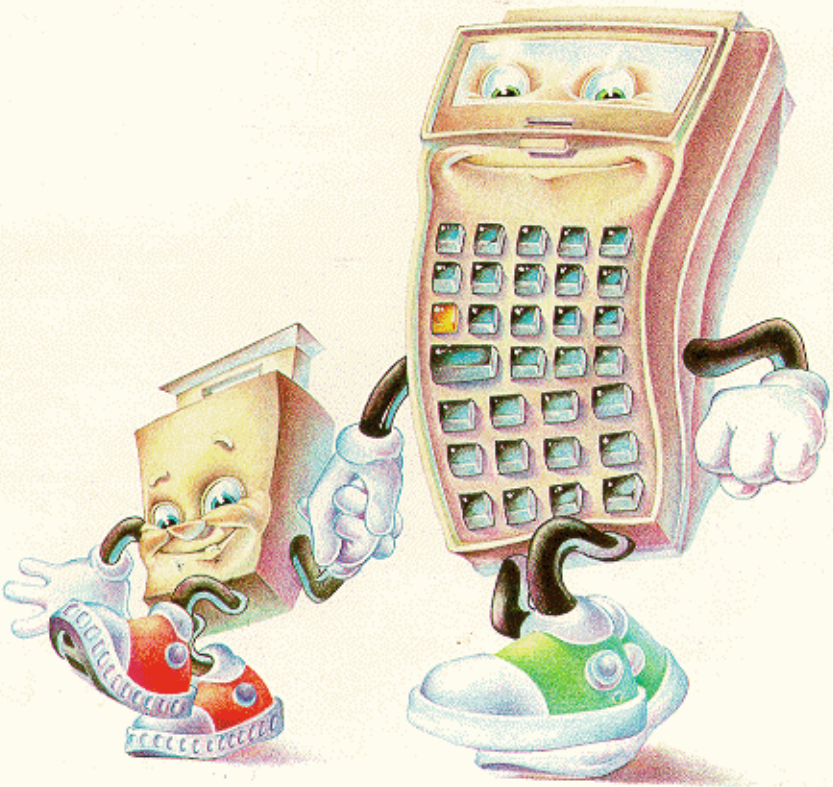
W&W SOFTWARE PRODUCTS GMBH

CCD - Modul

Benutzerhandbuch

für den HP-41



Vorwort

Das CCD-Modul wurde mit dem Ziel entwickelt, allen Anwendern des HP-41 ein Werkzeug zur besseren und einfacheren Programmierung in die Hand zu geben. Hierbei wurde besonderer Wert auf die Anwenderfreundlichkeit gelegt.
Die CCD-Modul-Funktionen sollen den Programmierer soweit unterstützen, daß er sich auf das eigentliche Problem konzentrieren kann. Andere Programmteile (formatierte Ein- und Ausgabe) können einfach mit den neuen Funktionen aufgebaut und auch bisher unlösbare Probleme gelöst werden.

Das vorliegende Modul wurde in über einjähriger Entwicklungszeit hergestellt. Besonderer Dank für die gewährte Unterstützung gilt allen Mitgliedern des CCD (Computerclub Deutschland, größter Deutscher Computerclub siehe *CCD e.V.*), die durch ihre zahlreiche Teilnahme an der durchgeführten Subskription, ihren Funktionsvorschlägen und nicht zuletzt auch durch ihre aufgebrachte Geduld dieses Modul in der vorliegenden Form erst ermöglichten. Weiterer Dank gebührt Herrn Dr. Baltes, dem geistigen Vater des Moduls, durch dessen Koordinierungsarbeit bei der Programmierung viel Platz für neue Funktionen geschaffen wurde. Ähnliches gilt auch für Herrn Holger Adelmann, durch dessen Anregungen und Programmierarbeit das CCD-Modul optimal wurde. Weiterhin gebührt mein Dank allen, die schon während der Entwicklungsphase unser Modul getestet haben und die durch Schreiben von Programmen zum Gelingen dieses Handbuchs beitrugen. Dieser Dank gilt besonders Herrn Gerhard Kruse, der eine Vielzahl optimaler Programme für das Handbuch geschrieben hat und Herrn Andreas Meyer für dessen schriftstellerische Arbeit und die hervorragend geplotteten Zeichnungen. Weiterhin danke ich der Firma Hewlett-Packard in Deutschland für die gewährte Unterstützung und zuletzt natürlich auch meinen ausländischen Freunden vom CHHU, PPC, PPCUK, JPC usw., die durch ihre Übersetzungshilfen dazu beigetragen haben, dieses Modul in der ganzen Welt bekannt zu machen.

W&W Software Products GmbH
Geschäftsleitung

(Wilfried Kötz)

Verzeichnis der Kapitel

Alphabetisches Inhaltsverzeichnis

Einleitung

Das CCD-Modul ist eine Erweiterung für den Taschenrechner HP-41. Es vergrößert den Wortschatz um etwa 100 neue Funktionen. Dazu kommt eine Vielzahl von Betriebssystem-Erweiterungen, wie neue Kataloge und die Möglichkeit, Kleinbuchstaben direkt einzugeben.

Bei der Planung des CCD-Moduls wurden Wünsche und Ideen vieler Mitglieder des Computerclub Deutschland (CCD e.V.) besonders berücksichtigt. Deshalb trägt das Modul den Namen dieser Anwendergruppe.

Vollkommen in Assembler programmiert, hebt sich das CCD-Modul von allen anderen Anwender-ROMs ab. Dies garantiert hohe Geschwindigkeit und Genauigkeit und war Vorraussetzung für viele Funktionen, die anders nicht realisierbar gewesen wären.

Das CCD-Modul stellt bereits für einen „einfachen" HP-41 C eine wertvolle Ergänzung dar; mit zunehmender Systemgröße steigen jedoch auch seine Nutzungsmöglichkeiten. Dabei unterstützt das CCD-Modul ganz besonders alle Druckertypen, das erweiterte Funktions- und Speicher-Modul und die Hewlett-Packard Interface Loop.

Kapitel 1

Aufbau des HP-41

Inhaltsverzeichnis Kapitel 1
Aufbau des HP-41

Aufbau des HP-41

Viele Funktionen des CCD-Moduls („Synthetik", PEEK und POKE) erlauben den direkten Zugriff auf den gesamten Schreib/Lesespeicher (RAM = Random Access Memory) des HP-41. Der RAM-Bereich umfaßt sowohl den Hauptspeicher (Datenregister, Programme, Tastenzuordnungen und I/O-Buffer) als auch die Statusregister (Rechen- und Rücksprungstapel, Flagregister und sonstige Informationen für das Betriebssystem) und den Speicherbereich der erweiterten Funktions- und Speichermodule.

Das Betriebssystem benutzt zahlreiche Speicherstellen, um sogenannte „Status"-Informationen abzulegen. Das unsachgemäße Verändern dieser Registerinhalte kann zu einem Absturz des Systems führen – jedoch **niemals** zu einer Beschädigung der Geräte.

In den folgenden Abschnitten werden zuerst der Aufbau und die Organisation des RAM-Bereichs beschrieben. Dem Leser sollten die Begriffe **Bit**, **Byte** und **Hexadezimalzahl** bekannt sein.

Der HP-41 beinhaltet zwei verschiedene Arten von Speichertypen:

- den Schreib/Lesespeicher (RAM) zum Ablegen von Daten und Statusinformationen und
- den Festwertspeicher (ROM = Read Only Memory), in dem das Betriebssystem enthalten ist.

(Das hier beschriebene Modul ist auch in einem ROM fest programmiert.)

Da nur der RAM-Bereich beschrieben und somit die in ihm enthaltene Information verändert werden kann, interessiert uns seine Struktur – das heißt der physikalische Aufbau und die logische Einteilung (Organisation) – besonders.

Der Schreib/Lesespeicher des HP-41

Richten wir zuerst unsere Aufmerksamkeit auf den Aufbau!

Der Schreib/Lesespeicher des HP-41 besteht aus 56-Bit-Registern. Das heißt, daß die CPU (Zentraleinheit) Informationen immer nur als gesamtes Register lesen oder schreiben kann. Will sie nur ein einzelnes Bit innerhalb eines Registers verändern, muß sie zuerst das ganze Register lesen, das Bit entsprechend setzen oder löschen und anschließend das ganze Register schreiben. Innerhalb der CPU kann der Registerinhalt im ganzen oder in Teilbereichen verändert werden. Als kleinsten Teilbereich kennen wir bereits das Bit; andere Teilbereiche sind ein Nybble (=4 Bits) oder ein Byte (= 8 Bits). Bits, Nybbles oder Bytes werden innerhalb eines Registers von rechts nach links durchnumeriert, wobei mit dem Wert 0 begonnen wird: das heißt,wir zählen die Bits von 0 bis 55, die Nybbles (es gibt 14 pro Register) von 0 bis 13 und die Bytes von 0 bis 6.

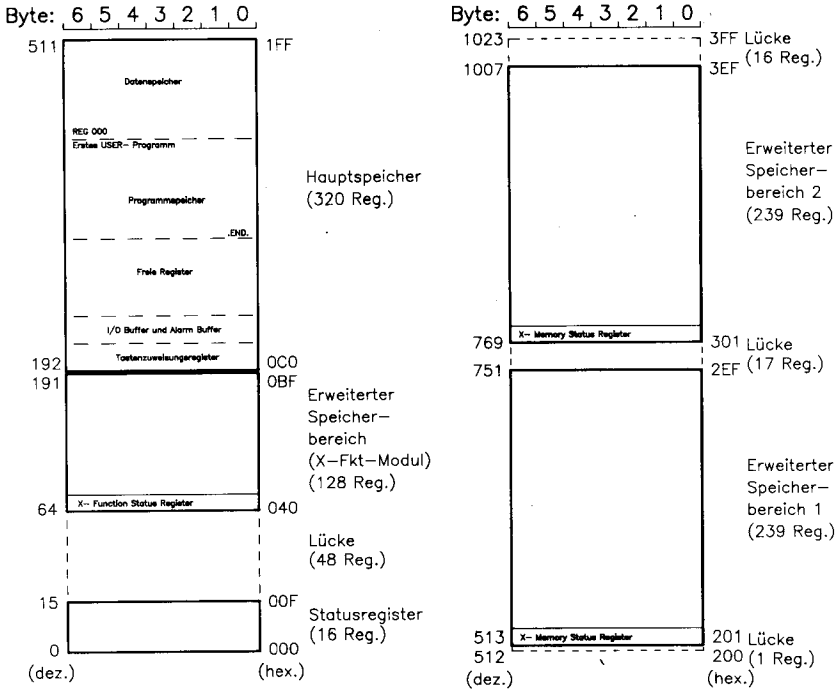
6		5		4		3		2		1		0		Byte–Nummer
13	12	11	10	9	8	7	6	5	4	3	2	1	0	Nybble–Nummer
55...	....	....	....	....	....	....	....	....	....	....	....	7654	3210	Bit–Nummer

Dabei sollte nicht vergessen werden, daß es sich hierbei nur um eine logische Unterteilung der Register handelt! Es gibt also – physikalisch gesehen – keine Grenzen zwischen Nybbles oder Bytes.

Die HP-41 CPU kann maximal 1024 solcher Register ansprechen. Dafür sind die Register von 0 bis 1023 durchnumeriert. Eine **Registernummer** wird auch als **absolute Adresse** bezeichnet; sie ist einzig und allein abhängig von der physikalischen Lage des Registers auf den RAM-Schaltkreisen. (Wir werden später noch den Begriff der **relativen Adresse** antreffen.) Daß die CPU 1024 verschiedene Register ansprechen kann, bedeutet nicht unbedingt, daß die Register auch vorhanden sind! Ganz im Gegenteil: Das Betriebssystem verlangt an ganz bestimmten Stellen Lücken im RAM-Bereich.

RAM-Speicherstruktur

Die Abbildung zeigt den Aufbau des Schreib/Lesespeichers.



Bisher haben wir nur den physikalischen Aufbau des RAM betrachtet. Nun wenden wir uns seiner logischen Organisation zu. Demnach unterscheiden wir – ihrer Verwendung entsprechend – drei Arten von Speicherbereichen:

1) Die Statusregister
2) Den Hauptspeicher
3) Den erweiterten Speicherbereich

Bezeichnung	Startadresse–Endadresse dezimal			Startadresse–Endadresse hexadezimal		
Statusregister	0	–	15	000	–	00F
Hauptspeicher	192	–	511	0C0	–	1FF
erweiterter Speicherbereich:						
X–Functions–M.	64	–	191	040	–	0BF
X–Memory 1	513	–	751	201	–	2EF
X–Memory 2	769	–	1007	301	–	3EF

Man erkennt, daß der X-Memory-Bereich in drei sich nicht berührende Blöcke gespalten ist (Extended Functions und zwei X-Memory Module).

Im HP-41 C reicht der Hauptspeicher nur von Adresse 192 bis 255. Er wird durch die einzelnen Einsteck-Speicher um jeweils 64 oder durch das QUAD-RAM um 256 Register nach oben erweitert, bis die Grenze 511 erreicht ist.

Im folgenden werden die einzelnen Speicherbereiche beschrieben. Dazu gehören detaillierte Informationen über die Arten von Daten, die in diesen Bereichen gespeichert werden,sowie die Beschreibung ihrer Codierung.

Der Hauptspeicher

Im Hauptspeicher werden vier verschiedene Arten von Daten gespeichert. Diese sind:

- alphanumerische Daten,

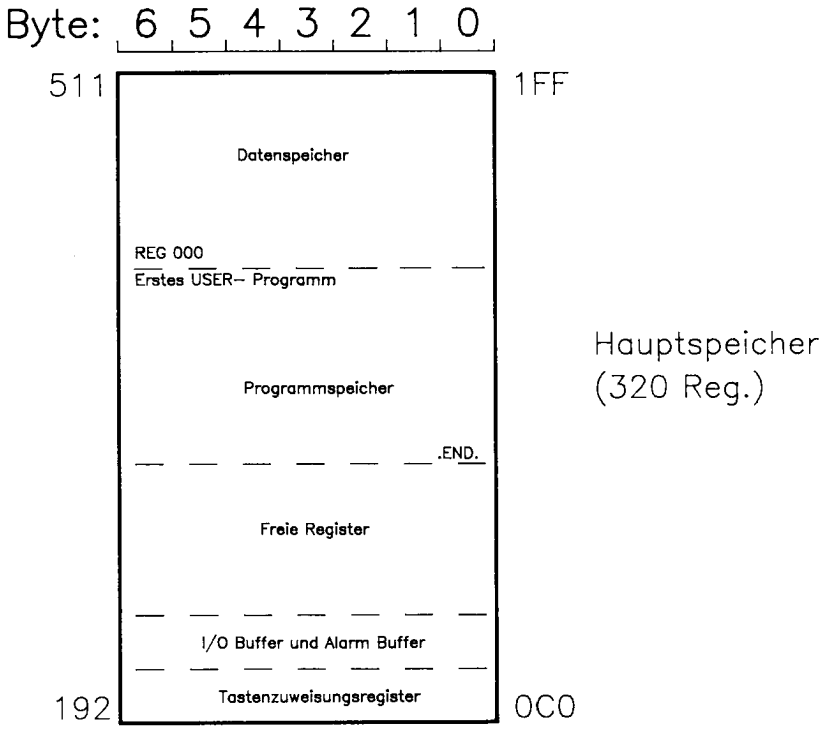
- Programme,

- Tastenzuordnungen (Key assignments=KA),

- I/O-Buffer.

Jedem Datentyp ist ein besonderer Teil des Speichers reserviert. Die Zuordnung dieser reservierten Bereiche geschieht durch das Betriebssystem, das dabei über die jeweilige Verteilung immer genau Protokoll führt. Ein solches System von variabler Speicherplatzzuordnung wird als **dynamische Speicherplatzverwaltung** bezeichnet.

In der Abbildung sind der Hauptspeicher und seine Aufteilung schematisch dargestellt.



Der Rechner verwaltet vier verschiedene Speicherbereiche:

- Datenspeicher,
- Programmspeicher,
- KA-Speicher,
- I/O-Buffer.

Ein fünfter Bereich wird als **freier Speicherbereich** bezeichnet: Das sind die Register zwischen Programmspeicher und I/O-Buffer, die weder dem einen noch dem anderen Bereich zugeordnet sind.

Datenregister (Aufbau)

Im oberen Bereich des Hauptspeichers werden die Datenregister angelegt. In ihnen speichern wir mit den Befehlen **STO** und **ASTO** alle numerischen und alphanumerischen Daten. Die dabei verwendeten Registernummern (von 0 bis SIZE – 1) werden wir in Zukunft als **relative Adressen** bezeichnen. Wie sie zustande kommen wird weiter unten erklärt.

Die obere Grenze der Datenregister liegt für HP-41 CV und CX bei der absoluten Adresse 511.Beim HP-41 C ist sie abhängig von der Anzahl der eingesteckten Memory-Module:

Anzahl Module	Bereich (Dezimal) von	bis	Bereich (Hexadezimal) von	bis	Bemerkungen
0	192	255	0C0	0FF	HP41C–Grundausstattung
1	256	319	100	13F	
2	320	383	140	17F	oder Quad–Ram
3	384	447	180	1BF	(im HP41CV und CX sind bereits alle Module eingebaut)
4	448	511	1C0	1FF	

Die untere Grenze wird vom Betriebssystem bei Ausführen des Befehls **SIZE** festgelegt; und zwar nach der Formel:

untere Grenze = obere Grenze + 1 – SIZE

Da in den meisten Fällen die obere Grenze gleich 511 ist, reduziert sich dann die Formel auf:

untere Grenze = 512 – SIZE

Diese untere Grenze wird auch als **Vorhang** bezeichnet, da sie die Grenze zum Programmspeicher bildet. Wir werden bei diesem Ausdruck bleiben, da er in Zukunft eine wichtige Rolle spielen wird:

Beim Durchzählen der Datenregister – das heißt bei der Zuordnung der **relativen Adressen** der Datenregister (das sind die, die bei Befehlen wie STO oder RCL als **Argument** angegeben werden) – beginnt man nämlich beim Vorhang mit der Zahl 0!
Den Zusammenhang zwischen absoluter und relativer Adresse zeigt folgende Tabelle:

absolute Adresse	**entspricht**	**relativer Adresse**
obere Grenze		SIZE – 1
Vorhang		0
Vorhang + nnn		Register nnn

Nun kann man sich leicht vorstellen, daß das Betriebssystem den **Vorhang** ständig benötigt, um **relative Adressen** in **absolute Adressen** und zurück umzuwandeln. Deshalb wird seine Adresse an einer bestimmten Stelle in den Statusregistern abgelegt. Der **Vorhang** ist nur eine *gedachte* Grenze zwischen Datenregistern und Programmspeicher!

Nachdem nun erklärt wurde, wo sich die Datenregister befinden und wie sie gezählt, numeriert und adressiert werden, wird im folgenden Abschnitt gezeigt, wie Daten in diesen Registern gespeichert sind.

Datenregister (innere Struktur)

Gleich zu Beginn dieses Abschnitts eine wichtige Regel:

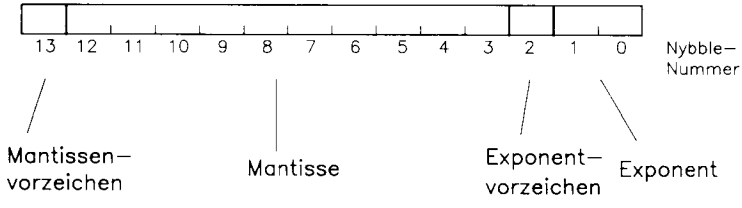
Jede Zahl oder Buchstabengruppe, die wir mittels STO oder ASTO in den Datenspeicher schreiben, belegt ein ganzes Register.

Wichtig: Wir sollten immer daran denken, daß die beschriebenen Registerteilfelder nur logische Gebilde sind und keine physikalisch existenten Grenzen darstellen.

Der Registeraufbau

Wir unterscheiden zwischen **numerischen Daten** – das sind Zahlen, die wir später zum Rechnen weiterverwenden wollen – und **alphanumerischen** oder **ASCII-Daten** – das sind Buchstabenketten.

Prinzipieller Aufbau eines Zahlenregisters

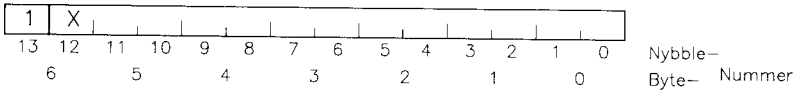


Beispiele für den Inhalt eines Zahlenregisters

9	8	7	5	1	2	0	0	0	0	0	0	0	0	≙ −8,7512
0	8	7	5	1	2	0	0	0	0	0	0	0	0	≙ 8,7512
0	1	0	0	0	0	0	0	0	0	0	0	1	2	≙ 1,0000 12
0	1	0	0	0	0	0	0	0	0	0	0	0	0	≙ 1,0000
9	8	7	5	1	2	0	0	0	0	0	9	8	5	≙ −8,7512 −15

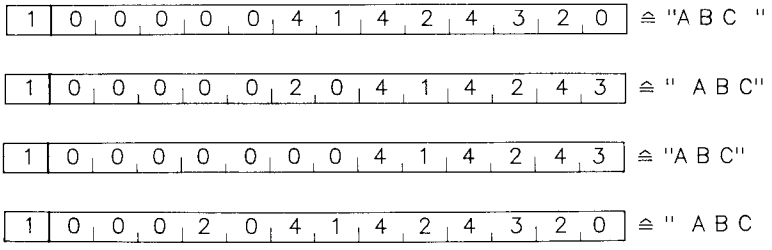
Aus den Abbildungen ist erkenntlich, daß für die beiden Vorzeichen jeweils ein Nybble (0 für ein positives und 9 für ein negatives Vorzeichen) und für jede Ziffer von Mantisse und Exponent auch ein Nybble reserviert ist. Diese Darstellung, bei der jedes Nybble eine Ziffer von 0 bis 9 enthält, nennt man BCD-Darstellung. (BCD bedeutet „Binary Coded Decimal".) Im Nybble 13 eines Zahlenregisters steht somit entweder eine 0 oder eine 9.

Prinzipieller Aufbau eines ASCII-Registers



Alle ASCII-Register haben die 1 im Nybble 13 gemeinsam. In den Bytes 5 bis 0 stehen bis zu 6 ASCII-Zeichen rechtsbündig. Wenn ein Register nur einen Buchstaben enthält, sind die Bytes 5 bis 1 Null-Bytes und Byte 0 enthält den Buchstaben.

Beispiele für den Inhalt eines ASCII-Registers



Es ist nun ein leichtes, Zahlen- von ASCII-Registern zu unterscheiden: Man muß nur das Nybble 13 analysieren. Ist es 1, so handelt es sich um ein ASCII-Register, ist es 0 oder 9, handelt es sich um eine Zahl. Das 12. Nybble kann bei älteren Betriebssystemen (Rechner bis zur Seriennummer 2035...) beliebige Werte haben, neue Betriebssysteme setzen dieses Nybble auf 0. Die Beispiele gehen also von einem neuen Betriebssystem aus, der Unterschied besteht aber nur im 12. Nybble.

(Das CCD-Modul enthält eine Funktion, die die Analyse eines Registerinhalts sehr vereinfacht: **DCD**, siehe Funktionsliste)

Programmspeicher (Aufbau)

Der Programmspeicher beginnt mit dem Register unterhalb des Vorhangs (siehe Hauptspeicherkarte). Er endet mit der **.END.**-Marke. Als erstes fällt dabei auf, daß der Programmspeicher von oben nach unten (von großen absoluten Adressen nach den kleinen hin) orientiert ist, während es bei den Datenregistern – wie wir im vorherigen Abschnitt gesehen haben – umgekehrt ist.

Die Bedeutung der END-Anweisung

Was ist eigentlich ein Programm im HP-41? Viele würden nun sagen: „Das beginnt mit einem ALPHA-Label und hört mit einer END-Anweisung auf!" Das ist falsch – zumindest teilweise. Wichtig an einem Programm ist nämlich lediglich die END-Anweisung. Der Programmanfang liegt immer hinter der END-Anweisung des vorherigen Programms oder unterhalb des Vorhangs. Dabei enthält jeder END-Befehl eine Information über die Länge des dazugehörigen Programms. (Das kürzeste Programm, das man schreiben kann, ist also nur ein END-Befehl.)

Wenn wir nun von einer „MEMORY LOST" Situation ausgehen (kein Programm im Speicher) und einige Programmschritte eingeben, so handelt es sich dabei bereits um ein Programm, egal ob wir ein END eingegeben haben oder nicht (Das ist leicht zu beweisen, denn nach Verlassen des Programm-Modus kann es mit den Befehlen RTN und R/S ausgeführt werden!). Das Betriebssystem unterhält nämlich einen permanenten END-Befehl: die **.END.**-Marke. Also hatten wir nach dem „MEMORY LOST" doch schon ein Programm im Speicher – lediglich aus dem **.END.**-Befehl bestehend.

Das **.END.** hat gegenüber einem normalen END eine Sonderfunktion: Es ist das letzte END im Programmspeicher und kann normalerweise nicht gelöscht werden.

Beim Schreiben oder Korrigieren eines Programms werden neue Befehle in den Programmspeicher geschrieben, indem eventuell vorhandene Null-Bytes überschrieben werden. Wenn sich an dieser Stelle keine Null-Bytes befinden, so werden welche erzeugt. Dies geschieht, indem der Teil des Programmspeichers unterhalb der augenblicklichen Adresse (also der Teil bis zum **.END.**) um ein Register in den freien Speicherbereich hinein nach unten geschoben wird. Dabei entstehen sieben Null-Bytes. Die Längeninformation im END-Befehl des bearbeiteten Programms wird dabei erneuert.

Beim „PACKING" werden überflüssige Null-Bytes aus dem Programmspeicher entfernt. Auch bei diesem Vorgang werden die Längeninformationen der END-Anweisungen erneuert.

Man erkennt nun, daß es sich beim Programmspeicher um ein außerordentlich dynamisches Gebilde handelt. Es verändert nicht nur seine Größe den Ansprüchen entsprechend, sondern wird auch als Gesamtes im Hauptspeicher nach oben oder unten verschoben (beim Ausführen von SIZE).

Programmspeicher (innere Struktur)

Die Abspeicherung von Programmbefehlen

Das Betriebssystem des HP-41 beinhaltet eine Liste (CAT 3) aller Befehle. Außer dem Namen ist für jeden Befehl intern ein Code aufgeführt. Tippen wir nun einen Befehl auf der Tastatur ein (mittels XEQ ALPHA *Befehl* ALPHA), so durchsucht das Betriebssystem diese Liste. Wird der Befehl gefunden, so merkt sich der Rechner lediglich den entsprechenden Code und verzweigt eventuell (z. B. bei dem Befehl STO) in die Abfrage nach dem **Argument** (bei STO wäre das die Registeradresse). Befehlscode und Argument können dann als Programmschritt abgespeichert werden.

Das Besondere am HP-41 ist nun, daß die Befehlsbibliothek erweitert werden kann; z. B. durch Einsteckmodule wie das CCD-Modul. Eine weitere Ergänzung ist durch die Vergabe von Namen für die vom Anwender geschriebenen Programme möglich (LBL-Befehl).

Die obengenannte Bibliothek kann auch gelistet werden. Das geschieht mit der **CAT**-Instruktion.

Sucht der Rechner nun ein Programm oder eine Funktion, dann ist die Reihenfolge, in der diese Bibliothek durchsucht wird, folgende:

- Anwenderprogramme in umgekehrter Reihenfolge, in der sie im Katalog erscheinen
- Befehle aus Einsteckmodulen, beginnend bei Block 5 (TIME-Modul)
- interne Befehlsliste

Da die Anwenderprogramme eine Ausnahme darstellen, werden sie weiter unten gesondert behandelt.

Nach der Befehlseingabe erfolgt zuerst eine Rechtschreibeprüfung (auch als **Syntax-Check** bezeichnet) . Dies macht es uns normalerweise unmöglich, z.B. die Befehle STO ! M und TONE 57 einzugeben. Im zweiten Schritt erfolgt die Abspeicherung von Befehls- und eventuell Argumentcodes. Dieser Teil des Betriebssystems ist der **Eingabeinterpreter**.

Beim Abarbeiten eines Programms liest das Betriebssystem nun Programmzeile für Programmzeile und interpretiert dabei jeweils den abgespeicherten Code, indem es zu dem Maschinenspracheprogramm (für jeden Befehl gibt es ein Programm) verzweigt. Dieser Teil des Betriebssystems ist der **Interpreter**.

Der Interpreter arbeitet, ohne zu wissen, wie der Programmcode zustande gekommen ist.

Diese Erkenntnis ist von grundlegender Bedeutung für alle Freunde der sogenannten „synthetischen Programmierung". Einfach ausgedrückt ist das nämlich nichts anderes, als das Erstellen eines Programmcodes unter Umgehung des Eingabeinterpreters. Insbesondere erzeugt man dabei neue („synthetische") Befehle, deren direkte Eingabe durch den Syntax Check unmöglich ist! Der Interpreter akzeptiert diese synthetischen Programmzeilen jedoch und führt sie auch korrekt aus.

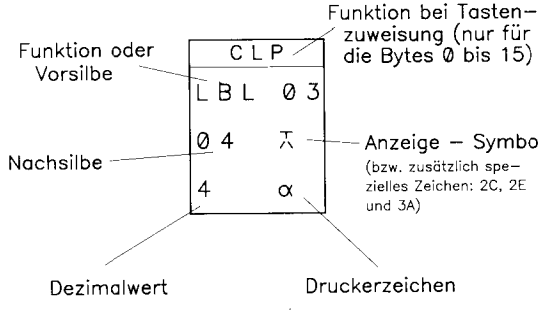
Bisher wurde beschrieben, wie ein Befehl als Programmschritt in den Speicher gelangt: nämlich auf dem Weg über den Eingabeinterpreter. Dabei wurde auch gesagt, daß ein Programmcode erstellt wird. Wie dieser Code aussieht, wird in den folgenden Abschnitten erklärt:

Der Programmcode des HP-41

Der Programmcode des HP-41 ist Byte-orientiert. Das heißt, der Interpreter arbeitet sich Byte für Byte durch den Programmspeicher. Dabei geht er von oben nach unten vor, oder anders ausgedrückt, von hohen Adressen zu den kleineren hin.

Mit einem einzigen Byte können bis zu 256 verschiedene Befehle codiert werden. Aber allein die drei Anweisungen STO, RCL und ASTO in Verbindung mit den Registernummern 0 bis 99 ergeben weit mehr Möglichkeiten – nämlich 300. Dieses Problem wird im HP-41 umgangen, indem bei manchen Befehlen dem ersten Byte ein zweites – und manchmal sogar ein drittes oder noch mehr – angehängt wird, wobei das nachfolgende Byte eine Ergänzung zum ersten darstellt; z.B. als Argument für den STO-Befehl. Entsprechend der Anzahl benötigter Bytes unterscheidet man **Ein-**, **Zwei-**, **Drei-** und **Mehr-Byte-Befehle**. Bei allen gibt das erste Byte den Befehlstyp und die Instruktion selbst an. Zusätzliche Bytes präzisieren dann die Instruktion.

In der nachstehenden Byte-Tafel sind alle aufgeführt. Die Tafel ist als 16*16 Raster ausgelegt. Das ist darauf zurückzuführen, daß ein Byte in zwei Nybbles zerlegt werden kann; dem ersten Nybble entspricht in der Tafel die Reihennummer, dem zweiten die Spaltennummer. Wir werden sehen, daß Befehle, die in einer Reihe liegen, immer gleichen Typs (Ein-, Zwei-, Drei- oder Mehr-Byte-Befehle) sind.
Auf den nächsten beiden Seiten finden Sie die **Hex-Byte-Tafel** für den HP-41. Jedes Kästchen dieser Tafel gliedert sich wie folgt:



HP-41C QUICK REFERENCE CARD FOR SYNTHETIC PROGRAMMING © 1982, SYNTHETIX

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
	CAT	@c (GTO..)	DEL	COPY	CLP	R/S	SIZE	BST	SST	ON	PACK	←(PRGM)	USR/P/A	2 __	SHIFT	ASN	
0	NULL 00 0 ◆	LBL 00 01 1 ×	LBL 01 02 2	LBL 02 03 3 ←	LBL 03 04 4 α	LBL 04 05 5 β	LBL 05 06 6 Γ	LBL 06 07 7 ↓	LBL 07 08 8 Δ	LBL 08 09 9 σ	LBL 09 10 10 ◆	LBL 10 11 11 λ	LBL 11 12 12 µ	LBL 12 13 13 ∡	LBL 13 14 14 τ	LBL 14 15 15 Φ	0
1	0 16 16 Θ	1 17 17 Ω	2 18 18 δ	3 19 19 Å	4 20 20 å	5 21 21 Ä	6 22 22 ä	7 23 23 Ö	8 24 24 ö	9 25 25 Ü	. 26 26 ü	EEX 27 27 Æ	NEG 28 28 œ	GTO T 29 29 ≠	XEQ T 30 30 £	W T 31 31	1
2	RCL 00 32 32	RCL 01 33 33 !	RCL 02 34 34 "	RCL 03 35 35 #	RCL 04 36 36 $	RCL 05 37 37 %	RCL 06 38 38 &	RCL 07 39 39 '	RCL 08 40 40 (	RCL 09 41 41)	RCL 10 42 42 *	RCL 11 43 43 +	RCL 12 44 44 ,	RCL 13 45 45 –	RCL 14 46 46 .	RCL 15 47 47 /	2
3	STO 00 48 48 0	STO 01 49 49 1	STO 02 50 50 2	STO 03 51 51 3	STO 04 52 52 4	STO 05 53 53 5	STO 06 54 54 6	STO 07 55 55 7	STO 08 56 56 8	STO 09 57 57 9	STO 10 58 58 :	STO 11 59 59 ;	STO 12 60 60 <	STO 13 61 61 =	STO 14 62 62 >	STO 15 63 63 ?	3
4	+ 64 64 @	– 65 65 A	* 66 66 B	/ 67 67 C	X<Y? 68 68 D	X>Y? 69 69 E	X≤Y? 70 70 F	Σ+ 71 71 G	Σ– 72 72 H	HMS+ 73 73 I	HMS– 74 74 J	MOD 75 75 K	% 76 76 L	%CH 77 77 M	P→R 78 78 N	R→P 79 79 O	4
5	LN 80 80 P	X↑2 81 81 Q	SQRT 82 82. R	Y↑X 83 83 S	CHS 84 84 T	E↑X 85 85 U	LOG 86 86 V	10↑X 87 87 W	E↑X-1 88 88 X	SIN 89 89 Y	COS 90 90 Z	TAN 91 91 [	ASIN 92 92 \	ACOS 93 93]	ATAN 94 94 ↑	→DEC 95 95 _	5
6	1/X 96 96 `	ABS 97 97 a	FACT 98 98 b	X≠0? 99 99 c	X>0? 100 100 d	LN1+X 101 101 e	X<0? A 102 f	X=0? B 103 g	INT C 104 h	FRC D 105 i	D→R E 106 j	R→D F 107 k	→HMS G 108 l	→HR H 109 m	RND I 110 n	→OCT J 111 o	6
7	CLΣ T 112 p	X<>Y Z 113 q	PI Y 114 r	CLST X 115 s	R↑ L 116 t	RDN M [117 u	LASTX N \ 118 v	CLX O] 119 w	X=Y? P ↑ 120 x	X≠Y? Q _ 121 y	SIGN ⊢ T 122 z	X≤0? a 123 π	MEAN b 124 |	SDEV c 125 →	AVIEW d 126 Σ	CLD e 127 ⊢	7
	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111	
	00 01 02 03	04 05 06 07	08 09 10 11	12 13 14 15	16 17 18 19	20 21 22 23	24 25 26 27	28 29 30 31	32 33 34 35	36 37 38 39	40 41 42 43	44 45 46 47	48 49 50 51	52 53 54 55	← bit numbers in a 7-byte register		

FLAGS (Register d)

- 00-10 general purpose
- 11 auto execute
- 12 doublewide
- 13 lower case
- 14 overwrite
- 15-16 IL printer
  - 0 0 MAN
  - 0 1 NORM
  - 1 0 TRACE
  - 1 1 TR/STACK
- 17 record incomplete
- 18, 19, 20] general use cleared at turn-on
- 21 prtr enable
- 22 num. entry
- 23 alpha entry
- 24 range ignore
- 25 error ignore
- 26 audio enable
- 27 USER mode
- 28 dec./comma
- 29 digit grouping
- 30 CAT
- 31 timer DMY/MDY
- 32 manual IL I/O
- 33 IL absolute manual
- 34 not used
- 35 not used
- 36-39 number of digits
- 40-41 display
  - 0 0 SCI
  - 0 1 ENG
  - 1 0 FIX
  - 1 1 FIX/ENG
- 42-43 trig mode
  - 0 0 DEG
  - 0 1 RAD
  - 1 0 GRAD
  - 1 1 RAD
- 44 cont. ON
- 45 system data entry
- 46 partial key sequence
- 47 SHIFT
- 48 ALPHA
- 49 low BAT
- 50 message
- 51 SST
- 52 PGRM
- 53 I/O
- 54 PSE
- 55 printer existence

HP-41C QUICK REFERENCE CARD FOR SYNTHETIC PROGRAMMING

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F	
8	DEG IND 00 128 ◆	RAD IND 01 129 ×	GRAD IND 02 130 ÷	ENTER↑ IND 03 131 ←	STOP IND 04 132 α	RTN IND 05 133 β	BEEP IND 06 134 Γ	CLA IND 07 135 ↓	ASHF IND 08 136 Δ	PSE IND 09 137 σ	CLRG IND 10 138 ◆	AOFF IND 11 139 ↑	AON IND 12 140 μ	OFF IND 13 141 Δ	PROMPT IND 14 142 π	ADV IND 15 143 ‡	8
9	RCL IND 16 144 θ	STO IND 17 145 Ω	ST+ IND 18 146 δ	ST− IND 19 147 ∞	ST × IND 20 148 ∂	ST/ IND 21 149 ∩	ISG IND 22 150 ∂	DSE IND 23 151 ∪	VIEW IND 24 152 ∂	Σ REG IND 25 153 ∩	ASTO IND 26 154 ∪	ARCL IND 27 155 ≠	FIX IND 28 156 ≤	SCI IND 29 157 ≥	ENG IND 30 158 £	TONE IND 31 159 ▒	9
A	XR 0-3 IND 32 160	XR 4-7 IND 33 161 !	XR8-11 IND 34 162 "	X12-15 IND 35 163 #	X16-19 IND 36 164 $	X20-23 IND 37 165 %	X24-27 IND 38 166 &	X28-31 IND 39 167 '	SF IND 40 168 (	CF IND 41 169)	FS?C IND 42 170 *	FC?C IND 43 171 +	FS? IND 44 172 ,	FC? IND 45 173 −	GTO XEQ IND IND 46 174 .	SPARE IND 47 175 /	A
B	SPARE IND 48 176 0	GTO 00 IND 49 177 1	GTO 01 IND 50 178 2	GTO 02 IND 51 179 3	GTO 03 IND 52 180 4	GTO 04 IND 53 181 5	GTO 05 IND 54 182 6	GTO 06 IND 55 183 7	GTO 07 IND 56 184 8	GTO 08 IND 57 185 9	GTO 09 IND 58 186 :	GTO 10 IND 59 187 ;	GTO 11 IND 60 188 <	GTO 12 IND 61 189 =	GTO 13 IND 62 190 >	GTO 14 IND 63 191 ?	B
C	GLOBAL IND 64 192 @	GLOBAL IND 65 193 A	GLOBAL IND 66 194 B	GLOBAL IND 67 195 C	GLOBAL IND 68 196 D	GLOBAL IND 69 197 E	GLOBAL IND 70 198 F	GLOBAL IND 71 199 G	GLOBAL IND 72 200 H	GLOBAL IND 73 201 I	GLOBAL IND 74 202 J	GLOBAL IND 75 203 K	GLOBAL IND 76 204 L	GLOBAL IND 77 205 M	X<>-- IND 78 206 N	LBL -- IND 79 207 O	C
D	GTO -- IND 80 208 P	GTO -- IND 81 209 Q	GTO -- IND 82 210 R	GTO -- IND 83 211 S	GTO -- IND 84 212 T	GTO -- IND 85 213 U	GTO -- IND 86 214 V	GTO -- IND 87 215 W	GTO -- IND 88 216 X	GTO -- IND 89 217 Y	GTO -- IND 90 218 Z	GTO -- IND 91 219 [	GTO -- IND 92 220 \	GTO -- IND 93 221]	GTO -- IND 94 222 ↑	GTO -- IND 95 223 _	D
E	XEQ -- IND 96 224 `	XEQ -- IND 97 225 a	XEQ -- IND 98 226 b	XEQ -- IND 99 227 c	XEQ -- IND100 228 d	XEQ -- IND101 229 e	XEQ -- IND102 230 f	XEQ -- IND103 231 g	XEQ -- IND104 232 h	XEQ -- IND105 233 i	XEQ -- IND106 234 j	XEQ -- IND107 235 k	XEQ -- IND108 236 l	XEQ -- IND109 237 m	XEQ -- IND110 238 n	XEQ -- IND111 239 o	E
F	TEXT 0 IND T 240 p	TEXT 1 IND Z 241 q	TEXT 2 IND Y 242 r	TEXT 3 IND X 243 s	TEXT 4 IND L 244 t	TEXT 5 INDM L 245 u	TEXT 6 IND N \ 246 v	TEXT 7 IND O] 247 w	TEXT 8 IND P ↑ 248 x	TEXT 9 IND Q _ 249 y	TEXT10 IND ⊢ 250 z	TEXT11 IND a 251 π	TEXT12 IND b 252 ↓	TEXT13 IND c 253 →	TEXT14 IND d 254 Σ	TEXT15 IND e 255 ⊢	F
	0 0000	1 0001	2 0010	3 0011	4 0100	5 0101	6 0110	7 0111	8 1000	9 1001	A 1010	B 1011	C 1100	D 1101	E 1110	F 1111	

For price information and a list of dealers in your area, send a self-addressed stamped envelope to SYNTHETIX, 1540 Mathews Ave., Manhattan Beach, CA 90266, USA

Structure of multi-byte instructions

Two-byte instructions

STO 16=145,16 , DSE IND 55 =151,183
LBL e =207,127 FS?C IND Y =170,242
RCL b =144,124 TONE 89 =159,89
X<>M =206,117 ST+ IND N =146,246
LBL Q =207,121 VIEW H(109)=152,109

Two-byte special cases

GTO IND=174,reg. XEQ IND=174,128+r
GTO IND 09=174,9 XEQ IND X=174,243
XROM i,j =160+i/4,64(i mod 4)+j
WSTS =XROM 30,10 =167,138
short form GTO =177+label,0
GTO 12 =189,0

Three-byte instructions

long form GTO =208,0,label
GTO 32 =208,0,32
XEQ =224,0,label
XEQ D =224,0,105
END =192,0,9+ sum of status indicators
32(.END.), 4(rePACK), 2(decompile)

Variable length instructions

TEXT =240+n, n character bytes
Append symbol counts as first char.
"&" =241,38 "⊢)? =243,127,41,63
GTO " =29,240+n, n character bytes
GTO "XYZ =29,243,88,89,90
XEQ " =30,240+n, n character bytes
XEQ "A =30,241,65 (synthetic)
LBL " =192,0,241+n, (key), n chars.
LBL " =192,0,242,0,58 (synthetic)

Ein-Byte-Befehle

Bis auf wenige Ausnahmen sind alle Befehle der Reihen 0 und 2-8 Ein-Byte-Befehle.

Register-Befehle

Dies sind die Befehle ARCL, ASTO, DSE, ISG, RCL, ΣREG, ST+, ST-, ST*, ST/ und Xα. Allen gemein ist , daß sie die Angabe einer Registerzahl erwarten. Der Code dieser Instruktionen besteht aus zwei Bytes. Das erste Byte gibt die durchzuführende Operation an, das zweite die Registernummer (**hexadezimal codiert**). Dabei entsprechen die Codes hex 00 bis hex 63 den Registernummern 0 bis 99 und die Codes hex 70 bis hex 74 den Stapelregistern T, Z, Y, X und L.

So wird z.B. der Befehl RCL 87 folgendermaßen codiert: hex 90 57; das erste Byte (Befehlsbyte) steht für RCL, das zweite (Argumentbyte) für 87.

Was passiert nun, wenn RCL IND 87 im Programmspeicher steht? Ganz einfach: Wenn ein Register als Zeiger dient, wird zum Argumentbyte hex 80 (entspricht dezimal 128) addiert. Hier also 57 + 80 = D7. In der Byte-Tafel sind die verschiedenen Argumente und ihre Hex-Äquivalente dargestellt.

Bei den RCL- und STO-Befehlen gibt es eine Besonderheit: Da diese Instruktionen in Verbindung mit den Registernummern 00 bis 15 die am häufigsten auftretenden Registerbefehle sind, hat man hierfür die Ein-Byte-Befehle der Reihen 2 und 3 der Befehlsbyte-Tabelle reserviert. Dies spart nicht nur Speicherplatz, sondern verringert auch die Ausführungszeit der Instruktionen.

Flag- und Anzeigeformat-Befehle

Dazu gehören die Befehle CF, FC?, FC?C, FS?, FS?C, SF, ENG, FIX und SCI. Sie werden wie die Register-Befehle zu einem Befehls- und Argumentbyte codiert.

Register-, Flag- und Anzeigeformat-Befehle besitzen eine gemeinsame Bit-Struktur, die in der Form

bbbb bbbb iaaa aaaa dargestellt werden kann.

Jeder Buchstabe steht für ein **Bit** des Codes. Die ersten acht Bits (das erste Byte) entsprechen dem Befehlsbyte. Das *i* bedeutet **indirekt**, und die sieben *a*'s entsprechen dem Argument.

Label-Befehle

Hier müssen wir zwischen numerischen und Alpha-Labels unterscheiden. Erstere werden wie Registerbefehle codiert, nur gibt es hier keine indirekten Argumente. Die häufig benötigten Labels 0 bis 14 werden auch hier als Ein-Byte-Befehle codiert.

Die allgemeine Bit-Struktur eines Alpha-Labels ist:

1100 *bbbr rrrrrrrr* 1111 *nnnn kkkkkkkk tttt tttt ...*

Im Feld *bbbrrrrrrrrr* wird die Entfernung zum davorliegenden Label gespeichert: *bbb* gibt die Bytedifferenz und *rrrrrrrr* die Registerdifferenz an. Das bedeutet:

Alle Alpha-Labels und END-Marken sind miteinander wie die Glieder einer Kette verbunden. Man spricht deshalb auch von der **Globalkette**. Die Verbindung zwischen den einzelnen Gliedern besteht darin, daß in jedem **Global-Befehl** (damit sind sowohl die Alpha-Labels als auch die END-Befehle gemeint) die Entfernung zum davorliegenden Global-Befehl gespeichert wird. (Damit ist die Aussage, die im Abschnitt „Programmspeicher (Auf-

bau)" gemacht wurde, nur dann richtig, wenn ein Programm keine Alpha-Labels enthält!) Das Betriebssystem verwaltet diese „**Links**" (= Verbindungen) , indem es sie beim Schreiben oder Ändern eines Programms stets auf dem neuesten Stand hält. Beim Suchen eines Programms braucht der Rechner nur die Globalkette zu durchsuchen – und zwar beginnend bei der **.END.**-Marke. Auch hier zeigt sich wieder die Wichtigkeit dieser Anweisung. Das Ende der Kette ist erreicht, wenn ein Global-Befehl mit dem „Link" 0 gefunden ist.

Da das Betriebssystem sich darauf verläßt, daß alle „Links" korrekt sind, hat eine Veränderung dieser Information verheerende Folgen.

Wie sieht der „Link" im Detail aus? Bildet man die Differenz zwischen den Adressen der ersten Bytes zweier benachbarter Global-Befehle und drückt diese in Bytes und Registern aus (so entspricht z.B. eine Distanz von 11Bytes einem Register und 4 Bytes), so erhält man die Information , die im *bbbr rrrr rrrr*-Feld abgespeichert wird. In unserem Beispiel (4 Bytes und 1 Register) wäre das bin 1000 0000 0001 oder hex 801.

Auf das Link-Feld folgen die Bits 1111*nnnn*. Die vier Einsen (hex F) kündigen eine Buchstabenkette an (besser gesagt, das Byte hex F*n*; siehe auch unter **Text-Befehle**!). Hier bedeutet das, daß es sich um ein Alpha-Label (und nicht um einen END-Befehl) handelt. Das *nnnn*-Feld gibt die Anzahl Buchstaben der Byte-Kette an, die auf dieses dritte Byte folgen. Der erste Buchstabe dieser Kette ist für eine eventuelle Tastenzuordnung dieses Labels reserviert: in ihm wird der Tastencode abgespeichert. Die folgenden Bytes sind die ASCII-Codes der Buchstaben des Namens. Die Programmzeile LBL ABCD ergibt somit:

hex C0 00 F5 00 41 42 43 44

(Distanz- und Tastencode-Feld sind hier auf 0 gesetzt)

END-Befehle

Die allgemeine Bit-Struktur eines **END**-Befehls ist:

1100 *bbbr rrrr rrrr* 00*e*0 *xpdx*

Wie bereits im Abschnitt „Label-Befehle" beschrieben, sind die vier ersten Nybbles dieses Befehls mit denen eines Alpha-Labels identisch. Die nächsten acht Bits haben hier jedoch eine ganz andere Bedeutung:

e-Bit: Dieses Bit ist normalerweise gleich 0. Nur die **.END.**-Marke hat dieses Bit gesetzt.

p-Bit: In einem neu geschriebenen Programm wird dieses Bit im **END**-Befehl gesetzt um anzugeben, ob das Programm gepackt werden darf. Ein Programm packen bedeutet, alle Null-Bytes daraus entfernen.

d-Bit: Dieses Bit wird vom Betriebssystem benötigt um festzuhalten, ob ein Programm verändert wurde.

