ø | 01000000 | | d | | |
| s | 41 | 01000001 | | s | d | |
| IZ ,@ | 42 | 01000010 | | d | | |
| s | 43 | 01000011 | | s | d | |
| BP ,@ | 44 | 01000100 | | d | | |
| s | 45 | 01000101 | | s | d | |
| IP ,@ | 46 | 01000110 | | d | | |
| s | 47 | 01000111 | | s | d | |
| BNZ ,@ | 48 | 01001000 | | d | | |
| s | 49 | 01001001 | | s | d | |
| INZ ,@ | 4A | 01001010 | | d | | |
| s | 4B | 01001011 | | s | d | |
| BNP ,@ | 4C | 01001100 | | d | | |
| s | 4D | 01001101 | | s | d | |
| INP ,@ | 4E | 01001110 | | d | | |
| s | 4F | 01001111 | | s | d | |
| BEC ,@ | 5Ø | 01010000 | | c | d | |
| s | 51 | 01010001 | | s | c | d |
| IEC ,@ | 52 | 01010010 | | c | d | |
| s | 53 | 01010011 | | s | c | d |
| BER ,@ | 54 | 01010100 | | r | d | |
| s | 55 | 01010101 | | s | r | d |
| IER ,@ | 56 | 01010110 | | r | d | |
| s | 57 | 01010111 | | s | r | d |
| BNEC,@ | 58 | 01011000 | | c | d | |
| s | 59 | 01011001 | | s | c | d |
| INEC,@ | 5A | 01011010 | | c | d | |
| s | 5B | 01011011 | | s | c | d |
| BNER,@ | 5C | 01011100 | | r | d | |
| s | 5D | 01011101 | | s | r | d |
| INER,@ | 5E | 01011110 | | r | d | |
| s | 5F | 01011111 | | s | r | d |
| BZC ,@ | 6Ø | 01100000 | | c | d | |
| s | 61 | 01100001 | | s | c | d |
| IZC ,@ | 62 | 01100010 | | c | d | |
| s | 63 | 01100011 | | s | c | d |
| BZR ,@ | 64 | 01100100 | | r | d | |
| s | 65 | 01100101 | | s | r | d |
| IZR ,@ | 66 | 01100110 | | r | d | |
| s | 67 | 01100111 | | s | r | d |
| BNZC,@ | 68 | 01101000 | | c | d | |
| s | 69 | 01101001 | | s | c | d |
| INZC,@ | 6A | 01101010 | | c | d | |
| s | 6B | 01101011 | | s | c | d |
| BNZR,@ | 6C | 01101100 | | r | d | |
| s | 6D | 01101101 | | s | r | d |
| INZR,@ | 6E | 01101110 | | r | d | |
| s | 6F | 01101111 | | s | r | d |
| BOC ,@ | 7Ø | 01110000 | | c | d | |
| s | 71 | 01110001 | | s | c | d |
| IOC ,@ | 72 | 01110010 | | c | d | |
| s | 73 | 01110011 | | s | c | d |
| BOR ,@ | 74 | 01110100 | | r | d | |
| s | 75 | 01110101 | | s | r | d |
| IOR ,@ | 76 | 01110110 | | r | d | |
| s | 77 | 01110111 | | s | r | d |
| BNOC,@ | 78 | 01111000 | | c | d | |
| s | 79 | 01111001 | | s | c | d |
| INOC,@ | 7A | 01111010 | | c | d | |
| s | 7B | 01111011 | | s | c | d |
| BNOR,@ | 7C | 01111100 | | r | d | |
| s | 7D | 01111101 | | s | r | d |
| INOR,@ | 7E | 01111110 | | s | r | d |
| s | 7F | 01111111 | | s | r | d |

| Symbol. Befehle | BEFEHLSBYTE hexa | bintär | String 1 | 2 | 3 | 4 | 5 | |
|---|---|---|---|---|---|---|---|---|
| LD. | 8. | 1000.... | | | | | | |
| AD. | 9. | 1001.... | | | | | | |
| SB. | A. | 1010.... | | | | | | |
| AN. | B. | 1011.... | | | | | | |
| OR. | C. | 1100.... | | | | | | |
| EO. | D. | 1101.... | | | | | | |
| ST. | E. | 1110.... | | | | | | |
| JP. | F. | 1111...0 | | | | | | |
| CS. | F. | 1111...1 | | | | | | |
| . .C,@ | .Ø | ....0000 | | c | | | | |
| s | .1 | ....0001 | | s | c | | | |
| . .X,@ | .2 | ....0010 | | x | | | | |
| s | .3 | ....0011 | | s | x | | | |
| . .R,@ | .4 | ....0100 | | r | | | | |
| s | .5 | ....0101 | | s | r | | | |
| . .R,@ | .6 | ....0110 | | r | x | | | indiziert |
| s | .7 | ....0111 | | s | r | x | | indiziert |
| . .L,@ | .8 | ....1000 | | d | | | | |
| s | .9 | ....1001 | | s | d | | | |
| . .L,@ | .A | ....1010 | | d | x | | | indiziert |
| s | .B | ....1011 | | s | d | x | | indiziert |
| . .A,@ | .C | ....1100 | | a | b | | | |
| s | .D | ....1101 | | s | a | b | | |
| . .A,@ | .E | ....1110 | | a | b | x | | indiziert |
| s | .F | ....1111 | | s | a | b | x | indiziert |

a Adresse niedrig
b Adresse hoch
c Konstante
d Differenz/Sprungweite
r Referenzregister
s Arbeitsregister
x Indexregister
z Anzahl