GTO- und XEQ-Befehle

Auch hier muß zwischen numerischen und Alpha-Argumenten unterschieden werden.

Alpha-**GTO**s und -**XEQ**s haben eine einfache Struktur:

GTO: bin 0001 1101 1111 *nnnn*

XEQ: bin 0001 1110 1111 *nnnn*

Das erste Byte gibt an, ob es sich um einen GTO (hex 1D) oder XEQ-Befehl (hex 1E) handelt. Das zweite Byte (hex F*n*) gibt – ähnlich wie beim Alpha-LBL – an, daß eine ASCII-Zeichenkette folgt. Diese *n* Buchstaben geben den Namen des Alpha-LBL an, zu dem gesprungen werden soll. Der Befehl enthält **keine** Information über die Entfernung zu dem Alpha-LBL und natürlich auch nicht über eine eventuelle Tastenzuordnung desselben!

Wird beim Ablauf eines Programms ein Alpha-GTO oder -XEQ ausgeführt, so durchsucht das Betriebssystem zuerst die **Globalkette**, beginnend beim **.END.**, nach oben bis zum ersten Label. Falls das entsprechende Alpha-Label nicht gefunden wird, durchsucht das Betriebssystem anschließend noch die Befehlsverzeichnisse **CAT 2** und **CAT 3.**

Nun zu den Befehlen GTO und XEQ mit numerischen Argumenten. Ihre Struktur ist:

Zwei-Byte GTO: bin 1011 *n'n'n'n' dbbb rrrr*

Drei-Byte GTO: bin 1101 *bbbr rrrr rrrr dnnn nnnn*

XEQ: bin 1110 *bbbr rrrr rrrr dnnn nnnn.*

Sicher erkennen Sie einige der Felder wieder. Das *nnnn*-Feld der beiden Drei-Byte-Befehle enthält die Nummer des Labels, zu dem die Programmausführung verzweigt werden soll. Beim Zwei-Byte-GTO entspricht das dem *n'n'n'n'*-Feld, nur steht hier die um den Wert 1 erhöhte Label-Nummer.

Beim ersten Ausführen eines dieser Befehle, sind die *b*-, *d*- und *r*-Bits auf Null gesetzt. Der Rechner geht auf die Suche nach dem entsprechenden numerischen Label. Sobald dieses gefunden ist, wird die Entfernung vom Sprungbefehl bis zum Label errechnet und im Sprungbefehl abgespeichert. Die Programmausführung geht beim Label weiter. Wenn der Sprungbefehl das nächste Mal ausgeführt wird, erkennt das Betriebssystem, daß die Sprungweite bereits bekannt ist und braucht deshalb das Label nicht nochmals zu suchen, es verzweigt direkt an die gewünschte Stelle. **Dabei prüft es nicht erneut, ob das angesprochene Label das richtige ist, oder ob dort überhaupt ein Label steht!**

Die Sprungweite wird von dem Byte, das den ersten Teil des Sprungweitencodes enthält (bei Zwei-Byte-GTOs das 2. Byte, bei Drei-Byte-Befehlen das 1. Byte), bis zu dem Byte, das dem angesprungenen Label unmittelbar vorangeht, gemessen. Das *d*-Bit gibt die Sprungrichtung an: $d = 0$, wenn nach unten (im Programm nach höheren Programmzeilen) gesprungen wird, und $d = 1$, wenn rückwärts gesprungen wird.

Wie bei den RCL-, STO- und LBL-Befehlen gibt es auch bei den GTO-Befehlen eine Kurzform: das Zwei-Byte-GTO. Einerseits hilft das, Programmspeicherplatz sparen, andererseits hat es aber hier eine weitere – leider unangenehme – Auswirkung: Für die Sprungweitenangabe ist hier weniger Platz vorhanden als im Drei-Byte-GTO, nämlich nur 4 Bits. Das entspricht einem maximalen Sprung von $15 * 7 + 6 = 111$ Bytes (*bbb rrrr* = 111 1111). Dies trifft für die Labels 0 bis 14 zu.

Text-Befehle

Alle Text-Befehle beginnen mit einem Byte der Form bin 1111 *nnnn*. Das *nnnn*-Feld gibt an, wieviele Buchstaben der Text-Befehl enthält – das Maximum ist bei 15 Buchstaben erreicht.

Wie werden Buchstaben abgespeichert? Entsprechend einer amerikanischen Norm wird für jeden Buchstaben ein Byte benutzt. Diese Norm ist der *American Standard Code for Information Interchange*. Er wird meistens knapp mit **ASCII** bezeichnet.

Mit dem CCD-Modul Kleinbuchstabenmodus ist es möglich, jeden beliebigen Byte-Wert einzugeben.

Das Byte hex 7F hat entsprechend seiner Position in der Zeichenkette eine unterschiedliche Bedeutung: Folgt es direkt auf das Text-Byte (hex F*n*), so wird es nicht als Buchstabe interpretiert, sondern bedeutet, daß die folgenden Buchstaben ans Alpha-Register anzuhängen sind, ohne es vorher zu löschen (**APPEND**-Befehl). An jeder anderen Stelle in der Zeichenkette wird dieses Byte als „liegendes T" interpretiert.

Zahlen-Eingabe

Wird eine Zahl als Programmzeile eingegeben, so wird für jede gedrückte Zifferntaste ein Byte in den Programmspeicher geschrieben. Jedes dieser Bytes stellt eigentlich einen Ein-Byte-Befehl dar, der den manuellen Tastendruck simuliert:

hex-Code	entspricht	Zifferntaste
10		0
11		1
12		2
13		3
.		.
.		.
.		.
19		9
1A		Dezimalkomma
1B		EEX-Taste
1C		Vorzeichen-Taste

So wie eine manuelle Zahleneingabe durch das Drücken der ENTER-Taste oder durch die Ausführung irgendeiner anderen Funktion abgeschlossen wird, so erkennt auch das Betriebssystem beim Ausführen eines Programms eine Zahleneingabe als abgeschlossen an, sobald ein Byte gelesen wird, das verschieden ist von den Werten hex 10 bis 1C. Das gilt insbesondere auch für das Null-Byte.

Befehle aus Einsteckmodulen

Die Möglichkeit, den Befehlsumfang des HP-41 durch Einsteckmodule zu erweitern, stellt eine der großen Stärken dieses Rechnersystems dar. Wie werden nun solche zusätzlichen Befehle codiert? Auch hier müssen wir etwas weiter ausholen und das Verhalten des Betriebssystems genauer erklären:

Wie schon anfangs gesagt, hat der HP-41 außer dem RAM-Speicher (für die Daten) auch noch einen Festwertspeicher (ROM) für das Betriebssystem. Die Einsteckmodule sind auch solche ROMs. Wie das RAM, so hat auch das ROM einen vordefinierten Adressenbereich, der in 16 gleich große Abschnitte (4k-Blöcke) eingeteilt ist. Die 8 ersten Abschnitte werden vom Betriebssystem selbst und einigen bestimmten Modulen belegt: TIME-, Drukker- und HP-IL-Modul. Die anderen 8 Abschnitte sind den Einsteckplätzen zugeordnet: jedem Einsteckplatz zwei Abschnitte (max. 8 kByte). Ein Modul belegt somit immer genau einen oder zwei dieser 4k-Blöcke. Ist ein Modul in den Rechner eingesteckt, so vermag die Zentraleinheit aus den entsprechenden Adressen Instruktionen zu lesen – sonst erscheint der ROM-Bereich leer.

Die ersten beiden Bytes innerhalb eines solchen Einsteck-ROMs haben eine besondere Bedeutung: Sie geben die Modul-Kennziffer – auch als ROM-ID bezeichnet – und die Anzahl Funktionen innerhalb dieses Moduls an. Die darauffolgenden Bytes bilden die Funktionsliste dieses Einsteckmoduls und anschließend den Funktionscode. Jede Funktion wird somit durch zwei Zahlen genau charakterisiert:

– durch die Kennziffer des Moduls, in dem sich diese Funktion befindet, und

– durch seinen Platz in der Modul-Liste: die Funktionsnummer.

Diese beiden Zahlen sind uns allen wohl bekannt: Sie tauchen im **XROM**-Befehl auf, den wir anstelle einer ROM-Funktion im Programmspeicher sehen, wenn wir vorher das entsprechende Modul aus dem Rechner entfernt haben.

Da der Rechner diese beiden Zahlen anzeigen kann, ohne daß das Modul eingesteckt ist, können wir davon ausgehen, daß sie in dem Code für die entsprechende Funktion enthalten sein müssen. Außerdem wissen wir, daß diese Zahlen die Funktion ausreichend charakterisieren und erwarten somit keine weitere Information in dem Befehl. Hier nun seine Struktur:

bin 1010 0*iii iiff ffff*.

Das *i*-Feld gibt die ROM-ID an, und das *f*-Feld die Funktionsnummer. Die ROM-ID und Nummern der Funktionen sind in den Handbüchern der Module aufgeführt. Hier ein Beispiel:

Die Funktion RNDM aus dem CCD-Modul hat die Nummer 4, das Modul die ID 9. Daraus ergibt sich

iiiii = 01001,
ffffff = 000100

XROM 09,04 = bin 1010 0010 0100 0100 = hex A2 44.

Diese Codierung erlaubt 31 verschiedene ROM-IDs (Die ID = 0 ist nicht erlaubt!) und 64 Funktionen pro ROM.

Zusammenfassung

Viele Befehle verlangen außer dem Befehlscode im engeren Sinne auch eine zusätzliche Information über eine Registeradresse, ein numerisches Label oder sonstiges. Mit dem CCD-Modul ist es möglich, die einzelnen Befehlscodes anhand der Beschreibung selbst zu erstellen, wobei man einfach die Parameterfelder entsprechend füllt. Dennoch dürfte eine Vielzahl von Fragen offen geblieben sein. Einige werden wir versuchen zu beantworten, andere kann man selbst durch Ausprobieren klären! Das CCD-Modul enthält die nötigen Funktionen, um das alles leicht durchzuführen.

Der Programmzeilenanfang

Beim Erläutern der Struktur eines jeden Befehls wurde nie darauf hingewiesen, wie der Anfang einer Programmzeile kenntlich gemacht wird. Der Grund dafür ist, daß es eine solche Kennung nicht gibt. Ein Programm, dessen Adresse durch die Globalkette festgelegt ist, wird immer von der ersten Zeile an interpretiert. Von dort aus muß das Betriebssystem Zeile für Zeile feststellen, wo ein Befehl beginnt und wieviele Bytes er benötigt. Die in den GTOs und XEQs gespeicherten Sprungweiten unterstützen natürlich diese Bestimmung während einer Programmverzweigung. Das bedeutet, daß jedes Byte von vorneherein als erstes oder weiteres Byte eines Mehr-Byte-Befehls angesehen werden muß. Wenn wir uns im Programmspeicher vorwärts bewegen, stellt das kein Problem dar. Wenn wir aber rückwärts gehen (z.B. mit BST), muß das Betriebssystem jedesmal – vom Anfang des Programms an – den Beginn der Zeile feststellen, die der augenblicklichen vorangeht.

Befehle aus Einsteckmodulen (Parameter)

Weshalb haben programmierbare Befehle aus Einsteckmodulen keine Parameter, wie z.B. die STO-Instruktion? Wenn das Betriebssystem einen XROM-Befehl erkennt, geht es davon aus, daß es sich dabei um einen Zwei-Byte-Befehl handelt. Natürlich wäre es möglich gewesen, einen zusätzlichen Parameter vorzusehen. Da nicht jeder Befehl um einen Parameter erweitert werden soll, müßte man zwischen solchen mit und ohne unterscheiden können, **auch wenn das entsprechende Modul nicht eingesteckt ist**, denn sonst würde das zusätzliche Argumentbyte bereits als neue Programmzeile interpretiert werden und somit das Abzählen der Programmzeilen durcheinander bringen (s.o.). Leider ist eine solche Unterscheidung seitens des Betriebssystems nicht vorgesehen; außerdem wäre in den XROM-Befehlen auch kaum Platz dafür.

Tastenzuordnungen (Aufbau und innere Struktur)

Bei Tastenzuordnungen unterscheidet man grundsätzlich zwei unterschiedliche Arten:
- Zuordnungen von USER-Programmen im Hauptspeicher
- Zuordnungen von Funktionen und Programmen aus Einsteckmodulen

Bei Zuordnungen von Programmen, die sich im Hauptspeicher befinden, wird der Tastencode (Code für die Tastenzuordnung) im 4. Byte des betreffenden LBL abgespeichert. Diese Information geht deshalb auch beim Abspeichern eines Programmes nicht verloren.

Bei Zuordnungen von Funktionen und Programmen aus Einsteckmodulen (XROM-Nummern) wird die Zuordnungsinformation in den Tastenzuweisungsregistern gespeichert. Diese Tastenzuweisungsregister beginnen im Hauptspeicher bei der Adresse 192 (hex 0C0). Sobald neue Zuweisungen vorgenommen werden, verschieben sich die alten nach oben (zu den höheren Adressen hin), und die neue Zuweisung steht bei der Adresse 192. Da also Register aus dem Hauptspeicherbereich benötigt werden, verringert sich die Anzahl freier Register, die zur Programmierung übrigbleiben, jeweils um ein Register pro zwei Tastenzuordnungen.

Um den Aufbau der Tastenzuordnungsregister zu zeigen, betrachten wir folgendes Beispiel:

Zuerst ordnen wir die Funktion **BEEP** der Taste -11 und die Funktion **SAS** (XROM 09,05) der Taste 15 zu. Diese Zuordnungen kann man anschliessend mit **CAT'6** überprüfen. Wenn man jetzt das Register auf Adresse 192 mit der Tastenfolge 192 **PEEKR** **DCD** **ALPHA** decodiert, erkennt man folgenden Aufbau (die Bytes sind hexadezimal dargestellt):

F0 A245 41 0486 09

oder ausführlich:

F0 : Erkennungsbyte für Tastenzuordnungsregister (immer F0)
A245 : Code der Zweibytefunktion **SAS** (XROM 09,05)
41 : Tastencode der Taste 15
0486 : Code der Funktion **BEEP**. Das Byte 04 dient nur als Füllerbyte und ist „Vorsilbe" aller Ein-Byte Funktionen aus dem **CAT 1**
09 : Tastencode der Taste -11

Nach Löschen der Tastenzuweisung **BEEP** auf der Taste -11 ändert sich der Aufbau des Tastenzuweisungsregisters wie folgt:

F0 A245 41 0486 00

Man sieht, daß beim Löschen einer Tastenzuweisung nur der Tastencode auf 00 gesetzt wird. Hieran erkennt das Betriebssystem, daß diese Tastenzuweisung nicht mehr aktiv ist. Die Existenz von Tastenzuweisungen genügt dem Rechner allerdings nicht zur Erkennung derselben. Um diese Zuweisungen schnell erkennen zu können, befinden sich in den Statusregistern F und E für alle Tasten sogenannte „Tastenzuweisungsbits". Das Betriebssystem des HP-41 sieht beim Drücken einer Taste zuerst nach, ob das entsprechende Tastenzuweisungsbit gesetzt ist. Ist dies der Fall, so wird die zugewiesene Funktion aktiv (nur im USER-Modus!). Ist das Bit gelöscht, wird erst gar nicht in den Tastenzuweisungsregistern und in den Programmlabels nach einer eventuellen Zuordnung gesucht. Dies erlaubt eine schnelle Unterscheidung, ob eine Zuweisung vorhanden ist oder nicht.

I/O-Buffer (Aufbau und innere Struktur)

Unter **I/O** versteht man im allgemeinen Ein- und Ausgabe. Hier bedeutet das: Dialog mit dem RAM-Speicher unter Umgehung des Betriebssystems. Der I/O-Buffer wird von Einsteckmodulen angesprochen. Manche Module – wie z.B. das TIME- und das CCD-Modul – legen einen I/O-Buffer an (jedes Modul einen eigenen) und verwalten ihn selbständig. Dem Betriebssystem erscheint ein I/O-Buffer als ein geschlossener Registerblock.

Verantwortlich dafür ist das Basisregister, das unterste Register eines jeden I/O-Buffers. Die vier Nybbles ganz links in diesem Register beinhalten die wichtigste Information:

Basisregister: hex *ii ll* .....

Im ersten und zweiten Nybble stehen jeweils eine Kopie der Bufferkennziffer – Buffer-ID genannt -, eine Zahl von hex 1 bis hex E (hex F ist für die Tastenzuordnungsregister reserviert). Die beiden Nybbles *ll* geben die Länge des Buffers an.

Beim Einschalten des Rechners sucht das Betriebssystem einen Buffer. Findet es ihn, löscht es die ID im Nybble 13. Dann springt es zum Register oberhalb des Buffers (indem es die Bufferlänge zur Adresse des Basisregisters addiert) und prüft, ob dort ein weiterer Buffer angelegt ist usw.. Wenn keine Buffer mehr gefunden werden, verzweigt es in die Einsteckmodule, die Buffer anlegen können. Findet jetzt ein Einsteckmodul einen Buffer, der von ihm angelegt wurde, so wird von diesem Modul die Buffer-ID wieder zurück ins Nybble 13 geschrieben. Wurden alle Module in dieser Weise angesprochen, wird wieder ins Betriebssystem verzweigt, welches dann mittels einer speziellen PACK-I/O-Routine die zurückgesetzten Buffer löscht und die Buffer-Register packt. Somit ist also leicht einzusehen, warum ein I/O-Buffer beim Einschalten des Rechners gelöscht wird, wenn das entsprechende Modul nicht im Rechner steckt.

Kapitel 2

Betriebssystem-erweiterungen

Inhaltsverzeichnis Kapitel 2
Betriebssystemerweiterungen

Erweiterungen des Betriebssystems

Allgemeines

Im Gegensatz zu den allgemein bekannten Modulen für den HP-41 erweitert das CCD-Modul das Betriebssystem dieses Rechners. Dies bedeutet, daß, sobald das CCD-Modul im Rechner steckt, die normalen Funktionen des HP-41 erweitert werden; zusätzlich zu den Funktionen des CAT 2. So wird der Rechner mit neuen Katalogen, verbesserten XEQ- und ASN-Funktionen, der Möglichkeit der direkten synthetischen Programmierung und einem Kleinbuchstabenmodus ausgestattet (diese Erweiterungen sind mit sehr alten Rechnern nicht direkt möglich, siehe Anhang *Kompatibilität*).

Die Kataloge

Mit dem CCD-Modul werden die drei (bzw. sechs im HP-41 CX) vorhandenen Kataloge des HP-41 auf sechzehn Kataloge erweitert und wesentlich verbessert. Alle neuen Kataloge lassen sich mit R/S anhalten und anschliessend mit SST und BST durchgehen. Beim Zurückgehen mit BST bleibt im Gegensatz zu den alten Katalogen das SHIFT-Flag an. Mit SHIFT R/S laufen die neuen Kataloge sogar rückwärts. Der laufende Katalog kann durch das Drücken einer beliebigen Taste (außer R/S und ON) beschleunigt werden. Es folgt eine genaue Beschreibung der einzelnen Kataloge:

CAT'0

Mit **CAT'0** werden die ID oder die AID aller Geräte in der angeschlossenen Interface Loop angezeigt. Wird der Katalog angehalten, so kann mit der ENTER-Taste das angezeigte Gerät selektiert werden. Mit der C-Taste kann es auf den Einschaltzustand zurückgesetzt werden (es wird eine Device

Clear Meldung gesendet). Mit der ← -Taste wird der angehaltene Katalog abgebrochen. Befindet sich kein HP-IL Modul im Rechner, so wird die Meldung „NO HPIL" angezeigt.

CAT'1

Dieser Katalog ruft den normalen **CAT 1** des verwendeten HP-41 auf. Nähere Beschreibung siehe Handbuch zum HP-41.

CAT'2

Dieser neue Katalog zeigt zunächst nur die „Überschriften" der eingesteckten oder eingebauten (wie im HP-41 CX) Module an. Wird der Katalog mit R/S angehalten, so gelangt man mit der Taste ENTER in den Funktionsblock der angezeigten „Überschrift". Hat man die gewünschte Funktion gefunden, kann man sie mit der Taste XEQ direkt aus dem Katalog heraus ausführen oder programmieren oder mit der A-Taste einer Taste zuordnen (siehe auch ASN im HP-41 Handbuch). Mit der Taste ENTER gelangt man aus dem Funktionsblock wieder zur „Überschrift" zurück.

CAT'3

Dieser Katalog ruft den normalen **CAT 3** des verwendeten HP-41 auf. Nähere Beschreibung siehe Handbuch zum HP-41.

CAT'4

Mit **CAT'4** werden die Namen, Längen und Typen aller Files im erweiterten Speicher angezeigt (wie **EMDIR** im Extended Functions Modul). Die zusätzlichen File-Typen des CCD-Moduls (B = Buffer, M = Matrix, K = Key) werden richtig angezeigt. Wenn kein erweiterter Speicher vorhanden ist, erfolgt die Fehlermeldung „NO XF/M".

CAT'5

Dieser Katalog ruft die Funktion **ALMCAT** des TIME-Moduls auf (wie **ALMCAT** im TIME-Modul). Befindet sich kein TIME-Modul im Rechner, so wird die Fehlermeldung „NO TIMER" angezeigt.

Hinweis:
Ist ein Kartenleser zusätzlich eingesteckt und der Alarmkatalog leer, so kann bei älteren HP-41 C oder CV mit älteren Kartenlesern die Meldung „CAT'5" im Display stehenbleiben. Dies ist jedoch unbedeutend und kein Fehler; mit der nächsten Operation (z.B. **CLX**) verschwindet „CAT'5" aus der Anzeige.

CAT'6

Mit diesem neuen **CAT'6** werden alle Tastenbelegungen des HP-41 in der Reihenfolge der Tastencodes angezeigt. Rechts in der Anzeige steht hierbei der Tastencode (siehe auch **ASN** im HP-41 Handbuch) und links davon die zugehörige Funktion. Selbst synthetische Tastenzuordnungen werden richtig (z.B. „RCL M" oder „TEXT 7") und nicht als XROM-Nummern angezeigt. Wird der Katalog angehalten, so kann man mit der Taste „C" die angezeigte Tastenzuordnung löschen. Wenn keine Tastenzuordnungen mehr vorhanden sind, wird die Meldung „NO KEYS" angezeigt.

CAT'7

Dieser Katalog ruft die Funktion **DIR** des eingesteckten HP-IL Moduls auf (nähere Beschreibung siehe Handbuch zum HP-IL Modul). Befindet sich kein HP-IL Modul im Rechner, so wird die Meldung „NO HPIL" angezeigt.

CAT'8 – CAT'F

Die Kataloge 8, 9, A, B, C, D, E und F beginnen den **CAT'2** mit dem Funktionsblock desjenigen Moduls, welches auf dem gewählten Adressblock liegt. Die 4 Ports des HP-41 sind adressenmäßig folgendermaßen geschaltet:

Port 1: Adressblock 8 und 9
Port 2: Adressblock A und B
Port 3: Adressblock C und D
Port 4: Adressblock E und F

Jeder Port des HP-41 kann maximal mit 8 kByte Programmaterial belegt werden. Wird nur ein 4 kByte Modul eingesteckt, so ist der restliche Platz praktisch verschenkt und der entsprechende Katalog zeigt für den leeren Adressblock „NO ROM" an.

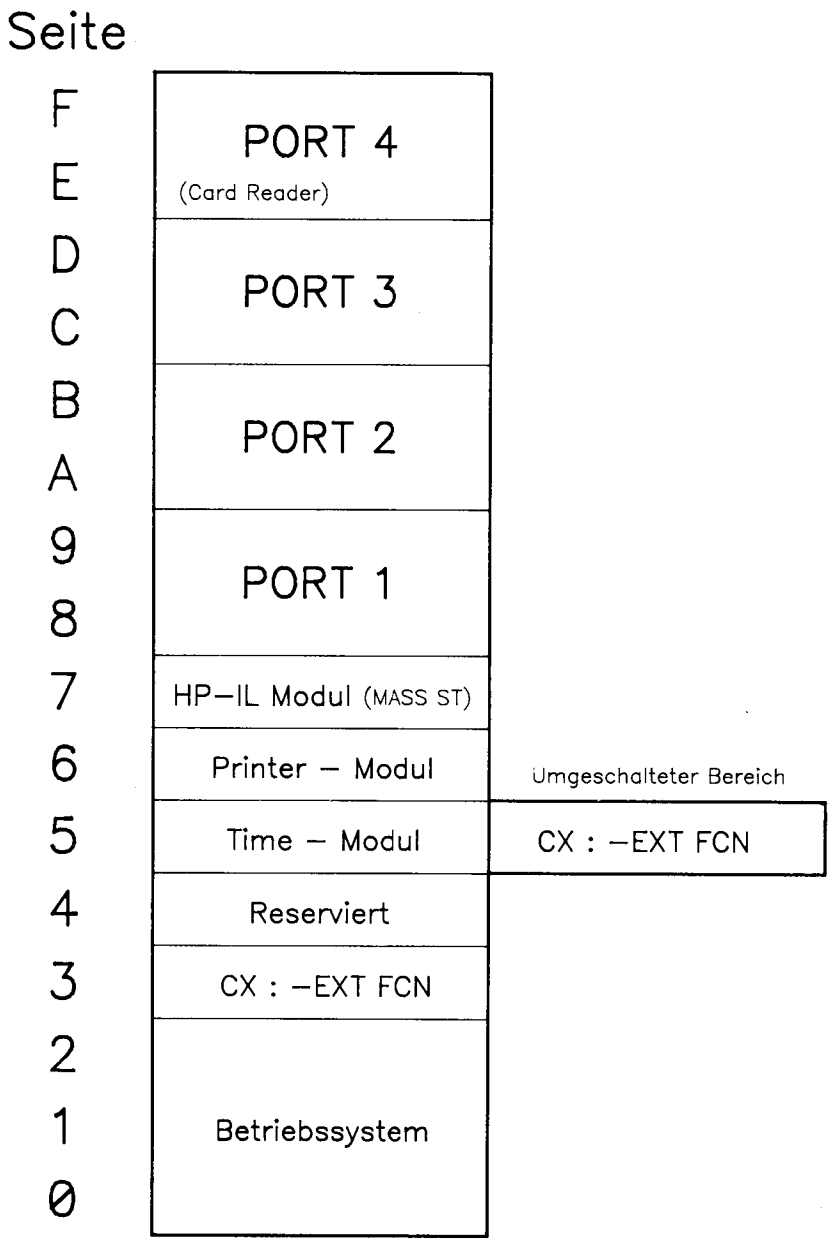
Der HP-41 verwaltet insgesamt sechzehn 4 kByte Adressblöcke (0 – F). Diese werden wie folgt genutzt:

Adressblock	**Verwendung für**
0	*Betriebssystem des HP-41 (System ROM 0)*
1	*Betriebssystem des HP-41 (System ROM 1)*
2	*Betriebssystem des HP-41 (System ROM 2)*
3	*Nicht genutzt vom HP-41 C und CV. Erweitertes Betriebssystem des HP-41 CX (System ROM 3)*
4	*Reserviert für ein Diagnose-Modul des HP Service*

5	*Beim HP-41 C und CV reserviert für das TIME-Modul. Erweitertes Betriebssystem des HP-41 CX (System ROMs 5a und 5b; werden intern umgeschaltet)*
6	*Reserviert für das Drucker-Modul*
7	*Reserviert für das HP-IL Modul (Anmerkung: Das Drucker-Modul ist im HP-IL Modul mit enthalten, kann aber mittels eines Schalters auf den Adressbereich 4 gelegt und somit abgeschaltet werden)*
8	*Port 1 untere 4 kByte*
9	*Port 1 obere 4 kByte*
A	*Port 2 untere 4 kByte*
B	*Port 2 obere 4 kByte*
C	*Port 3 untere 4 kByte*
D	*Port 3 obere 4 kByte*
E	*Port 4 untere 4 kByte*
F	*Port 4 obere 4 kByte*

Die Speicher und Speichererweiterungen des HP-41 werden gänzlich anders verwaltet und belegen deshalb keinen Port. Daher ist es möglich, alle Speichermodule in den Rechner einzubauen und so die Ports des Rechners wieder für andere Module frei zu haben. Man kann sogar das CCD-Modul fest in den Rechner auf Port 3 einbauen und dennoch das HP-IL Modul oder das Drucker-Modul in diesen Port einstecken, da diese Module intern adressiert sind (siehe oben). Auch ist es z.B. möglich, das TIME-Modul mit einem anderen Modul zusammenzubauen um somit einen Steckplatz frei zu bekommen. Nähere Informationen hierzu siehe Anhang *Hardwaremodifikationen*.

ROM – Speicherstruktur



Die Funktionen ASN und XEQ

Zum besseren Verständnis dieser Funktionen sollte unbedingt der Abschnitt „Funktionen für den fortgeschrittenen Programmierer" aufmerksam gelesen werden. Zum Verständnis der synthetischen Programmierung wird im Anhang *Literaturhinweise* auf entsprechende Literatur verwiesen.

ASN

Die erweiterte ASN-Funktion erlaubt folgende Eingaben:

a) **Die normale ASN-Funktion:** Drückt man nach ASN die ALPHA-Taste, so befindet man sich in der normalen ASN-Funktion. Es besteht kein Unterschied zu der ursprünglichen Funktion.

b) **Die Zuordnung beliebiger Zweibyte-Funktionen:** Drückt man mit eingestecktem CCD-Modul die Taste ASN, so erscheint in der Anzeige „ASN: _ _ _ : _ _ _ ". Der Rechner fordert zur Eingabe von zwei dezimalen Byte-Werten auf. Gibt man nun zwei Bytes ein (z.B. 144 und 117 für RCL M, siehe Byte-Tafel), so wird diese Funktion der anschließend gedrückten Taste zugeordnet. Drückt man allerdings vorher die Taste H, so erscheint in der Anzeige „ASN ' _ _ ' _ _ ", und der Rechner verlangt die entsprechende hexadezimale Byte-Eingabe. Mit ENTER oder . gelangt man wieder zur dezimalen Eingabeform zurück.

c) **Die Zuordnung einer XROM-Nummer:** Die erweiterte ASN-Funktion des CCD-Moduls erlaubt es auch, XROM-Nummern zuzuordnen, auch wenn das entsprechende Modul nicht im Rechner steckt. Hat man die Taste ASN betätigt und drückt anschließend die XEQ-Taste, so erscheint in der Anzeige „ASN XROM: _ _ ". Nun kann man die XROM-Nummer einer Funktion eingeben, auch wenn das betreffende Modul nicht im Rechner steckt und diese so trotzdem zuordnen. Nach Eingabe der ersten beiden Ziffern (Vorkommateil der XROM-Nummer), z.B. 23, erscheint in der Anzeige „ASN XROM:23: _ _ ", und der Nachkommateil der XROM-Nummer kann jetzt eingegeben werden, z.B. 01. In der Anzeige steht dann „XROM:23:01 _ ". Drückt man jetzt eine Taste, so wird diese XROM-Nummer (in unserem Beispiel die Funktion COPYFL aus dem Extended I/O-Modul) dieser Taste zugeordnet.

XEQ

Die erweiterte XEQ-Funktion erlaubt folgende Eingaben:

a) **Die normale XEQ-Funktion:** Drückt man nach XEQ die ALPHA-Taste oder eine Zifferntaste, so befindet man sich in der normalen XEQ-Funktion. Es besteht kein Unterschied zu der ursprünglichen Funktion.

b) **Die Ausführung beliebiger Zweibyte-Funktionen:** Drückt man mit eingestecktem CCD-Modul die Taste XEQ und anschließend die Taste ENTER, so erscheint in der Anzeige „XEQ:___:___". Der Rechner fordert zur Eingabe von zwei dezimalen Byte-Werten auf. Gibt man nun zwei Bytes ein (z. B. 144 und 117 für RCL M, siehe Byte-Tafel), so wird diese Funktion sofort ausgeführt oder programmiert. Drückt man allerdings vorher die Taste H, so erscheint in der Anzeige „XEQ'__'__", und der Rechner verlangt die entsprechende hexadezimale Byte-Eingabe. Mit ENTER oder · gelangt man wieder zur dezimalen Eingabeform zurück.

c) **Die Ausführung einer XROM-Nummer:** Die erweiterte XEQ-Funktion des CCD-Moduls ermöglicht es auch, Funktionen mit ihrer XROM-Nummer zu programmieren, auch wenn das entsprechende Modul nicht im Rechner steckt. Hat man die Tasten XEQ und ENTER betätigt und drückt anschließend die XEQ-Taste, so erscheint in der Anzeige „XEQ XROM: _ _ ". Nun kann man die XROM-Nummer einer Funktion eingeben, auch wenn das betreffende Modul nicht im Rechner steckt und diese so trotzdem programmieren. Nach Eingabe der ersten beiden Ziffern (Vorkommateil der XROM-Nummer), z.B. 23, erscheint in der Anzeige „XEQ XROM:23: _ _ ", und der Nachkommateil der XROM-Nummer kann jetzt eingegeben werden, z.B. 01. Im Normalmodus wird diese XROM-Nummer (in unserem Beispiel die Funktion COPYFL aus dem Extended I/O-Modul) sofort ausgeführt. Ist das entsprechende Modul nicht eingesteckt, so erscheint die Fehlermeldung „NONEXISTENT". Im Programmodus wird die XROM-Nummer programmiert und erscheint beim Einstecken des entsprechenden Moduls als die gewünschte Funktion.

Anmerkung:
Zur einfachen Unterscheidung beginnen im CCD-Modul alle Dezimaleingabeprompts mit dem „ : "-Zeichen und alle Hexadezimalprompts mit dem „ ' "-Zeichen.

Direkte und indirekte Speicherzugriffsfunktionen

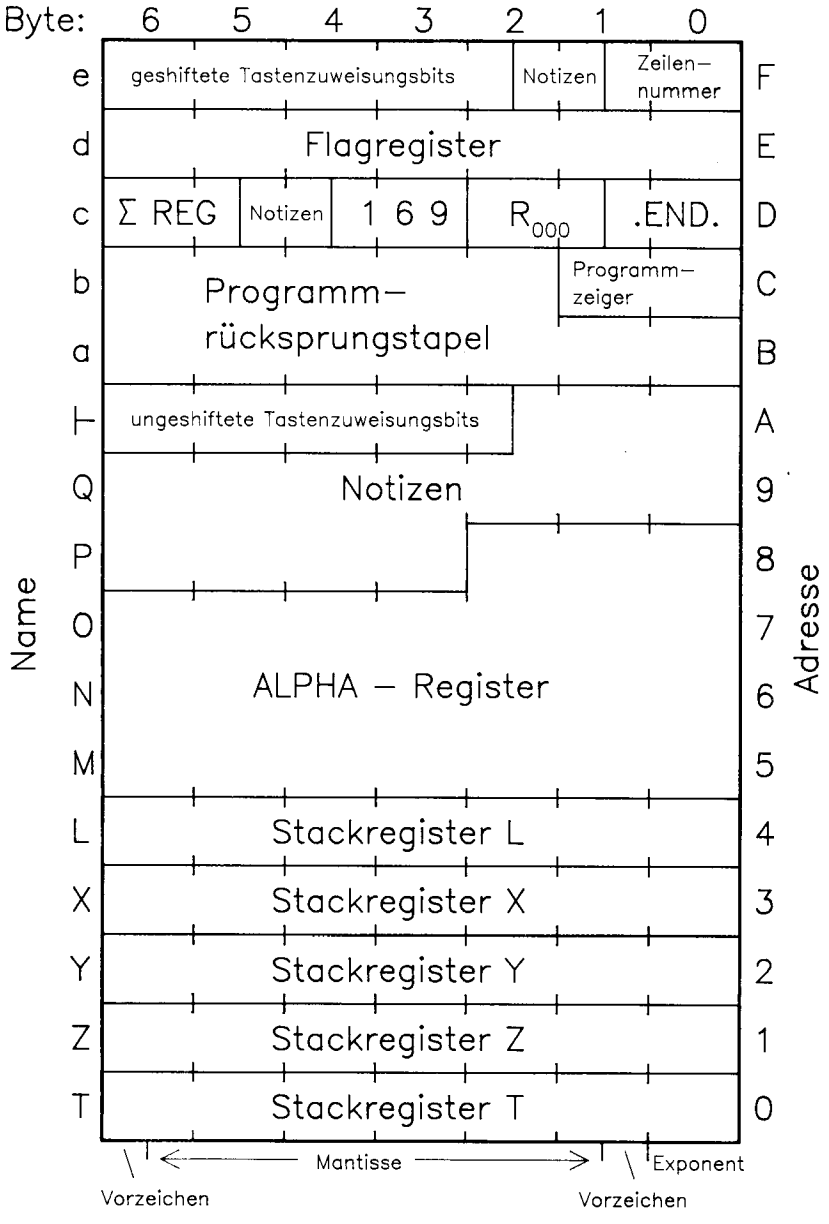
Um die Eingabe von synthetischen Programmzeilen einfach zu machen, wurde im CCD-Modul die Möglichkeit der direkten Eingabe solcher Befehle vorgesehen. (Bei sehr alten HP-41 Rechnern, siehe auch Anhang *Kompatibilität*, besteht diese Möglichkeit nicht; hier muß dann mit entsprechenden Tastenzuordnungen gearbeitet werden.)
Alle Speicherzugriffsfunktionen (RCL, STO, X<> usw.) können jetzt auch direkt von der Tastatur aus jedes Statusregister des HP-41 ansprechen. Der Zugriff auf die Register M, N, O, P, Q, ⊢, a, b, c, d und e ist somit genauso einfach geworden wie bei den Registern X, Y, Z, T und L. Die Tastenfolge hierfür ist z.B. RCL | d.

Eine Übersicht über die Statusregister des HP-41 ist auf der nächsten Seite abgebildet:

Achtung

Eine Veränderung der Register ⊢, a, b, c, d und e kann zu einem „MEMORY LOST" oder zu einem Systemabsturz des HP-41 führen. Beim Umgang mit diesen Registern ist Vorsicht geboten. Eine Änderung der Konstante 169 in Register c führt auf jeden Fall zu „MEMORY LOST". Bevor Sie mit diesen Registern experimentieren, sollten Sie sich zuerst Kenntnisse über die synthetische Programmierung aneignen (siehe Anhang *Literaturhinweise*).

Statusregister



Die Eingabe von beliebigen ALPHA-Zeichen

Mit dem CCD-Modul ist es nun möglich, alle 256 Bytes, die der HP-41 kennt, auch im ALPHA-Register darzustellen (siehe Byte-Tafel). Dies war bisher nur mit Funktionen wie XTOA aus dem Extended Functions Modul möglich, wobei eine direkte Eingabe in eine Programmzeile nicht möglich war. Gerade für Anwender, die sehr viel mit Druckern oder IL-Peripherie arbeiten, eröffnen sich mit dem Klein- und Sonderzeichenmodus des CCD-Moduls ungeahnte Möglichkeiten. Wie platzsparend man mit dem Kleinbuchstabenmodus programmieren kann, soll an einigen kurzen Beispielen veranschaulicht werden:

Ausdruck des Textes „Hewlett-Packard"

auf normale Weise

```
01♦LBL "HP"
02  "H"
03  ACA
04  SF 13
05  "EWLETT-
          "
06  ACA
07  CF 13
08  "P"
09  ACA
10  SF 13
11  "ACKARD"
12  ACA
13  CF 13
14  PRBUF
15  END

  PLNG "HP"
   46 BYTES
```

mit Hilfe des Kleinbuchstabenmodus programmiert

```
01♦LBL "HP"
02  "Hewlett
    -Packard"
03  AVIEW
04  END

  PLNG "HP"
   26 BYTES
```

Senden einer ESC-Sequenz an den ThinkJet-Drucker

auf normale Weise (wie im Handbuch beschrieben)

```
01◆LBL "WID
          E"
02 27
03 ACCHR
04 38
05 ACCHR
06 107
07 ACCHR
08 49
09 ACCHR
10 83
11 ACCHR
12 END

PLNG "WIDE"
   32 BYTES
```

mit Hilfe des Kleinbuchstabenmodus programmiert

```
01◆LBL "WID
          E"
02 "Æ&k1S"
03 ACA
04 END

PLNG "WIDE"
   19 BYTES
```

Wie man sieht, sind mit Hilfe von Kleinbuchstaben wesentlich bessere und kürzere Möglichkeiten gegeben, sinnvoll zu programmieren.

Der CCD-Modul-Kleinbuchstabenmodus ist aktiv, sobald das Modul in den Rechner eingesteckt und der USER-Modus ausgeschaltet ist (ALPHA an!). Nun ist es möglich, Texte, die Kleinbuchstaben, andere Sonderzeichen oder Druckersteuerzeichen enthalten, in das ALPHA-Register oder in ein Programm zu schreiben. Alle Sonderzeichen und Druckersteuerzeichen sind auf dem speziellen CCD-Modul-Overlay und der Tastatur gekennzeichnet:

USER-Mode an:	Das normale ALPHA-Tastenfeld ist aktiv
Blaue Schrift Tastatur:	Großbuchstaben A-Z und einige Sonderzeichen
Blaue Schrift Overlay:	Sonderzeichen, zugänglich über die SHIFT-Taste
USER-Mode aus:	Der Kleinbuchstabenmodus ist aktiv
Rote Schrift Overlay:	Sonderzeichen der ungeshifteten Tasten (die Kleinbuchstaben von A-Z sind nicht gesondert gekennzeichnet)
Grüne Schrift Overlay:	Sonderzeichen, zugänglich über die SHIFT-Taste

Befindet sich das gewünschte Sonderzeichen nicht auf der Tastatur, so kann man es mit seinem dezimalen Wert nach Drücken der Tastenkombination SHIFT ENTER eingeben. Nach Drücken der Tastenkombination SHIFT ENTER H ist sogar eine hexadezimale Eingabe möglich.

Achtung:
Diese im folgenden „Byteprompt" genannte Eingabe ist nicht rückgängig zu machen. Wurde der Byteprompt versehentlich ausgelöst, so ist zuerst ein beliebiges Zeichen einzugeben (z.B. entspricht 055 dem Zeichen „7") und dieses anschließend mit der ← Taste zu löschen.
Ist der Kleinbuchstabenmodus nicht gewünscht, so kann man ihn einfach inaktivieren (siehe Programm „TLC" bei *Funktionen für fortgeschrittene Programmierer*).

Wenn ein Sonderzeichen gleichzeitig ein Druckersteuerzeichen bzw. ein Control-Code für HP-IL Geräte ist, so sind diese Zeichen rechts neben der Taste in der gleichen Farbe mit ihrer Kurzbezeichnung abgebildet (siehe auch Handbücher der entsprechenden IL-Geräte).

Bitte beachten:
In der CCD-Modulversion **-W&W CCD A** darf man in einer Programmzeile kein Leerzeichen eingeben, wenn der Kleinbuchstabenmodus eingeschaltet ist. Dies könnte sonst zu Veränderungen im Programmspeicher führen. Leerzeichen in einer Programmzeile dürfen also nur mit ausgeschaltetem Kleinbuchstabenmodus oder mit dem Byteprompt eingegeben werden!

Kapitel 3

Funktionen aus Katalog 2

Inhaltsverzeichnis Kapitel 3
Im Katalog 2 erscheinende Funktionen

Im Katalog 2 erscheinende Funktionen

B?

Die Funktion **B?** (Buffer?) fragt nach der Existenz eines Ein-/Ausgabebuffers, im folgenden I/O-Buffer genannt. Die Buffer-Identitätsnummer (Buffer-ID genannt, eine Nummer zwischen 1 und 14) wird im X-Register erwartet. Wird die Funktion im laufenden Programm aufgerufen, so wird der nächste Programmschritt ausgeführt, wenn der gefragte Buffer vorhanden ist. Ist er nicht vorhanden, so wird der Programmschritt nach **B?** übersprungen. Wird **B?** von der Tastatur aus ausgeführt, erscheint in der Anzeige entweder „**YES**" oder „**NO**", je nachdem ob der gefragte I/O-Buffer existiert oder nicht. Ist der absolute Integerwert der Zahl im X-Register größer als 14, erscheint in der Anzeige die Fehlermeldung „DATA ERROR".

I/O-Buffer werden von verschiedenen Modulen (s.u.) angelegt, um dort Daten zwischenzuspeichern. Für diese Buffer wird der sonst freie Speicherplatz zwischen Tastenzuweisungsregistern und letztem Programm (ab dem **.END.**) benötigt. Je größer der I/O-Buffer ist, desto weniger freie Register sind für die Programmierung vorhanden (siehe auch *I/O-Buffer, Aufbau und innere Struktur*). Bisher sind folgende I/O-Buffer bekannt:

Modul	**ID**	**Verwendung**
CCD-Modul	5	Abspeicherung der Wordsize, der letzten Zufallszahl und der augenblicklich aktiven Matrix
TIME-Modul	10	Zeit Alarme
Plotter-Modul	11	Zwischenspeicher für Plotter- und Barcodebefehle
HP-IL Development	12	"Scope-Mode", Abspeicherung von HP-IL Kommandos

Eingabeparameter

X-Register: *aa* (Buffer-ID)

Beispiel

Es soll festgestellt werden, ob ein Alarm-Buffer des TIME-Moduls im HP-41 existiert. Hierzu sind folgende Schritte nötig:

10 **B?**

Falls Alarme im Rechner existieren, erfolgt die Meldung „**YES**". Falls kein Alarm existiert, erfolgt die Meldung „**NO**".

Weitere Hinweise

Die Funktion **B?** benötigt nur den absoluten Integerteil der Zahl im X-Register. Mit -5,678 **B?** ergibt sich also dasselbe Ergebnis wie mit 5 **B?**.

Verwandte Funktionen

CLB, **GETB**, **SAVEB**

CLB

Die Funktion **CLB** (Clear buffer) löscht einen I/O-Buffer. Die Buffer-ID (Zahl zwischen 1 und 14) wird im X-Register erwartet.

Eingabeparameter

X-Register: *aa* (Buffer-ID)

Beispiel

Um mehr freien Speicherplatz zu erhalten, sollen alle vorhandenen Zeitalarme des TIME-Moduls im HP-41 gelöscht werden. Dies erreicht man durch die Tastenfolge:

10 **CLB**

Weitere Hinweise

Die Funktion **CLB** benötigt nur den absoluten Integerteil der Zahl im X-Register. Mit -5,678 **CLB** ergibt sich also dasselbe Ergebnis wie mit 5 **CLB**.

Verwandte Funktionen

B?, **GETB**, **SAVEB**

SAS

Die Funktion **SAS** (Set autostart) setzt das Autostart-Flag des CCD-Moduls. Ist dieses Flag gesetzt, so beginnt der Rechner beim Einschalten auf jeden Fall mit dem Programmlauf. Somit kann man verhindern, daß ein Programm durch Ausschalten des Rechners unterbrochen wird, da beim Einschalten die Programmausführung sofort wieder aufgenommen wird.

Eingabeparameter

keine

Beispiel

Ein Programm soll an einer bestimmten Stelle auf keinen Fall unterbrochen werden können; auch nicht durch Betätigen der **ON**-Taste. Dieses Ziel wird durch einfaches Einfügen des Befehls **SAS** erreicht: Selbst wenn man den Rechner ausschaltet, wird beim Einschalten der Programmlauf an der gleichen Stelle wieder aufgenommen, ohne daß die Möglichkeit besteht, das Programm zu unterbrechen. In unserem Beispiel kann das Programm an der Stelle, wo **PMTK** ausgeführt wird, nicht unterbrochen werden.

```
01 SAS
02 "JA/NEIN
       JN"
03 PMTK
04 CAS
05 END
```

Weitere Hinweise

Das Flag für diesen Autostart befindet sich nicht im HP-41 Flag-Register, sondern im Status-Register c (Reg.-Byte 13,6 Bit 6).

Verwandte Funktion

CAS

CAS

Die Funktion **CAS** (Clear autostart) löscht das Autostart-Flag des CCD-Moduls.

Eingabeparameter

keine

Weitere Hinweise

Das Flag für diesen Autostart befindet sich nicht im HP-41 Flag-Register, sondern im Status-Register c (Reg.-Byte 13,6 Bit 6).

Verwandte Funktion

SAS

RNDM

Die Funktion **RNDM** (Random) erzeugt eine Zufallszahl, die ins X-Register geschrieben wird. Diese Zufallszahl hat einen Wert zwischen 0 und 1.

Eingabeparameter

keine, evtl. kann ein Startwert mit der Funktion **SEED** vorgegeben werden

Beispiele

Mit einem Startwert von 0,1 erzeugt man folgende Zufallszahlen:

0,1 **SEED**
RNDM 0,311327
RNDM 0,753794
RNDM 0,222201
RNDM 0,447348 usw.

Weitere Hinweise

Die letzte Zufallszahl wird im I/O-Buffer des CCD-Moduls abgespeichert. Jede neue Zufallszahl errechnet sich dann nach folgendem Algorithmus: X = FRC (Buffer * 9821 + 0,211327). Ist nicht genügend Speicherplatz für den I/O-Buffer vorhanden, erscheint die Fehlermeldung „NO ROOM".

Achtung:
In der CCD-Modulversion **-W&W CCD A** werden Werte, die kleiner als 0,1 sind, nicht richtig „normalisiert". Werden die Zufallszahlen sofort verarbeitet (z.B. mit einem anderen Wert multipliziert o.ä.), dann tritt kein Fehler auf. Sollen die Zufallszahlen jedoch sofort abgespeichert werden, so ist unmittelbar nach **RNDM** die Funktion **FRC** auszuführen, da ansonsten alle Zufallszahlen, die kleiner als 0,1 sind, zu Null umgewandelt werden.

Verwandte Funktion

SEED

SEED

Mit der Funktion **SEED** wird ein Startwert für die **RNDM**-Funktion des CCD-Moduls in den CCD-Modul I/O-Buffer geschrieben. Nur der Nachkommateil der Zahl im X-Register wird für diesen Startwert verwendet.

Eingabeparameter

X-Register: Startwert, nur der Nachkommateil findet Verwendung

Weitere Hinweise

Der Startwert wird im CCD-Modul I/O-Buffer abgelegt. Mit **CLX** **SEED** wird dieser Bufferteil gelöscht, und mindestens ein zusätzliches Register ist wieder für die Programmierung frei.

Verwandte Funktion

RNDM

SORT

Diese Funktion sortiert Registerinhalte, angefangen mit Register R_{iii} bis Register R_{jjj}. Mit dieser Funktion lassen sich auch ALPHA-Daten sortieren. Der größte Wert wird in Register R_{jjj} abgelegt. Durch geeigneten Aufbau der Kontrollzahl *iii,jjj* im X-Register ist es möglich, sowohl in absteigender als auch in aufsteigender Reihenfolge zu sortieren.

Eingabeparameter

X-Register: *iii,jjj*

Beispiele

Die Registerinhalte der Register 1–10 sollen einmal in aufsteigender und einmal in absteigender Reihenfolge sortiert werden:

Die ursprünglichen Registerinhalte sehen wie folgt aus:

```
R01=   0.1356
R02=   3.5462
R03=   9.7363
R04=   2.4138
R05=   7.8467
R06=   4.0629
R07=   4.0506
R08=   2.9577
R09=   9.2921
R10=   0.1220
```

1) 1,010 **SORT**

```
R01=   0.1220
R02=   0.1356
R03=   2.4138
R04=   2.9577
R05=   3.5462
R06=   4.0506
R07=   4.0629
R08=   7.8467
R09=   9.2921
R10=   9.7363
```

2) 10,001 **SORT**

```
R01=   9.7363
R02=   9.2921
R03=   7.8467
R04=   4.0629
R05=   4.0506
R06=   3.5462
R07=   2.9577
R08=   2.4138
R09=   0.1356
R10=   0.1220
```

Weitere Hinweise

Werden ALPHA-Daten und numerische Daten sortiert, so sind ALPHA-Daten größer als numerische.

Verwandte Funktion

SORTFL

Kapitel 4

Matrix-funktionen

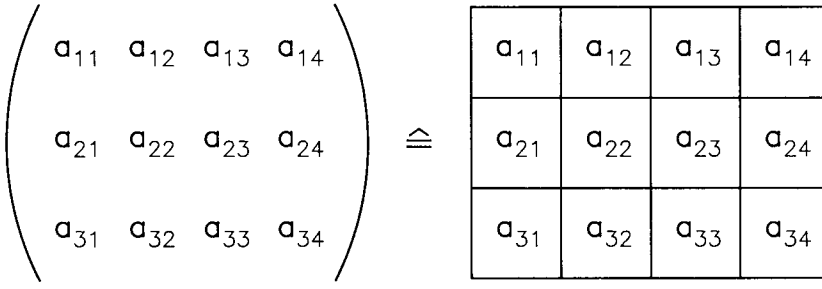
Inhaltsverzeichnis Kapitel 4
Matrixfunktionen

Feldfunktionen

In diesem Teil wird beschrieben, wie mit Hilfe des CCD-Moduls ein- oder zweidimensionale Datenfelder angelegt und manipuliert werden können.

Zwei Bemerkungen vorweg:

Diese Felder dürfen sowohl numerische als auch alphanumerische Informationen enthalten. Rein numerische Datenfelder werden als zweidimensionale Matrizen bezeichnet. Alle mathematischen Funktionen erlauben nur solche als Operanden. Im folgenden werden alle Matrizenelemente in umrandeten Kästchen dargestellt, um so den Zusammenhang zwischen diesen Elementen und dem verwendeten Speicherplatz zu verdeutlichen:



Weiterhin verwenden wir den Begriff „Reihe" anstelle von „Zeile", da so der Zusammenhang zwischen den Funktionsnamen (englisch „row" entspricht Zeile oder Reihe) und den Operationen besser ersichtlich ist.

Felder können sowohl im Hauptspeicher als auch im erweiterten Speicherbereich angelegt werden. Im ersten Fall nennen wir sie RAM-Felder, im zweiten XM-Felder. Außer einigen Unterschieden in der Namensgebung und beim Anlegen verhalten sich beide Arten von Feldern gleich. Im Katalog 4 des CCD-Moduls erkennt man XM-Felder an dem Buchstaben **M** (für Matrix).

Aufbau und Anlegen von Feldern

Jedes Feld besteht aus einer Statusinformation und seinen Elementenwerten.

Die Statusinformation beinhaltet den Feldnamen, die Felddimensionen und einen Elementzeiger. Bei RAM-Feldern wird ein Statusregister angelegt, das diese Information – außer dem Namen – enthält. Der Name ist durch die Lage des Registers gegeben: das RAM-Feld „R012" hat sein Statusregister in Register 12. Seine Elemente werden in den darauffolgenden Registern abgespeichert. Damit beim Zugriff auf Register 12 die Statusinformation nicht zerstört werden kann (z.B. durch **ST +** **12** oder **RCL** **12**), ist sie in Form von ALPHA-Daten abgelegt.

Bei XM-Feldern ist die Statusinformation innerhalb des vom erweiterten Funktionsmodul angelegten File-Statusregisters (auch File-Header genannt) untergebracht und benötigt daher keinen zusätzlichen Speicherplatz.

Die Werte der Feldelemente werden reihenweise gespeichert, wobei jedes Element ein Register belegt.

Jedes Element wird durch seine Position im Feld gekennzeichnet, d.h. durch die Angabe der Reihe und der Spalte, in denen es sich befindet. Die Reihen- und Spaltennummern nennen wir Reihen- und Spaltenindex und bezeichnen sie mit **i** und **j**. Der kleinstmögliche Wert für *i* und *j* ist 1, der größtmögliche für *i* ist **m** und für *j* ist er **n**. Somit gilt:

Reihenindex $1 <= i <= m$

Spaltenindex $1 <= j <= n$

Für alle Feldfunktionen gilt: **Wird für *i* oder *j* der Wert 0 angegeben, so wird er durch 1 ersetzt.**

Zum Anlegen eines Feldes wird der Befehl **MDIM** verwendet. Unter Angabe des Feldnamens (im ALPHA-Register) und der Felddimensionen (X-Wert = *mmm,nnn*) wird das entsprechende File angelegt. Seine Dimensionen *m* und *n* können später mit Hilfe von **DIM** wieder abgefragt werden.

Die Lage und Größe der Datenfelder (Matrizen, Vektoren usw.) wird mit Hilfe der neuen CCD-Modul-Funktionen automatisch verwaltet. Dies bedeutet, daß der Anwender sich die Position einzelner Register bzw. einzelner Elemente nicht mehr merken muß. So kann man zum Beispiel Matrizenfelder im erweiterten Speicherbereich (Extended Memory) anlegen und diese Felder mit Matrizenfeldern, die im Hauptspeicher positioniert sind, verknüpfen. Bei diesen Verknüpfungen mit den CCD-Modul-Funktionen gelten allerdings einige Eingabevorschriften, die im folgenden erläutert sind:

Eingabevorschriften für die Matrixfunktionen:

Alle Matrix-Verknüpfungsfunktionen des CCD-Moduls erwarten ihre Eingabeparameter im ALPHA-Register. Hat man zum Beispiel drei Matrizen gleicher Größe, „A", „B" und „C", welche vorher mit MDIM dimensioniert wurden (siehe auch MDIM), so sieht die Verknüpfungsvorschrift folgendermaßen aus:

1) Matrix „A" soll mit Matrix „B" verknüpft und das Ergebnis in Matrix „C" abgelegt werden. Im ALPHA-Register muß dann folgender Text eingegeben werden: „A,B,C". Die einzelnen Operanden sind also durch Kommas zu trennen. Jetzt kann die Verknüpfungsfunktion (z.B. M*M) ausgeführt werden.
2) Matrix „A" soll mit Matrix „B" verknüpft und das Ergebnis in Matrix „B" abgelegt werden. Das ALPHA-Register muß dann folgendermaßen aussehen: „A,B,B" oder einfach „A,B". Wird das Resultatfeld nicht angegeben, so wird das Resultat nach Ausführen der Verknüpfungsfunktion automatisch im zweiten Operanden abgelegt. (Achtung: Bei der Funktion M*M darf das Resultatfeld nicht identisch mit einem der Operanden sein; dies führt zu einer Fehlermeldung!)
3) Matrix „A" soll mit sich selbst verknüpft und das Ergebnis in Matrix „C" abgelegt werden:
   ALPHA-Register: „A,A,C" oder einfach „A,,C". Da der zweite Operand mit dem ersten identisch ist, braucht er nicht angegeben zu werden. Das separierende Komma darf allerdings auf keinen Fall vergessen werden. (Regel: Links neben dem Resultatnamen müssen immer zwei Kommas vorhanden sein, wenn dieser nicht mit einem der Operanden identisch sein soll!)

4) Matrix „A“ soll mit sich selbst verknüpft und das Ergebnis auch in „A“ abgelegt werden:
   ALPHA-Register: „A,A,A“ oder einfach „A“. Dies ist eigentlich die logische Schlußfolgerung aus dem oben Genannten. Findet eine Verknüpfungsfunktion nur einen Namen im ALPHA-Register vor, so wird die Operation nur in diesem Feld ausgeführt und das Ergebnis auch dort abgelegt.
5) Hat ein Operand den Namen „X“, so wird der entsprechend andere Operand mit dem Inhalt des X-Registers verknüpft.

Matrizen, die im Hauptspeicher liegen, haben immer einen Namen, der aus vier Zeichen besteht. Der erste Buchstabe muß immer ein „R“ (für Register) sein. Die nächsten drei Zeichen bestehen aus Ziffern. Diese drei Ziffern geben zusammen das Startregister der Matrix an (siehe auch MDIM). Im folgenden verwenden wir für die Namen der Datenfelder (Operand 1, Operand 2, Resultat) die verkürzte Schreibweise *OP1,OP2,RES*.

Funktionen zum Anlegen von Datenfeldern

MDIM

Mit **MDIM** werden ein- oder zweidimensionale Felder angelegt oder redimensioniert.

Eingabeparameter

ALPHA-Register : Feldname

Der Feldname ist ein Filename. Für Felder im Hauptspeicher hat er die Form „R*xxx*", wobei jedes *x* eine Ziffer von 0 bis 9 darstellt. (Das Register *xxx* wird später das Statusregister des Feldes beinhalten, und die darauffolgenden Register die Feldelemente.) Für Felder im erweiterten Speicherbereich gilt jede, bis zu sieben Buchstaben lange Zeichenkette. Der Name „X" darf nicht verwendet werden; er bezeichnet bei einigen Matrizenfunktionen das X-Register als Operanden.
Wird kein Feldname angegeben (ALPHA-Register leer), so wird das **aktive XM-File** angesprochen. Das Arbeits-File muß in diesem Fall den Filetyp „M" haben.

X-Register : *mmm,nnn* (Felddimensionen)

mmm gibt die Anzahl der Reihen und *nnn* die Anzahl der Spalten an. Wird für *m* oder *n* der Wert 0 angegeben, so wird er durch 1 ersetzt.

BEMERKUNG:
ROT= FILE WIRD AKTUELL

Beispiele

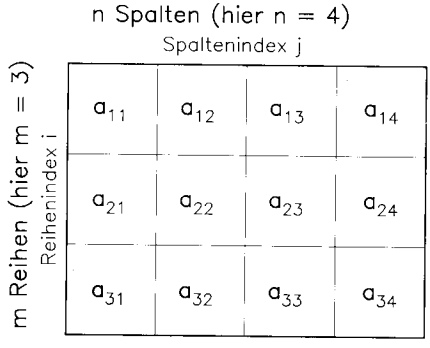
Um im erweiterten Speicherbereich ein Feld mit Namen „MATRIX“ und 3∗4 Elementen anzulegen, benötigt man folgende Sequenz:

‘MATRIX
3,004
MDIM

Das gleiche Feld im Hauptspeicher, beginnend mit Register 10, wird mit

‘R010
3,004
MDIM

angelegt, wobei Register 10 das Statusregister des Feldes ist und die Register 11 bis 22 die Feldelemente beinhalten. Das angelegte Feld hat die Elemente a_{11} bis a_{34}:



Mit

```
'MATRIX
4,004
MDIM
```

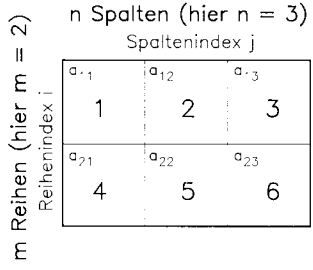
wird das bereits angelegte Feld „MATRIX“ redimensioniert; es hat nun die Elemente a_{11} bis a_{44}.

Weitere Hinweise

Beim Neuanlegen eines Feldes werden alle Elemente des Datenfeldes auf 0 gesetzt (also gelöscht). Dies gilt sowohl für Felder im erweiterten Speicherbereich als auch für RAM-Felder. Beim Redimensionieren von Feldern werden nur die neu hinzugekommenen Elemente auf 0 gesetzt; alle Elemente des alten Feldes bleiben erhalten. Beim Verkleinern eines Datenfeldes gehen natürlich alle überzähligen Elemente verloren.

Die Werte von Feldelementen werden reihenweise gespeichert, d.h. auf den Wert der ersten Reihe und der letzten Spalte folgt der Wert der zweiten Reihe und der ersten Spalte. Bei der Neudimensionierung eines Feldes bleibt zwar die *Reihenfolge* der Elemente erhalten, in der Regel wird deren *Position* aber verändert (wenn die Dimension sich ändert).

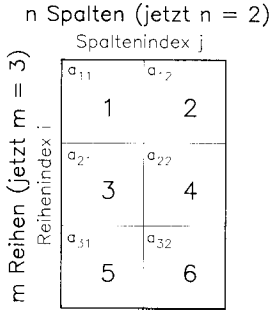
Beispiel: Feld „A“ ist mit der Dimension 2*3 deklariert und enthält die Werte 1 bis 6 in folgender Anordnung:



Nach der Redimensionierung mittels

```
'A
3,002
MDIM
```

besteht folgende Ordnung:



Wird beim Redimensionieren das Feld vergrößert, so werden die neu hinzugekommenen Elemente mit dem Wert 0 initialisiert. Wird das Feld dabei verkürzt, so gehen die überschüssigen Elemente verloren, falls es sich um ein Feld im erweiterten Speicherbereich handelt. Bei einem RAM-Feld wird lediglich die Dimension, die im Statusregister gespeichert ist, verändert – die Register mit ihren Inhalten bleiben erhalten.

Verwandte Funktion

DIM

DIM

DIM dient zur Abfrage der Dimensionen eines Feldes.

Eingabeparameter

ALPHA-Register : Feldname
Siehe unter MDIM: Eingabeparameter Feldname.

Ausgabe

X-Register : *mmm,nnn* (Diese Zahl stellt die Dimension des Feldes dar. *mmm* gibt die Anzahl der Reihen und *nnn* die Anzahl der Spalten an.)

Weitere Hinweise

Wurden beim Anlegen eines Feldes *m* oder *n* mit 0 angegeben, so wird der Ersatzwert 1 benutzt. Dies spiegelt sich in der Ausgabe von DIM wieder, wie folgendes Beispiel zeigt:

'R000 CLX
MDIM
DIM

Als Ergebnis sehen wir die Zahl 1,001 im X-Register.

Verwandte Funktion

MDIM

Funktionen zur Manipulation des Elementzeigers

?IJ

Die Funktion ?IJ schreibt die Zeigerposition *iii,jjj* des **aktuellen** Datenfeldes ins X-Register (der Stack wird angehoben).

Eingabeparameter

Keine

Beispiel

Bei einer bestehenden Matrix mit dem Namen „MAT" wird der Elementzeiger auf das zweite Element der ersten Reihe gesetzt. Dies geschieht mit der Tastenfolge:

'MAT
1,002
IJ=A

Im weiteren Verlauf eines Programms soll nun die Zeigerposition abgefragt werden. Dies geschieht einfach mit:

?IJ

Die Funktion ?IJ übergibt in diesem Fall den Wert 1,002 ins X-Register.

Weitere Hinweise

Die Funktion ?IJ benötigt keine ALPHA-Eingabe. Es wird immer die Zeigerposition des *aktuellen* Datenfeldes ausgegeben!

Verwandte Funktionen

?IJA, IJ=, IJ=A

?IJA

Die Funktion **?IJA** schreibt die Zeigerposition *iii,jjj* des **angegebenen** Datenfeldes in das X-Register (der Stack wird angehoben).

Eingabeparameter

ALPHA-Register : Name des Datenfeldes

Beispiel

Nach dem Dimensionieren einer Matrix soll überprüft werden, auf welchem Element der Elementzeiger dieser Matrix steht. Weiterhin soll sie die *aktuelle* Matrix werden. Dies geschieht z.B. mit:

```
'MAT
3,003
MDIM
?IJA
```

Die Matrix „MAT" ist nun die *aktuelle* Matrix. Da der Wert 1,001 ins X-Register übergeben wurde, steht der Elementzeiger bei einer Neudimensionierung auf dem Element a_{11}.

Weitere Hinweise

Nach Ausführung der Funktion **?IJA** ist das angegebene Datenfeld auch zum *aktuellen* Datenfeld geworden. Die Information über das *aktuelle* Datenfeld wird im Ein-/Ausgabebuffer des CCD-Moduls gespeichert, d.h., wenn keine freien Register für diesen Buffer vorhanden sind, erfolgen die Fehlermeldungen „PACKING" und „TRY AGAIN". Wird kein Name angegeben, dann erscheint die Fehlermeldung „NAME ERR".

Verwandte Funktionen

?IJ, **IJ=**, **IJ=A**

IJ=

Durch die Funktion IJ= wird der Zeiger des *aktuellen* Datenfeldes auf das Element mit dem Reihenindex *iii* und dem Spaltenindex *jjj* gestellt.

Eingabeparameter

X-Register : *iii,jjj*

Beispiel

Der Zeiger des aktuellen Datenfeldes soll auf das Element a_{12} positioniert werden, damit der Wert dieses Elementes ausgelesen werden kann. Dies geschieht mit der Tastenfolge:

1,002
IJ=
C› +

Weitere Hinweise

Wenn kein *aktuelles* Datenfeld vorhanden ist (z.B. durch Löschen des I/O-Buffers), wird die Fehlermeldung „NONEXISTENT" angezeigt.

Verwandte Funktionen

?IJ, ?IJA, IJ=A

IJ=A

Durch die Funktion IJ=A wird der Zeiger des **angegebenen** Datenfeldes auf das Element mit dem Reihenindex *iii* und dem Spaltenindex *jjj* gestellt.

Eingabeparameter

X-Register : *iii,jjj*
ALPHA-Register : Name des Datenfeldes

Beispiel

Nach der Dimensionierung einer Matrix mit Namen „R010" soll der Elementzeiger auf das letzte Element dieser Matrix positioniert werden. Dies geschieht z. B. mit folgenden Schritten:

```
'R000
5,005
MDIM
IJ=A
```

Weitere Hinweise

Nach Ausführung der Funktion IJ=A ist das angegebene Datenfeld auch zum *aktuellen* Datenfeld geworden. Die Information über das *aktuelle* Datenfeld ist im Ein-/Ausgabebuffer des CCD-Moduls gespeichert, d.h., wenn keine freien Register für diesen Buffer vorhanden sind, erfolgen die Fehlermeldungen „PACKING" und „TRY AGAIN". Wird kein Name angegeben, dann erscheint die Fehlermeldung „NAME ERR".

Verwandte Funktionen

?IJ, ?IJA, IJ=

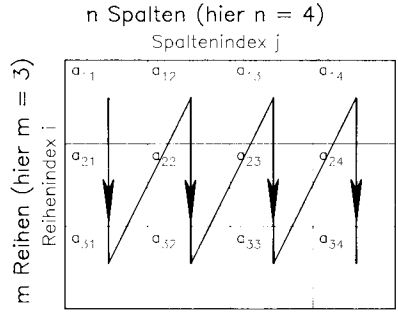
Ein-/Ausgabefunktionen für Datenfelder

›C+, **›R+**, **C›+**, **C›-**, **R›+**, **R›-**

Diese sechs Funktionen sind die Ein- und Ausgabefunktionen des CCD-Moduls für alle Datenfelder. Eingegeben oder ausgelesen wird immer auf der aktuellen Zeigerposition (siehe **IJ=** und **IJ=A**). Für die Namensgebung dieser Funktionen wurden folgende Regeln aufgestellt:
Ein voranstehendes „>"-Zeichen bedeutet Eingabefunktion: Das Element auf der aktuellen Zeigerposition wird mit dem X-Wert überschrieben.
Ein nachstehendes „>"-Zeichen bedeutet Ausgabefunktion: Der Stack wird angehoben und das Element auf der aktuellen Zeigerposition wird ins X-Register geschrieben.
Das „+"-Zeichen bedeutet, daß der Zeiger anschließend um eine Position inkrementiert wird, das „-"-Zeichen, daß er anschließend dekrementiert wird.
Der Buchstabe „C" (column) gibt an, daß spaltenweise eingegeben oder ausgelesen werden soll, der Buchstabe „R" (row) gibt an, daß reihenweise vorgegangen werden soll.

›C+

Die Funktion **›C+** ermöglicht die Eingabe von Datenfeldelementen Spalte für Spalte, d.h., bei einem Feld mit *i* Reihen und *j* Spalten wird nach der Elementeingabe *i* um 1 erhöht. Die Reihenfolge der Elementeingabe wird auch in der folgenden Abbildung deutlich:



Eingabeparameter

X-Register: einzugebendes Feldelement (auch ALPHA-Daten!)

Beispiel

Folgendes Programm zeigt, wie man auf einfache Weise die Elemente einer Matrix spaltenweise eingibt.

```
01◆LBL  "CEI
          N"
02  "R010"
03  CLX
04  IJ=A

05◆LBL  01
06  STOP
07  >C+
08  GTO  01
09  END
```

Ein entsprechendes Programm zur Ausgabe der Elemente einer Matrix ist unter C› + beschrieben.

Weitere Hinweise

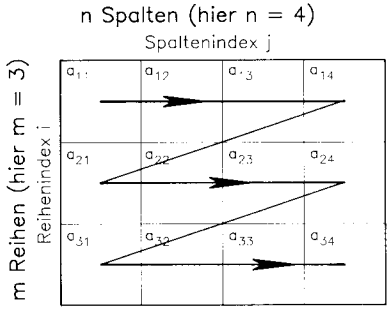
Hat man das letzte Element einer Matrix eingegeben, ist der Zeiger anschließend auf ein nicht existentes Element positioniert. Versucht man nun, weitere Elemente einzugeben oder auszulesen, erfolgt die Fehlermeldung „END OF FL".

Verwandte Funktionen

›R +, C› +, C›-, R› +, R›-

›R+

Die Funktion **›R+** ermöglicht die Eingabe von Datenfeldelementen Reihe für Reihe, d.h., bei einem Feld mit *i* Reihen und *j* Spalten wird nach der Elementeingabe *j* um 1 erhöht. Die Reihenfolge der Elementeingabe wird auch in der folgenden Abbildung deutlich:



Eingabeparameter

X-Register: einzugebendes Feldelement (auch ALPHA-Daten!)

Beispiel

Folgendes Programm zeigt, wie man auf einfache Weise die Elemente einer Matrix reihenweise eingibt.

```
01◆LBL "REI
         N"
02  "R010"
03  CLX
04  IJ=A

05◆LBL 01
06  STOP
07  >R+
08  GTO 01
09  END
```

Ein entsprechendes Programm zur Ausgabe der Elemente einer Matrix ist unter R› + beschrieben.

Weitere Hinweise

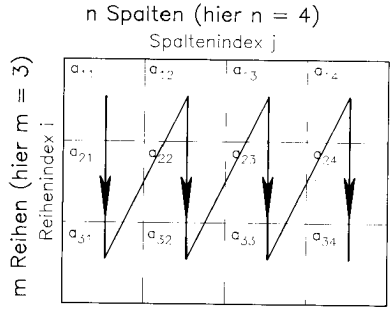
Hat man das letzte Element einer Matrix eingegeben, ist der Zeiger anschließend auf ein nicht existentes Element positioniert. Versucht man nun, weitere Elemente einzugeben oder auszulesen, erfolgt die Fehlermeldung „END OF FL".

Verwandte Funktionen

›C +, C› +, C›-, R› +, R›-

C›+

Die Funktion **C›+** ermöglicht die Ausgabe von Datenfeldelementen Spalte für Spalte, d.h., bei einem Feld mit *i* Reihen und *j* Spalten wird nach der Elementausgabe *i* um 1 erhöht. Der Stack wird nach oben geschoben und das Feldelement nach X geschrieben. Die Reihenfolge der Elementausgabe wird auch in der folgenden Abbildung deutlich:



Eingabeparameter

keine

Beispiel

Folgendes Programm zeigt, wie man auf einfache Weise die Elemente einer Matrix spaltenweise ausliest.

```
01*LBL  "CAU
          S"
02  "R010"
03  CLX
04  I.J=A

05*LBL  01
06  C>+
07  PSE
08  GTO  01
09  END
```

Weitere Hinweise

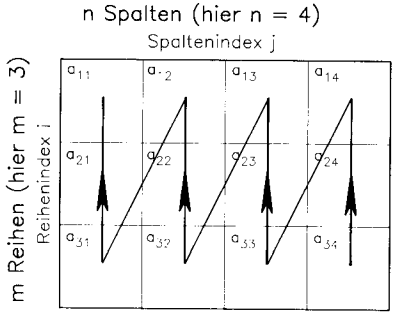
Hat man das letzte Element einer Matrix ausgegeben, ist der Zeiger anschließend auf ein nicht existentes Element positioniert. Versucht man nun, weitere Elemente auszulesen, erfolgt die Fehlermeldung „END OF FL".

Verwandte Funktionen

›C+, ›R+, C›-, R›+, R›-

C>-

Die Funktion C>- ermöglicht die Ausgabe von Datenfeldelementen Spalte für Spalte, d.h., bei einem Feld mit *i* Reihen und *j* Spalten wird nach der Elementausgabe *i* um 1 vermindert. Der Stack wird nach oben geschoben und das Feldelement nach X geschrieben. Die Reihenfolge der Elementausgabe wird auch in der folgenden Abbildung deutlich:



Eingabeparameter

keine

Beispiel

Folgendes Programm zeigt, wie man auf einfache Weise die Elemente einer Matrix spaltenweise rückwärts ausgibt.

```
01◆LBL  "CRA
         US"
02  "R010"
03  DIM
04  IJ=A

05◆LBL  01
06  C>-
07  PSE
08  GTO  01
09  END
```

Weitere Hinweise

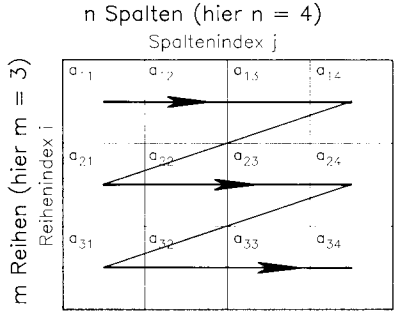
Hat man das letzte (erste) Element einer Matrix ausgegeben, ist der Zeiger anschließend auf ein nicht existentes Element positioniert (hier auf das Element 0,000). Versucht man nun, weitere Elemente auszulesen, erfolgt die Fehlermeldung „END OF FL".

Verwandte Funktionen

›C+, ›R+, C›+, R›+, R›-

R↓+

Die Funktion **R↓+** ermöglicht die Ausgabe von Datenfeldelementen Reihe für Reihe, d.h., bei einem Feld mit *i* Reihen und *j* Spalten wird nach der Elementausgabe *j* um 1 erhöht. Der Stack wird nach oben geschoben und das Feldelement nach X geschrieben. Die Reihenfolge der Elementausgabe wird auch in der folgenden Abbildung deutlich:



Eingabeparameter

keine

Beispiel

Folgendes Programm zeigt, wie man auf einfache Weise die Elemente einer Matrix reihenweise ausgibt.

```
01◆LBL  "RAU
          S"
02  "R010"
03  CLX
04  IJ=A

05◆LBL  01
06  R↓+
07  PSE
08  GTO  01
09  END
```

Weitere Hinweise

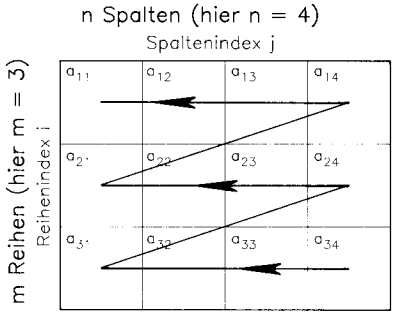
Hat man das letzte Element einer Matrix ausgegeben, ist der Zeiger anschließend auf ein nicht existentes Element positioniert. Versucht man nun, weitere Elemente auszulesen, erfolgt die Fehlermeldung „END OF FL".

Verwandte Funktionen

›C+, ›R+, C›+, C›-, R›-

R›-

Die Funktion R›- ermöglicht die Ausgabe von Datenfeldelementen Reihe für Reihe, d.h., bei einem Feld mit *i* Reihen und *j* Spalten wird nach der Elementausgabe *j* um 1 vermindert. Der Stack wird nach oben geschoben und das Feldelement nach X geschrieben. Die Reihenfolge der Elementausgabe wird auch in der folgenden Abbildung deutlich:



Eingabeparameter

keine

Beispiel

Folgendes Programm zeigt, wie man auf einfache Weise die Elemente einer Matrix reihenweise rückwärts ausliest.

```
01◆LBL  "RRA
         US"
02  "R010"
03  DIM
04  IJ=A

05◆LBL  01
06  R>-
07  PSE
08  GTO  01
09  END
```

Weitere Hinweise

Hat man das letzte (erste) Element einer Matrix ausgegeben, ist der Zeiger anschließend auf ein nicht existentes Element positioniert. Versucht man nun, weitere Elemente auszulesen, erfolgt die Fehlermeldung „END OF FL".

Verwandte Funktionen

Funktionen zum Verschieben und Vertauschen von Elementen

C‹›C

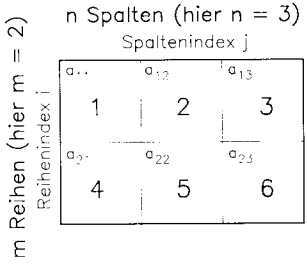
Die Funktion **C‹›C** vertauscht die Spalte *kkk* mit der Spalte *lll* des angegebenen Datenfeldes.

Eingabeparameter

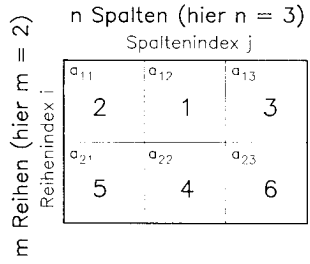
X-Register : *kkk,lll*
ALPHA-Register : Feldname

Beispiel

Die erste Spalte des abgebildeten Datenfeldes soll mit der zweiten vertauscht werden; das ursprüngliche Datenfeld ist:



Nach der Befehlsfolge 2,001 C↔C ergibt sich folgendes Bild:



Verwandte Funktionen

R↔R, MOVE, SWAP

R‹›R

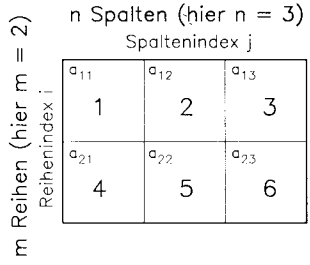
Die Funktion **R‹›R** vertauscht die Reihe *kkk* mit der Reihe *lll* des angegebenen Datenfeldes.

Eingabeparameter

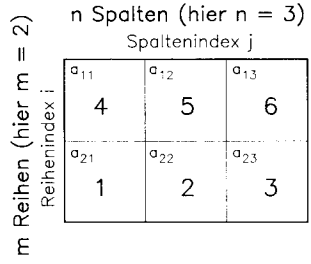
X-Register : *kkk,lll*
ALPHA-Register : Feldname

Beispiel

Die erste Reihe des abgebildeten Datenfeldes soll mit der zweiten vertauscht werden; das ursprüngliche Datenfeld ist:



Nach der Befehlsfolge 2,001 **R‹›R** ergibt sich folgendes Bild:



Weitere Hinweise

Das Vertauschen von Reihen geschieht z.B. bei der Pivotisierung einer Matrix.

Verwandte Funktionen

CoC, MOVE, SWAP

MOVE

Die Funktion **MOVE** kopiert alle angegebenen Elemente des spezifizierten Datenfeldes in ein anderes Datenfeld. Es können nur „rechteckige" Teilblöcke angegeben werden; der Block wird durch Angabe zweier sich gegenüberliegender Eckelemente *iii,jjj* und *kkk,lll* bestimmt. Im Zielfeld muß die obere linke Ecke, *mmm,nnn* des Zielblockes angegeben werden.

Eingabeparameter

X-Register : *iii,jjj*
Y-Register : *kkk,lll*
Z-Register : *mmm,nnn*
ALPHA-Register : *OP1,RES*

Beispiel

Wir wollen die Elemente a_{32}, a_{33}, a_{42} und a_{43} des abgebildeten 4∗3 Datenfeldes „A" in das Datenfeld „B" kopieren. Die Zielelemente im Feld „B" sollen b_{22}, b_{23}, b_{32} und b_{33} sein. Die Datenfelder „A" und „B" sind hier abgebildet:

Matrix A

n Spalten (hier n = 3)
Spaltenindex j

m Reihen (hier m = 4)
Reihenindex i

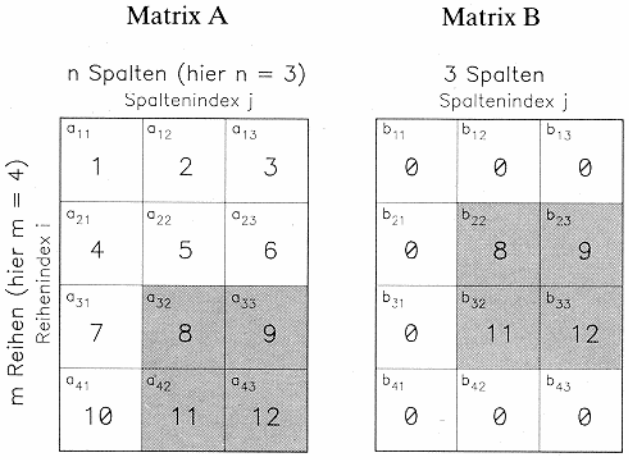
a_{11} 1	a_{12} 2	a_{13} 3
a_{21} 4	a_{22} 5	a_{23} 6
a_{31} 7	a_{32} 8	a_{33} 9
a_{41} 10	a_{42} 11	a_{43} 12

Matrix B

3 Spalten
Spaltenindex j

b_{11} 0	b_{12} 0	b_{13} 0
b_{21} 0	b_{22} 0	b_{23} 0
b_{31} 0	b_{32} 0	b_{33} 0
b_{41} 0	b_{42} 0	b_{43} 0

Die Datenfelder können natürlich auch verschiedene Dimensionen haben. Nach der Befehlsfolge **ALPHA** *OP1,RES* **ALPHA** 2,002 **ENTER** 4,003 **ENTER** 3,002 **MOVE** ergibt sich folgendes Bild der Datenfelder „A" und „B":



Das Feld „A" wurde überhaupt nicht verändert, im Feld „B" stehen die Elemente an der gewünschten Stelle.

Weitere Hinweise

Der zu verschiebende Block kann auf mehrere Arten angegeben werden:

1) das obere linke und das untere rechte Element oder
2) das untere linke und das obere rechte Element

Außerdem dürfen das X- und das Y-Register vertauscht werden, d.h. *kkk,lll* darf im X-Register und *iii,jjj* im Y-Register angegeben werden.

Verwandte Funktionen

CoC, **RoR**, **SWAP**

SWAP

Die Funktion **SWAP** vertauscht alle angegebenen Elemente des spezifizierten Datenfeldes mit den Elementen des anderen Datenfeldes. Es können nur „rechteckige" Teilblöcke angegeben, werden, der Block wird durch Angabe zweier sich gegenüberliegender Eckelemente *iii,jjj* und *kkk,lll* bestimmt. Vom zweiten Feld muß die obere linke Ecke *mmm,nnn* des zweiten Blockes angegeben werden.

Eingabeparameter

X-Register : *iii,jjj*
Y-Register : *kkk,lll*
Z-Register : *mmm,nnn*
ALPHA-Register : *OP1,OP2*

Beispiel

Wir wollen die Elemente a_{32}, a_{33}, a_{42} und a_{43} des abgebildeten 4*3 Datenfeldes „A" mit den Elementen b_{22}, b_{23}, b_{32} und b_{33} des Datenfeldes „B" vertauschen. Die Datenfelder „A" und „B" sind hier abgebildet:

Matrix A

n Spalten (hier n = 3)
Spaltenindex j

m Reihen (hier m = 4)
Reihenindex i

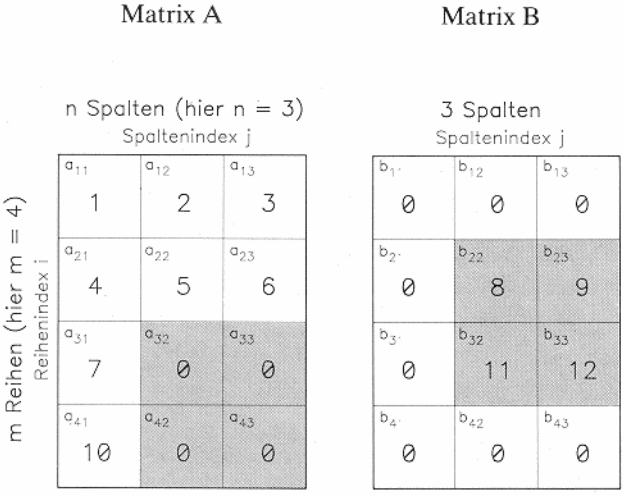
a_{11} 1	a_{12} 2	a_{13} 3
a_{21} 4	a_{22} 5	a_{23} 6
a_{31} 7	a_{32} 8	a_{33} 9
a_{41} 10	a_{42} 11	a_{43} 12

Matrix B

3 Spalten
Spaltenindex j

b_{11} 0	b_{12} 0	b_{13} 0
b_{21} 0	b_{22} 0	b_{23} 0
b_{31} 0	b_{32} 0	b_{33} 0
b_{41} 0	b_{42} 0	b_{43} 0

Die Datenfelder können natürlich auch verschiedene Dimensionen haben. Nach der Befehlsfolge **ALPHA** *OP1,RES* **ALPHA** 2,002 **ENTER** 4,003 **ENTER** 3,002 **SWAP** ergibt sich folgendes Bild der Datenfelder „A“ und „B“:



Die angegebenen Elemente der Felder „A“ und „B“ wurden vertauscht.

Weitere Hinweise

Der auszutauschende Block kann auf mehrere Arten angegeben werden:

1) das obere linke und das untere rechte Element oder
2) das untere linke und das obere rechte Element

Außerdem dürfen das X- und das Y-Register vertauscht werden, d.h. *kkk,lll* darf im X-Register und *iii,jjj* im Y-Register angegeben werden.

Verwandte Funktionen

C◦C, **R◦R**, **MOVE**

Funktionen zur Suche von Extremwerten der Feldelemente

Diese Funktionen werden für numerische Lösungsverfahren und Algorithmen benötigt. Beschreibungen dieser Verfahren finden sich in der Fachliteratur.

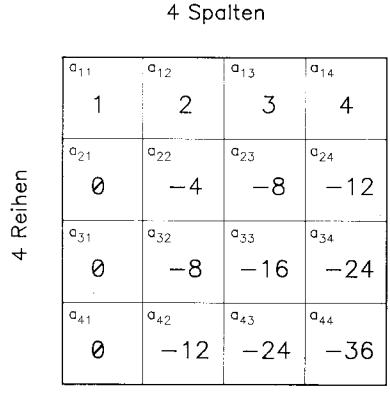
MAX

Die Funktion **MAX** stellt den Elementzeiger auf das größte Element des angegebenen Datenfeldes. Dieses Element wird außerdem im X-Register ausgegeben.

Eingabeparameter

ALPHA-Register : *OP1*

Beispiel



Bei Anwendung der Funktion **MAX** auf das oben dargestellte Datenfeld wird der Elementzeiger auf das Element a_{14} gestellt und dessen Wert 4 ausgegeben.

Weitere Hinweise

Die Funktion MAX gibt das größte Element aus und **nicht**, im Gegensatz zu MAXAB, das absolut größte Element. Ist kein Element größer als das Element a_{11} (z.B. alle Feldelemente gleich 0), so wird dieses ausgegeben. Der Elementzeiger ist dann ebenfalls auf das Element a_{11} positioniert.

Verwandte Funktionen

CMAXAB, MAX, MAXAB, MIN, PIV, R›R?, RMAXAB

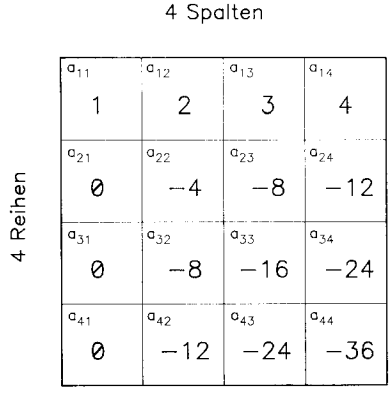
MAXAB

Die Funktion **MAXAB** stellt den Elementzeiger auf das absolut größte Element des angegebenen Datenfeldes. Der Absolutbetrag dieses Elementes wird außerdem im X-Register ausgegeben.

Eingabeparameter

ALPHA-Register : *OP1*

Beispiel



Das absolut größte Element des dargestellten Feldes ist -36, die Funktion **MAXAB** stellt den Elementzeiger auf das Element a_{44} und gibt dessen Absolutbetrag (also 36) im X-Register aus.

Weitere Hinweise

Das im X-Register ausgegebene Element ist nur gleich dem Element auf der angegebenen Position, wenn dieses Element positiv ist. Ist das Element negativ, so wird sein Betrag im X-Register abgelegt. Ist kein Element absolut größer als das Element a_{11} (z.B. alle Feldelemente gleich 0), so wird dieses ausgegeben. Der Elementzeiger ist dann ebenfalls auf das Element a_{11} positioniert.

Verwandte Funktionen

CMAXAB, MAX, MIN, PIV, R↓R?, RMAXAB

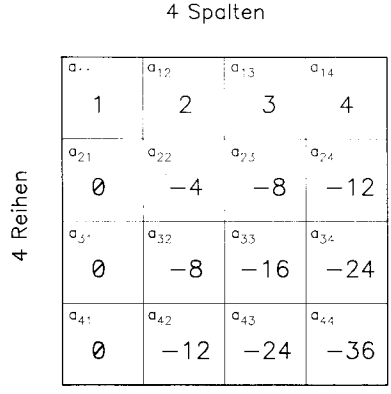
CMAXAB

Die Funktion **CMAXAB** stellt den Elementzeiger auf das absolut größte Element der spezifizierten Spalte *jjj* des angegebenen Datenfeldes. Dieses Element wird außerdem im X-Register ausgegeben.

Eingabeparameter

X-Register : *jjj*
ALPHA-Register : *OP1*

Beispiel



Nach der Tastenfolge 3 **CMAXAB** ist der Elementzeiger auf das Element a_{43} positioniert und dessen Absolutwert 24 im X-Register abgelegt.

Weitere Hinweise

Die Suche nach dem absolut größten Element geschieht nur in der angegebenen Spalte. Ist kein Element der Spalte *j* absolut größer als das Element a_{1j} (z.B. alle Spaltenelemente gleich 0), so wird dieses ausgegeben. Der Elementzeiger ist dann ebenfalls auf das Element a_{1j} positioniert.

Verwandte Funktionen

MAX, **MAXAB**, **MIN**, **PIV**, **R↓R?**, **RMAXAB**

RMAXAB

Die Funktion RMAXAB stellt den Elementzeiger auf das absolut größte Element der spezifizierten Reihe *iii* des angegebenen Datenfeldes. Dieses Element wird außerdem im X-Register ausgegeben.

Eingabeparameter

X-Register : *iii*
ALPHA-Register : *OP1*

Beispiel

4 Spalten

4 Reihen

a_{11} 1	a_{12} 2	a_{13} 3	a_{14} 4
a_{21} 0	a_{22} −4	a_{23} −8	a_{24} −12
a_{31} 0	a_{32} −8	a_{33} −16	a_{34} −24
a_{41} 0	a_{42} −12	a_{43} −24	a_{44} −36

Nach der Tastenfolge 2 RMAXAB ist der Elementzeiger auf a_{24} gestellt und die Zahl 12 im X-Register zu sehen.

Weitere Hinweise

Die Suche nach dem absolut größten Element geschieht nur in der angegebenen Reihe. Ist kein Element der Reihe *i* absolut größer als das Element a_{il} (z.B. alle Reihenelemente gleich 0), so wird dieses ausgegeben. Der Elementzeiger ist dann ebenfalls auf das Element a_{il} positioniert.

Verwandte Funktionen

CMAXAB, MAX, MAXAB, MIN, PIV, R›R?

MIN

Die Funktion MIN stellt den Elementzeiger auf das kleinste Element des angegebenen Datenfeldes. Dieses Element wird außerdem im X-Register ausgegeben.

Eingabeparameter

ALPHA-Register : *OP1*

Beispiel

4 Spalten

4 Reihen

a_{11} 1	a_{12} 2	a_{13} 3	a_{14} 4
a_{21} 0	a_{22} −4	a_{23} −8	a_{24} −12
a_{31} 0	a_{32} −8	a_{33} −16	a_{34} −24
a_{41} 0	a_{42} −12	a_{43} −24	a_{44} −36

Nach Ausführung der Funktion MIN ist der Elementzeiger auf a_{44} positioniert und der Wert -36 im X-Register abgelegt.

Weitere Hinweise

Ist a_{11} das kleinste Element des Feldes (z.B. alle Feldelemente gleich 0), so wird dieses ausgegeben. Der Elementzeiger ist dann ebenfalls auf das Element a_{11} positioniert.

Verwandte Funktionen

CMAXAB, MAX, MAXAB, PIV, R↓R?, RMAXAB

PIV

Die Funktion **PIV** stellt den Elementzeiger auf das absolut größte Element auf bzw. unter der Hauptdiagonalen der angegebenen Spalte *jjj*. Der Absolutbetrag dieses Elementes wird außerdem im X-Register ausgegeben.

Eingabeparameter

X-Register : *jjj*
ALPHA-Register : *OP1*

Beispiel

4 Spalten

4 Reihen

a_{11} 1	a_{12} 2	a_{13} 3	a_{14} 4
a_{21} 5	a_{22} 6	a_{23} 7	a_{24} 8
a_{31} 9	a_{32} 10	a_{33} 11	a_{34} 12
a_{41} 0	a_{42} −12	a_{43} −24	a_{44} −36

Mit der Tastenfolge 2 **PIV** wird der Elementzeiger auf das Element a_{42} gestellt und der Absolutbetrag dieses Elementes (also 12) im X-Register abgelegt.

Weitere Hinweise

Die Funktion **PIV** verwendet nur den Integerteil des X-Registers. Der Nachkommateil wird bei der Ausführung nicht beachtet. Die Suche nach dem absolut größten Element beginnt mit dem Element a_{jj} (also auf der Hauptdiagonalen) und wird dann in der Spalte *j* nach unten fortgesetzt.

Verwandte Funktionen

CMAXAB, **MAX**, **MAXAB**, **MIN**, **R↓R?**, **RMAXAB**

R>R?

Die Funktion **R>R?** vergleicht die Elemente der Reihe *kkk* mit denen der Reihe *lll* des angegebenen Datenfeldes. Hierbei werden die Elemente spaltenweise verglichen, beginnend mit Spalte eins. Sind die Elemente der gerade bearbeiteten Spalte gleich, so werden die beiden Elemente der nächsten Spalte verglichen, bis entweder zwei ungleiche Elemente gefunden werden oder das Ende der Reihe erreicht ist. Die Antwort ist „YES", sobald ein Element der Reihe *lll* größer als das Element in der gleichen Spalte der Reihe *kkk* ist. Wird ein Element der Reihe *lll* gefunden, welches kleiner als das entsprechende Element der Reihe *kkk* ist, so ist die Antwort „NO".

Eingabeparameter

X-Register : *kkk,lll*
ALPHA-Register : *OP1*

Beispiel

Es soll geprüft werden, ob die erste Reihe des abgebildeten Datenfeldes größer ist als die dritte:

4 Spalten

4 Reihen

a_{11} 1	a_{12} 2	a_{13} 3	a_{14} 4
a_{21} 5	a_{22} 6	a_{23} 7	a_{24} 8
a_{31} 9	a_{32} −8	a_{33} −16	a_{34} −24
a_{41} 13	a_{42} −12	a_{43} −24	a_{44} −36

Nach der Tastenfolge 1,003 R›R? erhalten wir als Antwort „NO“ in der Anzeige, da bereits in der ersten Spalte das Element aus Reihe 3 größer als das aus Reihe 1 ist.

Weitere Hinweise

Im Programmlauf wird bei der Antwort „YES“ die nächste Programmzeile ausgeführt, ansonsten die übernächste.

Verwandte Funktionen

MAX, MAXAB, MIN, CMAXAB, RMAXAB, PIV

Summen und Normen

Diese Funktionen werden, wie die Funktionen zur Extremwertsuche, für numerische Lösungsverfahren und Algorithmen benötigt. Beschreibungen dieser Verfahren finden sich in der Fachliteratur. Mit Hilfe von Normen läßt sich z.B. feststellen, ob ein Gleichungssystem singulär ist, d.h., ob es eindeutig lösbar ist oder nicht. Außerdem erhält man durch die Normen eine Aussage über die Zuverlässigkeit eines Ergebnisses; man spricht dann von der Konditionszahl für die Matrix.

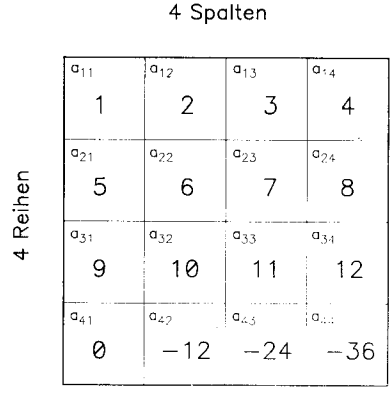
SUM

SUM addiert alle Elemente des angegebenen Datenfeldes und schreibt das Ergebnis in das X-Register.

Eingabeparameter

ALPHA-Register : *OP1*

Beispiel



Nach der Tastenfolge ALPHA *OP1* ALPHA SUM ist die Summe über alle Feldelemente, also 6, im X-Register abgelegt.

Weitere Hinweise

Die Funktion SUM hat keinen Einfluß auf den Elementzeiger, seine Stellung wird also nicht verändert.

Verwandte Funktionen

SUMAB, CSUM, RSUM

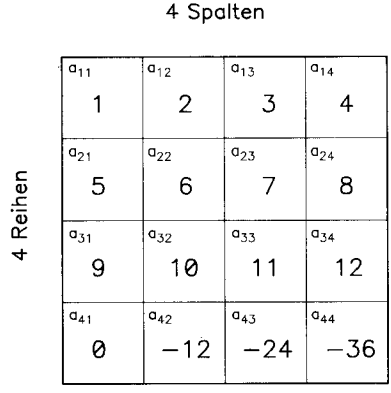
SUMAB

SUMAB addiert die Absolutwerte aller Elemente des angegebenen Datenfeldes zueinander und schreibt das Ergebnis in das X-Register.

Eingabeparameter

ALPHA-Register : *OP1*

Beispiel



Nach der Tastenfolge ALPHA *OP1* ALPHA SUMAB ist die Summe über die Absolutwerte aller Feldelemente, also 150, im X-Register abgelegt.

Weitere Hinweise

Die Funktion SUMAB hat keinen Einfluß auf den Elementzeiger, seine Stellung wird also nicht verändert.

Verwandte Funktionen

SUM, CSUM, RSUM

CSUM

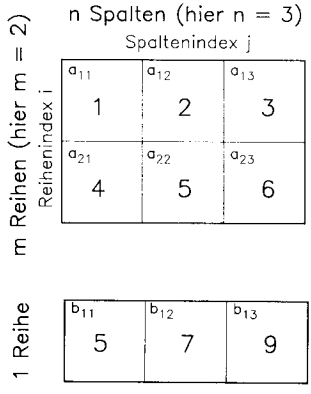
CSUM addiert alle Elemente jeder Spalte des angegebenen Datenfeldes zueinander und schreibt die Ergebnisse in das zweite angegebene Datenfeld.

Eingabeparameter

ALPHA-Register : *OP1,RES*

Beispiel

Um die Spaltensummen des abgebildeten Datenfeldes „A" (2 Reihen und 3 Spalten) zu bilden, benötigen wir ein weiteres Datenfeld, welches auch 3 Spalten, aber nur eine Reihe haben muß. Nach der Tastenfolge ALPHA *OP 1,RES* ALPHA CSUM enthält das Datenfeld „B" die jeweiligen Spaltensummen:



Weitere Hinweise

CSUM benötigt zur Ablage des Ergebnisses immer ein Datenfeld, welches die gleiche Spaltenanzahl besitzt wie das Datenfeld, dessen Spaltensummen berechnet werden sollen.

Verwandte Funktionen

SUM, SUMAB, RSUM

RSUM

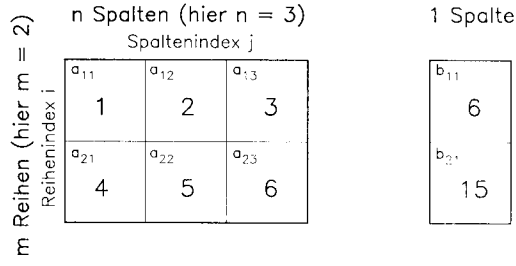
RSUM addiert alle Elemente jeder Reihe des angegebenen Datenfeldes und schreibt die Ergebnisse in das zweite angegebene Datenfeld.

Eingabeparameter

ALPHA-Register : *OP1,RES*

Beispiel

Um die Reihensummen des abgebildeten Datenfeldes „A" (2 Reihen und 3 Spalten) zu bilden, benötigen wir ein weiteres Datenfeld, welches auch 2 Reihen, aber nur eine Spalte haben muß. Nach der Tastenfolge ALPHA *OP 1,RES* ALPHA RSUM enthält das Datenfeld „B" die jeweiligen Reihensummen:



Weitere Hinweise

RSUM benötigt zur Ablage des Ergebnisses immer ein Datenfeld, welches die gleiche Reihenanzahl besitzt wie das Datenfeld, dessen Reihensummen berechnet werden sollen.

Verwandte Funktionen

SUM, SUMAB, CSUM

CNRM

Die Funktion CNRM berechnet die Spaltensummenmaximumnorm des angegebenen Datenfeldes. Das Ergebnis wird im X-Register abgelegt. Die Formel hierzu lautet:

$$\| A \|_C = \max_{1 \leq j \leq n} \sum_{i=1}^{m} | a_{ij} |$$

Eingabeparameter

ALPHA-Register : *OP1*

Beispiel

4 Spalten

4 Reihen

a_{11} 1	a_{12} 2	a_{13} 3	a_{14} 4
a_{21} 5	a_{22} 6	a_{23} 7	a_{24} 8
a_{31} 9	a_{32} 10	a_{33} 11	a_{34} 12
a_{41} 0	a_{42} −12	a_{43} −24	a_{44} −36

Die Tastenfolge ALPHA *OP1* ALPHA CNRM ergibt den Wert 60 im X-Register; dies ist die Spaltensumme der Absolutwerte der vierten Spalte des abgebildeten Datenfeldes.

Verwandte Funktionen

RNRM, FNRM

RNRM

Die Funktion **RNRM** berechnet die Reihensummenmaximumnorm des angegebenen Datenfeldes. Das Ergebnis wird in das X-Register geschrieben. Die Formel hierzu lautet:

$$\| A \|_R = \max_{1 \leq i \leq m} \sum_{j=1}^{n} | a_{ij} |$$

Eingabeparameter

ALPHA-Register : *OP1*

Beispiel

4 Spalten

4 Reihen

a_{11} 1	a_{12} 2	a_{13} 3	a_{14} 4
a_{21} 5	a_{22} 6	a_{23} 7	a_{24} 8
a_{31} 9	a_{32} 10	a_{33} 11	a_{34} 12
a_{41} 0	a_{42} −12	a_{43} −24	a_{44} −36

Mit der Tastenfolge **ALPHA** *OP1* **ALPHA** **RSUM** wird die Berechnung der Reihensummenmaximumnorm des abgebildeten Datenfeldes durchgeführt. Das Ergebnis ist hier die Summe der Absolutwerte der vierten Reihe. Diese Summe (hier 72) wird im X-Register abgelegt.

Verwandte Funktionen

CNRM, **FNRM**

FNRM

Die Funktion **FNRM** berechnet die Frobeniusnorm des angegebenen Datenfeldes und schreibt das Ergebnis in das X-Register. Die Formel hierzu lautet:

$$\| A \|_F = \left(\sum a_{ij}^2 \right)^{1/2}$$

Eingabeparameter

ALPHA-Register : *OP1*

Beispiel

4 Spalten

4 Reihen

a_{11} 1	a_{12} 2	a_{13} 3	a_{14} 4
a_{21} 5	a_{22} 6	a_{23} 7	a_{24} 8
a_{31} 9	a_{32} 10	a_{33} 11	a_{34} 12
a_{41} 0	a_{42} −12	a_{43} −24	a_{44} −36

Nach der Tastenfolge **ALPHA** *OP1* **ALPHA** **FNRM** erhalten wir im X-Register den Wert 51,6333. Diese Zahl ist der berechnete Wert der Frobeniusnorm des abgebildeten Datenfeldes.

Weitere Hinweise

Die Frobeniusnorm entspricht bei einer quadratischen Matrix deren euklidischer Länge.

Verwandte Funktionen

CNRM, **RNRM**

Mathematische Funktionen für Datenfelder

Die Funktionen M+, M-, M× und M/ sind sehr vielseitig, da mit der Angabe der Operanden viele unterschiedliche Möglichkeiten zur Manipulation der Datenfelder entstehen. Die Ausführungszeit ist im Vergleich zu herkömmlichen Programmen enorm kurz.

M+

Die Funktion M+ addiert die Elemente mit gleichem Index zweier Datenfelder (nur numerische Daten sind erlaubt) zueinander. Die Formel hierzu lautet:

$$c_{ij} = a_{ij} + b_{ij}$$

Eingabeparameter

ALPHA-Register : *OP1,OP2,RES*

Beispiel

Wir wollen die abgebildeten Felder „A" und „B" zueinander addieren:

Matrix A

n Spalten (hier n = 3)
Spaltenindex j

m Reihen (hier m = 2)
Reihenindex i

a_{11} 1	a_{12} 2	a_{13} 3
a_{21} 4	a_{22} 5	a_{23} 6

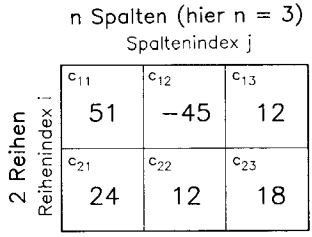
Matrix B

n Spalten (hier n = 3)
Spaltenindex j

b_{11} 50	b_{12} −47	b_{13} 9
b_{21} 20	b_{22} 7	b_{23} 12

Nach der Tastenfolge ALPHA *A,B,C* ALPHA M+ enthält das Feld „C“ folgende Elemente:

Matrix C



Weitere Hinweise

Wenn im ALPHA-Register nur *OP1* angegeben wird, so wird der Wert eines jeden Elementes im angegebenen Datenfeld verdoppelt. Tritt während der Funktionsausführung der Bereichsüberschreitungsfehler („OUT OF RANGE“) auf, dann sind in der Regel schon Teile des Datenfeldes bearbeitet worden, das Ergebnis ist dann also nicht verwendbar. Die angegebenen Datenfelder müssen die gleichen Dimensionen haben, wenn einer der Operanden nicht „X“ lautet und somit der X-Wert addiert werden soll.

Verwandte Funktionen

M-, M*, M/

M-

Die Funktion M- subtrahiert die Elemente mit gleichem Index zweier Datenfelder voneinander (nur numerische Daten sind erlaubt). Die Formel hierzu lautet:

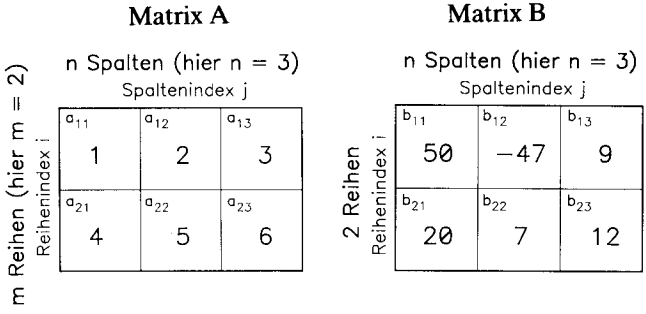
$$c_{ij} = a_{ij} - b_{ij}$$

Eingabeparameter

ALPHA-Register : *OP1,OP2,RES*

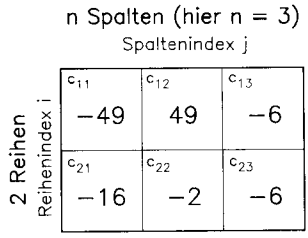
Beispiel

Wir wollen das abgebildete Feld „B" vom Feld „A" subtrahieren:



Nach der Tastenfolge ALPHA *A,B,C* ALPHA M- enthält das Feld „C“ folgende Elemente:

Matrix C



Weitere Hinweise

Das Zurücksetzen der Werte aller Elemente eines numerischen Datenfeldes auf den Wert 0 erreicht man, wenn im ALPHA-Register nur *OP1* angegeben wird. Tritt während der Funktionsausführung der Bereichsüberschreitungsfehler („OUT OF RANGE“) auf, dann sind in der Regel schon Teile des Datenfeldes bearbeitet worden, das Ergebnis ist dann also nicht verwendbar. Die angegebenen Datenfelder müssen die gleichen Dimensionen haben, wenn einer der Operanden nicht „X“ lautet.

Verwandte Funktionen

M+, M×, M/

M*

Die Funktion **M*** multipliziert die Elemente mit gleichem Index zweier Datenfelder miteinander (nur numerische Daten sind erlaubt). Die Formel hierzu lautet:

$$c_{ij} = a_{ij} \cdot b_{ij}$$

Eingabeparameter

ALPHA-Register : *OP1,OP2,RES*

Beispiel

Wir wollen die Elemente der abgebildeten Felder „A" und „B" miteinander multiplizieren:

Matrix A

n Spalten (hier n = 3)
Spaltenindex j

m Reihen (hier m = 2)
Reihenindex i

a_{11} 1	a_{12} 2	a_{13} 3
a_{21} 4	a_{22} 5	a_{23} 6

Matrix B

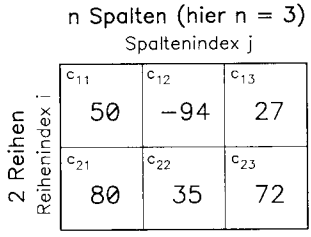
n Spalten (hier n = 3)
Spaltenindex j

2 Reihen
Reihenindex i

b_{11} 50	b_{12} −47	b_{13} 9
b_{21} 20	b_{22} 7	b_{23} 12

Nach der Tastenfolge ALPHA A,B,C ALPHA M* enthält das Feld „C“ folgende Elemente:

Matrix C



Weitere Hinweise

Das Quadrieren der Elemente eines Datenfeldes erreicht man, wenn im ALPHA-Register nur *OP1* angegeben wird. Tritt während der Funktionsausführung der Bereichsüberschreitungsfehler („OUT OF RANGE“) auf, dann sind in der Regel schon Teile des Datenfeldes bearbeitet worden, das Ergebnis ist dann also nicht verwendbar. Die angegebenen Datenfelder müssen die gleichen Dimensionen haben, wenn einer der Operanden nicht „X“ lautet und somit der X-Wert multipliziert werden soll (Skalare Multiplikation).

Verwandte Funktionen

M+, M-, M/

M/

Die Funktion M/ dividiert die Elemente mit gleichem Index zweier Datenfelder durcheinander (nur numerische Daten sind erlaubt). Die Formel hierzu lautet:

$$c_{ij} = a_{ij} / b_{ij}$$

Eingabeparameter

ALPHA-Register : *OP1,OP2,RES*

Beispiel

Zur Vorbereitung dieses Beispiels kopieren wir zunächst die Elemente des Feldes „C" in das Feld „A": ALPHA *C,A* ALPHA 1,001 ENTER 2,003 ENTER 1,001 MOVE. Danach enthalten die Felder „A" und „B" folgende Elemente:

Matrix A

n Spalten (hier n = 3)
Spaltenindex j

m Reihen (hier m = 2)
Reihenindex i

a_{11} 50	a_{12} −94	a_{13} 27
a_{21} 80	a_{22} 35	a_{23} 72

Matrix B

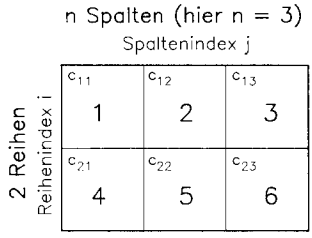
n Spalten (hier n = 3)
Spaltenindex j

2 Reihen
Reihenindex i

b_{11} 50	b_{12} −47	b_{13} 9
b_{21} 20	b_{22} 7	b_{23} 12

Wir dividieren nun mit Hilfe der Tastenfolge ALPHA *A,B,C* ALPHA M/ die Elemente des Feldes „A“ durch die des Feldes „B“ . Das Feld „C“ enthält nun folgende Elemente:

Matrix C



Weitere Hinweise

Wenn alle Feldelemente von 0 verschieden sind, erhält man die Einsmatrix, wenn im ALPHA-Register nur *OPI* angegeben wird. Tritt während der Funktionsausführung der Bereichsüberschreitungsfehler („OUT OF RANGE“) auf, dann sind in der Regel schon Teile des Datenfeldes bearbeitet worden, das Ergebnis ist dann also nicht verwendbar. Die angegebenen Datenfelder müssen die gleichen Dimensionen haben, wenn einer der Operanden nicht „X“ lautet.

Verwandte Funktionen

M+, M-, M*

M*M

Die Funktion M*M dient zur Matrizenmultiplikation. Diese Multiplikation unterliegt der folgenden Rechenvorschrift:

$$c_{ik} = \sum_{j=1}^{n} a_{ij} \cdot b_{jk}$$

Eingabeparameter

ALPHA-Register : *OP1,OP2,RES*

Beispiel

Bei gegebener inverser Matrix „A" und gegebenem Spaltenvektor „B" erhält man nach der Tastenfolge ALPHA *A,B,C* ALPHA M*M den Ergebnisvektor „C" eines eindeutig lösbaren Gleichungssystems. Hierbei müssen die Dimensionen der Matrizen verkettet sein. Hat „A" die Dimensionen 4*4, dann muß „B" auf jeden Fall 4 Reihen haben. Da „B" hier ein Spaltenvektor ist, hat dieses Feld nur eine Spalte. Das Ergebnisfeld ist dann zwangsweise ein Feld der Dimension 4*1. Die Felder „A", „B" und „C" sind mit ihren Elementen hier abgebildet:

n Spalten (hier n = 4)
Spaltenindex j

m Reihen (hier m = 4)
Reihenindex i

a_{11} 1	a_{12} 2	a_{13} 3	a_{14} 4
a_{21} 5	a_{22} 6	a_{23} 7	a_{24} 8
a_{31} 9	a_{32} 10	a_{33} 11	a_{34} 12
a_{41} 13	a_{42} 14	a_{43} 15	a_{44} 16

1 Spalte

b_{11} −3
b_{21} 5
b_{31} 1
b_{41} 2

1 Spalte

c_{11} 18
c_{21} 38
c_{31} 58
c_{41} 78

Weitere Hinweise

Die Dimensionen der angegebenen Datenfelder *OP1* und *OP2* müssen verkettet sein, d.h., wenn das Datenfeld *OP1* die Dimension *i*∗*j* hat, dann muß das Feld *OP2* die Dimension *j*∗*k* haben. Ist dies nicht der Fall, so erfolgt die Fehlermeldung „DIM ERR". Das Ergebnis dieser Operation ist ein Datenfeld der Dimension *i*∗*k*, das Feld *RES* muß also bereits vor der Funktionsausführung mit dieser Dimension existieren. Eine sinnvolle Anwendung der Funktion M*M liegt in der Lösung von nicht singulären, linearen Gleichungssystemen: wenn man die inverse (quadratische, also *i*∗*i*) Matrix mit einem Spaltenvektor multipliziert, erhält man den Ergebnisvektor. Besonders lohnend ist diese Operation, wenn zu einer festliegenden Matrix verschiedene Spaltenvektoren gehören können (z.B. mehrere Meßreihen). In diesem Fall muß nicht jedesmal das Gleichungssystem neu gelöst werden, es reicht aus, die Funktion M*M mit dem geänderten Spaltenvektor auszuführen.

Verwandte Funktionen

M+, M-, M*, M/

YC+C

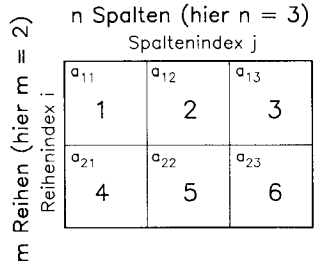
Die Funktion **YC+C** ermöglicht die Addition eines Vielfachen (Faktor in Y) der Spalte *i* zur Spalte *j*.

Eingabeparameter

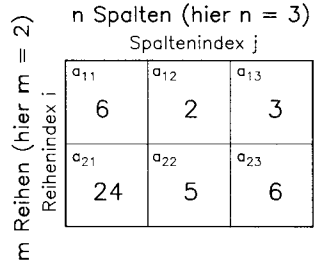
ALPHA-Register : *OP1*
X-Register : *iii,jjj*
Y-Register : Faktor

Beispiele

Das abgebildete Datenfeld „A" soll bearbeitet werden:

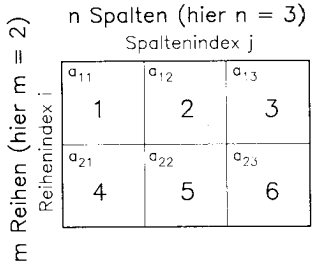


Die Tastenfolge **ALPHA** A **ALPHA** 5 **ENTER** 1,001 **YC+C** ergibt das veränderte Feld:

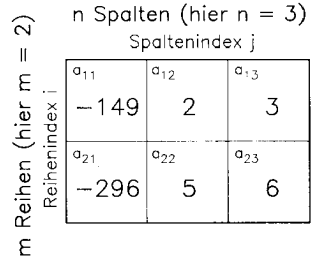


Die erste Spalte wurde mit 5 multipliziert und zu der ersten Spalte addiert.

Als zweites Beispiel gehen wir wieder von dem Feld „A“ aus:



Nach der Tastenfolge **ALPHA** A **ALPHA** -50 **ENTER** 3,001 **YC+C** ergibt sich folgendes Bild:



Die Elemente der dritten Spalte wurden mit -50 multipliziert und anschließend zu den Elementen der ersten Spalte addiert.

Weitere Hinweise

Tritt während der Funktionsausführung der Bereichsüberschreitungsfehler („OUT OF RANGE“) auf, dann sind in der Regel schon Teile des Datenfeldes bearbeitet worden, das Ergebnis ist dann also nicht verwendbar.

R-QR

Die Funktion **R-QR** führt einen Schritt des Gauß-Verfahrens durch. Bei diesem Verfahren wird die Matrix so umgeformt, daß alle Elemente unter der Hauptdiagonalen den Wert 0 haben. Auf diese Weise kann man dann durch Rückwärtseinsetzen das Gleichungssystem lösen. Die Funktionsausführung geschieht nach der Vorschrift:

$$\text{mit } Q = a_{kl} / a_{ll} \text{ gilt:}$$

$$\text{für } l < j \leqslant n : a_{kj} = a_{kj} - Q \cdot a_{lj}$$

Im X-Register wird *kkk,lll* angegeben. Nach der Funktionsausführung ist das Element a_{kl} immer Null. Hierbei gibt *lll* die Reihe an, die – mit Q multipliziert – von Zeile *kkk* subtrahiert wird. Dabei ist

$Q=a_{kl}/a_{ll}$

Das Element a_{ll} darf bei dieser Operation nicht Null sein!

Eingabeparameter

X-Register : *kkk,lll*

Beispiel

Als Beispiel wird hier das Gleichungssystem A∗x=0 bearbeitet, „A“ ist eine 4∗4 Matrix.

n Spalten (hier n = 4)
Spaltenindex j

m Reihen (hier m = 4)
Reihenindex i

a_{11} 1	a_{12} 2	a_{13} 3	a_{14} 4
a_{21} 5	a_{22} 6	a_{23} 7	a_{24} 8
a_{31} 9	a_{32} 10	a_{33} 11	a_{34} 12
a_{41} 13	a_{42} 14	a_{43} 15	a_{44} 16

mit 4,001 **R-QR** erhalten wir:

4 Spalten

4 Reihen

a_{11} 1	a_{12} 2	a_{13} 3	a_{14} 4
a_{21} 5	a_{22} 6	a_{23} 7	a_{24} 8
a_{31} 9	a_{32} 10	a_{33} 11	a_{34} 12
a_{41} 0	a_{42} −12	a_{43} −24	a_{44} −36

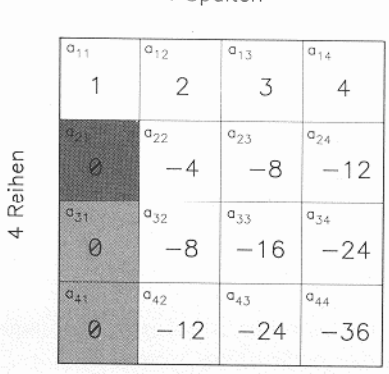
mit 3,001 **R-QR** erhalten wir:

4 Spalten

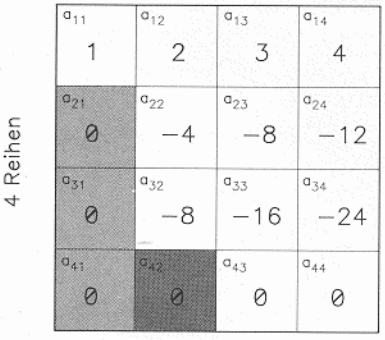
4 Reihen

a_{11} 1	a_{12} 2	a_{13} 3	a_{14} 4
a_{21} 5	a_{22} 6	a_{23} 7	a_{24} 8
a_{31} 0	a_{32} −8	a_{33} −16	a_{34} −24
a_{41} 0	a_{42} −12	a_{43} −24	a_{44} −36

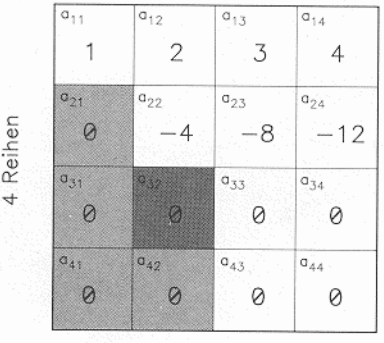
mit 2,001 **R-QR** erhalten wir:



mit 4,002 **R-QR** erhalten wir:



mit 3,002 **R-QR** erhalten wir:



Man sieht, daß durch mehrmaliges Anwenden von R-QR die grau hinterlegten Elemente zu Null werden. Man erhält also die obere Dreiecksmatrix. In diesem Beispiel ist es für die Lösung erforderlich, daß noch weitere Elemente zu Null werden. Das jeweils dunkelgrau hinterlegte Element wird auf jeden Fall immer Null! Auf ein Beispiel für ein inhomogenes Gleichungssystem der Form A*x=b (d.h. mit einer Matrix, die aus „A" und der Spalte „b" besteht) wurde verzichtet.

Weitere Hinweise

Im Gegensatz zu der sogenannten LR-Zerlegung (siehe R-PR) kann bei R-QR kein Spaltenvektor nachgeführt werden.

Verwandte Funktion

R-PR

R-PR

Die Funktion **R-PR** arbeitet ähnlich der Funktion **R-QR**. Da beim Gauß-Algorithmus alle Elemente der unteren Dreiecksmatrix Null werden, kann man anstelle der Null auch jeweils den Faktor Q abspeichern. Somit hat man die Möglichkeit, aus der umgeformten Matrix auch wieder die ursprüngliche Matrix zu erhalten. Weiterhin kann man so auch einen neuen Spaltenvektor nachführen. Der Algorithmus arbeitet genau wie bei **R-QR**:

$$\text{mit } Q = a_{kl} / a_{ll} \text{ gilt:}$$

$$\text{für } 1 \leqslant j \leqslant n : \; a_{kj} = a_{kj} - Q \cdot a_{lj}$$

$$\text{für } j = l \; : \; a_{kl} = Q$$

$$\text{für } 1 \leqslant j \leqslant l \; : \; a_{kj} = a_{kj}$$

Für die Elemente *l+1* bis *n* der Zeile *k* gilt die gleiche Operation wie bei **R-QR**; das Element a_{kl} erhält den Wert Q, und alle Elemente links von a_{kl} bleiben unverändert. Im X-Register wird *kkk,lll* angegeben. *lll* gibt die Reihe an, die – mit Q multipliziert – von Zeile *kkk* subtrahiert wird. Dabei ist

$Q=a_{kl}/a_{ll}$

Das Element a_{ll} darf bei dieser Operation nicht Null sein!

Eingabeparameter

X-Register : *kkk,lll*
ALPHA-Register : Feldname

Beispiel

Als Beispiel bearbeiten wir hier das Gleichungssystem A∗x=b. „A“ ist eine 4∗4 Matrix, der Spaltenvektor „b“ (inhomogener Teil) wird in die Matrix einbezogen, so daß eine 4∗5 Matrix entsteht:

Matrix A

n Spalten (hier n = 4)

Spaltenindex j

m Reihen (hier m = 4)

Reihenindex i

a_{11} 1	a_{12} 2	a_{13} 3	a_{14} 4
a_{21} 5	a_{22} 6	a_{23} 7	a_{24} 8
a_{31} 9	a_{32} 10	a_{33} 11	a_{34} 12
a_{41} 13	a_{42} 14	a_{43} 15	a_{44} 16

Vektor b

1 Spalte

4 Reihen

b_1 0
b_2 1
b_3 2
b_4 3

Oben sind die Matrix „A“ und der Spaltenvektor „b“, unten die neue 4∗5 Matrix mit einbezogenem Spaltenvektor (grau hinterlegt) abgebildet.

n Spalten (jetzt n = 5)

m Reihen (hier m = 4)

a_{11} 1	a_{12} 2	a_{13} 3	a_{14} 4	a_{15} 0
a_{21} 5	a_{22} 6	a_{23} 7	a_{24} 8	a_{25} 1
a_{31} 9	a_{32} 10	a_{33} 11	a_{34} 12	a_{35} 2
a_{41} 13	a_{42} 14	a_{43} 15	a_{44} 16	a_{45} 3

mit 4,001 R-PR erhalten wir: 5 Spalten

4 Reihen

a_{11} 1	a_{12} 2	a_{13} 3	a_{14} 4	a_{15} 0
a_{21} 5	a_{22} 6	a_{23} 7	a_{24} 8	a_{25} 1
a_{31} 9	a_{32} 10	a_{33} 11	a_{34} 12	a_{35} 2
a_{41} 13	a_{42} −12	a_{43} −24	a_{44} −36	a_{45} 3

mit 3,001 R-PR erhalten wir:

a_{11} 1	a_{12} 2	a_{13} 3	a_{14} 4	a_{15} 0
a_{21} 5	a_{22} 6	a_{23} 7	a_{24} 8	a_{25} 1
a_{31} 9	a_{32} −8	a_{33} −16	a_{34} −24	a_{35} 2
a_{41} 13	a_{42} −12	a_{43} −24	a_{44} −36	a_{45} 3

mit 2,001 R-PR erhalten wir:

4 Reihen

a_{11} 1	a_{12} 2	a_{13} 3	a_{14} 4	a_{15} 0
a_{21} 5	a_{22} −4	a_{23} −8	a_{24} −12	a_{25} 1
a_{31} 9	a_{32} −8	a_{33} −16	a_{34} −24	a_{35} 2
a_{41} 13	a_{42} −12	a_{43} −24	a_{44} −36	a_{45} 3

mit 4,002 **R-PR** erhalten wir:

4 Reihen

a_{11} 1	a_{12} 2	a_{13} 3	a_{14} 4	a_{15} 0
a_{21} 5	a_{22} −4	a_{23} −8	a_{24} −12	a_{25} 1
a_{31} 9	a_{32} −8	a_{33} −16	a_{34} −24	a_{35} 2
a_{41} 13	a_{42} 3	a_{43} 0	a_{44} 0	a_{45} 0

mit 3,002 **R-PR** erhalten wir:

4 Reihen

a_{11} 1	a_{12} 2	a_{13} 3	a_{14} 4	a_{15} 0
a_{21} 5	a_{22} −4	a_{23} −8	a_{24} −12	a_{25} 1
a_{31} 9	a_{32} 2	a_{33} 0	a_{34} 0	a_{35} 0
a_{41} 13	a_{42} 3	a_{43} 0	a_{44} 0	a_{45} 0

Weitere Hinweise

Die LR-Zerlegung ist praktisch gleich dem Gauß-Verfahren, nur wird hier das Nachführen eines neuen Spaltenvektors ermöglicht, da die Faktoren Q abgespeichert werden. Diese Faktoren Q werden in der unteren Dreiecksmatrix abgelegt, deren Elemente ja sonst Null wären. Für das Rückwärtseinsetzen darf man allerdings dann nur die obere Dreiecksmatrix betrachten!

Verwandte Funktion

R-QR

Programmbeispiele

Nachfolgend finden Sie zwei hilfreiche Programme, die Ihnen den Umgang mit den CCD-Modul-Matrixfunktionen erleichtern und zum besseren Verständnis der Funktionen beitragen sollen.

Gauß-Algorithmus

Das erste Programm dient zum Lösen eines linearen Gleichungssystems mit quadratischer Koeffizientenmatrix über das Gaußsche Eliminationsverfahren. Das Programm besteht aus mehreren austauschbaren Teilen, die in zwei beliebigen Registern Informationen benötigen. Hier sind die Register 00 und 05 gewählt worden.

Bedienung des Programms:

Mit dem Programmteil „ABIN" werden alle Werte der erweiterten Koeffizientenmatrix eingegeben. Die Transformation auf Dreiecksform sowie die anschließende Ausgabe des Ergebnisvektors erledigt der Programmteil „TRANS".

```
01◆LBL  "ABI   Start der Eingaberoutine (R01 wird als
          N"   Hilfsregister benutzt)
02  "DIM(N)?  Dimension n des Gleichungssystems
            "  abfragen und
03  PROMPT     in R00 abspeichern
04  STO  00
05    E        Für die Dimensionierung der Matrix wird die
06  +          Zahl nnn,mmm erzeugt, wobei mmm = n+1 ist
07    E3
08  /
09  RCL  00
10  +
11  "NAME?  "  Eingabe des Matrixnamens (z.B. „R006" für eine
12  PMTA       Matrix im Hauptspeicher) max. 6 Buchstaben
13  ASTO  05   und abspeichern dieses Namens in R05
14  MDIM       Erzeugung der Matrix
15  STO  01    Abspeicherung zu Hilfszwecken bei der Eingabe
16  CLX        Zeiger mit [J=A] auf a11 setzen (ist
               überflüssig bei Registermatrizen, siehe [MDIM])
```

```
17◆LBL "COR
        R"
18 CLA
19 ARCL 05
20 IJ=A

21◆LBL 00
22 RCL 01
23 ?IJ
24 X>Y?
25 RTN
26 "A("
27 ARCL I
28 FRC
29  E3
30 *
31 "├,"
32 X>Y?
33 "B("
34 X>Y?
35 ?IJ
36 ARCL I
37 "├): "
38 ?IJ
39 C>-
40 ARCL X
41 X<>Y
42 IJ=
43 RDN
44 PROMPT
45 >C+
46 GTO 00

47◆LBL "TRA
       NS"
48 CLA
49 ARCL 05
50 RCL 00
51 DSE X
```

Bei falscher Elementeingabe besteht eine Korrekturmöglichkeit. Eingabe in X: *iii,jjj*

Selektierung der gewählten Matrix (Name in R05) als Arbeitsmatrix

Beginn der Eingabeschleife

Wurde letztes Element schon gelesen?
Ja, Ende der Eingabe!
Erzeugung der Anzeige „A("
Anhängen von i :„A(i"
X: Aktueller Zeiger *iii,jjj*
Erzeugung von *j*

ALPHA: „A(i,"
ist *j* größer als die Dimension *n*?
Ja, Eingabe der Y-Werte. ALPHA: „Y("
ist *j* größer als die Dimension *n*?
Ja, in X wird *j* durch *iii,jjj* ersetzt
Anhängen des Integerwertes an ALPHA
ALPHA: „A(i,j):" oder „B(i):"
Lesen der aktuellen Zeigerposition und des Elementes auf dieser Position
Anhängen dieses Elementes an ALPHA

Herstellung der ursprünglichen Zeigerposition
Altes Element in X
Eingabe eines neuen Wertes oder Übernahme des alten sowie anschließendes Abspeichern
Zurück zum Beginn der Eingabeschleife

Start des Programms zur Transformation der Matrix auf Dreiecksform
Der Matrixname muß für **PIV** und **R-QR** (bzw. **R-PR**) in ALPHA stehen
Insgesamt wird die Pivotisierung *n-1* mal durchgeführt

```
52◆LBL  02
53  RCL  00
54  RCL  Y
55  -
56  RCL  X
57    E3
58  /
59  +
60  PIV
61  X=0?
62  RTN
63  RDN
64  ?I J
65  X≠Y?
66  R<>R
67  RDN
68  IJ=

69◆LBL  03
70  RDN
71  ?I J
72  RCL  Y
73  +
74  R-PR
75  DSE  Y
76  GTO  03
77  RDN
78  RDN
79  DSE  X
80  GTO  02
81  SIGN

82◆LBL  "X"
83  CLA
84  ARCL  05
85  DIM
86  FRC
87  STO  [
88  LASTX
```

Äußere Schleife $i := 1 - (n-1)$
Erzeugung der Pivotzeilennummer
Y läuft von *(n-1)* bis *1*
$i = n-Y$
Erzeugung *iii,iii*
Durch dieses **E3** wird der Y-Inhalt, wie er nach **LBL 02** vorliegt, nach T geschoben.
In Z steht nun der äußere Zähler
Suchen des Pivotelementes
Ist dieses Element gleich Null, so existiert keine eindeutige Lösung: Rücksprung mit X = 0
Pivotelement interessiert nicht weiter
Falls der Zeiger des Pivotelementes ungleich dem berechneten ist:
Zeilen vertauschen
Aktuellen Zeiger auf *iii,iii* setzen

Innere Schleife: $j := n - i + 1$
Erzeugung des Zeigers *jjj,iii*
Pivotzeiger
Zähler von *(n-1) – 1*
oder **R-QR** Transformation der Zeile *j* ab Spalte *i*
Falls Zähler größer Null,
nächsthöhere Zeile bearbeiten
Äußeren Zähler nach X holen
Falls Zähler größer Null, nächste Untermatrix bearbeiten
X ungleich Null bei eindeutiger Lösung (falls als Unterprogramm aufgerufen)

Berechnung des Ergebnisvektors
ALPHA: Matrixname
Erzeugung von *0,(nnn+1)*
Verwendung von Reg. M als Hilfsregister
X: *nnn,(nnn+1)*

```
 89◆LBL  04
 90  INT
 91  RCL  X
 92   E3
 93  /
 94  +

 95◆LBL  05
 96  IJ=
 97  INT
 98  RCL  [
 99  +
100  C>-
101  ?IJ
102  X<>  Z
103  IJ=
104  C>-
105  X<>Y
106  IJ=
107  RDN
108  X<>Y
109  /
110  STO  \
111  >C+
112  RDN
113  X=0?
114  GTO  07

115◆LBL  06
116  IJ=
117  C>-
118  RCL  \
119  *
120  ?IJ
121  X<>  Z
122  INT
123  RCL  [
124  +
```

Hauptschleife: *i:= n – 1*
Erzeugung von *iii,iii*

Zeiger auf *iii,iii* setzen
Zeiger des zu berechnenden Ergebniselementes erzeugen: *iii,(nnn+1)*

Element *iii,iii* lesen, Zeilenzeiger dekrementieren. Dieser Zeiger ist 0, wenn die oberste Zeile bearbeitet wurde – siehe Vergleich X=0?
Zeiger auf *iii,(nnn+1)* setzen
Element lesen
Zeiger wieder auf *iii,(nnn+1)* setzen

Ergebniselement x_i berechnen
$x_i := a_{i,n+1} / a_{i,n}$
Abspeichern in Hilfsregister N sowie an seinen Ergebnisort *iii,(nnn+1)*

Wenn die erste Zeile bearbeitet wurde: fertig, Ausgabe!

Innere Schleife: *j := i-1 – 1*
Zeiger auf *jjj,iii* setzen
Produkt $a_{j,i} * x_i$

Zeiger *(jjj-1),iii* lesen und retten
Erzeugung von *jjj,(nnn+1)*

```
125  IJ=
126  X<>Y
127  R>-
128  X<>Y
129  -
130  X<>Y
131  IJ=
132  X<>Y
133  >R+
134  X<> Z
135  X<Y?
136  GTO 06
137  FRC
138   E3
139  *
140  GTO 04

141◆LBL 07
142◆LBL "XOU
          T"
143  CLA
144  ARCL 05
145  DIM
146  FRC
147  ISG X
148  IJ=A

149◆LBL 08
150  ?IJ
151  "X("
152  ARCLI
153  RDN
154  "⊢)= "
155  C>+
156  ARCL X
157  SF 25
```

Subtraktion des Produktes $a_{j,i} * x_i$ vom momentanen Wert x_i

$a_{j,n+1}$ lesen

Produkt subtrahieren

Zeiger auf *jjj,(nnn+1)* setzen

Differenz abspeichern

Falls Zeiger auf $a_{j,i}$ noch nicht in Zeile *n* gesprungen ist (durch C-), ist *j* noch nicht bei 1 angelangt

Zeile *i-1* ist als nächstes zu bearbeiten

Zurück zur Hauptschleife

Ende der Berechnung und Ausgabe des Ergebnisvektors (x_i-Werte)

Matrixname in ALPHA

Zeiger *1,(nnn+1)* erzeugen

Zeiger auf x_1 setzen
Beginn der Ausgabeschleife

Erzeugung der Anzeige „X(i)="

Lesen des Wertes x_i und Anhängen an ALPHA

Ausdrucken, wenn Drucker vorhanden

```
158 PRA
159 FC?C 25
160 PROMPT      Sonst Anzeigen der x_i-Werte
161 GTO 08      Zurück zum Beginn der Ausgabeschleife
162 END         Programmende
```

```
PLNG "ABIN"
  301 BYTES
```

Berechnung der inversen Matrix

Das Programm „INV" berechnet die Inverse „A^{-1}" einer gegebenen quadratischen Matrix „A". Nach Berechnung der Inversen lassen sich lineare Gleichungssysteme der Form $A*x=b$ mit Hilfe einer einfachen Matrizenmultiplikation (zum Beispiel mit dem Befehl M*M des CCD-Moduls) lösen:

$$x=A^{-1}*b$$

Das Programm „INV" führt auch bei singulären Matrizen zu einer Lösung. Deshalb berechnet „INV" eine Konditionszahl, die eine Aussage über die Genauigkeit der Lösung erlaubt. Der relative Fehler der Lösung x läßt sich nun ermitteln, indem man die Konditionszahl mit dem relativen Fehler des Spaltenvektors b (auf dem HP-41 also minimal 1E-10) multipliziert.

Bedienung:
Im Alpha-Register wird der Name eines vorher mit MDIM angelegten quadratischen Datenfeldes erwartet. Nach Ausführung von „INV" beinhaltet dieses Datenfeld die Inverse des ursprünglichen Feldes. Im X-Register wird die Konditionszahl ausgegeben. „INV" verändert den Stack sowie die Register R00 bis R02. Des weiteren benötigt „INV" genau so viele **zusätzliche** Register , wie das im Alpha-Register angegebene Datenfeld belegt. Befindet sich das Datenfeld im Datenspeicherbereich des HP-41 (Name R*xxx*), so erwartet „INV" diese zusätzlichen Register hinter dem Datenfeld. Beispiel: Damit „INV" die Inverse eines Datenfeldes mit dem Namen R010 der Dimension 5 * 5 berechnen kann, muß vorher mindestens SIZE 61 eingegeben werden, sonst erscheint die Meldung „NONEXISTENT". Liegt das Datenfeld im erweiterten Speicherbereich, so muß nach Anlegen des Datenfeldes die Funktion EMDIR mindestens 25 freie Register anzeigen, sonst erscheint die Meldung „NO ROOM".

Funktionsbeschreibung:
Das hier verwendete Verfahren berechnet die Inverse der Matrix „A" spaltenweise durch Lösen der Gleichungssysteme $A*y_k=e_k$. e_k sind die Spalten der Einheitsmatrix „E", die Lösungen y_k die Spalten der Inversen „A^{-1}".

Zur Lösung der Gleichungssysteme wird ein Algorithmus nach Gauß-Jordan verwendet. Da die dabei auftretenden Umformungen der Matrix „A“ auch auf alle „rechten Seiten“ e_k angewendet werden müssen, wird zunächst eine Matrix erzeugt, deren linke Hälfte die Matrix „A“ und deren rechte Hälfte die Einheitsmatrix „E“ enthält. Da die Funktionen des CCD-Moduls Matrizen zeilenweise abspeichern, wird zunächst die Zeilenzahl der zu invertierenden Matrix verdoppelt (Programmzeilen 4-8). MDIM löscht alle dazugekommenen Elemente, so daß eine Matrix wie folgt entsteht:

a_{11}	a_{12}	a_{13}
a_{21}	a_{22}	a_{23}
a_{31}	a_{32}	a_{33}
0	0	0
0	0	0
0	0	0

Es werden nun die Zeilen so getauscht, daß zwischen zwei Zeilen der ursprünglichen Matrix immer eine Leerzeile gelangt (Programmzeilen 9-16):

a_{11}	a_{12}	a_{13}
0	0	0
a_{21}	a_{22}	a_{23}
0	0	0
a_{31}	a_{32}	a_{33}
0	0	0

Ein Unterprogramm, beginnend bei LBL 20, vertauscht nun die Zeilenzahl der Dimension mit ihrer Spaltenzahl:

a_{11}	a_{12}	a_{13}	0	0	0
a_{21}	a_{22}	a_{23}	0	0	0
a_{31}	a_{32}	a_{33}	0	0	0

Die Programmzeilen 18-30 erzeugen in der noch leeren rechten Hälfte die Einheitsmatrix.

a_{11}	a_{12}	a_{13}	1	0	0
a_{21}	a_{22}	a_{23}	0	1	0
a_{31}	a_{32}	a_{33}	0	0	1

Die Umformung der Matrix nach Gauß-Jordan geschieht mit zwei ineinanderliegenden Programmschleifen. Die äußere, mit dem Zeiger *l* in R01 (Initialisierung in den Programmzeilen 31-35 und beginnend bei LBL 02) führt zunächst die sogenannte Pivotisierung durch: Der Befehl **PIV** (Programmzeile 43) durchsucht die *l*-te Spalte von der *l*-ten bis zur letzten Zeile nach dem absolut größten Element. Es werden dann zwei Zeilen so vertauscht, daß dieses Element in die *l*-te Zeile gelangt (Programmzeile 49). Dort wird es als sogenanntes Pivot in der folgenden inneren Schleife, mit dem Zeiger *k* in R02 (Initialisierung in den Programmzeilen 37-41) und beginnend bei LBL 03, verwendet. Der Befehl **R-QR** (Programmzeile 60) subtrahiert ein Vielfaches der Zeile des Pivots so von der *k*-ten Zeile, daß das in der Spalte des Pivots liegende Element zu Null wird. Den dazu notwendigen Faktor berechnet **R-QR** vorher mit einer Division des später zu Null werdenden Elementes durch das Pivot (siehe Beschreibung der Funktion **R-QR**). Damit die bei dieser Division auftretenden Fehler möglichst gering werden, wird vorher die Pivotisierung durchgeführt. Wird dabei als Pivot die Zahl Null gefunden, so war die Matrix „A" entweder singulär oder zumindest „nahezu" singulär; der Rechner kann dies wegen der auftretenden Rundungsfehler nicht unterscheiden. Damit das Programm bei der Division nicht mit der Meldung „DATA ERROR" abbricht, wird ein Null-Pivot durch eine Zahl ersetzt, die 1E-10 mal der absolut größten in der Spalte des Pivot auftretenden Zahl, mindestens aber 1E-99 ist (Unterprogramm beginnend bei LBL 30). Diese Zahl liegt normalerweise in der Größenordnung der auftretenden Rundungsfehler.

Nach Abarbeiten der beiden Schleifen ist die linke Hälfte der Matrix, also die ursprüngliche Matrix „A", in eine Diagonalmatrix umgeformt:

a_{11}	0	0	z_{11}	z_{12}	z_{13}
0	a_{22}	0	z_{21}	z_{22}	z_{23}
0	0	a_{33}	z_{31}	z_{32}	z_{33}

Man erhält jetzt die Lösungsmatrix „A^{-1}“, wenn man alle Elemente der rechten Hälfte durch das in derselben Zeile liegende Diagonalelement der linken Hälfte dividiert. Dies geschieht in den beiden ineinanderliegenden Schleifen, beginnend mit LBL 05 und LBL 06, sowie den Zeigern R01 auf das Diagonalelement und R02 auf die Spalten in der rechten Hälfte. Ist ein Diagonalelement gleich Null, wird es ersetzt durch das absolut größte Element der Spalte, mindestens aber durch 1E-99 (Unterprogramm beginnend mit LBL 35). Das ist möglich, weil die vorangegangene Umformung ja alle übrigen Elemente der Spalte bis auf Rundungsfehler zu Null gemacht hat. Man findet also eine Zahl, die ungleich Null, aber in der Größenordnung der auftretenden Rundungsfehler ist.

Nach Abarbeiten der Schleifen liegt die Lösung „A^{-1}“ in der rechten Hälfte der Matrix. Diese Lösung wird nun wieder auf den Platz der ursprünglichen Matrix „A“ kopiert durch Vertauschen der Dimensionen (Programmzeile 106), Zeilentauschen (Programmzeilen 107-116) und anschließendes Umdimensionieren zu einer quadratischen Matrix (Programmzeilen 117-120). Die Konditionszahl wird errechnet durch Multiplikation der Frobeniusnorm der Matrix „A“ vor der Umformung (Programmzeilen 2-3) mit der Frobeniusnorm ihrer Inversen (Programmzeilen 121-123).

```
01◆LBL "INV
       "
02 FNRM
03 STO 00
04 DIM
05 INT
06 LASTX
07 +
08 MDIM
09 1
10 -

11◆LBL 11
12 R<>R
13 1.001
14 -
15 DSE X
16 GTO 11
17 XEQ 20
18 INT
19  E3
20 /
21 1
22 +
23 IJ=A
24 LASTX

25◆LBL 01
26 R>+
27 RDN
28 >C+
29 ISG Y
30 GTO 01
31 RDN
32 FRC
33 1
```

```
34 +
35 STO 01

36◆LBL 02
37 RCL 01
38 FRC
39 1
40 +
41 STO 02
42 RCL 01
43 PIV
44 ?IJ
45 X<>Y
46 X=0?
47 XEQ 30
48 RDN
49 R<>R

50◆LBL 03
51 RCL 02
52 INT
53 RCL 01
54 INT
55 X=Y?
56 GTO 04
57  E3
58 /
59 +
60 R-QR

61◆LBL 04
62 ISG 02
63 GTO 03
64 ISG 01
65 GTO 02
66 RCL 01
67 FRC
68 1
69 +
70 STO 01

71◆LBL 05
72 DIM
73 1
74 +
75 STO 02
76 RCL 01
77 INT
78 1,001
79 *
80 IJ=
81 R>+
82 X=0?
83 XEQ 35
84 RCL 02
85  E3
86 /
87 RCL 01
88 INT
89 +
90 IJ=
91 RDN
92◆LBL 06
93 ?IJ
94 R>+
95 X<>Y
96 IJ=
97 RDN
98 X<>Y
99 /
100 >R+
101 LASTX
102 ISG 02
103 GTO 06
104 ISG 01
105 GTO 05
106 XEQ 20
107 INT
108 2
109 /
110 0
```

```
111◆LBL 12
112 1,002
113 +
114 R<>R
115 DSE Y
116 GTO 12
117 INT
118 1,001
119 *
120 MDIM
121 FNRM
122 RCL 00
123 *
124 RTN

125◆LBL 20
126 DIM
127 FRC
128 LASTX
129 INT
130  E3
131 /
132 X<>Y
133 LASTX
134 *
135 +
136 MDIM
137 RTN

138◆LBL 30
139 RDN
140 RCL 01
141 CMAXAB
142  E10
143 /
144  E-99
145 +
146 RCL Z
147 IJ=
148 X<>Y
149 >R+
150 RTN

151◆LBL 35
152 RCL 01
153 CMAXAB
154  E-99
155 +
156 END
```

```
PLNG "INV"
 247 BYTES
```

Kapitel 5

Binärfunktionen
(Hexadezimalfunktionen)

Inhaltsverzeichnis Kapitel 5
Hexadezimalfunktionen

Hexadezimalfunktionen

Der hexadezimale oder auch logische Funktionsblock besteht aus den Funktionen 1CMP, 2CMP, AND, bC?, bS?, Cb, NOT, OR, R‹, R›, S‹, S›, Sb, UNS, WSIZE und XOR. Außerdem werden diese Funktionen noch durch ARCLH, PMTH, VIEWH und XTOAH aus dem Ein-/Ausgabe Funktionsblock (-I/O FNS) des CCD-Moduls ergänzt.

Zum Verständnis der Beschreibung dieser Funktionen wird die Kenntnis der binären, oktalen, dezimalen und hexadezimalen Zahlensysteme sowie der Vorzeichenmodi vorausgesetzt.

Um die verwendeten Begriffe in ihrer Bedeutung einzugrenzen, werden sie unten dennoch kurz erläutert.

Alle hexadezimalen Funktionen folgen in ihrer Ausführung dem weiter unten dargestellten Flußdiagramm, richten sich also immer nach der gewählten Wortbreite und dem Vorzeichenmodus!

Die Zahlensysteme

Das binäre oder duale Zahlensystem

Die Basis des binären Zahlensystems ist die 2; alle Zahlen werden durch die beiden Ziffern **0** und **1** dargestellt. Im folgenden werden Binärzahlen durch die Schreibweise *bin* kenntlich gemacht: bin 1001 = dez 9. Der Übertrag auf die nächsthöhere Stelle erfolgt im binären Zahlensystem bei dez 2. Die Zuordnung zu den Dezimalzahlen steht für den Bereich dez 0 bis dez 10 in der folgenden Tabelle.

Dezimal	Binär
0	0000
1	0001
2	0010
3	0011
4	0100
5	0101
6	0110
7	0111
8	1000
9	1001
10	0001 0000

Das oktale Zahlensystem

Die Basis des oktalen Zahlensystems ist die 8, alle Zahlen werden durch die Ziffern **0**, **1**, **2**, **3**, **4**, **5**, **6** und **7** dargestellt. Die Zuordnung zu den Dezimalzahlen ist:

Dezimal	Oktal
0	0
1	1
2	2
3	3
4	4
5	5
6	6
7	7
8	10
9	11
10	12

Der Übertrag auf die nächsthöhere Stelle erfolgt im oktalen Zahlensystem bei dez 8. Die im folgenden verwendete Schreibweise für Oktalzahlen ist *okt*. Oktalzahlen „entstehen" durch das Zusammenfassen von 3 binären Stellen (3 Bits); dadurch nutzt das oktale Zahlensystem den vollen Wertebereich der 3 Stellen aus (dez 0 bis 7 bzw. bin 000 bis 111).

Die Funktionen **OCT** und **DEC** des HP-41 Betriebssystems ermöglichen bereits Berechnungen im oktalen Zahlenbereich.

Das dezimale Zahlensystem

Das dezimale Zahlensystem ist das allgemein gebräuchliche Zahlensystem. Die Funktionen des HP-41 unterstützen die Zahlen zur Basis 10 in allen Operationen. Zahlen zur Basis 10 werden durch die Schreibweise *dez* kenntlich gemacht: dez 136 = Dezimalzahl 136. Zahlen, bei denen die Zahlenbasis nicht ausdrücklich angegeben ist, sind als Dezimalzahlen zu verstehen.

Das hexadezimale Zahlensystem

Die Basis dieses Zahlensystems ist dez 16. Es werden die Ziffern **0**, **1**, **2**, **3**, **4**, **5**, **6**, **7**, **8** und **9** sowie die Buchstaben **A**, **B**, **C**, **D**, **E** und **F** zur Darstellung benutzt. Der Übertrag auf die nächsthöhere Stelle erfolgt bei dez 16. Die Zuordnung zu den binären und dezimalen Zahlen ist:

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

Warnung:
Diese Tabelle ermöglicht den Vergleich von dezimalen zu binären Zahlen und umgekehrt nur bis dez 9! Zum Beispiel ist die Darstellung von dez 14 im Speicher des HP-41 bin 0001 0100 und nicht etwa bin 1110!

Hexadezimalzahlen „entstehen" durch das Zusammenfassen von 4 binären Stellen (4 Bits); dadurch nutzt das hexadezimale Zahlensystem die 4 Bits eines Nybbles im vollen Wertebereich aus. Jedes Nybble kann Werte von bin 0000 bis 1111 annehmen, in hexadezimaler Schreibweise von hex 0 bis F. Dieses Zahlensystem wird im Computerbereich sehr oft benutzt.

Darstellung von negativen Zahlen

Bei einer vorgegebenen beschränkten Stellenzahl m (hier zwischen 0 und 32 binären Stellen) ist der Wertebereich der darstellbaren Zahlen $0<=X<=2^m-1$, es sind also keine negativen Zahlen im Wertebereich enthalten. Um diesem Mangel abzuhelfen, führt man das Komplement ein.

Das Komplement

Das Komplement der Zahl X ist definiert als:

$$KOM\ (X) = K - X$$

Der Wert von K wird durch das gewählte Komplement festgelegt. Übliche Werte für K sind im Dualsystem $K=2^m$ und $K=2^m-1$, man spricht dann vom Zweier- bzw. Einerkomplement (s.u.).

Allgemein gibt es das Komplement natürlich in jedem Zahlensystem mit der Basis B (zum Beispiel $B=10$ im Dezimalsystem, $B=16$ im Hexadezimalsystem,etc.). Für eine beliebige Basis B gibt es dann ein $(B-1)$- und ein B-Komplement, im Dezimalsystem also ein 9er und ein 10er Komplement.

Aufteilung des Wertebereiches

Die Bildung eines Komplements mag bisher unsinnig erscheinen, denn wir können bisher immer noch keine negativen Zahlen darstellen. Dieses gelingt erst durch eine willkürliche Aufteilung des Wertebereiches: Im binären Zahlensystem legt man fest, daß alle Zahlen, deren höchstwertigstes Bit gesetzt ist, negative Zahlen sind. Durch die Aufteilung und die Komplementierung verschiebt sich der Wertebereich wie folgt:

vorher (ohne Komplementierung): $0<=X<=(B^m)-1$

nachher (B-Komplementierung) : $-B^{(m-1)}<=X<=B^{(m-1)}-1$

nachher ($(B-1)$-Komplementierung): $-B^{s(m-1)}-1<=X<=B^{(m-1)}-1$

Die verschiedenen Modi

Die Umwandlung der dezimal dargestellten Zahlen in binäre und umgekehrt geschieht also abhängig von dem verwendeten Modus.Die drei Modi sind:

- **Der Einerkomplement Modus**
- **Der Zweierkomplement Modus**
- **Der Modus ohne Vorzeichen** (im folgenden *Unsigned Modus* genannt)

Mit Hilfe des Komplements ist es für eine Recheneinheit leichter, Subtraktionen durchzuführen, da diese sich zu Additionen vereinfachen. Außerdem ist das Komplement einer Zahl für den Prozessor in binärer Schreibweise sehr leicht zu bilden (siehe unten).

Der Einerkomplement Modus

Das Einerkomplement einer Zahl erhält man durch Subtraktion dieser Zahl von der größten darstellbaren Zahl, d.h., bei einer Wortbreite von 5 Bit ist das Einerkomplement von bin -a = 11111 – a. Das Einerkomplement von bin 00000 ist die Zahl bin 11111. Der Prozessor invertiert einfach alle Bits der ursprünglichen Zahl; er führt somit die logische Funktion „not" aus. Durch die willkürliche aber eindeutige Aufteilung des Wertebereiches ist bei allen negativen Zahlen das ganz linke Bit gesetzt und übernimmt daher die Rolle des „-"-Zeichens. Im Einerkomplement Modus gibt es gleich viele positive und negative Zahlen, also auch zwei Darstellungen für die Null: 0 und -0, dies sind bei 5-stelliger binärer Rechnung bin 00000 und 11111.

Der Zweierkomplement Modus

Das Zweierkomplement einer Zahl erhält man durch Subtraktion dieser Zahl von der größten darstellbaren Zahl und anschließender Addition von 1, d.h., bei einer Wortbreite von 5 Bit ist das Zweierkomplement von bin -a = 11111 – a + 1. Das Zweierkomplement von bin 10001 ist die Zahl bin 01111. Wir bemerken, daß durch die Aufteilung auch im Zweierkomplement Modus bei allen negativen Zahlen das ganz linke Bit gesetzt ist und daher die Rolle des „-"-Zeichens übernimmt. Im Zweierkomplement Modus gibt es ein negatives Element mehr als im positiven Zahlenbereich, die Null hat nur die Darstellung 0.

Der Modus ohne Vorzeichen (Unsigned Modus)

Da die Komplementmodi ein Bit als Vorzeichenbit verwenden, reicht der Wertebereich bei einer Wortbreite von 8 Bit beim Einerkomplement nur von dez 127 bis -127 bzw. bis -128 beim Zweierkomplement. Dies sind zwar auch 256 Werte, jedoch benötigt man häufig nur den positiven Zahlenbereich. Für diesen Fall gibt es den Unsigned Modus, bei dem das Vorzeichenbit entfällt. Daher ist nun bei gleicher Wortbreite von dez 8 Bit der Wertebereich von dez 0 bis 255 abgedeckt.

In der nachstehenden Tabelle sind die jeweiligen Dezimalwerte den binären Werten zugeordnet (für eine Wortbreite von 4 Bit).

Dezimalwerte aller 4–Bit Binärkombinationen

Binär	Einer–komplement–Modus	Zweier–komplement–Modus	Unsigned–Modus
0111	7	7	7
0110	6	6	6
0101	5	5	5
0100	4	4	4
0011	3	3	3
0010	2	2	2
0001	1	1	1
0000	0	0	0
1111	–0	–1	15
1110	–1	–2	14
1101	–2	–3	13
1100	–3	–4	12
1011	–4	–5	11
1010	–5	–6	10
1001	–6	–7	9
1000	–7	–8	8

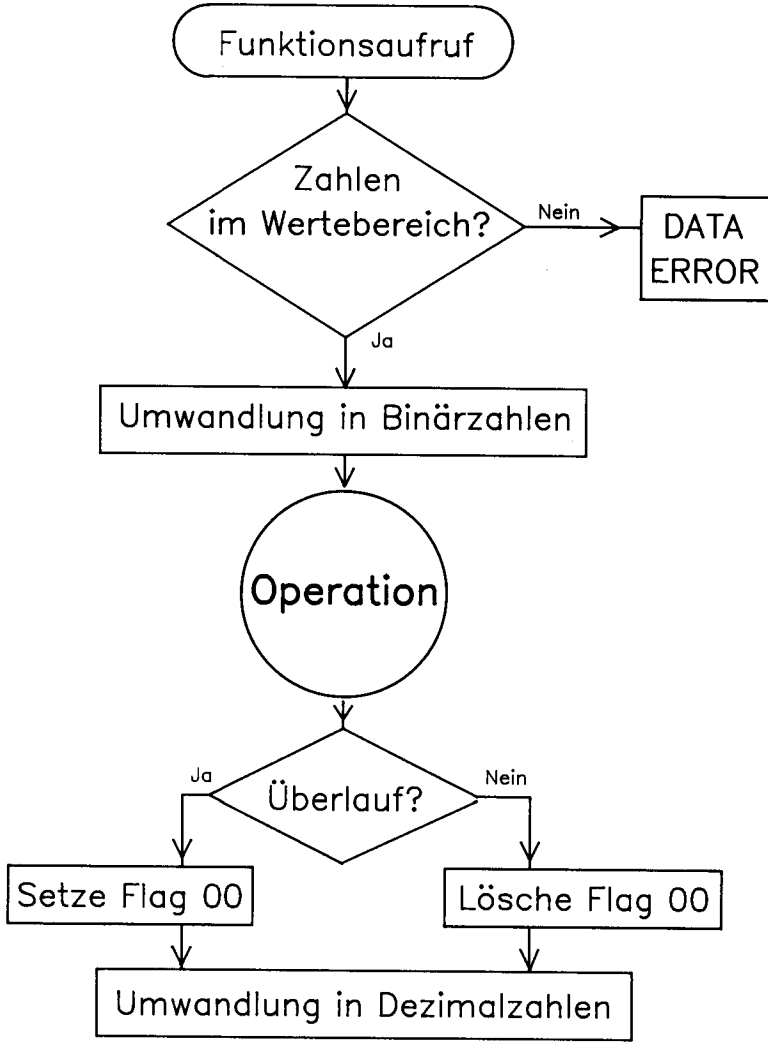
Allgemeine Konventionen für die hexadezimalen Funktionen

Nach der kurzen Einführung in die Zahlensysteme, Komplementmodi und den Unsigned Modus werden hier die Vereinbarungen für die hexadezimalen Funktionen getroffen. Alle Funktionen beachten die folgenden Voreinstellungen:

1) Die zu verarbeitende Wortbreite. Sie wird mit der Funktion **WSIZE** (wordsize = Wortbreite) festgelegt.
2) Den eingestellten Vorzeichenmodus. Die dezimale Darstellung und die interne hexadezimale Darstellung sind über den Komplement bzw. den Unsigned Modus verknüpft. Abhängig von dem gewählten Modus wird die immer dezimal dargestellte Zahl vor der gewählten Verknüpfung in eine binäre Zahl umgewandelt.Nach der Verknüpfung wird die binäre Zahl wieder abhängig vom Modus in eine Dezimalzahl zurückverwandelt und angezeigt.
3) Der ganzzahlige Anteil. Die Funktionen beachten nur den ganzzahligen Anteil einer Dezimalzahl (Integer-Teil).

Der Ablauf der Funktionsausführung für alle hexadezimalen Funktionen ist im folgenden Flußdiagramm dargestellt:

Flußdiagramm für die Funktionen AND, OR, XOR, NOT, S<, S>, R< und R>



Erläuterung zum Flußdiagramm:

1) Es wird überprüft, ob alle für die Operation benötigten Werte in der aktuellen Wortbreite und dem gewählten Vorzeichenmodus darstellbar sind.
2) Die im Stack des HP-41 dezimal dargestellten Zahlen werden in Binärzahlen umgewandelt (abhängig vom Vorzeichenmodus).
3) Die eigentliche Operation oder Verknüpfung wird durchgeführt.
4) Es wird überprüft, ob bei der ausgeführten Operation ein Überlauf eingetreten ist (es wird überprüft, ob das Carry gesetzt worden ist). Als Überlauf-Bit oder auch Carry-Bit wird das Flag 0 verwendet.
5) Die Binärzahlen werden in Dezimalzahlen umgewandelt und im Stack abgelegt.

Grundeinstellung

Folgende Einstellungen sind nach erstmaligem Stecken des CCD-Moduls oder nach „MEMORY LOST" vorhanden:

- Es ist für diesen Funktionsblock keine Wortbreite definiert, d.h., das Modul geht von der Wortbreite dez 8 aus.

- Es ist kein Modus definiert, d.h., der Modus ohne Vorzeichen wird benutzt(UNS).

Beschreibung der Funktionen

Um eine definierte Ausgangssituation mit den oben genannten Grundeinstellungen zu erhalten, führen wir folgende Operationen durch:

UNS	Schaltet den Unsigned Modus ein.
CLX WSIZE	Stellt die Wortbreite auf 8 Bit ein.

Funktionen für die Grundeinstellung der Wortbreite und des Vorzeichenmodus

WSIZE

Die Funktion WSIZE legt die Wortbreite *bb* für alle Hexadezimalfunktionen des CCD-Moduls fest. Die Wortbreite *bb* kann 1 bis 32 Bit betragen; sie wird im I/O-Buffer des CCD-Moduls gespeichert. CLX WSIZE löscht den Eintrag der Wortbreite im I/O-Buffer, der Ersatzwert ist dann dez 8. Der Stack wird durch die Funktionsausführung nicht verändert.

Eingabeparameter

X-Register : *bb*

Beispiele

Tastenfolge	Kommentar
9 WSIZE	Wortbreite von 9 Bit gewählt, Operationen im Bereich dez 0 bis 511 möglich.
33 WSIZE	Fehlermeldung: „DATA ERROR". Die Wortbreite wurde nicht verändert, da Werte größer als 32 nicht möglich sind.
16 WSIZE	Wortbreite von 16 Bit gewählt.

Weitere Hinweise

Die Zahl im X-Register darf sowohl negativ sein als auch Nachkommastellen enthalten; es wird nur der absolute Integerwert beachtet. Bei Zahlen außerhalb des Bereiches dez 32 bis -32 erscheint die Fehlermeldung „DATA ERROR", über dez 1000 die Fehlermeldung „NONEXISTENT". Die eingestellte Wortbreite hat keinen Einfluß auf die Inhalte des Stacks oder der Speicherregister. Bei der Ausführung von WSIZE kann die Fehlermeldung „NO ROOM" auftreten, da zur Abspeicherung der Wortbreite in der Regel Platz im I/O-Buffer benötigt wird.In diesem Fall muß vor Einstellung der Wortbreite erst wieder freier Speicherplatz geschaffen werden.

1CMP

Mit der Funktion **1CMP** wird der HP-41 in den Einerkomplement Modus versetzt. Andere Vorzeichenmodi (Zweierkomplement oder Unsigned Modus) werden aufgehoben. Der Inhalt des X-Registers wird im LAST X-Register abgelegt und anschließend durch die neue Darstellung der Hexadezimalzahl ersetzt; d.h., die Zahl aus dem X-Register wird zunächst im alten Modus in eine Binärzahl umgewandelt und dann nach Einschalten des neuen Modus wieder in eine Dezimalzahl zurückverwandelt.

Eingabeparameter

keine

Beispiel

Tastenfolge	Kommentar
1CMP	Der HP-41 ist in den Einerkomplement Modus versetzt (nur gültig für Binärfunktionen des CCD-Moduls).

Weitere Hinweise

Kann die Zahl aus dem X-Register im neuen Vorzeichenmodus nicht dargestellt werden, so erfolgt die Fehlermeldung „DATA ERROR X“ und der Modus wird nicht verändert.

Verwandte Funktionen

2CMP, **UNS**

2CMP

Mit der Funktion **2CMP** wird der HP-41 in den Zweierkomplement Modus versetzt. Andere Vorzeichenmodi (Einerkomplement oder Unsigned Modus) werden aufgehoben. Der Inhalt des X-Registers wird im LAST X-Register abgelegt und anschließend durch die neue Darstellung der Hexadezimalzahl ersetzt; d.h., die Zahl aus dem X-Register wird zunächst im alten Modus in eine Binärzahl umgewandelt und dann nach Einschalten des neuen Modus wieder in eine Dezimalzahl zurückverwandelt.

Eingabeparameter

keine

Beispiel

Tastenfolge	Kommentar
2CMP	Der HP-41 ist in den Zweierkomplement Modus versetzt (nur gültig für Binärfunktionen des CCD-Moduls).

Weitere Hinweise

Kann die Zahl aus dem X-Register im neuen Vorzeichenmodus nicht dargestellt werden, so erfolgt die Fehlermeldung „DATA ERROR X" und der Modus wird nicht verändert.

Verwandte Funktionen

1CMP, **UNS**

UNS

Mit der Funktion **UNS** wird der HP-41 in den Unsigned Modus versetzt. Andere Vorzeichenmodi (Einer- oder Zweierkomplement Modus) werden aufgehoben. Der Inhalt des X-Registers wird im LAST X-Register abgelegt und anschließend durch die neue Darstellung der Hexadezimalzahl ersetzt; d.h., die Zahl aus dem X-Register wird zunächst im alten Modus in eine Binärzahl umgewandelt und dann, nach Einschalten des neuen Modus, wieder in eine Dezimalzahl zurückverwandelt.

Eingabeparameter

keine

Beispiel

Tastenfolge	Kommentar
UNS	Der HP-41 ist in den Unsigned Modus versetzt (nur gültig für Binärfunktionen des CCD-Moduls).

Weitere Hinweise

Kann die Zahl aus dem X-Register im neuen Vorzeichenmodus nicht dargestellt werden, so erfolgt die Fehlermeldung „DATA ERROR X“ und der Modus wird nicht verändert.

Verwandte Funktionen

1CMP, **2CMP**

Ein-/Ausgabefunktionen für -HEX FNS

Diese Funktionen werden auch im Kapitel **-I/O FNS** beschrieben, der Vollständigkeit halber seien die für die hexadezimalen Funktionen wichtigen Ein-/Ausgabehilfen auch hier erklärt.

PMTH

Die Funktion **PMTH** dient zur Eingabe von hexadezimalen Zahlen. Sie schreibt den äquivalenten Dezimalwert in das X-Register. Der Stack wird vorher angehoben.

Eingabeparameter:

Die Anzahl der einzugebenden Stellen ist abhängig von der aktuellen Wortbreite.

Beispiel

Tastenfolge	Kommentar
PMTH	Eingabe von einem Wert zwischen hex 00 und hex 8E. Unmittelbar nach der Eingabe der letzten Ziffer (hier E) erscheint im X- Register der dezimale Wert, in diesem Fall dez 142.

Beispielprogramm:

```
'ADRESSE
PMTH
```

Mit diesen zwei Programmschritten erscheint im Programmlauf die Anzeige ADRESSE'_ _ ; nach Eingabe von zwei Ziffern wird der Programmlauf fortgesetzt.

Weitere Hinweise

Im Programmlauf kann zusätzlicher Text mit angezeigt werden. Die Funktion weist zu große Eingabewerte ab. Nur die Tasten *0* bis *9* und *A* bis *F* sind aktiv. Ein Abbruch der Funktion kann nur mit der ← oder der ON-Taste erfolgen.

VIEWH

Die Funktion **VIEWH** zeigt das hexadezimale Äquivalent des X-Registers an.

Eingabeparameter

X-Register : Dezimalzahl im erlaubten Wertebereich (abhängig von der aktuellen Wortbreite und dem gewählten Modus).

Beispiel

Tastenfolge	Kommentar
142 **VIEWH**	Die Zahl aus dem X-Register (142) wird in ihr hexadezimales Äquivalent umgewandelt und angezeigt (in diesem Fall 8E).

Weitere Hinweise

Bei ALPHA-Daten erfolgt die Fehlermeldung „ALPHA DATA". Wenn die Dezimalzahl aus dem X-Register in der aktuellen Wortbreite und dem Vorzeichenmodus nicht darstellbar ist, erscheint die Fehlermeldung „DATA ERROR X".

Verwandte Funktion

ARCLH

ARCLH

ARCLH hängt die hexadezimale Darstellung der Zahl aus dem X-Register an den Inhalt des ALPHA-Registers an.

Eingabeparameter

X-Register : Dezimalzahl im erlaubten Wertebereich (abhängig von der aktuellen Wortbreite und dem gewählten Modus).

Beispiel

Tastenfolge	Kommentar
UNS	Schaltet den Unsigned Modus ein.
CLX WSIZE	Stellt die Wortbreite auf 8 Bit ein.
'ABC	Eingabe des Textes „ABC" in das ALPHA-Register.
ARCLH	Hängt zwei hexadezimale Ziffern an das ALPHA-Register an; in diesem Fall 08, d.h., im ALPHA-Register steht nun „ABC08".
55 ARCLH	Die Zahl hex 37 wird an das ALPHA-Register angehängt.

Weitere Hinweise

Bei ALPHA-Daten erfolgt die Fehlermeldung „ALPHA DATA". Wenn die Dezimalzahl aus dem X-Register in der aktuellen Wortbreite und dem Vorzeichenmodus nicht darstellbar ist, erscheint die Fehlermeldung „DATA ERROR X".

Verwandte Funktion

VIEWH

XTOAH

Die Funktion XTOAH hängt ein oder mehrere Zeichen mit dem Wert des X-Registers an den Inhalt des ALPHA-Registers an.

Eingabeparameter

X-Register : Wert der anzuhängenden Zeichen

Beispiel

Tastenfolge	Kommentar
CLA 10 WSIZE	Das ALPHA-Register wird gelöscht und eine Wortbreite von 10 Bit eingestellt.
340 XTOAH	Dez 340 entspricht hex 154. Also werden die Zeichen „Männchen" (Byte hex 01) und „T" (Byte hex 54) an das ALPHA-Register angehängt.

Weitere Hinweise

Bei ALPHA-Daten erfolgt die Fehlermeldung „ALPHA DATA". Wenn die Dezimalzahl aus dem X-Register in der aktuellen Wortbreite und dem Vorzeichenmodus nicht darstellbar ist, erscheint die Fehlermeldung „DATA ERROR X".

Verwandte Funktion

XTOA (Funktion aus dem X-Functions Modul)

Logische Operationen

AND

Die Funktion AND verknüpft X und Y mit der logischen UND-Funktion, d.h., im Ergebnis sind alle Bits gesetzt, die in beiden Binärzahlen gleichzeitig gesetzt waren; alle übrigen Bits sind gelöscht. Der Stack wird heruntergeschoben und der alte X-Wert im LAST X-Register abgelegt. (Binär heißt das z.B.: 1011 und 0111 ergibt 0011.)

Eingabeparameter

Y-Register : Operand 1
X-Register : Operand 2

Beispiel

Tastenfolge	Anzeige	Kommentar
4 WSIZE	4,0000	Einstellen der Wortbreite 4 Bit.
3 ENTER 1 AND	1,0000	0011 und 0001 ergibt 0001.
7 ENTER 8 AND	0,0000	0111 und 1000 ergibt 0000.
7 ENTER 15 AND	7,0000	0111 und 1111 ergibt 0111.

Weitere Hinweise

Bei ALPHA-Daten erfolgt die Fehlermeldung „ALPHA DATA". Wenn die Dezimalzahl aus dem X- oder aus dem Y-Register in der aktuellen Wortbreite und dem Vorzeichenmodus nicht darstellbar ist, erscheint die Fehlermeldung „DATA ERROR X" bzw. „DATA ERROR Y".

Verwandte Funktionen

OR, XOR, NOT

OR

Die Funktion OR verknüpft X und Y mit der logischen OR-Funktion, d.h., im Ergebnis sind alle Bits gesetzt, die vorher in den beiden einzelnen Binärzahlen gesetzt waren. Der Stack wird heruntergeschoben und der alte X-Wert im LAST X-Register abgelegt. (Binär heißt das z.B.: 1011 oder 0111 ergibt 1111.)

Eingabeparameter

Y-Register : Operand 1
X-Register : Operand 2

Beispiele

Tastenfolge	Anzeige	Kommentar
4 WSIZE	4,0000	Einstellen der Wortbreite 4 Bit.
3 ENTER 1 OR	3,0000	0011 oder 0001 ergibt 0011.
7 ENTER 8 OR	15,0000	0111 oder 1000 ergibt 1111.
7 ENTER 15 OR	15,0000	0111 oder 1111 ergibt 1111.

Weitere Hinweise

Bei ALPHA-Daten erfolgt die Fehlermeldung „ALPHA DATA". Wenn die Dezimalzahl aus dem X- oder aus dem Y-Register in der aktuellen Wortbreite und dem Vorzeichenmodus nicht darstellbar ist, erscheint die Fehlermeldung „DATA ERROR X" bzw. „DATA ERROR Y".

Verwandte Funktionen

AND, XOR, NOT

XOR

Die Funktion XOR verknüpft X und Y mit der logischen EXKLUSIV-ODER-Funktion, d.h.,im Ergebnis sind alle Bits gesetzt, die nur in einer der beiden ursprünglichen Binärzahlen gesetzt waren. Alle übrigen Bits sind gelöscht. Der Stack wird heruntergeschoben und der alte X-Wert im LAST X-Register abgelegt. (Binär heißt das z.B.: 1011 exklusiv-oder 0111 ergibt 1100).

Eingabeparameter

Y-Register : Operand 1
X-Register : Operand 2

Beispiele

Tastenfolge	Anzeige	Kommentar
4 WSIZE	4,0000	Einstellen der Wortbreite 4 Bit.
3 ENTER 1 XOR	2,0000	0011 exklusiv-oder 0001 ergibt 0010.
7 ENTER 8 XOR	15,0000	0111 exklusiv-oder 1000 ergibt 1111.
7 ENTER 15 XOR	8,0000	0111 exklusiv-oder 1111 ergibt 1000.

Weitere Hinweise

Bei ALPHA-Daten erfolgt die Fehlermeldung „ALPHA DATA". Wenn die Dezimalzahl aus dem X- oder aus dem Y-Register in der aktuellen Wortbreite und dem Vorzeichenmodus nicht darstellbar ist, erscheint die Fehlermeldung „DATA ERROR X" bzw. „DATA ERROR Y".

Verwandte Funktionen

AND, OR, NOT

NOT

Die Funktion **NOT** invertiert alle Bits der Zahl aus dem X-Register. Der alte X-Wert wird im LAST X-Register abgelegt. (Binär heißt das z.B.: nicht 1011 ergibt 0100.)

Eingabeparameter

X-Register : Dezimalzahl

Beispiele

Tastenfolge	Anzeige	Kommentar
4 **WSIZE**	4,0000	Einstellen der Wortbreite 4 Bit.
3 **NOT**	12,0000	Nicht 0011 ergibt 1100.
7 **NOT**	8,0000	Nicht 0111 ergibt 1000.
15 **NOT**	0,0000	Nicht 1111 ergibt 0000.

Weitere Hinweise

Bei ALPHA-Daten erfolgt die Fehlermeldung „ALPHA DATA". Wenn die Dezimalzahl aus dem X-Register in der aktuellen Wortbreite und dem Vorzeichenmodus nicht darstellbar ist, erscheint die Fehlermeldung „DATA ERROR X". Im Einerkomplement Modus hat die Ausführung von **NOT** einen Vorzeichenwechsel zur Folge.

Verwandte Funktionen

AND, **OR**, **XOR**

Funktionen zur Bitmanipulation

S<

Die Funktion S< verschiebt auf binärer Ebene die Bits des X-Registers um ein Bit (eine binäre Stelle) nach links. Das herausgeschobene Bit wird im Übertrags-Bit, also im Flag 0, abgespeichert. War bei der Zahl vor dem Verschieben das ganz linke Bit gesetzt, so wird Flag 0 gesetzt, wenn es gelöscht war, wird Flag 0 ebenfalls gelöscht. Beim Verschieben wird von rechts immer das Bit mit dem Wert 0 hereingeschoben.

Eingabeparameter

X-Register : Dezimalzahl, die in der gewählten Wortbreite und dem eingestellten Vorzeichenmodus binär darstellbar ist.

Beispiel

Tastenfolge	Anzeige	Kommentar
4 WSIZE	4,0000	Einstellen der Wortbreite auf 4 Bit.
UNS 1	1,0000	Wählen des Unsigned Modus.
S<	2,0000	Aus 0001 wird 0010, Flag 0 wird gelöscht (bleibt gelöscht).
S<	4,0000	Aus 0010 wird 0100, Flag 0 wird gelöscht (bleibt gelöscht).
S<	8,0000	Aus 0100 wird 1000, Flag 0 wird gelöscht (bleibt gelöscht).

S‹	0,0000	Aus 1000 wird 0000, Flag 0 wird gesetzt.
S‹	0,0000	0000 bleibt 0000, Flag 0 wird gelöscht.

Weitere Hinweise

S‹ entspricht im Unsigned und im Zweierkomplement Modus einer Multiplikation mit dez 2. Im Einerkomplement Modus entspricht die Verschiebung nach links im positiven Zahlenbereich ebenfalls einer Multiplikation mit 2, im negativen Bereich allerdings einer Multiplikation mit 2 und anschließender Subtraktion von 1. Bei ALPHA-Daten erfolgt die Fehlermeldung „ALPHA DATA". Wenn die Dezimalzahl aus dem X-Register in der aktuellen Wortbreite und dem Vorzeichenmodus nicht darstellbar ist, erscheint die Fehlermeldung „DATA ERROR X".

Verwandte Funktionen

S›, R‹, R›

S›

Die Funktion S› verschiebt auf binärer Ebene die Bits des X-Registers um ein Bit (eine binäre Stelle) nach rechts. Das herausgeschobene Bit wird im Übertrags-Bit, also im Flag 0, abgespeichert. War bei der Zahl vor dem Verschieben das ganz rechte Bit gesetzt, so wird Flag 0 gesetzt, wenn es gelöscht war, wird Flag 0 ebenfalls gelöscht. Beim Verschieben wird von links immer das Bit mit dem Wert 0 hereingeschoben.

Eingabeparameter

X-Register : Dezimalzahl, die in der gewählten Wortbreite und dem eingestellten Vorzeichenmodus binär darstellbar ist.

Beispiel

Tastenfolge	Anzeige	Kommentar
4 WSIZE	4,0000	Einstellen der Wortbreite auf 4 Bit.
UNS 8	8,0000	Wählen des Unsigned Modus.
S›	4,0000	Aus 1000 wird 0100, Flag 0 wird gelöscht (bleibt gelöscht).
S›	2,0000	Aus 0100 wird 0010, Flag 0 wird gelöscht (bleibt gelöscht).
S›	1,0000	Aus 0010 wird 0001, Flag 0 wird gelöscht (bleibt gelöscht).
S›	0,0000	Aus 0001 wird 0000, Flag 0 wird gesetzt.
S›	0,0000	0000 bleibt 0000, Flag 0 wird gelöscht.

Weitere Hinweise

Bei ALPHA-Daten erfolgt die Fehlermeldung „ALPHA DATA". Wenn die Dezimalzahl aus dem X-Register in der aktuellen Wortbreite und dem Vorzeichenmodus nicht darstellbar ist, erscheint die Fehlermeldung „DATA ERROR X".

Verwandte Funktionen

S‹, R‹, R›

R<

Die Funktion R< rotiert auf binärer Ebene die Bits des X-Registers um ein Bit (eine binäre Stelle) nach links. Das herausgeschobene Bit wird im Übertrags-Bit, also im Flag 0, abgespeichert und anschließend wieder von rechts hereingeschoben. War bei der Zahl vor dem Rotieren das ganz linke Bit gesetzt, so wird Flag 0 gesetzt, wenn es gelöscht war, wird Flag 0 ebenfalls gelöscht.

Eingabeparameter

X-Register : Dezimalzahl, die in der gewählten Wortbreite und dem eingestellten Vorzeichenmodus binär darstellbar ist.

Beispiel

Tastenfolge	Anzeige	Kommentar
4 WSIZE	4,0000	Einstellen der Wortbreite auf 4 Bit.
UNS 1	1,0000	Wählen des Unsigned Modus.
R<	2,0000	Aus 0001 wird 0010, Flag 0 wird gelöscht (bleibt gelöscht).
R<	4,0000	Aus 0010 wird 0100, Flag 0 wird gelöscht (bleibt gelöscht).
R<	8,0000	Aus 0100 wird 1000, Flag 0 wird gelöscht (bleibt gelöscht).
R<	1,0000	Aus 1000 wird 0001, Flag 0 wird gesetzt.
R<	2,0000	Aus 0001 wird 0010, Flag 0 wird gelöscht.

Weitere Hinweise

Bei ALPHA-Daten erfolgt die Fehlermeldung „ALPHA DATA". Wenn die Dezimalzahl aus dem X-Register in der aktuellen Wortbreite und dem Vorzeichenmodus nicht darstellbar ist, erscheint die Fehlermeldung „DATA ERROR X".

Verwandte Funktionen

S‹, S›, R›

R›

Die Funktion R› rotiert auf binärer Ebene die Bits des X-Registers um ein Bit (eine binäre Stelle) nach rechts. Das herausgeschobene Bit wird im Übertrags-Bit, also im Flag 0, abgespeichert und anschließend von links wieder hereingeschoben. War bei der Zahl vor dem Rotieren das ganz rechte Bit gesetzt, so wird Flag 0 gesetzt, wenn es gelöscht war, wird Flag 0 ebenfalls gelöscht.

Eingabeparameter

X-Register : Dezimalzahl, die in der gewählten Wortbreite und dem eingestellten Vorzeichenmodus binär darstellbar ist.

Beispiel

Tastenfolge	Anzeige	Kommentar
4 WSIZE	4,0000	Einstellen der Wortbreite auf 4 Bit.
UNS 8	8,0000	Wählen des Unsigned Modus.
R›	4,0000	Aus 1000 wird 0100, Flag 0 wird gelöscht (bleibt gelöscht).
R›	2,0000	Aus 0100 wird 0010, Flag 0 wird gelöscht (bleibt gelöscht).
R›	1,0000	Aus 0010 wird 0001, Flag 0 wird gelöscht (bleibt gelöscht).
R›	8,0000	Aus 0001 wird 1000, Flag 0 wird gesetzt.
R›	4,0000	Aus 1000 wird 0100, Flag 0 wird gelöscht.

Weitere Hinweise

Bei ALPHA-Daten erfolgt die Fehlermeldung „ALPHA DATA". Wenn die Dezimalzahl aus dem X-Register in der aktuellen Wortbreite und dem Vorzeichenmodus nicht darstellbar ist, erscheint die Fehlermeldung „DATA ERROR X".

Verwandte Funktionen

S‹, S›, R‹

bS?

Die Funktion **bS?** ermöglicht die Abfrage jedes einzelnen Bits der im Y-Register dezimal abgelegten Binärzahl. Diese Frage ist vergleichbar einer Abfrage eines einzelnen Flags, nachdem die im X-Register abgelegte Zahl mit **X→F** in die Flags 0-7 übertragen wurde (**X→F** ist im X-Functions Modul bzw. im HP-41 CX enthalten). Der Umweg über die Flags ist bei den Bitmanipulationsfunktionen des CCD-Moduls jedoch nicht erforderlich. Die Antwort auf **bS?** ist „YES", wenn das im X-Register angegebene Bit *bb* gesetzt ist, sie ist „NO", wenn es gelöscht ist. Es erfolgt ein Stack-Drop, der alte X-Wert wird in das LAST X-Register geschrieben.

Eingabeparameter

X-Register : *bb* (Dezimalzahl; 0 <= *bb* <= aktuelle Wortbreite – 1)
Y-Register : Dezimalzahl, die in der gewählten Wortbreite und dem eingestellten Vorzeichenmodus binär darstellbar ist.

Beispiele

Tastenfolge	Anzeige	Kommentar
4 **WSIZE** **UNS**	4,0000	Einstellen der Wortbreite 4 Bit und des Unsigned Modus.
7 **ENTER** 0 **bS?**	YES	Das Bit 0 der Binärzahl bin 0111 bzw. dez 7 ist gesetzt.
←	7,0000	Löschen der Anzeige „YES"
7 **ENTER** 3 **bS?**	NO	Das Bit 3 der Binärzahl bin 0111 bzw. dez 7 ist nicht gesetzt.

Weitere Hinweise

Bei ALPHA-Daten erfolgt die Fehlermeldung „ALPHA DATA". Wenn die Dezimalzahl aus dem Y-Register in der aktuellen Wortbreite und dem Vorzeichenmodus nicht darstellbar ist, erscheint die Fehlermeldung „DATA ERROR Y". Die Fehlermeldung „DATA ERROR X" erscheint, wenn das angegebene Bit *bb* außerhalb des erlaubten Wertebereichs liegt. Die Numerierung der Bits geschieht von rechts nach links, das rechteste Bit ist das nullte Bit. Die höchste Bit-Nummer, die angegeben werden kann, ist immer um eins kleiner als die aktuelle Wortbreite (bei Wortbreite 8 können die Bits 0 bis 7 angegeben werden, also alle 8 Bits der Binärzahl). Im Programmlauf wird der nächste Programmschritt ausgeführt, wenn das angegebene Bit gesetzt ist; wenn es gelöscht ist, wird der nächste Programmschritt übersprungen.

Verwandte Funktionen

bC?, Sb, Cb

bC?

Die Funktion **bC?** ermöglicht die Abfrage jedes einzelnen Bits der im Y-Register dezimal abgelegten Binärzahl. Die Antwort ist „YES", wenn das im X-Register angegebene Bit *bb* gelöscht ist, sie ist „NO", wenn es gesetzt ist. Es erfolgt ein Stack-Drop, der alte X-Wert wird in das LAST X-Register geschrieben.

Eingabeparameter

X-Register : *bb* (Dezimalzahl; 0 <= *bb* <= aktuelle Wortbreite – 1)
Y-Register : Dezimalzahl, die in der gewählten Wortbreite und dem eingestellten Vorzeichenmodus binär darstellbar ist.

Beispiele

Tastenfolge	Anzeige	Kommentar
4 **WSIZE** **UNS**	4,0000	Einstellen der Wortbreite 4 Bit und des Unsigned Modus.
7 **ENTER** 0 **bC?**	NO	Das Bit 0 der Binärzahl bin 0111 bzw. dez 7 ist nicht gelöscht.
←	7,0000	Löschen der Anzeige „NO".
7 **ENTER** 3 **bC?**	YES	Das Bit 3 der Binärzahl bin 0111 bzw. dez 7 ist gelöscht.

Weitere Hinweise

Bei ALPHA-Daten erfolgt die Fehlermeldung „ALPHA DATA". Wenn die Dezimalzahl aus dem Y-Register in der aktuellen Wortbreite und dem Vorzeichenmodus nicht darstellbar ist, erscheint die Fehlermeldung „DATA ERROR Y". Die Fehlermeldung „DATA ERROR X" erscheint, wenn das angegebene Bit *bb* außerhalb des erlaubten Wertebereichs liegt. Die Numerierung der Bits geschieht von rechts nach links, das rechteste Bit ist das nullte Bit. Die höchste Bit-Nummer, die angegeben werden kann, ist immer um eins kleiner als die aktuelle Wortbreite (bei Wortbreite 8 können die Bits 0 bis 7 angegeben werden, also alle 8 Bits der Binärzahl). Im Programmlauf wird der nächste Programmschritt ausgeführt, wenn das angegebene Bit gelöscht ist; wenn es gesetzt ist, wird der nächste Programmschritt übersprungen.

Verwandte Funktionen

bS?, **Sb**, **Cb**

Cb

Die Funktion **Cb** ermöglicht das Löschen jedes einzelnen Bits der im Y-Register dezimal abgelegten Binärzahl. Das im X-Register angegebene Bit *bb* der Zahl im Y-Register wird gelöscht. Es erfolgt ein Stack-Drop, der alte X-Wert wird in das LAST X-Register geschrieben.

Eingabeparameter

X-Register : *bb* (Dezimalzahl; 0 <= *bb* <= aktuelle Wortbreite – 1)
Y-Register : Dezimalzahl, die in der gewählten Wortbreite und dem eingestellten Vorzeichenmodus binär darstellbar ist.

Beispiele

Tastenfolge	Anzeige	Kommentar
4 **WSIZE** **UNS**	4,0000	Einstellen der Wortbreite 4 Bit und des Unsigned Modus.
7 **ENTER** 0 **Cb**	6,0000	Das Bit 0 der Binärzahl bin 0111 wurde gelöscht, d.h. aus 0111 ist 0110 geworden.
7 **ENTER** 3 **Cb**	7,0000	Das Bit 3 der Binärzahl bin 0111 wurde nicht verändert, da es bereits gelöscht war.

Weitere Hinweise

Bei ALPHA-Daten erfolgt die Fehlermeldung „ALPHA DATA". Wenn die Dezimalzahl aus dem Y-Register in der aktuellen Wortbreite und dem Vorzeichenmodus nicht darstellbar ist, erscheint die Fehlermeldung „DATA ERROR Y". Die Fehlermeldung „DATA ERROR X" erscheint, wenn das angegebene Bit *bb* außerhalb des erlaubten Wertebereichs liegt. Die Numerierung der Bits geschieht von rechts nach links, das rechteste Bit ist das nullte Bit. Die höchste Bit-Nummer, die angegeben werden kann, ist immer um eins kleiner als die aktuelle Wortbreite (bei Wortbreite 8 können die Bits 0 bis 7 angegeben werden, also alle 8 Bits der Binärzahl).

Verwandte Funktionen

bC?, bS?, Sb

Sb

Die Funktion Sb ermöglicht das Setzen jedes einzelnen Bits der im Y-Register dezimal abgelegten Binärzahl. Das im X-Register angegebene Bit *bb* der Zahl im Y-Register wird gesetzt. Es erfolgt ein Stack-Drop, der alte X-Wert wird in das LAST X-Register geschrieben.

Eingabeparameter

X-Register	: *bb* (Dezimalzahl; 0 <= *bb* <= aktuelle Wortbreite – 1)
Y-Register	: Dezimalzahl, die in der gewählten Wortbreite und dem eingestellten Vorzeichenmodus binär darstellbar ist.

Beispiele

Tastenfolge	Anzeige	Kommentar
4 WSIZE UNS	4,0000	Einstellen der Wortbreite 4 Bit und des Unsigned Modus.
7 ENTER 0 Sb	6,0000	Das Bit 0 der Binärzahl bin 0111 wurde nicht verändert, da es bereits gesetzt war.
7 ENTER 3 Sb	8,0000	Das Bit 3 der Binärzahl bin 0111 wurde gesetzt, aus 0111 wurde 1111.

Weitere Hinweise

Bei ALPHA-Daten erfolgt die Fehlermeldung „ALPHA DATA". Wenn die Dezimalzahl aus dem Y-Register in der aktuellen Wortbreite und dem Vorzeichenmodus nicht darstellbar ist, erscheint die Fehlermeldung „DATA ERROR Y". Die Fehlermeldung „DATA ERROR X" erscheint, wenn das angegebene Bit *bb* außerhalb des erlaubten Wertebereichs liegt. Die Numerierung der Bits geschieht von rechts nach links, das rechteste Bit ist das nullte Bit. Die höchste Bit-Nummer, die angegeben werden kann, ist immer um eins kleiner als die aktuelle Wortbreite (bei Wortbreite 8 können die Bits 0 bis 7 angegeben werden, also alle 8 Bits der Binärzahl).

Verwandte Funktionen

bC?, bS?, Cb

Programmbeispiele

Nachfolgend finden Sie zwei hilfreiche Programme, die Ihnen den Umgang mit den CCD-Modul-Binärfunktionen erleichtern und zum besseren Verständnis der Funktionen beitragen sollen.

Bestimmung der eingestellten Wortbreite

Mit folgendem Programm läßt sich die zur Zeit aktive Wortbreite berechnen:

```
01◆LBL  "W?"
02 CLX
03 UNS
04 NOT
05 LN1+X
06 4
07 LN
08 /
09 ST+ X
10 END
```

Löschen von Flag 55

Bei eingestecktem Drucker ist normalerweise das Druckeranwesenheitsflag 55 gesetzt. In Programmteilen, die keinen Drucker benötigen, ist es manchmal sinnvoll,Flag 55 zu löschen, da die Programme mit gelöschtem Flag 55 schneller ablaufen. Dieses Löschen kann man z.B. mit folgendem Programm erreichen:

```
01◆LBL  "CF5
          5"
02  14
03  PEEKB
04  0
05  Cb
06  POKEB
07  END
```

Beim Ausführen einer Druckerfunktion oder einer Flag 55 Abfrage wird – bei angeschlossenem Drucker – das Flag 55 automatisch gesetzt.

Kapitel 6

Ein-/Ausgabe-funktionen

Inhaltsverzeichnis Kapitel 6
Ein-/Ausgabefunktionen

Ein- und Ausgabefunktionen

Die nachfolgenden Funktionen sollen dem Anwender beim dialogorientierten Programmieren des HP-41 helfen. Mit diesen Funktionen sind alle Möglichkeiten der formatierten Ein- und Ausgabe gegeben. Der Anwender kann sich somit voll auf die eigentliche Lösung des Problems konzentrieren, ohne sich übermäßig um Ein- und Ausgabe Gedanken machen zu müssen. Besonders die Ausgabe auf großen HP-IL Ausgabegeräten, wie ThinkJet Printer oder Video Interface, ist mit den neuen Funktionen des CCD-Moduls so einfach wie noch nie geworden. Weiterhin helfen die CCD-Modul-Funktionen sehr viel Speicherplatz sparen, der sonst für Unterprogramme benötigt wird, wenn man ähnliches in normalem USER-Code schreibt. Des weiteren kann die Ausführungszeit der Programme durch die Anwendung der CCD-Modul-Funktionen wesentlich verringert werden.

Eingabefunktionen

INPT

Die Funktion **INPT** ist eine universelle Dateneingabefunktion. Mit ihrer Hilfe kann man nun Datenblöcke sehr komfortabel füllen. Diese Funktion ersetzt folgenden USER-Code:

```
01◆LBL 05
02 TONE 0

03◆LBL "INP
        "
04 "⊢:  "
```

An den Inhalt des ALPHA-Registers werden ein Doppelpunkt sowie ein Leerzeichen angehängt.

```
05 RCL IND
      00
06 ARCL X
07 PROMPT
08 CLA-
09 ABSP
10 ABSP
11 1
12 X<>Y
13 X<NN?
14 GTO 05
15 2
16 X<>Y
17 X>NN?
18 GTO 05
19 STO IND
      00
20 ISG 00
21 END
```

Der durch das Kontrollregister 00 angegebene Registerinhalt wird ins X-Register geschrieben und an das ALPHA-Register angehängt.

ALPHA wird angezeigt und die Programmausführung gestoppt; der Rechner wartet nun auf eine Zahleneingabe.

Das ALPHA-Register wird nun bis auf den ursprünglichen Inhalt gelöscht (Löschen von links bis zum Doppelpunkt einschl.).

stehen,verglichen. Liegt der Wert außerhalb dieser Grenzen, so beginnt das Programm von vorne (Die Vergleichsfunktionen **X‹NN?** und **X›NN?** sind HP-41 CX Funktionen).

Der Wert liegt innerhalb der vorgegebenen Eingabegrenzen. Mit diesem Wert wird nun der alte Inhalt des durch das Register 00 spezifizierten Registers überschrieben.

Die Kontrollzahl im Register 00 wird nun mit **ISG** inkrementiert (siehe **ISG**).

```
PLNG "INP"
  44 BYTES
```

Wie man sieht, spart man mit der Funktion **INPT** ein ganzes komplexes Programm, welches relativ viel Speicherplatz verbrauchen würde. In Wirklichkeit besteht die Funktion **INPT** aus zwei verschiedenen Funktionsteilen. Beim erstmaligen Ausführen der Funktion wird nur der erste Funktionsteil ausgeführt. Dies entspricht in unserem obigen USER-Programm dem Code bis zur Zeile 07 **PROMPT**. Dann wird von der Funktion **INPT** aus einmal **BST** ausgeführt, so daß der Programmzeiger wieder auf **INPT** zeigt. Damit die Funktion beim zweiten Ausführen nun weiß, das nicht der erste, sondern der zweite Teil der Funktion ausgeführt werden soll, setzt sie das CCD-Modul Input-Flag (Bit 5 vom Reg.-Byte 13,4). Jetzt wird der Anwender zur Dateneingabe aufgefordert, und nach Drücken der R/S-Taste wird der zweite Teil der **INPT**-Funktion ausgeführt. Dieser Teil vergleicht den

Eingabewert mit den Eingabegrenzen aus den Registern 01 und 02 und speichert ihn, entsprechend der Kontrollzahl in Register 00, ab. Wurde kein neuer Wert eingegeben, so wird der alte übernommen, wenn er innerhalb der Eingabegrenzen liegt. Sodann wird die Kontrollzahl *iii,fffcc* aus dem Register 00 um den Wert *cc* erhöht und die nächste Programmzeile übersprungen, wenn gilt: *iii>fff* (siehe auch **ISG**).

Eingabeparameter

R000: Kontrollzahl *iii,fffcc*
R001: minimal möglicher Eingabewert
R002: maximal möglicher Eingabewert

Beispiele

Während eines chemischen Experiments werden 10 verschiedene pH-Werte gemessen. Diese sollen zur Auswertung mit einem Programm in die Datenregister 10 bis 19 eingegeben werden. Ein Programm hierzu könnte folgendermaßen aussehen:

```
01◆LBL  "PHI
        NPT"
02  10.019
03  STO  00
04  CLX
05  STO  01
06  14
07  STO  02
08  "PH"

09◆LBL  01
10  INPT
11  GTO  01
12  BEEP
13  END
PLNG  "PHINPT
           "
     34  BYTES
```

Die Kontrollzahl *iii,fffcc* wird in R000 abgespeichert.
Die Eingabegrenzen werden abgespeichert (min = 0, max = 14). Dadurch werden unmögliche Eingaben ausgeschlossen.

In dieser Schleife werden nun alle 10 Werte nacheinander eingegeben.

Fertig!

Weitere Hinweise

Die Funktion INPT führt, wie alle Prompt-Funktionen des CCD-Moduls, während der Ausführung im Programm einmal BST aus. Wird BST am Ende eines Programms ausgeführt, so dauert das bei großen Programmen relativ lange. Daher empfiehlt es sich, alle Prompt-Funktionen des CCD-Moduls an den Anfang des betreffenden Programms zu setzen (eventuell in einem Unterprogramm) oder kurz hinter ein globales Label. Des weiteren setzt die Funktion INPT, wie auch die Funktion PMTA, das Input-Flag des CCD-Moduls (Bit 5 vom Reg.-Byte 13,4). Dies bedeutet, daß, wenn die Funktion INPT unrichtig abgebrochen wird (richtig sind nur die Tasten R/S, ← und ON), dieses Flag noch gesetzt ist, so daß bei erneutem Ausführen der Funktion nur der zweite Teil ausgeführt wird. Will man sicher sein, daß das Input-Flag an einer bestimmten Stelle im Programm gelöscht ist, so fügt man am besten folgende Programmzeilen vor der INPT-Instruktion ein:

```
13,4
1
POKEB
```

Diese Anweisungen löschen das CCD-Modul-Input-Flag.

Möchte man einmal in einem Programm nur den zweiten Teil der INPT-Funktion ausführen, so ist die Zahl 1 durch die Zahl 33 zu ersetzen.

Verwandte Funktionen

ISG, PMTA

PMTA

Die Funktion PMTA eröffnet die Möglichkeit einer komfortablen ALPHA-Eingabe. Wie die Funktion INPT besteht auch diese Funktion aus zwei Teilen. Wird PMTA in einem laufenden Programm ausgeführt, so wird der Programmablauf unterbrochen und der Programmzeiger um eine Programmzeile auf die Funktion PMTA zurückgesetzt. Jetzt wird das CCD-Modul Input-Flag (Bit 5 vom Reg.-Byte 13,4) gesetzt sowie das ALPHA-Register und ein Prompt-Zeichen angezeigt (ALPHA wird angeschaltet!). Mit ← und ON kann nun die Funktion abgebrochen werden, ohne daß der ursprüngliche ALPHA-Inhalt verlorengeht. Drückt man eine andere Taste, so wird auf jeden Fall das ALPHA-Register gelöscht, was jedoch keinen Einfluß auf die Anzeige hat. Ist diese Taste eine Buchstabentaste, so wird das ALPHA-Register mit dem entsprechenden Buchstaben überschrieben und dieser an die Anzeige angehängt. Nach Drücken der Taste R/S wird PMTA nun zum zweiten Mal ausgeführt. Dies erkennt die Funktion daran, daß das Input-Flag des CCD-Moduls noch gesetzt ist. Jetzt wird dieses Flag gelöscht, ALPHA ausgeschaltet und das Flag 23 gesetzt, wenn in ALPHA etwas eingegeben wurde.

Eingabeparameter

keine

Beispiel

Folgendes Unterprogramm veranschaulicht die Wirkungsweise der Funktion PMTA. Der Anwender wird in diesem Unterprogramm nach seinem Namen gefragt.

```
01*LBL "NAM
         E"
02*LBL 01
03 CF 23
04 "NAME: "
05 PMTA
06 FC? 23
07 GTO 01
08 END
```

Das ALPHA-Eingabe-Flag wird gelöscht und nach dem Namen gefragt.

Wurde ein Name in ALPHA eingegeben?
Wenn nicht,
gehe zurück und frage noch einmal.
Es wurde ein Name eingegeben. Fertig!

Weitere Hinweise

Ist das ALPHA-Register beim Ausführen der Funktion PMTA leer, so wird statt dessen der String „TEXT: _ " angezeigt. Die Funktion PMTA führt, wie alle Prompt-Funktionen des CCD-Moduls, während der Ausführung im Programm einmal BST aus. Wird BST am Ende eines Programms ausgeführt, so dauert das bei großen Programmen relativ lange. Daher empfiehlt es sich, alle Prompt-Funktionen des CCD-Moduls an den Anfang des betreffenden Programms zu setzen (eventuell in einem Unterprogramm), oder kurz hinter ein globales Label. PMTA benutzt auch das Input-Flag des CCD-Moduls (siehe hierzu auch INPT).

Verwandte Funktionen

INPT, PMTH, PMTK, PROMPT

PMTH

Die Funktion PMTH ist die Eingabefunktion des CCD-Moduls für Hexadezimalzahlen. Wird PMTH von der Tastatur aus ausgeführt, so wird, entsprechend der eingestellten Wortbreite (siehe *Binärfunktionen*), der Anwender aufgefordert, Hexadezimalzahlen einzugeben. Jetzt sind die Tasten 1-9 und A-F aktiv. Wird eine Hexadezimalzahl eingegeben, für die die eingestellte Wortbreite zu klein ist, so beginnt die Funktion von vorn und fordert erneut zur Eingabe auf. Ist die Eingabe korrekt, so wird der Stack angehoben und die eingegebene Zahl, nach Umwandlung in eine Dezimalzahl, ins X-Register geschrieben. Wird die Funktion in einem laufenden Programm ausgeführt, so wird zusätzlich noch das ALPHA-Register angezeigt und der Programmlauf unterbrochen. Nach Eingabe des letzten Hexdigits wird die Programmausführung automatisch fortgesetzt.

Eingabeparameter

Die Anzahl der einzugebenden Hexdigits ist abhängig von der aktuellen Wortbreite.

Beispiel

Das folgende Programm wandelt eine Hexadezimalzahl in eine Oktalzahl um und umgekehrt. Es wird automatisch zur Eingabe der entsprechenden Zahlen aufgefordert.

```
01◆LBL  "H-O
              "
02  16
03  WSIZE
04  UNS
05  "HEX"
06  PMTH
07  OCT
08  "OKT:  "
09  ARCLI
10  PROMPT
```

Umwandlung von HEX nach OKT
Einstellen der Wortbreite auf 16 Bit. Die größte HEX-Zahl ist somit FFFF entspricht okt 177777.
Einstellen des Unsigned-Modus.
Mit „HEX“ in ALPHA wird nun mit der Funktion PMTH zur HEX-Eingabe aufgefordert. Die entstandene Dezimalzahl wird zu einer Oktalzahl umgewandelt und an den ALPHA- String „OKT:“ als Integerzahl angehängt. Nun wird das oktale Ergebnis angezeigt. Fertig!

```
11*LBL "O-H
       "
12  "OKT _"
13  PROMPT
14  DEC
15  "HEX'"
16  ARCLH
17  PROMPT
18  END

 PLNG "H-O"
    55 BYTES
```

Umwandlung von OKT nach HEX

Mit dem String „OKT _ " wird zur Eingabe einer Oktalzahl aufgefordert.
Diese wird in ihr dezimales Äquivalent umgewandelt. Die hexadezimale Darstellung dieser Zahl wird nun an den ALPHA-String „HEX'" angehängt.
Nun wird das hexadezimale Ergebnis angezeigt.
Fertig!

Weitere Hinweise

Die Funktion **PMTH** führt, wie alle Prompt-Funktionen des CCD-Moduls, während der Ausführung im Programm einmal **BST** aus. Wird **BST** am Ende eines Programms ausgeführt, so dauert das bei großen Programmen relativ lange. Daher empfiehlt es sich, alle Prompt-Funktionen des CCD-Moduls an den Anfang des betreffenden Programms zu setzen (eventuell in einem Unterprogramm) oder kurz hinter ein globales Label.

Verwandte Funktion

ARCLH

PMTK

Mit der Funktion **PMTK** ist nun endlich auch eine Menüsteuerung für den HP-41 möglich. Das ALPHA-Register wird angezeigt und die Programmausführung unterbrochen. Dann wartet der Rechner darauf, daß der Anwender eine Taste drückt. Hierbei gibt es vier verschiedene Möglichkeiten:

1) Die **ON**-Taste schaltet den Rechner aus. Der Programmzeiger steht immer noch auf der Funktion **PMTK**, so daß beim Starten des Programms an dieser Stelle die Funktion nochmals ausgeführt wird.
2) Es wird eine unzulässige Taste gedrückt. Dies beantwortet der Rechner mit einem kurzen Ton (nur bei gesetztem Flag 26).
3) Es wird eine zulässige Taste gedrückt. Zulässig sind die Tasten, deren zugehöriges ALPHA-Zeichen (von der normalen ALPHA- und der normalen SHIFT-ALPHA-Tastatur) in der Anzeige erscheint. Diese Zeichen sind durch Doppelpunkte voneinander getrennt. Zusätzlich kann aber auch noch erklärender Text im ALPHA-Register eingegeben werden, der auf die Menüsteuerung keinen Einfluß hat. Dieser Text muß zuerst ins ALPHA-Register eingegeben werden (linksbündig), gefolgt von mindestens einem Leerzeichen und den zulässigen ALPHA-Zeichen (rechtsbündig). Die Funktion **PMTK** unterscheidet nun zwischen dem erklärenden Text und den zulässigen Zeichen anhand eines Leerzeichens, welches die beiden Textgruppen voneinander trennt. Wird nun eine Taste, deren ALPHA-Zeichen zulässig ist, gedrückt, so wird der Stellenwert des Zeichens ins X-Register geschrieben. Der Stack wird angehoben. Diese Zahl, die tastenabhängig ist, kann nun zur Programmverzweigung genutzt werden. Das ALPHA-Register wird bis auf den Kommentartext und ein Leerzeichen gelöscht.
4) Bei leerem ALPHA-Register wird „KEY?“ angezeigt und der Tastencode (siehe auch **ASN** und **GETKEY** aus dem Extended Functions Modul) der als nächstes gedrückten Taste ins X-Register geschrieben. Der Stack wird angehoben.

Eingabeparameter

ALPHA-Register. Kommentartext und zulässige Tastenzeichen, getrennt durch mindestens ein Leerzeichen oder leeres ALPHA-Register.

Beispiel

An einem einfachen Beispiel soll die Menüsteuerung erklärt werden. Gesteuert durch verschiedene Tastendrücke, soll der HP-41 zwischen 1 und 4 mal BEEP ausführen.

```
01◆LBL "KEY
       "
02 "1234"
03 PMTK

04 GTO IND
          X
05◆LBL 04
06 BEEP

07◆LBL 03
08 BEEP

09◆LBL 02
10 BEEP

11◆LBL 01
12 BEEP
13 END
```

Die zulässigen Tasten werden ins ALPHA-Register geschrieben.
Mit PMTK erscheint „1:2:3:4" in der Anzeige und der Rechner erwartet einen Tastendruck.

Je nach dem Stellenwert, der hier dem ALPHA Zeichen entspricht, wird nun zum LBL 1-4 verzweigt.

Im folgenden Beispiel wird eine Anwendung gezeigt, wo zusätzlich ein Kommentartext mit angezeigt werden soll. Der Anwender wird gefragt, ob die eingegebenen Daten auf Band abgespeichert werden sollen oder nicht.

```
01*LBL "?>C
       AS"
02 "->BAND?
       JN"
03 PMTK
04 X<>Y
05 GTO IND
       Y
06*LBL 01
07 WRTRX
08*LBL 02
09 END
```

Kommentartext und zulässige Zeichen (J und N) werden ins ALPHA-Register geschrieben.

Mit PMTK erscheint in der Anzeige: "->BAND? J:N"; J und N sind durch einen Doppelpunkt voneinander getrennt.

Die Kontrollzahl für die Funktion WRTRX wird mit dem Wert 1 oder 2, der durch die Funktion PMTK erzeugt wurde, vertauscht.

Je nachdem ob J für Ja oder N für Nein gedrückt wurde, wird zum entsprechenden Label 01 oder 02 verzweigt.

Schreibe auf Band.

Bei LBL 02 erfolgt keine Operation.

Fertig!

Bei leerem ALPHA-Register arbeitet die Funktion PMTK wie folgt:

Tastenfolge	Anzeige	Kommentar
CLA PMTK	KEY?	Es wird zu einem Tastendruck aufgefordert.
SHIFT	31,0000	Der Tastencode der SHIFT-Taste (Reihe 3, Spalte 1 auf der Tastatur) wird ins X-Register geschrieben.

Weitere Hinweise

Drückt man versehentlich eine falsche Taste, so kann dies rückgängig gemacht werden, indem man sie so lange gedrückt hält, bis „NULL" in der Anzeige erscheint. Bei leerem ALPHA-Register wird in diesem Fall der Tastencode der gedrückten Taste angezeigt. Soll im ALPHA-Register hinter dem Leerzeichen, welches den Kommentartext und die zulässigen Zeichen trennt, ein Punkt, Doppelpunkt oder Komma eingegeben werden, so sind mindestens zwei Leerzeichen zur Trennung erforderlich.

Folgende zusätzliche Zeichen sind ebenfalls brauchbar:

Zeichencode dezimal	**Taste**
005	R/S
007, 008	SST
012	USER, PRGM, ALPHA
014	SHIFT

Die Funktion PMTK führt, wie alle Prompt-Funktionen des CCD-Moduls, während der Ausführung im Programm einmal BST aus. Wird BST am Ende eines Programms ausgeführt, so dauert das bei großen Programmen relativ lange. Daher empfiehlt es sich, alle Prompt-Funktionen des CCD-Moduls an den Anfang des betreffenden Programms zu setzen (eventuell in einem Unterprogramm), oder kurz hinter ein globales Label.

Verwandte Funktionen

PMTA, PMTH, GETKEY

Ausgabefunktionen

Funktionen zur Druckerausgabe

ACAXY

Mit **ACAXY** (Accumulate ALPHA and X by Y) ist es möglich, Text und Zahl aus dem X-Register formatiert auf der einmal eingestellten Druckbreite in den Druckbuffer zu übertragen. Dabei werden der Text (in ALPHA) linksbündig und der X-Wert rechtsbündig übertragen. Der Zwischenraum wird automatisch mit Leerzeichen gefüllt. Ist Flag 20 gesetzt, so wird dieser Zwischenraum mit Punkten gefüllt. Wenn Text und X-Wert zusammen mehr Platz benötigen, als an Druckbreite vorgegeben wurde, so wird die Druckbreite so lange verdoppelt, bis genügend Platz für einen Zwischenraum vorhanden ist. Bei leerem ALPHA-Register wird nur der X-Wert übertragen (rechtsbündig). Dies erlaubt auf einfache Weise einen formatierten Tabellenausdruck. Im FIX/ENG-Modus wird der X-Wert im **ENG**-Format mit einem Buchstaben (entsprechend den international festgelegten SI-Vorsätzen) statt einem Exponenten übertragen. Nach dem Ausführen der Funktion wird der X-Wert in das LAST X-Register geschrieben und der Stack nach unten verschoben (Stack-Drop). **ACAXY** arbeitet nur mit angeschlossenem Drucker bzw. mit eingestecktem HP-IL Modul.

Eingabeparameter

X-Register	: Zahl, die rechtsbündig formatiert übertragen werden soll.
Y-Register	: Druckbreite (Zahl zwischen 0 und 99).
ALPHA-Register	: Text, der linksbündig ausgedruckt werden soll.

Beispiele

Beispiel 1
Es soll auf einem 24-Zeichen Drucker folgende Zeile ausgedruckt werden:

```
BREITE:     200.00 METER
```

Im Programm-Modus sind hierzu nur folgende Programmschritte notwendig:

```
01◆LBL "PR"
02 "BREITE:
           "
03 18
04 RCL 00
05 ACAXY
06 " METER"
07 ACA
08 PRBUF
09 END
```

Beispiel 2
Die Registerinhalte der Register 1–9 sollen in Tabellenform auf einem 24-Zeichen Drucker ausgedruckt werden:

Registerinhalte:

```
R01=   200.00
R02=   15.00
R03=   16.20
R04=   150.00
R05=   20.50
R06=   12.20
R07=   159.00
R08=   18.80
R09=   42.32
```

Ausdruck mit dem Programm TAB:

```
200.00   15.00   16.20
150.00   20.50   12.20
159.00   18.80   42.32
```

Programmlisting:

```
01◆LBL "TAB
       "
02 CLA
03 1.009
04 8

05◆LBL 01
06 RCL IND
         Y
07 ACAXY
08 ISG Y
09 GTO 01
10 PRBUF
11 END
```

Weitere Hinweise

Falls die Druckbreite im Y-Register Null ist, wird die Standarddruckbreite von 24 Zeichen angenommen. Bei negativen Druckbreiten wird der Absolutwert gewählt.
Eine Tabelle der international festgelegten SI-Vorsätze findet sich bei der Funktion ARCLE.

Verwandte Funktion

PRAXY

PRAXY

Mit **PRAXY** (Print ALPHA and X by Y) ist es möglich, Text und Zahl aus dem X-Register formatiert auf der einmal eingestellten Druckbreite zu drucken. Dabei werden der Text (in ALPHA) linksbündig und der X-Wert rechtsbündig gedruckt. Der Zwischenraum wird automatisch mit Leerzeichen gefüllt. Ist Flag 20 gesetzt, so wird dieser Zwischenraum mit Punkten gefüllt. Wenn Text und X-Wert zusammen mehr Platz benötigen, als an Druckbreite vorgegeben wurde, so wird die Druckbreite|so lange| verdoppelt, bis genügend Platz für einen Zwischenraum vorhanden ist. Bei leerem ALPHA-Register wird nur der X-Wert gedruckt (rechtsbündig). Dies erlaubt auf einfache Weise einen formatierten Ausdruck. Im FIX/ENG-Modus wird der X-Wert im **ENG**-Format mit einem Buchstaben (entsprechend den international festgelegten SI-Vorsätzen) statt einem Exponenten übertragen. Nach dem Ausführen der Funktion wird der X-Wert in das LAST X-Register geschrieben und der Stack nach unten verschoben (Stack-Drop). **PRAXY** arbeitet nur mit angeschlossenem Drucker bzw. mit eingestecktem HP-IL Modul.

Eingabeparameter

X-Register	: Wert, der rechtsbündig formatiert ausgedruckt werden soll.
Y-Register	: Druckbreite (Zahl zwischen 0 und 99).
ALPHA-Register	: Text, der linksbündig ausgedruckt werden soll.

Beispiel

Es soll auf einem 40-Zeichen Drucker folgende Zeile ausgedruckt werden:

```
Stueckzahl:..................... 120
```

Im Programm-Modus sind hierzu nur folgende Programmschritte notwendig:

```
01◆LBL "PR3
       "
02 FIX 0
03 "Stueckz
     ahl:"
04 SF 20
05 40
06 120
07 PRAXY
08 END
```

Weitere Hinweise

Falls die Druckbreite im Y-Register Null ist, wird die Standarddruckbreite von 24 Zeichen angenommen. Bei negativen Druckbreiten wird der Absolutwert gewählt.
Eine Tabelle der international festgelegten SI-Vorsätze findet sich bei der Funktion ARCLE.

Verwandte Funktion

ACAXY

ACLX

ACLX (Accumulate line by X) überträgt eine Anzahl gleicher Zeichen in den Druckbuffer. Die Anzahl sowie der Zeichencode werden im X-Register in der Form *aa,bbb* spezifiziert. *aa* gibt die Anzahl der Zeichen an, *bbb* den Zeichencode (0-255). Mit dieser Funktion ist es möglich, Linien und Unterstreichungen beliebiger Länge auf jedem Ausgabegerät speicherplatzsparend und schnell auszugeben. **ACLX** arbeitet nur mit angeschlossenem Drucker bzw. mit eingestecktem HP-IL Modul.

Eingabeparameter

X-Register : *aa,bbb*

Beispiel

Auf einem 24-Zeichen Drucker sollen eine Zahlenkolonne rechts unterstrichen und der Druckstreifen durch zwei Linien abgegrenzt werden:

Ausdruck:

```
++++++++++++
      200.00
      ------
************
```

Programmlisting:

```
01*LBL "PR1
                "
02 SF 12
03 CLX
04 ACLX
05 RCL 00
06 ACX
07 ADV
08 6.045
09 ACLX
10 ADV
11 .042
12 ACLX
13 ADV
14 END
```

Weitere Hinweise

Ist der Wert *aa* im X-Register gleich Null, so wird die Standarddruckbreite von 24 Zeichen angenommen. Bei negativen Werten wird der Absolutwert verwendet.

Verwandte Funktion

PRL

PRL

Die Funktion **PRL** (Print line) druckt eine Linie von 24 „-"-Zeichen. Waren vorher noch Zeichen im Druckbuffer vorhanden, so werden diese zuerst ausgedruckt, gefolgt von einem Zeilenvorschub. Ist Flag 12 gesetzt, so werden nur 12 „-"-Zeichen gedruckt.(Bei Druckern, die das „Doppeltbreit"-Flag 12 nicht erkennen, wird ein voranstehendes Leerzeichen ausgedruckt.)

Eingabeparameter

keine

Beispiel

Folgendes Programm verdeutlicht die Wirkungsweise von **PRL**:

Ausdruck:

```
                    TEST1
------------------------
TEST2
------------
```

Programmlisting:

```
01◆LBL "PR4
"
02 CF 12
03 "TEST1"
04 ACA
05 PRL
06 SF 12
07 "TEST2"
08 ACA
09 PRL
10 END
```

Weitere Hinweise

Um Unterstreichungen beliebiger Länge zu erreichen, muß man die Funktion **ACLX** anwenden. **PRL** ist nur für die 24-Zeichen Drucker der Fa. Hewlett-Packard gedacht.

Verwandte Funktion

ACLX

VIEWH

Die Funktion VIEWH (View hex) zeigt den Integerteil der Dezimalzahl im X-Register als Hexadezimalzahl an. Hierbei muß allerdings die Zahl im erlaubten Wertebereich (abhängig von der aktuellen Wortbreite und dem gewählten Modus, siehe *Binärfunktionen)* liegen, andernfalls erfolgt die Fehlermeldung „DATA ERROR X".

Eingabeparameter

X-Register : Dezimalzahl im erlaubten Wertebereich (abhängig von der aktuellen Wortbreite und dem gewählten Modus).

Beispiel

Tastenfolge	Kommentar
142 VIEWH	Die Zahl aus dem X-Register (142) wird in ihr hexadezimales Äquivalent umgewandelt und angezeigt ('8E).

Weitere Hinweise

Bei ALPHA-Daten im X-Register erfolgt die Fehlermeldung „ALPHA DATA"; zu große Werte führen zu der Fehlermeldung „DATA ERROR X".

Verwandte Funktion

ARCLH

F/E

Die Funktion F/E setzt den sogenannten Fix/Eng-Modus. Es werden Flag 40 und 41 gesetzt. Der Rechner zeigt nun alle Zahlen wie im FIX-Format an. Ist die Zahl allerdings so groß, daß sie mit Exponenten angezeigt werden muß (> 9.999.999.999), so wird sie im ENG-Format angezeigt. Die Anzahl der Nachkommastellen wird von dem vorher eingestellten Modus übernommen und verändert sich nicht. Dieser F/E-Modus wird von den Funktionen ACAXY und PRAXY als Indikator für eine spezielle Druckart angesehen (siehe ACAXY und PRAXY).

Eingabeparameter

keine

Beispiel

Zahlendarstellungen in den verschiedenen Modi (im Beispiel mit zwei Nachkommastellen):

FIX	ENG	F/E
0,00	0,00 00	0,00
1500,00	1,50 03	1500,00
89.456,25	89,5 03	89.456,25
9.999.999.999	10,0 09	9.999.999.999
1,00 11	100, 09	100, 09
1,59 52	15,9 51	15,9 51
-7,95 53	-795, 51	-795, 51

Weitere Hinweise

Der F/E-Modus bleibt so lange aktiv, wie kein anderer Modus (FIX, ENG oder SCI) gewählt wird. Soll die Anzahl der Nachkommastellen im F/E-Modus geändert werden, so ist sie zuerst mit der Funktion FIX zu ändern. Anschließend muß der Fix/Eng-Modus wieder mit F/E eingestellt werden.

Verwandte Funktionen

FIX, SCI, ENG, ACAXY, PRAXY

ALPHA-Funktionen

Funktionen zur Manipulation des ALPHA-Registers

ABSP

Die Funktion ABSP (ALPHA backspace) löscht von rechts ein Zeichen aus dem ALPHA-Register.

Eingabeparameter

keine

Beispiel

Wenn im ALPHA-Register der Text „ABCDEFG" steht, so wird mit der Funktion ABSP das rechtsbündige Zeichen (hier „G") gelöscht, und im ALPHA-Register steht jetzt nur noch „ABCDEF".

Weitere Hinweise

Die Funktion ABSP arbeitet auch dann einwandfrei, wenn das letzte Byte rechts im ALPHA-Register den Wert 0 hat.

Verwandte Funktion

CLA-

CLA-

Die Funktion CLA- löscht das ALPHA-Register von rechts, bis es einen Buchstaben findet, dem rechts ein Leerzeichen folgt. Das diesem Buchstaben zugehörige Leerzeichen wird nicht mehr gelöscht.

Eingabeparameter

keine

Beispiel

Steht in ALPHA z.B. „TASTE 123", so steht nach Ausführung der Funktion CLA- im ALPHA-Register „TASTE" . Die Buchstabenkette „123" wurde gelöscht!

Weitere Hinweise

Ist das ALPHA-Register leer oder befinden sich nur Leerzeichen oder nur Buchstaben in ALPHA, so wird das gesamte ALPHA-Register gelöscht.

Verwandte Funktionen

CLA, ABSP

XTOAH

Die Funktion XTOAH (X to ALPHA hex) hängt entsprechend der eingestellten Wortbreite (siehe *Binärfunktionen*) ein oder mehrere Zeichen an das ALPHA-Register an. Dabei entsprechen diese Zeichen dem hexadezimalen Äquivalent des X-Registers.

Eingabeparameter

X-Register : Wert der anzuhängenden Zeichen

Beispiel

Tastenfolge	Kommentar
CLA 10 WSIZE	Das ALPHA-Register wird gelöscht und eine Wortbreite von 10 Bit eingestellt.
340 VIEWH	Der Wert 340 wird hexadezimal als '154 angezeigt.
XTOAH	Die Zeichen '01 und '54 („Männchen" und „T") werden an das ALPHA-Register angehängt.

Weitere Hinweise

Mit einer eingestellten Wortbreite von 8 Bit arbeitet die Funktion XTOAH genau wie die Funktion XTOA aus dem Extended Functions Modul.

Verwandte Funktion

XTOA (aus dem Extended Functions Modul)

ARCLE

Die Funktion ARCLE (ALPHA recall engeneer) arbeitet ähnlich der im HP-41 schon vorhandenen Funktion ARCL X (im ENG-Format), ersetzt jedoch den Exponenten durch einen Buchstaben, entsprechend den international festgelegten SI-Vorsätzen (siehe Tafel).

Eingabeparameter

X-Register : Zahl, die ans ALPHA-Register angehängt werden soll.

Beispiel

In einen Text soll eine Zahl mit einer SI-Einheit eingefügt und anschließend auf einem 80-Zeichen Drucker ausgedruckt werden:

Programm:

```
01◆LBL "PR2
       "
02 "Die "
03 ACA
04 "Strasse
  nlaenge "
05 ACA
06 "betraeg
       t "
07 RCL 00
08 ARCLE
09 "km."
10 ACA
11 PRBUF
12 END

R00=   15.000,00
```

Ausdruck:

```
Die  Strassenlaenge  betraegt 15,00 km.
```

Tafel der international festgelegten SI-Vorsätze

Faktor	Vorsatz	
	Name	Zeichen
10^{18}	Exa	E
10^{15}	Peta	P
10^{12}	Tera	T
10^{9}	Giga	G
10^{6}	Mega	M
10^{3}	Kilo	k
10^{-3}	Milli	m
10^{-6}	Micro	u (statt μ)
10^{-9}	Nano	n
10^{-12}	Pico	p
10^{-15}	Femto	f
10^{-18}	Atto	a

Weitere Hinweise

Ist der Exponent der Zahl im X-Register kleiner als 10^{-18} oder größer als 10^{18}, so wird diese Zahl, wie bei **ARCL** **X** mit ihrem Exponenten an das ALPHA-Register angehängt.

Verwandte Funktionen

ARCLH, **ARCLI**

ARCLH

ARCLH (ALPHA recall hex) hängt die hexadezimale Darstellung der Zahl im X-Register an den Inhalt des ALPHA-Registers an. Diese Darstellung ist abhängig von der gewählten Wortbreite und dem eingestellten Komplement-Modus (siehe *Binärfunktionen*). Es werden nur Zeichen zwischen 0 und 9 und A bis Z ans ALPHA-Register angehängt. Bei unveränderter Wortbreite ist die Anzahl angefügter Zeichen stets gleich. Je nach Wortbreite kann sie zwischen 1 und 8 Zeichen betragen.

Eingabeparameter

X-Register : dezimale Darstellung einer Hex-Zahl

Beispiel

Eine mit der Funktion PMTH eingegebene Hexadezimalzahl soll an einen bestehenden Text in ALPHA angehängt werden:

Tastenfolge	Kommentar
CLX WSIZE	Einstellen der Wortbreite auf 8 Bit
'WERT	Der Text „WERT " wird ins ALPHA-Register geschrieben.
PMTH 8E	Die Hexadezimalzahl 8E wird dezimal als Wert 142 ins X-Register geschrieben.
ARCLH	Der Wert 142 wird als Hexadezimalzahl 8E an das ALPHA-Register angehängt. Dort steht nun „WERT 8E".

Weitere Hinweise

Um Fehler bei der Anwendung dieser Funktion zu vermeiden, sollte man unbedingt vorher den Abschnitt *Binärfunktionen* lesen.

Verwandte Funktionen

VIEWH, PMTH, XTOAH, ARCLE, ARCLI, ARCL X

ARCLI

Mit der Funktion ARCLI (ALPHA recall integer) ist es jetzt möglich, nur den Integerteil einer Zahl an das ALPHA-Register anzuhängen, ohne das FIX-Format des Rechners zu verändern.

Eingabeparameter

keine

Beispiel

In ALPHA steht z.B. „SPALTE", im X-Register der Wert 15,002. Nach Ausführen der Funktion ARCLI steht im ALPHA-Register: „SPALTE 15".

Weitere Hinweise

keine

Verwandte Funktionen

ARCLX, ARCLE, ARCLH

Kapitel 7

Funktionen für fortgeschrittene Programmierer

Inhaltsverzeichnis Kapitel 7
Funktionen für fortgeschrittene Programmierer

Funktionen für fortgeschrittene Programmierer

Die 17 Funktionen dieses Blocks bilden die Grundlage für eine fortschrittliche Programmierung, die bisher entweder gar nicht oder nur mit erheblichem Aufwand möglich war. In dem folgenden Kapitel, das diese Funktionen beschreibt, wird durch einige Programmbeispiele verdeutlicht, welche Anwendungen möglich sind. Zum Verständnis dieses Kapitels ist es wichtig, über den Aufbau und die Adressierung der Register des HP-41 Bescheid zu wissen. Diese Grundlagen sind im Kapitel *Aufbau des HP-41* ausführlich erläutert.

PLNG

Die nicht programmierbare Funktion **PLNG** fordert zur Eingabe eines Programmnamens (Global-Label) auf, wie z. B. auch die Funktion **CLP**. Es ist also nach der Eingabe dieser Funktion **ALPHA** *Name des Labels* **ALPHA** zu drücken. Dann wird die Länge des spezifizierten Programmes in Bytes angezeigt. Dabei werden keine Statusregister verändert, der Programmzeiger bleibt also unverändert, ebenfalls das X-Register und alle Stack-Register. Wie bei der Funktion **CLP** ist auch für **PLNG** die Eingabefolge **PLNG** **ALPHA** **ALPHA** für das Programm, auf das der Programmzeiger gerade gerichtet ist, möglich.

Eingabeparameter

Es wird der Programmname erfragt.

Beispiel

Wenn sich das Programm „CDE“ im Rechner befindet, ergibt die Tastenfolge **PLNG** **ALPHA** *CDE* **ALPHA** in der Anzeige die Meldung„ 60 BYTES“. Ist ein Drucker am Rechner angeschlossen, der sich im TRACE- oder NORM-Modus befindet, so wird diese Meldung zusätzlich ausgedruckt.

Weitere Hinweise

PLNG ist normalerweise nicht programmierbar. Wird die Bytefolge dieser Funktion (162, 220) trotzdem in eine Programmzeile eingegeben, ergibt die Ausführung in der Regel die Fehlermeldung „NONEXISTENT“. Wenn allerdings im Register Q ein rückwärts geschriebener Programmname steht, wird die Länge dieses Programms angezeigt und die Programmausführung unterbrochen. Wird das Q-Register gelöscht, so wird die Länge des Programms, in dem **PLNG** gerade ausgeführt wird, angezeigt und der Programmlauf nicht angehalten.

Verwandte Funktion

PPLNG

PPLNG

Dies ist die programmierbare Funktion (programmable program length) zur Angabe der Programmlänge in Bytes. Zur Ausführung dieser Funktion muß der Programmname (Global-Label) vorher in das ALPHA-Register eingegeben worden sein. Enthält ein Programm mehrere Global-Labels, sokann jedes angegeben werden. Das Ergebnis der Funktion **PPLNG** wird in das X-Register geschrieben, die Stackregister werden dabei wie bei einem **RCL**-Befehl nach oben verschoben.

Eingabeparameter

ALPHA-Register : Programmname

Beispiel

Die Programmzeilen
'CDE
PPLNG
schreiben den Wert 60 ins X-Register, wenn sich das Programm „CDE" im Hauptspeicher des Rechners befindet.

Weitere Hinweise

Bei leerem ALPHA-Register gibt **PPLNG** die Länge des Programms an, auf das der Programmzeiger gerade positioniert ist. Bei der Ausführung von **PPLNG** von der Tastatur aus wird der Programmzeiger nicht verändert; bei der Ausführung in einem Programm wird anschließend der nach **PPLNG** folgende Programmbefehl ausgeführt. Es erfolgt also kein Sprung in das Programm, dessen Länge berechnet wurde.

Verwandte Funktion

PLNG

PHD

Die Funktion PHD (program head) – programmierbar oder von der Tastatur aus zu benutzen – gibt die absolute Adresse des ersten Bytes eines Programms an. Dazu ist wie bei PPLNG ein Global-Label vorher im ALPHA-Register anzugeben. Bei leerem ALPHA-Register wird die absolute Adresse des Programms angegeben, auf das der Programmzeiger gerade positioniert ist. Die absolute Adresse des ersten Bytes (program head) wird in das X-Register geschrieben. Dabei werden die Stack-Register wie bei einem RCL-Befehl nach oben verschoben. Das Format der Adresse ist so, daß es sofort für die Befehle PEEKB oder POKEB oder X›PC oder X›RTN verwendet werden kann.

Eingabeparameter

ALPHA-Register : Programmname

Beispiel

Bei eingestelltem SIZE 000 ergibt bei einem HP-41 CV oder CX die Tastenfolge ALPHA *Name des ersten Programms aus CAT 1* ALPHA PHD im X-Register den Wert 511,6.

Weitere Hinweise

Zu beachten ist, daß sich die absolute Adresse eines Programmbefehls durch viele Faktoren verändern kann. Das Ergebnis der Funktion PHD kann also, je nach Lage des Programms, durchaus sehr verschieden sein. Die Adresse ändert sich immer, wenn die Anzahl der Datenregister neu definiert wird (SIZE oder PSIZE).Das Packen des Programmspeichers kann ebenfalls die absoluten Adressen verändern. Das Einfügen oder Löschen von Programmbefehlen verändert die absolute Adresse nur weiter unten im Programmspeicher, also an Programmteilen, die später im **CAT 1** erscheinen.

Verwandte Funktion

PC›X

Rechenfunktionen für absolute Adressen

Die Rechenfunktionen für absolute Adressen sind die Funktionen A+, A+B, A-, A-A. Diese arbeiten für den gesamten Bereich, den das Betriebssystem des HP-41 verwalten kann, also von 0,0 bis 1023,6. Dabei ist der Vorkommateil die absolute Registeradresse *aaa* und der Nachkommateil die absolute Byteadresse *c* innerhalb des betreffenden Registers. Bei Werten im X- oder Y-Register, die nicht dem angegebenen Bereich entsprechen, erscheint die Fehlermeldung „DATA ERROR X" oder „DATA ERROR Y". Die nach der ersten Nachkommastelle folgenden weiteren Nachkommstellen werden von den Rechenfunktionen für absolute Adressen nicht verändert; sie rufen keine Fehlermeldungen hervor.

A+

Diese Funktion erhöht die im X-Register angegebene absolute Adresse *aaa,c* um ein Byte. Das Ergebnis wird in das X-Register, der ursprüngliche Wert des X-Registers, vor Ausführung von A+, in das LAST X-Register geschrieben.

Eingabeparameter

X-Register : *aaa,c*

Beispiele

192,0 A+ ergibt 192,1
192,6 A+ ergibt 193,0

Weitere Hinweise

Die Funktion **A+** arbeitet unabhängig davon, ob es die absolute Adresse tatsächlich gibt, für den gesamten Rechenbereich von 0,0 bis 1023,6.

Verwandte Funktionen

A+B, **A-**, **A-A**

A+B

Diese Funktion addiert zu der im Y-Register enthaltenen absoluten Adresse *aaa,c* die im X-Register angegebene Anzahl *n* Bytes. Wenn das X-Register eine negative Zahl enthält, wird entsprechend abgezogen. Die Stackregister werden wie bei einer „normalen" Rechenfunktion, z.B. **+**, nach unten verschoben:
Das Ergebnis der Rechnung „Adresse plus Bytes" steht im X-Register, die Anzahl der addierten Bytes (vorheriger Wert des X-Registers) im LAST X-Register.

Eingabeparameter

X-Register : *nnnn*
Y-Register : *aaa,c*

Beispiele

412 **ENTER** 70 **CHS** **A+B** ergibt 402
192 **ENTER** 2239 **A+B** ergibt 511,6
(=gesamter Hauptspeicherbereich des HP-41 CV oder HP-41 CX)

Weitere Hinweise

Die Funktion **A+B** arbeitet unabhängig davon, ob es die absolute Adresse tatsächlich gibt, für den gesamten Rechenbereich von 0,0 bis 1023,6. Die größte zulässige Eingabe für das X-Register (Anzahl der Bytes, die addiert werden sollen) ist 7167 (bzw. -7167).

Verwandte Funktionen

A+, **A-**, **A-A**

A-

Diese Funktion vermindert die im X-Register angegebene absolute Adresse *aaa,c* um ein Byte. Das Ergebnis wird in das X-Register, der ursprüngliche Wert des X-Registers, vor Ausführung von A-, in das LAST X-Register geschrieben.

Eingabeparameter

X-Register : *aaa,c*

Beispiele

192,1 A- ergibt 192,0
193,0 A- ergibt 192,6

Weitere Hinweise

Die Funktion A- arbeitet unabhängig davon, ob es die absolute Adresse tatsächlich gibt, für den gesamten Rechenbereich von 0,0 bis 1023,6.

Verwandte Funktionen

A+, A+B, A-A

A-A

Diese Funktion errechnet die Differenz zwischen einer absoluten Adresse *aaa,c* im Y-Register und einer absoluten Adresse *ddd,e* imX-Register. Das Ergebnis in Bytes wird in das X-Register geschrieben. Die Stackregister werden wie bei einer „normalen" Subtraktion nach unten verschoben; die absolute Adresse, die vor Ausführung von **A-A** im X-Register enthalten war, wird in das LAST X-Register geschrieben. Sofern der Wert im Y-Register vor Ausführung von **A-A** größer als der Wert im X-Register war, wird das Ergebnis negativ.

Eingabeparameter

X-Register : *ddd,e*
Y-Register : *aaa,c*

Beispiele

511,6 **ENTER** 192 **A-A** ergibt 2239
402 **ENTER** 412 **A-A** ergibt -70

Weitere Hinweise

Die Funktion **A-A** arbeitet unabhängig davon, ob es die absoluten Adressen tatsächlich gibt, für den gesamten Rechenbereich von 0,0 bis 1023,6.

Verwandte Funktionen

A+, **A+B**, **A-**

DCD

Die Funktion **DCD** (decode) decodiert den Wert im X-Register und hängt die hexadezimal decodierte Darstellung an das ALPHA-Register an. Zur Analyse von Statusregistern ist diese Funktion besonders gut geeignet.

Eingabeparameter

X-Register : Wert, der decodiert werden soll

Beispiel

CLA **RCL b** **DCD** **ALPHA**
Ergebnis z.B. 000000000060F7, dies bedeutet: Das Register b enthielt keine Rücksprungadresse, der Programmzeiger stand auf 247,5 (gemäß Abfrage mit **PC›X**).

Weitere Hinweise

In der Ergebnisdarstellung von **DCD** entsprechen je zwei hexadezimale Ziffern einem Byte, daher ist die Darstellung immer 14-stellig und jedes einzelne Byte ist eindeutig identifizierbar.

Die Umkehrfunktion von **DCD** wäre „CDE" (code). Folgendes Programm codiert nun eine hexadezimale Darstellung aus dem ALPHA-Register und schreibt das Ergebnis in das X- und in das M-Register (als Text sichtbar):

```
01◆LBL "CDE
       "
02 CLX
03 WSIZE
04 6.5
05 SF 22

06◆LBL 00
07 5
08 PEEKB
09 ABSP
10 X<>Y
11 RDN
12 57
13 -
14 X>0?
15 2
16 X<=0?
17 9
18 +
19 X<0?
20 CLX
21 FS?C 22
22 GTO 00
23 16
24 *
25 +
26 POKEB
27 SF 22
28 CLX
29 5.5
30 X<>Y
31 A-
32 X≠Y?
33 GTO 00
34 RCL [
35 END

 PLNG "CDE"
   60 BYTES
```

Funktionen zur Manipulation des Programmzeigers

Die Gruppe der Funktionen, die Manipulationen des Programmzeigers und der Rücksprungadressen ermöglichen, soll als nächstes behandelt werden. Die Funktionen PC›RTN, PC›X, X›PC, X›RTN, XR›RTN erlauben – programmgesteuert – beliebige Sprünge und Rücksprünge im gesamten Hauptspeicherbereich, in den Statusregistern und vor allem auch im erweiterten Speicherbereich des Extended Functions Moduls und der X-Memory Module. XR›RTN erlaubt außerdem, ein Programm in einem Softwaremodul an jeder gewünschten Programmzeile aufzurufen.

PC>X

Die Funktion PC›X (program counter to X) liest aus dem Statusregister b die absolute Adresse des Programmzeigers und schreibt das Ergebnis in das X-Register. Da der Wert ein „ganz normaler" Zahlenwert ist, kann er in einem Datenregister gespeichert werden. Mit diesem Befehl ist es problemlos möglich, beliebige Byteabstände zu berechnen. So kann zum Beispiel erkannt werden, ob ein Zwei-Byte-GTO für die Compilierung des Sprungbefehls ausreicht.

Eingabeparameter

keine

Beispiel

Es soll eine Sprungweite berechnet werden, um festzustellen, ob ein Zwei-Byte-GTO für diese Entfernung ausreicht. Diese Berechnung kann man z.B. mit folgenden Schritten ausführen:
Im Programm auf die Programmzeile gehen, die nach GTO NN steht; PRGM aus, PC›X. Dann zum LBL NN gehen; PC›X , A-A.

Ist der angezeigte Wert kleiner als 112, reicht ein Zwei-Byte-GTO aus, damit der Sprungbefehl compiliert werden kann. Falls das Ergebnis negativ war (das LBL NN steht im Programm vor dem GTO NN), muß man vor dem ersten PC›X auf die Programmzeile gehen, die vor dem GTO NN steht, um die Sprungdistanz genau zu berechen.

Weitere Hinweise

keine

Verwandte Funktionen

PHD, X›PC

X>PC

Diese Funktion (X to program counter) setzt den Programmzeiger auf die im X-Register angegebene absolute Adresse *aaa,c*. Dabei kann jede tatsächlich existente absolute Adresse angegeben werden, also nicht nur im „normalen" Programmspeicherbereich. Insbesondere ist X>PC dafür gedacht, eine bestimmte Stelle in einem Programm wieder zu erreichen, wobei die absolute Adresse des Programmzeigers zuvor durch PC>X ermittelt wurde. Wenn diese Methode im Programmspeicherbereich angewendet wird, darf in der Zwischenzeit nicht die Anzahl der Datenregister verändert werden, da sich sonst die absoluten Adressen im Programmspeicherbereich verändern.

Eingabeparameter

X-Register : *aaa,c*

Beispiel

```
01 PC>X
02 TONE 9
03 X>PC
```

Dies ist eine unendliche Programmschleife und entspricht im Prinzip der Befehlsfolge RCL b, TONE N, STO b, nur läßt sich die von PC>X ermittelte absolute Adresse im Gegensatz zu dem Wert aus RCL b in einem Datenregister zwischenspeichern, und der Wert für X>PC kann problemlos beliebig vorgegeben werden, was mit dem Wert für STO b nicht so einfach möglich ist.

Weitere Hinweise

Insbesondere kann mit X>PC jeder Bereich im erweiterten Speicher des Extended Functions oder X-Memory Moduls direkt erreicht werden. Die absolute Adresse des ersten Bytes im Extended Functions Modul ist 191,6. Dort beginnt der erste Header des ersten Files. Jedes File belegt zunächst zwei Header-Register und beginnt dann mit seinem eigentlichen Inhalt. Die absoluten Adressen sind wie im Programmspeicher auch absteigend.

Beispiel:

CAT'4 zeigt an:
KA K004 (Keyfile 4 Register)
CLK K000 (Keyfile 0 Register)
ALM B006 (Butterfile 6 Register)
K? P007 (Programmfile 7 Register)
99,0000 (freie Register im Extended Functions Modul)

(Die Erklärungen zu den Filearten **K** und **B** werden später im Funktionsblock *XF/Memory Funktionen* gegeben; dort findet sich auch das Programm zur Erzeugung des Files „CLK" mit 0 Registern.)

Nun soll das Programm „K?" im Programmfile „K?" mit der Funktion **X›PC** direkt ausgeführt werden. Dazu muß die absolute Adresse des Programmbeginns in dem Programmfile errechnet werden:

Erstes Byte im Extended Functions Modul überhaupt	191,6
abzüglich zwei Headerregister von „KA" ergibt	189,6
abzüglich vier Register File „KA" ergibt	185,6
abzüglich zwei Headerregister von „CLK" ergibt	183,6
abzüglich null Register von „CLK" ergibt	183,6
abzüglich zwei Headerregister von „ALM" ergibt	181,6
abzüglich sechs Register von „ALM" ergibt	175,6
abzüglich zwei Headerregister von „K?" ergibt	**173,6**

Dies ist die absolute Adresse des ersten Bytes des Programms „K?" als Programmfile im Extended Functions Modul. Mit 173,6 im X-Register und **X›PC** gelangt man in diesem Beispielfall direkt in das Programm „K?" und kann es sofort ausführen. Soll das Programm nicht von der ersten Programmzeile an ausgeführt werden, so kann die absolute Adresse jeder Programmzeile ermittelt werden (z.B. Programm mit **SST** durchgehen und dann mit **PC›X** den Programmzeiger lesen).

Verwandte Funktionen

PC›X, **PHD**

X>RTN

Die Funktion X›RTN (X to returnstack) setzt die erste Rücksprungadresse auf die im X-Register angegebene absolute Adresse *aaa,c*.

Eingabeparameter

X-Register : *aaa,c*

Weitere Hinweise

Auch bei dieser Funktion können, wie bei X›PC, alle tatsächlich existenten absoluten Adressen (nicht nur die aus dem Programmspeicherbereich) angegeben werden. Beim nächsten RTN oder END wird der Programmzeiger dann auf die angegebene Adresse gesetzt. Jede der verwendeten Rücksprungadressen kann in einem Datenregister zwischengespeichert werden; somit ist ein praktisch unbegrenzter Rücksprungstapel zu konstruieren. Da erst beim nächsten RTN oder END verzweigt wird, kann man zuvor noch alle Stackregister wie erforderlich belegen.

Verwandte Funktionen

PC›RTN, XR›RTN

PC<>RTN

Diese Funktion (program counter exchange return address) vertauscht den gegenwärtigen Wert des Programmzeigers mit dem gegenwärtigen Wert der ersten Rücksprungadresse. Die Funktion PC◇RTN darf daher nur ausgeführt werden, wenn mindestens eine Rücksprungadresse besteht (sonst erhält man „DATA ERROR").

Eingabeparameter

keine

Beispiel

Die Befehlsfolge

Programmteil A
rrr,b
X◇RTN
PC◇RTN
Programmteil B

hat folgende Funktion:
Zuerst wird der *Programmteil A* ausgeführt, dann wird die absolute Adresse *rrr,b* als erste Rücksprungadresse gespeichert, und PC◇RTN vertauscht den Programmzeiger mit dieser ersten Rücksprungadresse. Also wird die Programmausführung bei der absoluten Adresse *rrr,b* fortgesetzt (z.B. auch im Extended Functions Modul!), bis dort ein RTN oder END erreicht wird. Dann wird der *Programmteil B* ausgeführt, da ja PC◇RTN den Programmzeiger am Anfang des *Programmteils B* als erste Rücksprungadresse gespeichert hatte.

Weitere Hinweise

Mit der Funktion **PC‹RTN** läßt sich ein Unterprogrammaufruf zu jeder beliebigen absoluten Adresse programmieren. Auch ein Unterprogrammaufruf und der Rücksprung vom Extended Functions Modul aus sind jetzt möglich geworden.

Verwandte Funktionen

X›RTN, **XR›RTN**

XR>RTN

Diese Funktion erlaubt den Unterprogrammaufruf oder Sprung in das Programm mit der XROM-Nummer *kk,ll* eines Softwaremoduls an jeder beliebigen Programmzeile *nnnn* (also nicht nur dort, wo ein Label ist!).Diese Funktion ermöglicht es daher, in manchen Modulen die Eingabeaufforderungen (PROMPT-Befehle) zu umgehen oder Programme nur zum Teil als Unterprogramme zu verwenden.

Eingabeparameter

X-Register	: Programmzeilennummer *nnnn*
Y-Register	: XROM-Nummer *kk,ll*

Beispiel

Das Programm „T1" im PPC ROM beginnt mit der Programmzeile 140, enthält dann 13 synthetische Töne, dann ein RTN. Es hat die XROM-Nummer 10,47. Normalerweise kann man nur mit XROM „T1" alle 13 Töne zusammen ausführen. Der Befehl XR›RTN erlaubt nun, das Programm „T1" auch in beliebigen Teilen auszuführen.
Die Befehlsfolge 10,47 ENTER 150 XR›RTN führt z.B. nur vier Töne (Zeile 150-153 einschl.) aus.

Weitere Hinweise

Soll ein Rücksprung ins Hauptprogramm erfolgen, so ist nach XR›RTN noch PC›RTN zu programmieren.

Verwandte Funktionen

PC›RTN, X›RTN

PEEK- und POKE-Funktionen

Die Funktionen zur Programmzeigermanipulation bieten eine Vielzahl ganz neuer Programmiertechniken, die im vorigen Abschnitt nur angedeutet werden konnten. So kann man z. B. auch unterhalb des **.END.** Programme ablaufen lassen, die nicht im **CAT'1** gelistet werden und auch sonst nicht sofort zu entdecken sind. Um dort programmieren zu können, benötigt man aber die Funktionen **PEEKB**, **PEEKR**, **POKEB** und **POKER**, die in diesem Abschnitt behandelt werden. Diese Befehle ermöglichen jede denkbare Bytemanipulation im gesamten Rechnerbereich von der absoluten Adresse 0,0 bis zur absoluten Adresse 1023,6. Damit werden weitere ungeahnte Möglichkeiten eröffnet, wie hier an einigen Programmen gezeigt werden soll.

PEEKB

Dieser Befehl könnte mit „Lese Byte" übersetzt werden. Er liest den dezimalen Wert des Bytes, dessen absolute Adresse *aaa,c* im X-Register angegeben wird. Das Ergebnis wird in das X-Register geschrieben und der Stack wird angehoben. Das LAST X-Register wird nicht verändert. Die absolute Adresse eines Bytes ist im Format *aaa,c* einzugeben, wobei *aaa* die absolute Registeradresse und *c* die Byteposition innerhalb des Registers *aaa* ist.

Eingabeparameter

X-Register : *aaa,c*

Beispiele

Sofern eine echte Tastenzuordnung (aus **CAT 2** oder **CAT 3**) besteht, hat das Byte 192,6 den Wert 240. So läßt sich durch **PEEKB** feststellen, wieviele Tastenzuweisungsregister belegt sind, denn das Byte 6 jedes Tastenzuweisungsregisters ist 240. Man muß also zunächst 192,6 testen, dann 193,6 usw., bis sich ein von 240 verschiedener Wert durch **PEEKB** ergibt.

Programmbeispiel dazu: „A?"

Das Programm „A?" ermittelt, wieviele Register insgesamt mit Tastenzuweisungen und I/O-Buffer (Alarmregister, Buffer des CCD-Moduls oder andere Buffer) belegt sind. Es liefert nur dann das richtige Ergebnis, wenn mindestens ein freies Register vorhanden ist. Falls kein freies Register vorhanden ist, kann nicht so einfach programmiert werden. Wenn nur die Anzahl der belegten Tastenzuordnungsregister berechnet werden soll, sind die Programmzeilen 15 – 21 (einschl.) zu löschen.

```
01◆LBL "A?"
02 191.6
03 R↑
04 240

05◆LBL 00
06 R↑
07 R↑
08 ISG X
09 PEEKB
10 R↑
11 X=Y?
12 GTO 00
13 R↑
14 R↑
15 -.1
```

```
16◆LBL 01
17 +
18 PEEKB
19 X≠0?
20 GTO 01
21 CLX
22 192
23 -
24 INT
25 END

  PLNG "A?"
   47 BYTES
```

Weiteres Programmbeispiel zu PEEKB:
„VB" (VIEW BYTES)

Mit dem Programm „VB" lassen sich z.B. Textzeilen in Programmen eindeutig identifizieren oder Statusregister in dezimaler oder hexadezimaler Form decodieren. Wenn eine Ausgabe im Hexadezimalformat gewünscht wird, ist Zeile 11 durch ARCLH zu ersetzen, vor dem Programmstart sollte man dann CLX, WSIZE ausführen, um die Wortbreite auf 8 Bit einzustellen.

Das Programm „VB" zeigt eine beliebige Anzahl von Bytes in folgender Form im ALPHA-Register an:
absolute Byteadresse _ _ _ *Dezimalwert des Bytes*
Das Programm kann dazu benutzt werden, jeden Bereich des Rechners anzuzeigen (außer das ALPHA-Register, da das Programm das ALPHA-Register wie auch die Stackregister verändert).

Es ist einzugeben:
Anzahl der gewünschten Bytes, ENTER, absolute Adresse des ersten Bytes.
Soll ein Programm komplett decodiert werden, ist folgende Funktionsfolge möglich:
Programmname ins ALPHA-Register eingeben, dann PPLNG, PHD, XEQ"VB".

```
01◆LBL "VB"
02◆LBL 00
03 " "
04 RCL d
05 FIX 1
06 ARCL Y
07 STO d
08 RDN
09 "⊢___"
10 PEEKB
11 ARCLI
12 AVIEW
13 RDN
14 A-
15 DSE Y
16 GTO 00
17 CLST
18 CLD
19 END

   PLNG "VB"
    40 BYTES
```

Der Ausdruck des Programms „VB" selbst ist z.B. wie folgt (wobei die angegebenen absoluten Adressen jeweils anders angegeben werden, je nach Rechnerkonfiguration und Programmen im **CAT 1**):

```
280,1___198
280,0___0
279,6___243
279,5___0
279,4___86
```

Noch ein recht nützliches Programm, das PEEKB benutzt: „GE"

Dieses Programm stellt den Programmzeiger auf die erste Zeile des letzten Programms im Hauptspeicher, also auf das Programm, das das **.END.** als letzte Programmzeile enthält. Es ist damit in der Funktion identisch dem gleichnamigen Programm aus dem PPC ROM, nur sehr viel kürzer und ohne Synthetik. Man erreicht also schnell das letzte Programm (auch wenn es kein Global-Label hat!), ohne den gesamten **CAT 1** durchlaufen zu müssen.

In den Zeilen 02 bis 13 wird aus den Bytes 13 und 13,1 zunächst die absolute Adresse des **.END.** decodiert (Register c). Dann ergibt zweimal A+ die absolute Adresse des Beginns des **.END.**; diese Adresse wird zur ersten Rücksprungadresse und nach CLST führt das END den Sprung zum **.END.** aus, das die Programmausführung anhält. Wenn man nun in den PRGM-Modus schaltet, sieht man Zeile 01 des letzten Programms. (Falls vorher GTO. ausgeführt wurde, enthält dies letzte Programm nur das **.END.**)

```
01*LBL "GE"
02 13
03 PEEKB
04 X<>Y
05 A+
06 PEEKB
07 16
08 MOD
09 LASTX
10 X↑2
11 *
12 R↑
13 +
14 A+
15 A+
16 X>RTN
17 CLST
18 END

  PLNG "GE"
   33 BYTES
```

Weitere Hinweise

keine

Verwandte Funktionen

PEEKR, POKEB, POKER

PEEKR

Dieser Befehl ist mit einem **RCL**-Befehl vergleichbar. Er ermöglicht es jedoch, den Inhalt eines jeden Registers ohne Normalisation in das X-Register zu lesen. Damit ist eine der wesentlichen Schwierigkeiten bei der synthetischen Programmierung nun beseitigt. Die Adresse des zu lesenden Registers ist als absolute Adresse *aaa* in das X-Register einzugeben. Die Stackregister werden wie bei einem **RCL IND X**-Befehl verändert.

Eingabeparameter

X-Register : *aaa*

Beispiel

Programmbeispiel zu PEEKR: „VR" (VIEW REGISTER)

Das Programm „VR" zeigt eine beliebige Anzahl von Registerinhalten in hexadezimal decodierter Form an oder druckt die Ergebnisse aus. Dabei wird jeweils die absolute Adresse mitangegeben.
Es ist einzugeben:
Absolute Adresse des ersten gewünschten Registers als Vorkommateil, des letzten gewünschten Registers als Nachkommateil. 192,205 z.B. zeigt Register 192 bis 205 (einschl.) an. Das Programm „VR" ist in erster Linie zur Decodierung der Tastenzuweisungsregister oder Bufferregister geschrieben worden, deshalb werden die Register in aufsteigender Reihenfolge behandelt. Zeile 05 hängt drei Leerstellen (Spaces) an das ALPHA-Register an. Das Programm „VR" ist in dieser einfachen Form nur möglich, da **PEEKR** nicht normalisiert und mit **DCD** einfach decodiert werden kann. Die Zeilen 09 bis 11 wurden eingefügt, damit das Programm bei angeschlossenem Drukker schneller läuft (kein Scrolling).

```
01♦LBL "VR"
02♦LBL 00
03 CLA
04 ARCLI
05 "⊢    "
06 PEEKR
07 DCD
08 RDN
09 SF 25
10 PRA
11 FC?C 25
12 AVIEW
13 ISG X
14 GTO 00
15 END

   PLNG "VR"
    34 BYTES
```

Weitere Hinweise

Auch PEEKR arbeitet für jede tatsächlich existente Registeradresse von 0 bis 1023, je nach Rechnerkonfiguration.
Will man aus den (relativen) Datenregistern ohne Normalisierung mit PEEKR lesen, so muß man die absoluten Adressen der Datenregister berechnen.

Verwandte Funktionen

PEEKB, POKEB, POKER

POKEB

Diese Funktion überschreibt das Byte, dessen absolute Adresse *aaa,c* im Y-Register angegeben ist, mit dem im X-Register angegebenen Wert *bbb*. POKEB arbeitet für den gesamten existenten Bereich des Rechners. Die Stackregister bleiben unverändert, sofern sie nicht durch die absolute Adresse im Y-Register angesprochen werden.

Eingabeparameter

X-Register : *bbb*
Y-Register : *aaa,c*

Beispiele

Als erstes Programmbeispiel für POKEB soll ein Programm zur Erzeugung beliebiger synthetischer Töne und zur Messung der Tondauer gezeigt werden, das sich je nach Eingabewert selbst verändert. Es handelt sich also um ein „sich selbst programmierendes" Programm.

Das Programm „TD" (TONDAUER)

```
01◆LBL "TD"
02 " "
03 ARCLI
04 "⊢___"
05 PC>X
06 11
07 CHS
08 A+B
09 X<>Y
10 POKEB
11 TIME
12 TONE X
13 TIME
14 X<>Y
15 HMS-
16 7 E-6
17 -
18  E4
19 *
20 RCL d
21 FIX 2
22 ARCL Y
23 STO d
24 R↑
25 R↑
26 AVIEW
27 END

  PLNG "TD"
   55 BYTES
```

Das Programm „TD“ führt jeden Ton im Bereich 0 bis 127 aus, indem das zweite Byte des Befehls in Zeile 12 entsprechend geändert wird. Startet man das Programm z.B. mit 120 im X-Register, so wird Zeile 12 in TONE P geändert; diese Änderung bewirkt der Befehl POKEB, der in Zeile 10 steht! Der Rest des Programms besteht aus der Zeitmessung mit Hilfe der Funktion TIME des TIME-Moduls. Wenn das Prinzip von POKEB, wie hier dargestellt, ohne die Zeitmessung mit dem TIME-Modul benutzt werden soll, ist das Programm wie folgt zu kürzen:

```
01◆LBL  "T"
02 PC>X
03 -8
04 A+B
05 X<>Y
06 POKEB
07 TONE X
08 END
```

Dabei kann bei der Programmierung in Zeile 07 jeder beliebige TONE-Befehl eingegeben werden.
Vor dem Ausführen von „TD“ oder „T“ muß der Programmspeicher gepackt werden, um jede Möglichkeit einer Fehlfunktion auszuschließen!

Ein Programm, das die Befehle PEEKB und POKEB benutzt, ist „TLC“.Es dient dazu, den Kleinbuchstabenmodus des CCD-Moduls umzuschalten. Bei eingeschaltetem ALPHA-Modus und ausgeschaltetem USER-Modus können mit dem CCD-Modul beliebige ALPHA-Zeichen eingegeben werden. Diese Sonderfunktion kann mit dem Programm „TLC“ inaktiviert oder wieder aktiviert werden. Die entsprechende Codierung geschieht im Register c im Byte 13,4. Eingaben sind für „TLC“ nicht erforderlich.

```
01◆LBL "TLC
       "
02 13,4
03 PEEKB
04 128
05 XOR
06 POKEB
07 CLST
08 END

 PLNG "TLC"
   24 BYTES
```

```
01◆LBL "TLC
       "
02 13,4
03 PEEKB
04 128
05 X<Y?
06 CHS
07 +
08 POKEB
09 CLST
10 END

 PLNG "TLC"
   25 BYTES
```

```
01◆LBL "TLC
       "
02 13,4
03 PEEKB
04 X<>F
05 FC?C 07
06 SF 07
```

```
07 X<>F
08 POKEB
09 CLST
10 END

 PLNG "TLC"
   27 BYTES
```

Wie man an diesem Programmbeispiel sieht, gibt es verschiedene Möglichkeiten, ein Bit umzuschalten.

Ein weiteres, besonders nützliches Programm, das PEEKR, PEEKB und POKEB verwendet, ist „CHK". Es berichtigt die Prüfsumme eines Programmfiles im Extended Functions Modul. Wenn man ein nicht compiliertes Programm (nicht alle Sprungbefehle wurden vor dem Speichern ins Extended Functions Modul ausgeführt) direkt im Extended Functions Modul ausführt, läßt sich das Programm später nicht mehr mit GETP oder GETSUB in den Hauptspeicher zurückladen. Wenn man dies versucht, erhält man die Fehlermeldung „CHKSUM ERR". Man muß dann nur mit dem Namen des Programmfiles im ALPHA-Register das Programm „CHK" ausführen, und die Prüfsumme wird berichtigt. Das Programmfile muß dabei vollständig im Extended Functions Modul liegen ! Das Programm „CHK" kann auch für Programme benutzt werden, die komplett in einem X-Memory Modul

liegen. Dazu ist Zeile 03 in 751 oder 1007 zu ändern, je nachdem in welchen Port das X-Memory Modul eingesteckt wurde. Zeile 04 hängt sechs Leerstellen (Spaces) an das ALPHA-Register an.

```
01◆LBL "CHK
              "
02 RCLPTA
03 191
04 "⊢
              "
05 7
06 AROT
07 RCL [

08◆LBL 00
09 R↑
10 R↑
11 PEEKR
12 DSE Y
13 R↑
14 X≠Y?
15 GTO 00
16 CLX
17 STO \
18 RDN
19◆LBL 01
20 RDN
21 A-
22 PEEKB
23 ST+ T
24 DSE Z
25 GTO 01
26 RDN
27 A-
28 RCL Z
29 256
30 MOD
31 POKEB
32 END

 PLNG "CHK"
   66 BYTES
```

Weitere Hinweise

Da mit **POKEB** von der Tastatur aus oder programmgesteuert jedes beliebige Byte verändert werden kann, sollte man diese Funktion nur anwenden, wenn man die Rechnerstruktur genau kennt. Andernfalls sind ungewollte Veränderungen in Programmen, Datenregistern, Statusregistern usw. oder „MEMORY LOST" die Folge. Besonders bei gezielter Veränderung in Programmen mit **POKEB** muß beachtet werden, daß sich die absoluten Adressen durch vielfältige Faktoren ebenfalls verändern können!

Verwandte Funktionen

PEEKB, **PEEKR**, **POKER**

POKER

Diese Funktion überschreibt das absolute Register, dessen Adresse *aaa* im Y-Register angegeben ist, mit dem Inhalt des X-Registers. POKER arbeitet für den gesamten existenten Registerbereich des Rechners. Die Stackregister bleiben unverändert, sofern sie nicht durch die absolute Adresse im Y-Register angesprochen werden.

Eingabeparameter

X-Register : abzuspeichernder Wert
Y-Register : *aaa*

Beispiele

Programmbeispiel zu POKER: Das Programm „CB" (Clear Buffer)

Das Programm „CB" entspricht in der Funktion dem Programm „CK" aus dem PPC ROM. Es löscht den gesamten I/O-Buffer-Bereich. Gelöscht werden alle Register ab dem Register mit der absoluten Adresse 192 aufsteigend bis zum Register unmittelbar unterhalb des **.END.**. Damit werden alle I/O-Buffer und alle Tastenzuordnungen aus **CAT 2** oder **CAT 3** gelöscht.Im **CAT'6** muß anschließend mit der Taste C noch das Tastenzuweisungsbit zu jeder Taste gelöscht werden (Zuordnungen erscheinen als ABS). Man kann auch mit 15 ENTER CLX POKER und 10 ENTER CLX POKER alle Tastenzuweisungsbits auf einmal löschen; damit werden dann aber auch alle Tastenzuordnungen der Global-Labels unwirksam.
Der eigentliche Zweck des Programms „CB" ist jedoch, ungewollte Speicherungen im Bereich des I/O-Buffers zu löschen, die sonst vom Betriebssystem niemals gelöscht würden.

```
01*LBL "CB"
02 13
03 PEEKB
04 X<>Y
05 A+
06 PEEKB
07 16
08 MOD
09 LASTX
10 X↑2
11 *
12 R↑
13 +
14 DSE X
15  E3
16 /
17 193
18 +
19 DSE X
20 CLST
21 .

22*LBL 00
23 POKER
24 ISG Y
25 GTO 00
26 END

   PLNG "CB"
    46 BYTES
```

Das Programm „CB" berechnet bis einschließlich Zeile 13 die absolute Adresse des Registers, in dem das **.END.** gespeichert ist. Anschließend wird in Zeile 14 diese Adresse um 1 vermindert. Wenn im Rechner einige freie Register vorhanden sind (RTN, PRGM an: zeigt 00 REG NN), so kann man diese freien Register mit einem Schlag versperren, indem man in das Register unmittelbar unterhalb des **.END.** eine Zahl speichert. Man läßt also „CB" nur bis zur Zeile 14 (einschl.) laufen und gibt dann ein: ENTER, POKER. Nun zeigt das Betriebssystem des HP-41 keine freien Register mehr an. Dieser fehlerhafte Zustand läßt sich wieder aufheben, indem man das Programm „CB" ausführt.
Jede ungewollte Speicherung innerhalb der freien Register kann mit „CB" wieder gelöscht werden. Das Programm läßt sich auch so modifizieren, daß eine bestimmte Anzahl von Tastenzuweisungsregistern und Bufferregistern erhalten bleibt; dazu ist die Zahl der Register, die erhalten bleiben sollen, von 193 abzuziehen und das Ergebnis in Zeile 17 des Programms „CB" einzusetzen. Man kann dazu auch die Programme „A?" und „CB" entsprechend kombinieren.

Weitere Hinweise

Da mit **POKER** von der Tastatur aus oder programmgesteuert jedes beliebige Register verändert werden kann, sollte man diese Funktion nur anwenden, wenn man die Rechnerstruktur genau kennt. Andernfalls sind ungewollte Veränderungen in Programmen, Datenregistern, Statusregistern usw. oder „MEMORY LOST“ die Folge.

Verwandte Funktionen

PEEKB, **PEEKR**, **POKEB**

Der gesamte Funktionsblock der -ADV FNS bietet vor allem im Zusammenwirken der einzelnen Funktionen eine Fülle neuer Möglichkeiten für die fortschrittliche Programmierung des HP-41. Ein weiteres Beispiel dafür ist das Programm „ST". Es dient dazu, sämtliche synthetische Textzeilen eines Programms herauszufinden und decodiert auszudrucken. Nach der Eingabeaufforderung „PRGM ?" ist ein Global-Label des zu untersuchenden Programms einzugeben und mit R/S zu starten. In der Anzeige des HP-41 ist jeweils die gerade untersuchte Zeilennummer zu sehen. Es ist SIZE 020 erforderlich. Das Programm „ST" enthält keine synthetischen Textzeilen, jedoch sind drei synthetische Drei-Byte-GTO-Befehle enthalten:
Zeile 031 GTO 13
Zeile 121 GTO 08
Zeile 142 GTO 08

Das Programm „ST" (Synthetic Textlines)

```
01◆LBL "ST"
02 CLST
03 WSIZE
04 STO 01
05 "PRGM? "
06 PMTA
07 PRA
08 PHD
09 STO 00
10 CF 22
11 GTO 08

12◆LBL 01
13 RCL 00
14 A-
15 STO 00
16 LASTX
17 PEEKB
18 RTN

19◆LBL 02
20 XEQ 01
21◆LBL 03
22 XEQ 01
23 GTO 08

24◆LBL 04
25 XEQ 01

26◆LBL 05
27 XEQ 01
28 240
29 -
30 X<0?
31 GTO 13
32 STO 03

33◆LBL 06
34 XEQ 01
35 DSE 03
36 GTO 06
37 GTO 08
```

```
38◆LBL 07
39 FS? 22
40 GTO 09
41 SF 22

42◆LBL 08
43 ISG 01
44 CLX
45 CF 21
46 VIEW 01

47◆LBL 09
48 XEQ 01
49 STO 04
50 X=0?
51 CF 22
52 X=0?
53 GTO 09
54 16
55 X>Y?
56 GTO 08
57 13
58 +
59 X>Y?
60 GTO 07
61 CF 22
62 2
63 +
64 X>Y?
65 GTO 05
66 113
67 +
68 X>Y?
69 GTO 08
70 48
71 +
72 X>Y?
73 GTO 03
74 14
75 +
76 X=Y?
77 GTO 03
78 1
79 +
80 X=Y?
81 GTO 03
82 X>Y?
83 GTO 04
84 33
85 +
86 X>Y?
87 GTO 02
88 -
89 X=0?
90 SF 21
91 4
92 +
93  E3
94 /
95 5
96 +
97 STO 02
98 1
99 -
100 STO 03
101 FS? 21
102 GTO 11

103◆LBL 10
104 XEQ 01
105 127
106 X=Y?
107 GTO 11
108 32
109 -
110 X<Y?
111 SF 21
112 X<> L
113 X>Y?
114 SF 21
```

```
115◆LBL 11
116 RDN
117 STO IND
          02
118 ISG 02
119 GTO 10
120 FC? 21
121 GTO 08
122 CLA
123 RCL 01
124  E2
125 X>Y?
126 "0"
127 SQRT
128 X>Y?
129 "⊢0"
130 X<>Y
131 ARCLI
132 "⊢:"
```

```
133◆LBL 12
134 RCL IND
          03
135 ARCLH
136 "⊢ "
137 ACA
138 CLA
139 ISG 03
140 GTO 12
141 PRBUF
142 GTO 08

143◆LBL 13
144 SF 21
145 ADV
146 BEEP
147 END

    PLNG "ST"
    251 BYTES
```

Barcodes von Programmen direkt aus dem Programmspeicher mit CCD-Modul und ThinkJet Drucker

Das Programm „PBC" ermöglicht es, von jedem Programm direkt aus dem Programmspeicher (RAM) des HP-41 Barcodes zu drucken. Dazu sind außer dem CCD-Modul das Extended Functions Modul und der ThinkJet Drucker (mit IL-Modul) erforderlich.

Programmbedienung:

1. Den Drucker ans Netz anschließen und über **CAT'0** selektieren und zurücksetzen (Taste ENTER und Taste C im **CAT'0**)
2. SIZE 019 ist erforderlich
3. Ein Global-Label des Programms, von dem die Barcodes gedruckt werden sollen, ins ALPHA-Register eingeben. Bei leerem ALPHA-Register druckt das Programm „PBC" seine eigenen Barcodes.
4. Das Programm „PBC" starten.

```
01♦LBL "PBC
       "
02 PHD
03 CLRG
04 STO 00
05 1
06 PPLNG
07 STO 02
08 -
09 A+B
10 PEEKB
11 SF 23
12 6
13 bS?
14 CF 23
15 9.018
16 STO 03
17 SF 21
18 PRA
19 "Æ*r784S
       "
20 ACA
21 "µ"
22 23
23 CRFLAS

24♦LBL 00
25 RCL 00
26 A-
27 X<> 00
28 PEEKB
29 STO IND
          03
30 ST+ 06
31 DSE 01
32 GTO 01
33 143
34 -
35 X<=0?
36 97
37 64
38 -
39 X<=0?
40 34
41 32
42 -
43 X<=0?
44 3
45 STO 01
46 STO 04

47♦LBL 01
48 ISG 03
49 X<0?
50 GTO 02
51 DSE 02
52 GTO 00

53♦LBL 02
54 RCL 05
55 16
56 MOD
57 LASTX
58 FC? 23
59 ST+ X
60 +
61 ST+ 06
62 STO 07
63 ISG 05
64 CLX
65 RCL 04
66 RCL 01
67 DSE X
68 -
69 RCL 08
70 +
71 STO 08
72 RCL 06
73 +
74 255
```

```
 75 MOD
 76 X=0?
 77 LASTX
 78 STO 06
 79 RCL 03
 80 INT
 81 DSE X
 82  E3
 83 /
 84 9
 85 +
 86 STO 03
 87 3
 88 -
 89 CLA
 90 RCL 05
 91 CHS
 92 ARCLI
 93 "⊢-"
 94 PRA
 95 "ᵽᵽ"
 96 SF 22

 97◆LBL 03
 98 X<>Y
 99 RCL IND
              X
100 7
101 CHS

102◆LBL 04
103 bC?
104 XEQ 07
105 LASTX
106 bS?
107 XEQ 08
108 LASTX
109 ISG X
110 GTO 04
111 APPCHR
112 CLA
113 RDN
114 ISG Y
115 GTO 03
116 XEQ 08
117 XEQ 07
118 APPCHR
119 26

120◆LBL 05
121 RCLPT
122  E3
123 *
124 "Æ*b"
125 ARCLI
126 "⊢W"
127 ACA
128 CLX
129 SEEKPT

130◆LBL 06
131 GETREC
132 OUTA
133 FS? 17
134 GTO 06
135 X<>Y
136 DSE X
137 GTO 05
138 6
139 "Æ*b1W◆"
140 GTO 10

141◆LBL 07
142 FS? 22
143 "⊢ᵽ"
144 FC? 22
145 "⊢↓"
146 RTN
```

```
147*LBL 08
148 FC?C 22
149 GTO 09
150 "⊢⊢"
151 RTN

152*LBL 09
 153 "⊢↓"
154 SF 22
155 RTN

156*LBL 10
157 ACA
158 DSE X
159 GTO 10
160 "μ"
161 CLFL
162 RCL 05
163 18
164 MOD
165 X=0?
166 OUTA
167 16
168 RCL 01
169 DSE X
170 *
171 STO 08
172 DSE 02
173 GTO 00
174 PURFL
175 "Æ*rB"
176 OUTA
177 BEEP
178 END

  PLNG "PBC"
   319 BYTES
```

Programmbeschreibung:
Das Programm benutzt zwei Routinen aus dem Barcodeprogramm von Winfried Maschke, wie es im PRISMA 10/11-1982 und im PPC Calculator Journal V9N4P45 veröffentlicht worden ist. Die Barcodes werden entsprechend dem Status des Progamms, von dem sie gedruckt werden, in normaler oder privat-geschützter Form erstellt. Um sicherzustellen, daß die Graphik-Kapazität des ThinkJet Druckers niemals überschritten werden kann, werden nur zehn Programmbytes (+ drei Steuerbytes) in eine Reihe gedruckt. Das Programm enthält elf synthetische Textzeilen:

Zeile	Dezimalwerte	Bedeutung
019	247, 027, 042, 114, 055, 056, 052, 083	Escape-Sequenz für den Graphik-Modus
021	241, 012	File-Name
095	242, 112, 112	Start-Bits 0 0
124	243, 027, 042, 098	Escape-Sequenz für die Anzahl der Graphik-Bytes zusammen mit Zeilen 125, 126
139	246, 027, 042, 098, 049, 087, 000	Escape-Sequenz für einen Graphik-Vorschub
143	242, 127, 112	Graphik-Byte für 0 – Bit
145	242, 127, 007	Graphik-Byte für 0 – Bit
150	242, 127, 127	Graphik-Byte für 1 – Bit
153	243, 127, 007, 240	Graphik-Bytes für 1 – Bit
160	241, 012	File-Name und Blatt-Vorschub
175	244, 027, 042, 114, 066	Escape-Sequenz für Ende des Graphik-Modus

Die dezimalen Bytewerte des Programms, von dem Barcodes gedruckt werden sollen, werden mit PEEKB (Zeile 28) aus dem Programmspeicher gelesen und zunächst in den Datenregistern 9 bis 18 zwischengespeichert (Zeile 29). Das Register 00 enthält jeweils die absolute Byteadresse des nächsten zu verarbeitenden Bytes. Im Datenregister 02 ist die Anzahl der Programmbytes gespeichert; dieses Register wird als Zähler verwendet (Zeilen 51 und 172). Wenn zehn Bytewerte zwischengespeichert sind, werden die Werte für die drei Steuerbytes berechnet (Zeilen 54 bis 78); diese werden in den Datenregistern 06, 07 und 08 gespeichert; Register 05 enthält jeweils die Reihennummer.

Beginnend mit Zeile 95 wird für jedes einzelne Bit der Barcodereihe ein Graphikbyte in ein Text-File gespeichert. Zeile 95 enthält die beiden Bytes für die zwei Startbits; alle weiteren Bytes werden für jedes Bit des gelesenen Programms in den Labels 07, 08 oder 09 gebildet. Jedes Byte des Programms ergibt 8 bis 12 Graphikbytes. Um die gewünschte Höhe der Barcodereihen zu erreichen, wird die gesamte Graphik-Information einer Reihe 26 mal zum Drucker geschickt (Label 05 und Label 06). Für eine individuelle Gestaltung der Höhe kann die Steuerzahl (Zeile 119) verändert werden. Die Zeilen 156 bis 159 bewirken in Verbindung mit der Escape-Sequenz aus Zeile 139 einen Abstand zur nächsten Barcodereihe. Die zugehörige Steuerzahl (6) steht in Zeile 138 und kann ebenfalls beliebig verändert werden.

Das Programm druckt eine Reihe sofort lesbarer Barcodes in erstklassiger Qualität in etwa 3 1/2 Minuten. Würde man die Graphik-Bytes nicht in einem Text-File zwischenspeichern, wäre ein Programm mit gleicher Funktion wie „PBC" auch ohne das Extended Functions Modul möglich; dann würde jedoch der Ausdruck einer Barcodereihe etwa 35 Minuten dauern! Nachdem jeweils 18 Barcodereihen auf einer Seite gedruckt worden sind, führt das Programm automatisch einen Blattwechsel durch (Zeilen 162 bis 166); Steuerzahl dazu in Zeile 163. So druckt das Programm mit Endlospapier vollkommen automatisch Barcodes für beliebig lange Programme. Bei Einzelblattbetrieb muß man nach dem Papierauswurf von Hand ein neues Blatt einlegen und dann die blaue Taste „TOF" betätigen. (Das Programm braucht bei diesem Papierwechsel nicht angehalten zu werden.)

Es werden die Flags 17, 22 und 23 benutzt. Das Programm kann jederzeit angehalten werden; man kann es auch mit SST ausführen. Wenn der Rechner während des Programmlaufs ausgeschaltet werden soll (z.B. zum Batteriewechsel), ist vorher der Zustand der Flags 17, 22 und 23 abzufragen. Dieser Zustand muß beim Neustart wiederhergestellt werden. Die Stackregister und das ALPHA-Register, die vom Programm benutzten Datenregister (0 bis 18) und SIZE dürfen natürlich ebenfalls nicht verändert werden, wenn das Programm erfolgreich fortgesetzt werden soll. Soll das Programm abgebrochen werden, ist der Drucker (über **CAT'0**) zurückzusetzen und das Text-File zu löschen.

Wer ein Extended Functions Modul der Revision B besitzt, sollte nach dem Programmende unbedingt ein Arbeitsfile definieren (wegen PURFL in Zeile 174). Das Programm enthält ein Long-Form-GTO (208, 000, 000) in Zeile 173.

Programm (319 Bytes) und Text von Gerhard Kruse

Kapitel 8

XF/Memory-Funktionen

Inhaltsverzeichnis Kapitel 8
XF/Memory Funktionen

Der Extended Functions/Memory Funktionsblock

Die Funktionen dieses Blocks vergrößern die Nutzungsmöglichkeiten des Extended Functions Moduls. Sie sind daher nur in Verbindung mit diesem Modul oder einem HP-41 CX verwendbar. Andernfalls erscheint die Fehlermeldung „NO XF/M".

SAVEB

Diese Funktion speichert den I/O-Buffer mit der im X-Register angegebenen Identitätszahl *aa* als Bufferfile im erweiterten Speicherbereich des Extended Functions- oder X-Memory Moduls.
Die Bedienung ist analog zu SAVEP. Der Filename ist im ALPHA-Register anzugeben. Bei leerem ALPHA-Register wird der momentane File angesprochen.
Die Funktion SAVEB erlaubt, beliebig viele Files (mit unterschiedlichem Namen) vom gleichen Typ und mit gleicher Identitätszahl anzulegen.
(Aufstellung einiger Identitätszahlen zu verschiedenen Buffertypen siehe Funktion B?.)

Eingabeparameter

X-Register : *aa*
ALPHA-Register : Filename

Beispiel

Die gegenwärtigen Alarmdaten sollen als File mit dem Namen „ALM" gespeichert werden.
Eingaben: ALPHA *ALM* ALPHA 10 SAVEB.
Diese Eingaben bewirken, daß zusätzlich zu dem Alarmbuffer, der bestehen bleibt, das Bufferfile „ALM" angelegt wird. Mit 10 CLB könnten die Alarmdaten gelöscht werden, um diese Register freizusetzen.

Weitere Hinweise

Ein Bufferfile erscheint im **CAT'4** mit der Typenbezeichnung **B**. Das CCD-Modul unterscheidet verschiedene Bufferfiles, auch wenn dies nicht im **CAT'4** erkennbar ist; so kann ein Bufferfile, das Alarmdaten für das TIME-Modul enthält (Identitätszahl 10), nicht mit einem Bufferfile mit einer anderen Identitätszahl überschrieben werden.

Verwandte Funktionen

GETB, SAVEK, GETK

GETB

Diese Funktion holt die Bufferdaten wieder aus dem Bufferfile zurück in den I/O-Buffer. Der Filename ist im ALPHA-Register anzugeben, bei leerem ALPHA-Register wird der momentane File angesprochen. Sofern ein I/O-Buffer, der dem Bufferfile entspricht, das zurückgeholt werden soll, bereits existiert, wird dieser I/O-Buffer zunächst gelöscht, bevor mit den Filedaten ein neuer Buffer angelegt wird.

Eingabeparameter

ALPHA-Register : Filename

Beispiel

Programm zu I/O-Buffern: „BS?" (Buffer Size?)

Das Programm „BS?" ist mit der gewünschten Identitätszahl im X-Register zu starten. Es errechnet dann, wieviele Register der entsprechende Buffer belegt. Die Ergebnisanzeige versteht sich einschließlich vorhandener Headerregister. Der Befehl **SF 99** in Zeile 04 soll die Fehleranzeige „NONEXISTENT" auslösen, falls der Buffer mit der eingegebenen Identitätszahl nicht existiert. Das Programm arbeitet einwandfrei, unabhängig davon, wieviele verschiedene Buffer bestehen. Auch können beliebig viele (oder gar keine) Tastenzuweisungsregister belegt sein. Die Identitätszahl des Buffers, der durch das CCD-Modul selbst angelegt wird, ist 5. So gibt 5 XEQ „BS?" an, wieviele Register dieser Buffer belegt.

```
01◆LBL "BS?
       "
02 B?
03 X=0?
04 SF 99
05 17
06 *
07 191.6
08 R↑
09 240

10◆LBL 00
11 R↑
12 R↑
13 ISG X
14 PEEKB
15 R↑
16 X=Y?
17 GTO 00
18 X<> T
19 X=Y?
20 GTO 02

21◆LBL 01
22 XEQ 02
23 +
24 A+
25 PEEKB
26 RCL Z
27 X≠Y?
28 GTO 01

29◆LBL 02
30 RCL Z
31 A-
32 PEEKB
33 END

 PLNG "BS?"
   64 BYTES
```

Weitere Hinweise

I/O-Buffer werden vom Betriebssystem des HP-41 immer automatisch gelöscht, wenn der Rechner ohne das zugehörige Modul eingeschaltet wird. Die Funktionen SAVEB und GETB arbeiten jedoch unabhängig davon, ob das zugehörige Modul für den behandelten Buffer eingesteckt ist.
Die Funktion CLFL sollte nicht für Bufferfiles ausgeführt werden; zum Löschen der Files ist PURFL zu verwenden.

Verwandte Funktionen

SAVEB, SAVEK, GETK, MRGK

SAVEK

Diese Funktion speichert alle Tastenzuordnungen von Befehlen aus **CAT 2** und **CAT 3** als Keyfile im erweiterten Speicherbereich des Extended Functions Moduls oder des X-Memory Moduls.
Die Bedienung ist analog zu SAVEP. Der Filename ist im ALPHA-Register anzugeben. Bei leerem ALPHA-Register wird der momentane File angesprochen.

Eingabeparameter

ALPHA-Register : Filename

Beispiel

Die vorhandenen Tastenzuordnungen aus **CAT 2** und **CAT 3** sollen in einem Keyfile mit Namen „KEYS" abgespeichert werden. Hierzu sind folgende Schritte notwendig:
ALPHA *KEYS* ALPHA SAVEK

Weitere Hinweise

Ein Keyfile erscheint im **CAT'4** mit der Bezeichnung **K**. Die Tastenzuordnungen bleiben unverändert erhalten.

Verwandte Funktionen

GETK, MRGK, GETB, SAVEB

GETK

Diese Funktion löscht alle Tastenzuordnungen von Befehlen aus **CAT 2** und **CAT 3** und aktiviert dann die in dem angesprochenen Keyfile gespeicherten Tastenzuordnungen. Der Filename ist im ALPHA-Register einzugeben. Bei leerem ALPHA-Register wird der momentane File angesprochen.

Eingabeparameter

ALPHA-Register : Filename

Beispiel

Ein Programm zu GETK: „CLK" (Anlegen eines CLEAR KEYS FILES)

Das Programm „CLK" legt ein File an, das im **CAT'4** als „CLK K000" erscheint, also ein Keyfile mit der Größe Null; es besteht nur aus den Headerregistern. Dieses File, auf das mit GETK immer wieder zugegriffen werden kann, löscht alle Tastenzuordnungen von Befehlen aus **CAT 2** und **CAT 3**. (CLKEYS würde auch Tastenzuordnungen von Global-Labeln löschen.)
Das Programm erfordert keine Eingaben. Es kann nur benutzt werden, wenn innerhalb der Register des erweiterten Speicherbereichs des Extended Functions Moduls noch mindestens zwei freie Register vorhanden sind. Andernfalls hält das Programm in Zeile 10 mit „NONEXISTENT" oder in Zeile 17 mit „DATA ERROR" an.
Das Programm „CLK" enthält zwei synthetische Textzeilen:
Zeile 03: F7, FF, FF, FF, FF, FF, FF, FF
Zeile 30: F0 (NOP) .
Zeile 18 enthält den Filenamen „CLK" und vier Leerstellen (Spaces); das ALPHA-Register hat hier also die Länge 7 !
Unmittelbar nach einem „MEMORY LOST" oder nachdem das Extended Functions Modul neu eingesteckt wurde, muß einmal **CAT'4** ausgeführt werden, bevor das Programm „CLK" benutzt werden kann.

Unterhalb des letzten benutzten Registers im erweiterten Speicherbereich wird ein Register mit sieben hex FF-Bytes (dez 255) als Grenze gespeichert. Das Programm „CLK" sucht zunächst diese Grenze (Zeilen 02 bis 13). Dann wird dieses Grenzregister mit dem Filenamen „CLK" überschrieben (Zeilen 18 bis 20), und das Register unmittelbar darunter erhält die Kennzeichnung als Keyfile (Zeilen 22 bis 28). Das darunterliegende Register wird dann als neues Grenzregister gekennzeichnet (Zeilen 29 bis 32). Anschließend löscht **GETK** mit Hilfe des erzeugten Keyfiles alle Tastenzuordnungen von Funktionen aus **CAT 2** und **CAT 3**.

```
01◆LBL "CLK
       "
02 192
03 ""
04 RCL [
05 ENTER↑

06◆LBL 00
07 R↑
08 R↑
09 DSE X
10 PEEKR
11 R↑
12 X≠Y?
13 GTO 00
14 66
15 R↑
16 X<=Y?
17 ASIN
18 "CLK
       "
19 RCL [
20 POKER
21 DSE Y
22 CLX
23 POKER
24 CLX
25 .6
26 +
27 96
28 POKEB
29 DSE Y
30 ""
31 X<> T
32 POKER
33 CLST
34 GETK
35 END

 PLNG "CLK"
    75 BYTES
```

Weitere Hinweise

Tastenzuordnungen von Programmen werden nur gelöscht, wenn durch das Keyfile auf die entsprechende Taste eine andere Zuordnung gelegt wird.

Verwandte Funktionen

SAVEK, **MRGK**, **SAVEB**, **GETB**

MRGK

Diese Funktion aktiviert die in dem angesprochenen Keyfile gespeicherten Tastenzuordnungen. Dabei werden bestehende Tastenzuordnungen nur dann gelöscht, wenn dieselbe Taste, die mit einer Zuordnung belegt ist, auch in den Zuordnungen des Keyfiles enthalten ist. Im übrigen bleiben bestehende Zuordnungen unverändert.

Eingabeparameter

ALPHA-Register : Filename

Beispiele

Da die Funktion GETK alle bestehenden Tastenzuordnungen aus **CAT 2** und **CAT 3** zunächst löscht, ergeben die Funktionen SAVEK und GETK eine einfache und schnelle Möglichkeit, um die Tastenzuweisungsregister zu pakken.
Die Tastenfolge dazu ist:
Beliebigen Namen ins ALPHA-Register eingeben, SAVEK, GETK, PURFL.
Sofern man ein Extended Functions Modul der Revision B besitzt, ist anschließend **CAT'4** oder EMDIR auszuführen, um die Files im erweiterten Speicher vor einer versehentlichen Löschung zu sichern.

Das folgende Programm packt die Tastenzuordnungsregister nur durch Anwendung der Funktionen PEEKB und POKEB. Man braucht hierzu also kein Extended Functions Modul wie bei der ersten Methode. Das Programm benötigt keine Eingabe und kann beliebig angehalten oder mit SST ausgeführt werden. Sobald einmal der Befehl POKEB erreicht worden ist, muß das Programm aber bis zum Ende ablaufen, da man sonst Unordnung oder Doppelspeicherungen in den Zuordnungsregistern erhält. Das Programm „PK" erfordert SIZE 002. Sämtliche Buffer bleiben unberührt.

```
01◆LBL "PK"
02 192
03 STO 00
04 STO 01

05◆LBL 00
06 RCL 00
07 ,6
08 +
09 PEEKB
10 240
11 X≠Y?
12 GTO 01
13 RCL 00
14 PEEKB
15 X≠0?
16 XEQ 02
17 RCL 00
18 ,3
19 +
20 PEEKB
21 X≠0?
22 XEQ 02
23 ISG 00
24 CLX
25 GTO 00

26◆LBL 01
27 RCL 00
28 RCL 01
29 X=Y?
30 GTO 05
31 0
32 POKEB
33 X<>Y
34 A+
35 X<>Y
36 POKEB
37 X<>Y
38 A+
39 XEQ 03
40 GTO 01

41◆LBL 02
42 RCL 01
43 XEQ 04
44 XEQ 04

45◆LBL 03
46 X<>Y
47 POKEB
48 X<>Y
49 ENTER↑
50 FRC
51 +
52 ENTER↑
53 FRC
54 4
55 /
56 -
57 STO 01
58 RTN

59◆LBL 04
60 X<>Y
61 POKEB
62 RCL Z
63 A+
64 PEEKB
65 R↑
66 A+
67 RTN

68◆LBL 05
69 CLST
70 SEED
71 END

PLNG "PK"
113 BYTES
```

Weitere Hinweise

keine

Verwandte Funktionen

SAVEK, GETK, SAVEB, GETB

SORTFL

Diese Funktion sortiert die Register eines Datenfiles. Der Filename ist im ALPHA-Register anzugeben. Bei leerem ALPHA-Register wird der momentane File angesprochen. Es werden sowohl numerische als auch ALPHA-Daten sortiert. Die Stackregister werden nicht verändert. Nach Ausführen von **SORTFL** befindet sich der kleinste Wert des spezifizierten Datenfiles im ersten Datenregister des Files.

Eingabeparameter

ALPHA-Register : Filename

Weitere Hinweise

Im Gegensatz zur Funktion **SORT** ist ein absteigendes Sortieren nicht möglich!

Verwandte Funktion

SORT

Zum Abschluß dieses Funktionsblocks noch ein Programm, das ein sicher schon oft aufgetauchtes Problem einfach löst:

Das Speichern eines Textfiles auf Magnetkarte.

Das Programm „WF / RF", das nur einen synthetischen Befehl enthält, kann für jeden Filetyp benutzt werden, wurde aber eigentlich für Textfiles (ASCII-Files) geschrieben. Zeile 18 hängt sechs Leerstellen (Spaces) an das ALPHA-Register an. Das zweite Headerregister des Files muß im Bereich des Extended Functions Moduls (absolute Adresse 65 bis 190) liegen.
Das Programm „WF" (Write File) kopiert alle Register des Files in Datenregister. Dazu wird die Anzahl der Datenregister an die Anzahl der Fileregister angepaßt (FLSIZE, PSIZE). Deshalb kann das Programm nur bei solchen Files angewendet werden, die vollständig in den Datenregistern (zuzüglich vorhandener freier Register) untergebracht werden können. Jedoch können in einem HP-41 CV oder HP-41 CX immerhin Files bis zu einer Größe von über 300 Registern verarbeitet werden!

Programmbedienung:
Der Filename ist in das ALPHA-Register einzugeben. Dann ist „WF" zu starten, um das File auf Magnetkarten zu speichern. Bei der Eingabeaufforderung „RDY 01 OF NN" muß man unbedingt eine (mehrere) Magnetkarte(n) einschieben oder das Programm mit zweimal R/S wieder starten. Andernfalls wird der Filetyp, der vorübergehend geändert wurde, nicht wieder in den ursprünglichen Filetyp zurückgestellt.
Bevor von Magnetkarten der Fileinhalt mit dem Programm „RF" (Read File) wieder zurückgelesen werden kann, muß das gewünschte File mit dem richtigen Filetyp und in ausreichender Größe angelegt werden. Dann ist der Filename wieder in das ALPHA-Register einzugeben und „RF" zu starten. Bei der Eingabeaufforderung „CARD" muß man unbedingt eine (mehrere) Magnetkarte(n) einschieben oder das Programm mit zweimal R/S wieder starten (s.o.).
Man kann das Programm auch nur dazu verwenden, um die Filedaten in Datenregister zu übertragen und umgekehrt. Dazu sind die Befehle WDTA und RDTA zu löschen. Dann ist es z.B. auch möglich, ein Textfile zu vergrößern, ohne den gesamten Inhalt neu eingeben zu müssen:
Zuerst „WF", File löschen, dann File mit größerer Anzahl Registern neu anlegen, „RF".

Das Programm „WF/RF" ist auf **alle** Filetypen anwendbar. Für Programmfiles ist die Anwendung natürlich sinnlos. Wenn das Programm für Bufferfiles verwendet wird, muß vor dem Zurücklesen das Bufferfile exakt vom gleichen Typ und mit gleicher Identitätszahl angelegt werden, wie dies den Daten auf der (den) Magnetkarte(n) entspricht. Auch die Größe sollte gleich sein, um Schwierigkeiten zu vermeiden. Wenn man dies beachtet, ist es kein Problem, alle Arten von Buffern über den Umweg eines Bufferfiles auf Magnetkarten zu speichern.

```
01◆LBL "WF"
02 XEQ 00
03 GETR
04 WDTA
05 POKEB
06 RTN

07◆LBL "RF"
08 XEQ 00
09 RDTA
10 SAVER
11 POKEB
12 RTN

13◆LBL 00
14 RCLPTA
15 FLSIZE
16 PSIZE
17 191.6
18 "⊢
                "
19 7
20 AROT
21 RCL [
22 CLA
23◆LBL 01
24 R↑
25 R↑
26 PEEKR
27 DSE Y
28 CLX
29 R↑
30 X≠Y?
31 GTO 01
32 RCL Z
33 PEEKB
34 X<>Y
35 32
36 POKEB
37 CLX
38 SEEKPT
39 X<> Z
40 TONE 8
41 END

   PLNG "WF"
    89 BYTES
```

Kapitel 9

Barcodes

Inhaltsverzeichnis Kapitel 9
Barcodes

A?

Benötigte Programmregister: 7

Zeile 1 (1-4) Barcodes

Zeile 2 (4-12) Barcodes

Zeile 3 (13-21) Barcodes

Zeile 4 (22-27) Barcodes

ABIN

Benötigte Programmregister: 43

Zeile 1 (1-2) Barcodes

Zeile 2 (2-11) Barcodes

Zeile 3 (11-15) Barcodes

Zeile 4 (16-20) Barcodes

Zeile 5 (20-28) Barcodes

Zeile 6 (29-35) Barcodes

Zeile 7 (36-40) Barcodes

Zeile 8 (41-47) Barcodes

Zeile 9 (47-53) Barcodes

Zeile 10 (54-62) Barcodes

Zeile 11 (63-71) Barcodes

Zeile 12 (72-79) Barcodes

Zeile 13 (80-85) Barcodes

Zeile 14 (86-95) Barcodes

Zeile 15 (96-103) Barcodes

Zeile 16 (103-111) Barcodes

Zeile 17 (112-120) Barcodes

Zeile 18 (120-128) Barcodes

Zeile 19 (129-137) Barcodes

Zeile 20 (138-142) Barcodes

Zeile 21 (142-150) Barcodes

Zeile 22 (150-155) Barcodes

Zeile 23 (155-162) Barcodes

Zeile 24 (162) Barcodes

BS ?

Benötigte Programmregister: 10

Zeile 1 (1-5) Barcodes

Zeile 2 (5-11) Barcodes

Zeile 3 (12-20) Barcodes

Zeile 4 (20-27) Barcodes

Zeile 5 (28-36) Barcodes

Zeile 6 (37-39) Barcodes

CB

Benötigte Programmregister: 7

Zeile 1 (1-5) Barcodes

Zeile 2 (6-15) Barcodes

Zeile 3 (15-23) Barcodes

Zeile 4 (24-29) Barcodes

CDE

Benötigte Programmregister: 9

Zeile 1 (1-4) Barcodes

Zeile 2 (5-13) Barcodes

Zeile 3 (14-23) Barcodes

Zeile 4 (24-31) Barcodes

Zeile 5 (32-38) Barcodes

CF55

Benötigte Programmregister: 3

Zeile 1 (1-4) Barcodes

Zeile 2 (5-8) Barcodes

CHK

Benötigte Programmregister: 10

Zeile 1 (1-4) Barcodes

Zeile 2 (4-8) Barcodes

Zeile 3 (9-17) Barcodes

Zeile 4 (18-25) Barcodes

Zeile 5 (26-33) Barcodes

Zeile 6 (34-36) Barcodes

CLK

Benötigte Programmregister: 11

Zeile 1 (1-3) Barcodes

Zeile 2 (3-9) Barcodes

Zeile 3 (10-18) Barcodes

Zeile 4 (18-22) Barcodes

Zeile 5 (23-30) Barcodes

Zeile 6 (31-37) Barcodes

GE

Benötigte Programmregister: 5

Zeile 1 (1-5) Barcodes

Zeile 2 (6-15) Barcodes

Zeile 3 (15-20) Barcodes

H-0

Benötigte Programmregister: 8

Zeile 1 (1-4) Barcodes

Zeile 2 (5-8) Barcodes

Zeile 3 (9-12) Barcodes

Zeile 4 (12-17) Barcodes

Zeile 5 (18-19) Barcodes

INP

Benötigte Programmregister: 7

Zeile 1 (1-4) Barcodes

Zeile 2 (4-11) Barcodes

Zeile 3 (12-19) Barcodes

Zeile 4 (20-26) Barcodes

INV

Benötigte Programmregister: 36

Zeile 1 (1-5) Barcodes

Zeile 2 (6-13) Barcodes

Zeile 3 (13-20) Barcodes

Zeile 4 (21-29) Barcodes

Zeile 5 (30-41) Barcodes

Zeile 6 (42-49) Barcodes

Zeile 7 (50-60) Barcodes

Zeile 8 (60-68) Barcodes

Zeile 9 (69-78) Barcodes

Zeile 10 (78-85) Barcodes

Zeile 11 (85-94) Barcodes

Zeile 12 (95-103) Barcodes

Zeile 13 (104-112) Barcodes

Zeile 14 (112-118) Barcodes

Zeile 15 (118-125) Barcodes

Zeile 16 (125-135) Barcodes

Zeile 17 (136-143) Barcodes

Zeile 18 (144-150) Barcodes

Zeile 19 (151-156) Barcodes

PBC

Benötigte Programmregister: 46

Zeile 1 (1-6) Barcodes

Zeile 2 (6-14) Barcodes

Zeile 3 (14-19) Barcodes

Zeile 4 (19-23) Barcodes

Zeile 5 (23-30) Barcodes

Zeile 6 (31-38) Barcodes

Zeile 7 (38-48) Barcodes

Zeile 8 (49-57) Barcodes

Zeile 9 (57-65) Barcodes

Zeile 10 (66-76) Barcodes

Zeile 11 (76-86) Barcodes

Zeile 12 (87-96) Barcodes

Zeile 13 (96-104) Barcodes

Zeile 14 (105-111) Barcodes

Zeile 15 (111-118) Barcodes

Zeile 16 (118-124) Barcodes

Zeile 17 (125-130) Barcodes

Zeile 18 (131-138) Barcodes

Zeile 19 (138-142) Barcodes

Zeile 20 (143-149) Barcodes

Zeile 21 (150-155) Barcodes

Zeile 22 (156-163) Barcodes

Zeile 23 (163-171) Barcodes

Zeile 24 (172-177) Barcodes

Zeile 25 (177-183) Barcodes

PHINPT

Benötigte Programmregister: 5

Zeile 1 (1-2) Barcodes

Zeile 2 (2-9) Barcodes

Zeile 3 (10-14) Barcodes

PK

Benötigte Programmregister: 17

Zeile 1 (1-6) Barcodes

Zeile 2 (7-14) Barcodes

Zeile 3 (14-22) Barcodes

Zeile 4 (22-30) Barcodes

Zeile 5 (31-39) Barcodes

Zeile 6 (39-45) Barcodes

Zeile 7 (46-57) Barcodes

Zeile 8 (58-66) Barcodes

Zeile 9 (66-77) Barcodes

Zeile 10 (77) Barcodes

PR1

Benötigte Programmregister: 5

Zeile 1 (1-5) Barcodes

Zeile 2 (6-11) Barcodes

Zeile 3 (11-15) Barcodes

PR2

Benötigte Programmregister: 8

Zeile 1 (1-3) Barcodes

Zeile 2 (3-4) Barcodes

Zeile 3 (4-6) Barcodes

Zeile 4 (6-11) Barcodes

Zeile 5 (11-12) Barcodes

PR3

Benötigte Programmregister: 5

Zeile 1 (1-3) Barcodes

Zeile 2 (3-6) Barcodes

Zeile 3 (7-10) Barcodes

PR4

Benötigte Programmregister: 5

Zeile 1 (1-3) Barcodes

Zeile 2 (3-7) Barcodes

Zeile 3 (7-11) Barcodes

ST

Benötigte Programmregister: 36

Zeile 1 (1-5) Barcodes

Zeile 2 (5-10) Barcodes

Zeile 3 (11-20) Barcodes

Zeile 4 (20-26) Barcodes

Zeile 5 (27-33) Barcodes

Zeile 6 (34-40) Barcodes

Zeile 7 (40-48) Barcodes

Zeile 8 (48-56) Barcodes

Zeile 9 (56-65) Barcodes

Zeile 10 (65-73) Barcodes

Zeile 11 (73-82) Barcodes

Zeile 12 (83-90) Barcodes

Zeile 13 (91-101) Barcodes

Zeile 14 (102-108) Barcodes

Zeile 15 (108-117) Barcodes

Zeile 16 (117-124) Barcodes

Zeile 17 (124-132) Barcodes

Zeile 18 (132-138) Barcodes

Zeile 19 (139-145) Barcodes

Zeile 20 (146-147) Barcodes

TD

Benötigte Programmregister: 8

Zeile 1 (1-4) Barcodes

Zeile 2 (4-11) Barcodes

Zeile 3 (11-18) Barcodes

Zeile 4 (18-26) Barcodes

Zeile 5 (27-28) Barcodes

TLC

Benötigte Programmregister: 4

Zeile 1 (1-3) Barcodes

Zeile 2 (4-8) Barcodes

TLC1

Benötigte Programmregister: 4

Zeile 1 (1-3) Barcodes

Zeile 2 (4-10) Barcodes

TLC2

Benötigte Programmregister: 4

Zeile 1 (1-3) Barcodes

Zeile 2 (4-10) Barcodes

Zeile 3 (10) Barcodes

VB

Benötigte Programmregister: 6

Zeile 1 (1-5) Barcodes

Zeile 2 (6-11) Barcodes

Zeile 3 (11-20) Barcodes

Zeile 4 (21) Barcodes

VR

Benötigte Programmregister: 5

Zeile 1 (1-5) Barcodes

Zeile 2 (5-11) Barcodes

Zeile 3 (12-16) Barcodes

W?

Benötigte Programmregister: 3

Zeile 1 (1-6) Barcodes

Zeile 2 (7-11) Barcodes

WF

Benötigte Programmregister: 13

Zeile 1 (1-4) Barcodes

Zeile 2 (5-9) Barcodes

Zeile 3 (9-16) Barcodes

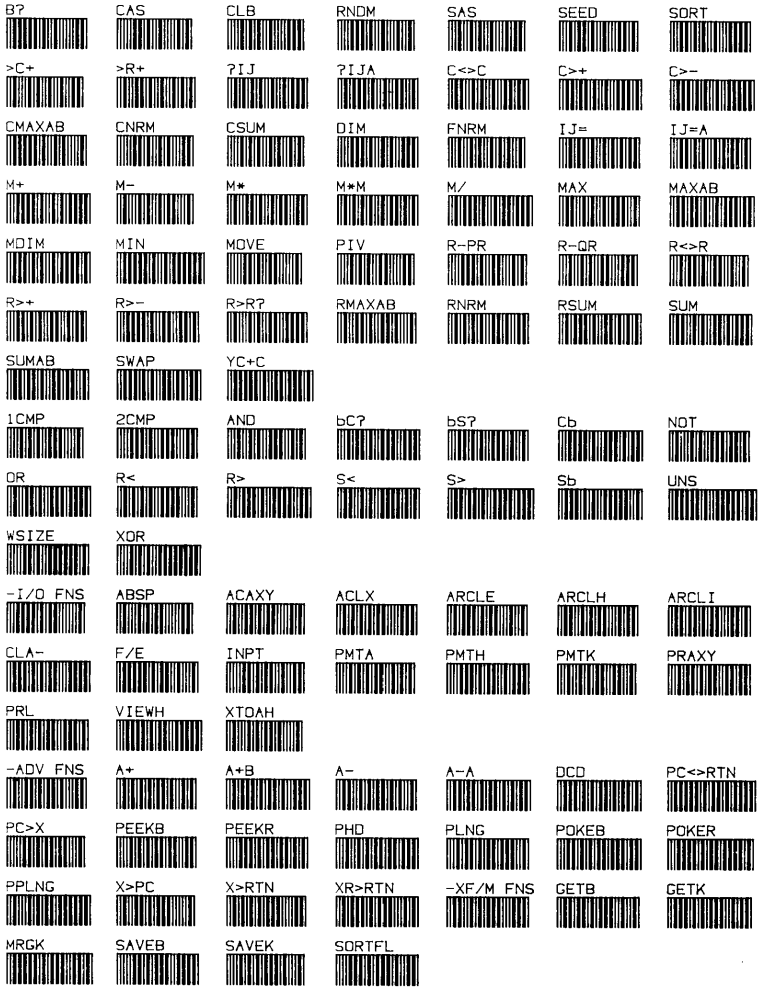
Zeile 4 (17-18) Barcodes

Zeile 5 (19-27) Barcodes

Zeile 6 (28-36) Barcodes

Zeile 7 (36-43) Barcodes

HP 41	BAR-CODES CCD-ROM	BLATT CCD-1



AUSGABE	HINWEISE:	ROLF SCHMIDT
0/2	X-ROM 9.00 - 9.63* sowie 11.00 - 11.41	ESSLINGEN
Jan. 1985	* X-ROM 9.00, 9.08 u. 9.47 nicht als Barcodes ausgedruckt.	CCD
		Nr. 1789

Kapitel 10

Kurzübersicht

Kurzübersicht

Folgende **CAT 2** Funktionsliste des CCD-Moduls zeigt die XROM-Nummern, Bytekombinationen und die Stackbedingungen aller CCD-Modul-Funktionen.

Zeichenerklärung:

- – Der Stack wird nicht verändert
- ↑ Der Stack wird nach oben verschoben; ein Ergebniswert wird in das X-Register geschrieben
- ↓ Der Stack wird nach unten verschoben (Stack-Drop)
- L Der ursprüngliche X-Wert wird in das LAST X-Register geschrieben

-W&W CCD A

FCN-NAME	XROM-#	BYTES	STACK	Seite
-W&W CCD A	09.00	162=064		
B?	09.01	162=065	–	3.05
CAS	09.02	162=066	–	3.09
CLB	09.03	162=067	–	3.07
RNDM	09.04	162=068	↑	3.10
SAS	09.05	162=069	–	3.08
SEED	09.06	162=070	–	3.12
SORT	09.07	162=071	–	3.13

-ARR FNS

-ARR FNS	09.08	162=072		
>C+	09.09	162=073	–	4.19
>R+	09.10	162=074	–	4.21
?IJ	09.11	162=075	↑	4.14
?IJA	09.12	162=076	↑	4.15
C<>C	09.13	162=077	–	4.31
C>+	09.14	162=078	↑	4.23
C>-	09.15	162=079	↑	4.25

FCN-NAME	XROM-#	BYTES	STACK	Seite
CMAXAB	09.16	162=080	↑	4.43
CNRM	09.17	162=081	↑	4.54
CSUM	09.18	162=082	–	4.52
DIM	09.19	162=083	↑	4.13
FNRM	09.20	162=084	↑	4.56
IJ=	09.21	162=085	–	4.16
IJ=A	09.22	162=086	–	4.17
M+	09.23	162=087	–	4.57
M-	09.24	162=088	–	4.59
M*	09.25	162=089	–	4.61
M*M	09.26	162=090	–	4.65
M/	09.27	162=091	–	4.63
MAX	09.28	162=092	↑	4.39
MAXAB	09.29	162=093	↑	4.41
MDIM	09.30	162=094	–	4.09
MIN	09.31	162=095	–	4.45
MOVE	09.32	162=096	–	4.35
PIV	09.33	162=097	↑	4.46
R-PR	09.34	162=098	–	4.73
R-QR	09.35	162=099	–	4.69
R<>R	09.36	162=100	–	4.33
R>+	09.37	162=101	↑	4.27
R>-	09.38	162=102	↑	4.29
R>R?	09.39	162=103	–	4.47
RMAXAB	09.40	162=104	↑	4.44
RNRM	09.41	162=105	↑	4.55
RSUM	09.42	162=106	–	4.53
SUM	09.43	162=107	↑	4.49
SUMAB	09.44	162=108	↑	4.51
SWAP	09.45	162=109	–	4.37
YC+C	09.46	162=110	–	4.67

-HEX FNS

-HEX FNS	09.47	162=111		
1CMP	09.48	162=112	L	5.19
2CMP	09.49	162=113	↓L	5.20
AND	09.50	162=114	↓L	5.27

FCN-NAME	XROM-#	BYTES	STACK	Seite
bC?	09.51	162=115	↓L	5.43
bS?	09.52	162=116	↓L	5.41
Cb	09.53	162=117	↓L	5.45
NOT	09.54	162=118	L	5.32
OR	09.55	162=119	↓L	5.29
R<	09.56	162=120	L	5.37
R>	09.57	162=121	L	5.39
S<	09.58	162=122	L	5.33
S>	09.59	162=123	L	5.35
Sb	09.60	162=124	↓L	5.47
UNS	09.61	162=125	L	5.21
USIZE	09.62	162=126	–	5.17
XOR	09.63	162=127	↓L	5.30

-I/O FNS

-I/O FNS	11.00	162=192		
ABSP	11.01	162=193	–	6.29
ACAXY	11.02	162=194	↓L	6.17
ACLX	11.03	162=195	–	6.22
ARCLE	11.04	162=196	–	6.32
ARCLH	11.05	162=197	–	6.35, 5.25
ARCLI	11.06	162=198	–	6.37
CLA-	11.07	162=199	–	6.30
F/E	11.08	162=200	–	6.27
INPT	11.09	162=201	–	6.05
PMTA	11.10	162=202	–	6.09
PMTH	11.11	162=203	↑	6.11, 5.22
PMTK	11.12	162=204	↑	6.13
PRAXY	11.13	162=205	↓L	6.20
PRL	11.14	162=206	–	6.24
VIEWH	11.15	162=207	–	6.26, 5.24
XTOAH	11.16	162=208	–	6.31, 5.26

-ADV FNS

-ADV FNS	11.17	162=209

FCN-NAME	XROM-#	BYTES	STACK	Seite
A+	11.18	162=210	L	7.09
A+B	11.19	162=211	↓L	7.11
A-	11.20	162=212	L	7.12
A-A	11.21	162=213	↓L	7.13
DCD	11.22	162=214	–	7.14
PC<>RTN	11.23	162=215	–	7.21
PC>X	11.24	162=216	↑	7.16
PEEKB	11.25	162=217	↑	7.24
PEEKR	11.26	162=218	↑	7.29
PHD	11.27	162=219	↑	7.08
PLNG	11.28	162=220	↑	7.05
POKEB	11.29	162=221	–	7.31
POKER	11.30	162=222	–	7.35
PPLNG	11.31	162=223	↑	7.07
X>PC	11.32	162=224	–	7.18
X>RTN	11.33	162=225	–	7.20
XR>RTN	11.34	162=226	–	7.23

-XF/M FNS

-XF/M FNS	11.35	162=227		
GETB	11.36	162=228	–	8.07
GETK	11.37	162=229	–	8.10
MRGK	11.38	162=230	–	8.12
SAVEB	11.39	162=231	–	8.05
SAVEK	11.40	162=232	–	8.09
SORTFL	11.41	162=233	–	8.15

Kapitel 11

Kompatibilität

Kompatibilität

Wenn Sie das CCD-Modul in Ihren Rechner gesteckt haben, kann es unter nachfolgend aufgeführten Umständen passieren, daß einige Funktionen nicht – wie beschrieben – anwendbar sind. Um diesem Zustand abzuhelfen, prüfen Sie bitte, ob einer der nachstehenden Punkte auf Sie zutrifft:

1) Sie haben ein altes Betriebssystem in Ihrem Rechner. Die Betriebssystemerweiterungen des CCD-Moduls sind nur möglich, wenn das Modul nach jedem Tastendruck für kurze Zeit die Kontrolle über den Rechner übernehmen kann. Diese Möglichkeit ist allerdings nur mit einem neueren Betriebssystem des HP-41 gegeben. Sollten Sie also einen älteren HP-41 besitzen (d. h., die Seriennummer ist kleiner als 2035 . . .), kann es sein, daß einige CCD-Modul-Betriebssystemerweiterungen nicht funktionieren. In diesem Fall können Sie
   a) einige Funktionen simulieren, indem Sie das CCD-Modul auf Port 1 stecken und die Barcodes für „ASN", „CAT" u. „XEQ" (siehe Barcodes) einlesen,

ASN

Benötigte Programmregister: 6

Zeile 1 (1-3) Barcodes

Zeile 2 (4-11) Barcodes

Zeile 3 (11-19) Barcodes

Zeile 4 (20) Barcodes

CAT

Benötigte Programmregister: 3

Zeile 1 (1-4) Barcodes

Zeile 2 (4-7) Barcodes

XEQ

Benötigte Programmregister: 3

Zeile 1 (1-4) Barcodes

Zeile 2 (4-7) Barcodes

b) bei eingestecktem Barcodeleser seine Taste betätigen, wenn ein „Prompt" erscheint. Hierdurch übernimmt das CCD-Modul die Kontrolle und die erweiterte Funktion tritt in Kraft,
c) Ihr Betriebssystem beim HP-Service gegen die üblichen Kosten austauschen lassen.

Für Inkompatibilitäten mit evtl. neu entwickelten HP-41 Modellen wird keine Haftung übernommen. Das CCD-Modul funktioniert mit allen bis Januar 1985 gebauten HP-41 Modellen.

2) **Sie betreiben das CCD-Modul zusammen mit dem HP-Development Modul.**
Wenn Sie ein HP-Development Modul zusammen mit dem CCD-Modul benutzen wollen, **vergewissern Sie sich bitte, daß das CCD-Modul vor dem HP-Development Modul steckt.** Ist dies nicht der Fall, ist ein einwandfreies Funktionieren der CCD-Modul Betriebssystemerweiterungen ebenfalls nicht gewährleistet. Sind die Module in der oben beschriebenen, richtigen Reihenfolge eingesteckt, ist nichts zu beachten.

3) Sie betreiben das CCD-Modul zusammen mit dem ZENROM.
Bei der Benutzung mit dem ZENROM gibt es eine Inkompatibilität im Kleinbuchstabenmodus, da das ZENROM ebenfalls Kleinbuchstaben erzeugen kann. Dies ist nur zu umgehen, wenn der Kleinbuchstabenmodus des CCD-Moduls abgeschaltet wird (siehe Programm „TLC").

4) Sie betreiben das CCD-Modul zusammen mit einem Modul gleicher XROM-Nummer.
Wenn Sie das CCD-Modul eingesteckt haben, dürfen weitere Module mit den XROM-Nummern 9 oder 11 nicht mehr eingesteckt werden.

Bis jetzt sind das folgende:

- Home Management Modul
- Real Estate Modul
- PANAME Modul

Kapitel 12

Literatur-hinweise

Literaturhinweise

Wenn Sie mehr über synthetische bzw. optimale Programmierung Ihres HP-41 wissen möchten, können wir Ihnen folgende Bücher empfehlen:

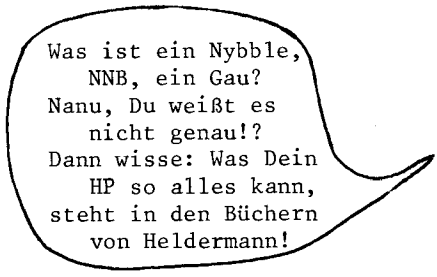
1) Optimales Programmieren mit dem HP-41, erschienen im Vieweg Verlag
   Autor: Gerhard Kruse

2) Synthetische Programmierung auf dem HP-41C/CV, 2. erweiterte Auflage, erschienen im Heldermann Verlag
   Autor: W. C. Wickes, deutsch von H. Dalkowski

3) Synthetisches Programmieren auf dem HP-41 – leicht gemacht, erschienen im Heldermann Verlag
   Autor: K. Jarett, Deutsche Ausgabe von H. Dalkowski

Alle Bücher über den HP-41 können Sie auch bei uns bestellen – Postkarte genügt:

Fa. W&W Software Products GmbH
Postfach 80 01 33
5060 Bergisch Gladbach 2



Albers,K.: HP-41 Barcodes mit dem HP-IL-System. Ca. 200 Seiten, ca. 38.00 DM (1986), ISBN 3-88538-804-9.

Dearing,J.S.: Tricks, Tips und Routinen für Taschenrechner der Serie HP-41. Deutsche Ausgabe von Heinz Dalkowski. 220 Seiten, 34.00 DM (1984), ISBN 3-88538-801-4.

Jarett,K.: Synthetisches Programmieren auf dem HP-41 - leicht gemacht. Deutsche Ausgabe von Heinz Dalkowski. 170 Seiten, 40.00 DM (1985) Quick Reference Card beiliegend, ISBN 3-8853 ISBN 3-88538-802-2.

Jarett,K.: Erweiterte Funktionen des HP-41 - leicht gemacht. Deutsche Ausgabe von Heinz Dalkowski. Ca. 200 Seiten, ca. 38.00 DM (1985), ISBN 3-88538-803-0.

Meschede,W.: Plotten und Drucken auf dem HP-41 Thermodrucker. 176 Seiten, 36.00 DM (1985), ISBN 3-88538-805-7.

Stroinski,W.(Hrsg.): Zsfg. der Bedienungshandbücher für das I/O-ROM, IB- und IL-Interface der Rechner HP-83/85 und HP-86/87. Ca. 200 Seiten, ca. 38.00 DM (1985), ISBN 3-88538-806-5.

Wickes,W.C.: Synthetische Programmierung auf dem HP-41C/CV. Deutsche Ausgabe von Heinz Dalkowski. 165 Seiten, 34.00 DM (1983), ISBN 3-88538-800-6.

Bitte richten Sie Ihre Bestellung direkt an den

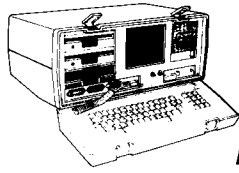
Heldermann Verlag Berlin
Herderstr. 6-7
D - 1000 Berlin 41

Kapitel 13

CCD e.V.

CCD COMPUTERCLUB DEUTSCHLAND E.V.
Postfach 2129, 6242 Kronberg 2

OSBORNE (CP/M)



HP 71/75

MS DOS Rechner

HP 41 C/CV/CX

Der CCD bietet seinen Mitgliedern:

- Die kostenlose Zusendung der Clubzeitschrift Prisma
- Programmierhilfen für Einsteiger und Fortgeschrittene
- Anwenderprogramme aller Art mit Beschreibung, Listing und Balkencode
- Informationen über neue Produkte und Erfahrungsberichte
- Informationen über Hardware -Umbauten und -Ergänzungen
- Einblick in die synthetische Programmierung
- Einblick in die Programmierung in Maschienensprache
- Zugriff auf eine umfangreiche Programmbibliothek zahlreicher Anwendungsgebiete
- Vergünstigungen bei der Anschaffung von Geräten und Zubehör
- Kostenlose Kleininserate in der Clubbörse
- Vieles Andere mehr

Die ständig wachsende Leistungsfähigkeit der Microcomputer erfordert immer tiefer gehende Kenntnisse über Aufbau, Arbeitsweise und Einsatzmöglichkeiten dieser Geräte. Um dies zu erreichen, haben wir 1981 den CCD e.V. als unabhängigen Anwenderclub gegründet und haben zur Zeit über 2500 Mitglieder. Über unsere Clubzeitschrift PRISMA unterhalten wir einen intensiven Erfahrungsaustausch.

Unser Ziel ist, den Benutzern der genannten Systeme beim Einsatz der Geräte Hilfestellung zu leisten und Ihnen zu ermöglichen, die Fähigkeiten dieser Computer voll auszuschöpfen. Weiterhin wollen wir unseren Mitgliedern Kenntnisse vermitteln, die weit über die Hersteller-Informationen hinausgehen.

Die durch PRISMA vermittelte Information kann im Rahmen von Ortsgruppen (bisher in Hamburg, München, Köln, Göttingen und Berlin) besprochen und vervollständigt werden. Falls ein Mitglied zur Lösung eines Problems Hilfe braucht, kann es über die Clubleitung und/oder die Clubzeitschrift Kontakt zu anderen Mitgliedern schaffen. Somit profitiert jedes Mitglied vom Wissen der anderen und kann mit eigenen Ideen und Vorschlägen die Clubarbeit unterstützen.

Die bestehende Osborne-Benutzergruppe erweitert den Informationstausch durch monatlich erscheinende Informationen auf Disketten.

Die MS-DOS-Gruppe ist im Entstehen und bietet Informationen zu den verschiedenen PC's unter diesem Betriebssystem.

Über die Mailbox des CCD werden weitere Informationen ausgetauscht. Dabei haben wir nicht eine weitere Telefonmailbox geschaffen, die mehr Ärger über ständig belegte Anschlüsse hervorruft als den Mitgliedern nützlich zu sein. Wir haben eine Datex P Mailbox mit mehreren Ports, so daß Besetztsituationen so gut wie nie auftreten.

Wie wird man Mitglied im CCD?

Sie senden einen Aufnahmeantrag mit Ihrer kompletten Anschrift an die unten genannte Adresse; Verrechnungsscheck über die Aufnahmegebühr und den ersten Beitrag beilegen. Sie werden dann Mitglied zum Beginn des nächsten Quartals. Die Berechnung der Folgebeiträge erfolgt jedes Jahr zu diesem Quartal.
Bei Minderjährigen muß der Erziehungsberechtigte den Aufnahmeantrag unterschreiben.

Probehefte:
Bei Detlev Bock, Petrikirchstr. 36, 3400 Göttingen kann gegen eine Schutzgebühr von DM 10,00 ein Probeheft angefordert werden. Bitte Verrechnungsscheck oder Bargeld beilegen.

Beiträge:

Aufnahmegebühr:	Firmen	160,00 DM
	Schüler und Studenten	20,00 DM
	alle anderen	40,00 DM
Jahresbeitrag:	Taschenrechner	60,00 DM
	CP/M - Systeme incl. 12 Disketten	180,00 DM
Zuschlag:	Ausland Europa	20,00 DM
	übriges Ausland	50,00 DM

Überweisungsbetrag: _____

Bei Schüler- und Studentenbeitrag bitte Kopie des entsprechenden Ausweises beilegen.

✂ . . . . . .

Probeheft

☐ *Ich möchte ein Probeheft der Zeitschrift Prisma. Die Schutzgebühr von DM 10,-- lege ich als Verrechnungsscheck oder in bar bei.*

Meine Adresse lautet:

Vorname _____

Name _____

Firma _____

Strasse _____

PLZ Ort _____

Unterschrift

CCD-Computerclub Deutschland e.V.
Detlev Bock
Petrikirchstraße 2

3400 Göttingen

✂ . . . . . .

Aufnahmeantrag

☐ *Ich will dem CCD e.V. beitreten und lege die Aufnahmegebühr und einen Jahresbeitrag als Verrechnungsscheck über _____ DM bei.*

Meine Adresse lautet:

Vorname _____

Name _____

Firma _____

Strasse _____

PLZ Ort _____

Unterschrift (bei Minderjährigen Unterschrift der Eltern)

CCD-Computerclub Deutschland e.V.
Postfach 2129

6242 Kronberg 2

Preisliste

Stand: August 1985

für den HP-41

CCD-Modul	DM	350,–
Turbo-Umbau (Erhöhung der Rechnergeschwindigkeit)	DM	120,–
Modulein- bzw. zusammenbau auf die Umbauten gewähren wir 1 Jahr Garantie!	DM	80,–
32k RAM-Box im Kartenlesergehäuse (Speichererweiterungseinheit, Modulsimulator)	DM	963,16
32k EPROM-Box im Kartenlesergehäuse (Festspeichererweiterung); ohne EPROMs	DM	349,12
Assembler-Handbuch (nur zur Verwendung mit der 32k-Speichererweiterung oder dem ERAMCO-MLDL)	DM	74,58*
EPROM-Brennservice auf Anfrage		
ERAMCO-MLDL (Modulsimulator)	DM	875,44

für den HP-71

PRTLEX-File (mit Digitalcassette od. 3½" Diskette) LEX-File zur Druckerunterstützung, für verschiedene Druckertypen erhältlich	DM	70,–
DISP-LEX-File (mit Digitalcassette od. 3½" Diskette) LEX-File zur Unterstützung von Display-Einheiten (z. B. Pack Screen)	DM	70,–
Telefonverwaltungsprogramm	DM	261,40

HHP-Speichererweiterungen im Kartenleserschacht:

HHP-71 M/M 32k RAM (mit Lithiumbatterie ausgestattet)	DM	1 382,46
HHP-71 M/M 64k RAM (mit Lithiumbatterie ausgestattet)	DM	2 432,46
HHP-71 M/M 96k RAM	DM	3 482,46
HHP-71 M/M 32k EPROM	DM	346,49
HHP-71 M/M 32k RAM/EPROM	DM	1 732,46

für den HP-75

Assembler-Handbuch (mit Digitalcassette oder 3½" Diskette)	DM	185,05*

HP-IL

Pac Screen Video-Interface (graphikfähig)	DM	1 350,88
μ-LOGGER W41 (Meß- und Steuerungssystem)	DM	3 260,–
EPSON-Drucker FX-80+	DM	2 087,72
IL-Karte für EPSON FX-80 und FX-80+	DM	491,23
Leercassetten für HP-IL Laufwerk (Mindeststückzahl 5)	à DM	17,37
Netz-/Ladeteil für Diskettenlaufwerk HP-9114	DM	138,60

Bücher

aus dem Vieweg-Verlag	Preis auf Anfrage*	
aus dem Heldermann-Verlag	Preis auf Anfrage*	
PPC-ROM Quick-Reference Guide	DM	21,31*
Synthetic Quick Reference Guide für den HP-41	DM	15,98*

– Preise zzgl. 14 (*7) % MwSt, Porto u. Verpackung –

Falls Ihr HP-Händler unsere Produkte nicht vorrätig hat, richten Sie Ihre Bestellung direkt an:

W&W Software Products GmbH
Im Aehlemaar 20
Postfach 80 01 33
5060 Bergisch Gladbach 2
Telefon: 0 22 02/8 50 68

Lieferung nur gegen Vorkasse oder Nachnahme. Es gelten ausschließlich unsere Liefer- und Zahlungsbedingungen.

Bankverbindungen: Raiffeisenb. Odenthal e. G. (BLZ 370 695 77) Kto.-Nr.: 2 104 646 010
Deutsche Bank AG, Köln (BLZ 370 700 60) Kto.-Nr.: 1 357 508
Postgiroamt Köln (BLZ 370 100 50) Kto.-Nr.: 469 897-504

W&W SOFTWARE PRODUCTS GMBH

Hardwareumbauten

Produktinformationen

für Hewlett Packard Rechner

Aufstieg in die Schwergewichtsklasse

NEU!

32 kByte-Speichererweiterung

Diese Speichererweiterung in den Abmessungen des HP-Kartenlesers bietet nicht nur ein Maximum an Zusatzspeicher, sondern gewährleistet auch die volle Portabilität des HP-41 Systems. Die Box wird wie ein Kartenleser auf Ihren HP-41 aufgesteckt. Mit dem eingebauten 4 kByte Betriebssystem können Sie nun 28 kByte mit Ihren eigenen Programmen beschreiben. Wenn Sie im Besitz eines ERAMCO-MILDs sind, stehen Ihnen die vollen 32 kByte zur Verfügung. Die Programme sind dann wie z. B. im Mathe-Modul sofort lauffähig. Durch eine eingebaute Batterie bleibt der Inhalt dieser Speichererweiterung auch erhalten, wenn man sie vom Rechner abzieht. Natürlich besteht auch die Möglichkeit, fertige „Module" auf Band abzuspeichern und zurückzuladen. Im gleichen Gehäuse gibt es auch eine 32 kByte EPROM-BOX.

Neue 32 kByte EPROM-Box für den HP-41

Für diese Entwicklung gelten im wesentlichen die bei der 32 kByte Speichererweiterung genannten Ausführungen. Bei dieser Erweiterung handelt es sich jedoch nicht um einen veränderbaren Speicher, sondern um eine ROM-Erweiterung. Die EPROMs müssen speziell mit der Kundensoftware „gebrannt" werden. Die Softwareentwicklung kann der Kunde mit der 32 kByte Speichererweiterung selbst vornehmen.

HP-IL Interfacekarte für EPSON-Drucker

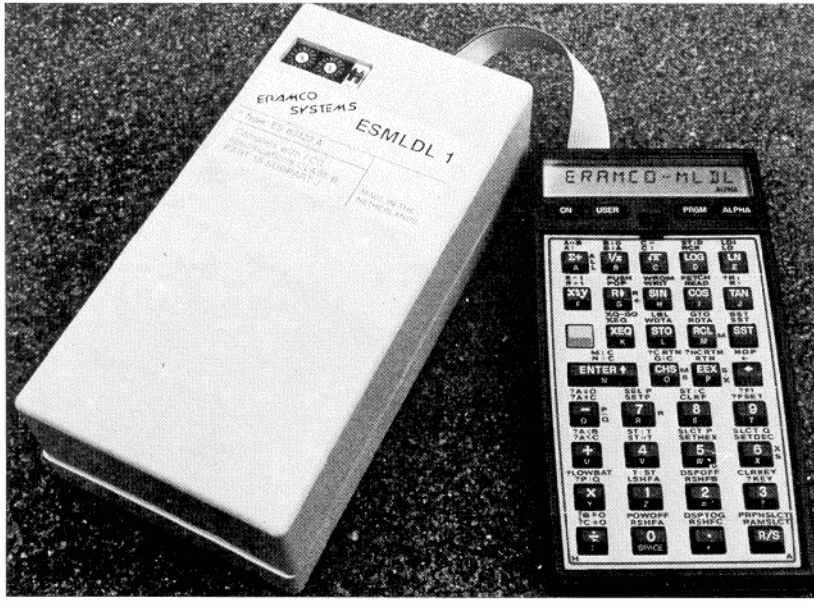
Viele Kunden und Besitzer eines Personal Computers möchten gerne ihren schon vorhandenen EPSON-Drucker auch mit ihrem portablen System betreiben. Hierzu gibt es jetzt eine preiswerte Möglichkeit. Unsere HP-IL-EPSON-Karte wird einfach in den Drucker eingesteckt, und schon kann man ihn mit einem IL-Controller ansteuern. Die ID dieser Karte lautet „W&W FX80", die AID ist 46, also eine Drucker-AID. Mit dieser Erweiterungskarte kann man nun die zusätzlichen Möglichkeiten der EPSON-Drucker mit den Vorteilen des transportablen HP-IL-Systems verbinden.

Netzteil für das Diskettenlaufwerk HP-9114

Ein von vielen Kunden bemängelter Schwachpunkt des neuen HP-9114 Diskettenlaufwerkes ist seine unzureichende Stromversorgung. Da das mitgelieferte Ladegerät nur 400 mA liefert, das Laufwerk aber bis zu 2 Ampere verbraucht, ist der eingebaute Akku nach ca. einer Stunde intensiven Betriebes leer. Eine 14-stündige Ladezeit ist in vielen Fällen unzumutbar. Durch die periodischen Ent- und Aufladungen des Bleiakkus ist die geschätzte Lebenserwartung dieses Teils nicht größer als maximal zwei Jahre. Unser neues Netzteil behebt alle diese Nachteile und verhilft einem an sich excellenten Gerät zu seiner vollen Funktion.

Einbau:
Um unser Netzteil betreiben zu können, wird außen am Akkugehäuse eine zusätzliche Buchse eingebaut (ein Loch bohren oder schneiden). Zwei Drähte, die schon an dieser Buchse angelötet sind, werden an die Akku-Klemmen angesteckt. Dieser Einbau kann auf Wunsch des Kunden von uns vorgenommen werden; allerdings ist die HP-Garantie erloschen. Der Akku ist nun zum Betrieb mit dem neuen Netzteil vorbereitet.

Netzteil Spezifikationen:
Abmessungen des Netzteils: 20*9,8*7 cm.
Stromversorgung durch Netzkabel.
Ladekabel mit Stecker.
Das Netzteil selbst ist ein mehrstufiges Schaltnetzteil, welches speziell auf den Bleiakku des HP-9114 Diskettenlaufwerks eingestellt ist. Der maximale Ladestrom beträgt 1,25 Ampere, was dem Durchschnittsverbrauch des Laufwerks entspricht. Somit wird der Akku im Pufferbetrieb gefahren und einem Dauerbetrieb steht nichts mehr im Wege. Da der Akku nicht mehr den starken Belastungen ausgesetzt ist, erhöht sich natürlich auch seine Lebenserwartung. Ein zusätzlicher Pluspunkt ist es außerdem, daß mit dieser Lösung die Portabilität des Systems in keiner Weise eingeschränkt ist.



Das ERAMCO-MLDL ist eine Erweiterung für Ihr HP-41-System.
Mit diesem Zusatzgerät erweitern Sie Ihren Rechner um einen 24k-EPROM-Teil und einen 8k-RAM-Teil. War Ihnen bisher der Speicherplatz Ihres HP-41 zu klein, so haben Sie mit unserem ERAMCO-MLDL insgesamt 32k an Speicherplatz mehr zur Verfügung (entspricht ca. 4570 Programmregistern!). Da das MLDL – wie auch unsere 32k-Speichererweiterung – eine Art MODULSIMULATOR ist, sind alle in ihm gespeicherten Programme sofort zugreifbar (wie zum Beispiel beim Mathematik-Modul). Weiterhin kann man das ERAMCO-MLDL in Maschinensprache programmieren. Programmierhilfe-EPROMS mit entsprechender Dokumentation sind im Lieferumfang enthalten. Selbstverständlich bieten wir Ihnen auch unseren preisgünstigen EPROM-Brennservice an.

Mit diesem Gerät haben Sie die Leistungsfähigkeit Ihres HP-41 voll ausgeschöpft.

Zum Lieferumfang gehören:
Eine englische Bedienungsanleitung (leicht verständlich) sowie das MLDL-OPERATING SYSTEM EPROM mit Bedienungshandbuch und ein 8k-EPROM-Satz zur Maschinensprache-Programmierung.

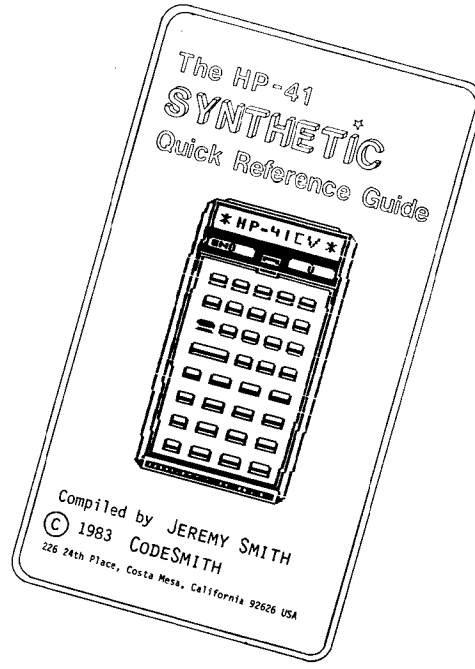
MEHR TEXT.
MEHR GRAFIK.
ADVANCED

Es gibt viele Gründe, sich für Pac Screen als Video Interface für die HP-Rechner-Serie zu entscheiden. Beim **Advanced Pac Screen** sind es noch einige gute Gründe mehr geworden.

Erstens: Advanced Pac Screen ist klein. Mit nur guten 400 cm^3 ist Advanced Pac Screen vollgepackt mit modernster Elektronik und trotzdem so klein, daß es neben dem Bildschirm kaum auffällt. Auffällig sind aber die unglaublichen Möglichkeiten, die in ihm stecken.

Nachschlageheft zur synthetischen Programmierung

Dieses Heft ist unentbehrlich für alle Anwender der synthetischen Programmierung. Es enthält Bytetafeln, Modullisten, Barcodes und viele nützliche Tabellen.



HP-75 Assembler Handbuch

Wollen Sie Ihrem HP-75 die letzten Geheimnisse entlocken?

Alles Notwendige hierfür finden Sie in unserem neuen HP-75 ASSEMBLER Handbuch. Geschrieben von Michael Hartmann, einem Spezialisten für diesen Rechner, enthält es nicht nur Informationen über den internen Aufbau des Rechners, sondern mit den auf Digitalkassette oder Diskette mitgelieferten Programmen und LEX-Files finden Sie einen einfachen Einstieg in die ASSEMBLER-Programmierung.

Im Lieferumfang sind enthalten:
DIN A4 Handbuch im Ringbuchordner, eine Digitalkassette oder eine Programmdiskette
Bestellnummer:
WW85751K Buch mit Digitalkassette
WW85751D Buch mit 3½" Programmdiskette

Hardwareumbauten

Folgende Hardwareumbauten sind bei einem HP-41 möglich und können bei der Fa. W&W Software Products GmbH, Bergisch Gladbach, durchgeführt werden:

1) „TURBO"-Umbau
   Durch den sogenannten TURBO-Umbau wird die Rechengeschwindigkeit des HP-41 auf das 1,6 bis 1,8-fache der normalen Geschwindigkeit erhöht. Dadurch laufen alle Programme (auch solche in Maschinensprache, wie z. B. „PACK") schneller. Allerdings kann der Kartenleser im TURBO-Gang keine Magnetkarten aufzeichnen. Für diesen Fall wird einfach mittels eines hochwertigen Microschalters, der an der Außenseite des Rechners eingelassen ist, wieder auf Normalgeschwindigkeit umgeschaltet. Das Einlesen von Magnetkarten ist auch im TURBO-Gang uneingeschränkt möglich. Weiterhin läßt sich der Rechner schneller programmieren, da auch die Tastendrücke schneller angenommen werden.

2) Moduleinbau in den Rechner
   Der Einbau von Modulen in den Rechner ist die eleganteste Methode, die Ports für weitere Module freizumachen und gleichzeitig die Kapazität des Rechners auf das Maximum zu erhöhen. Es können bis zu 7 Module in den Rechner eingebaut werden. Hierbei ist zu beachten, daß alle ROM-Module (wie Mathematik-, Games-, CCD-Modul usw.) einen Port belegen, egal ob sie eingebaut sind oder nicht. Keinen Port belegen nach einem Einbau alle Speichermodule, das TIME-Modul sowie das IL- bzw. Druckermodul. Daher empfiehlt es sich auch, den Rechner nur bis zu maximal folgender Stufe auszubauen:
   - Einbau des TIME-Moduls. Es ist nichts zu beachten.
   - Einbau des X-Functions- und der X-Memory-Module. In Port 3 des Rechners darf nur noch der Drucker- bzw. IL-Stecker eingesetzt werden, da auf diesen Port das X-Functions-Modul geschaltet wird! Ansonsten ist nichts zu beachten.
     Gegen einen geringen Aufpreis kann zusätzlich ein Microschalter eingebaut werden, mit dem der X-Memory-Bereich abgeschaltet werden kann, ohne daß die Speicherinhalte verlorengehen (Schutz vor „MEMORY LOST").

Alle Module werden mittels einer hochwertigen Folienplatine eingebaut, die einen optimalen Kontakt gewährleistet und der hohen HP-Qualität entspricht.

3) Doppelmodulumbau
Es werden zwei Module in das Gehäuse eines einzigen eingebaut. Hierbei gelten dieselben Einschränkungen wie beim Moduleinbau (s. o.). Beim Zusammenbau von zwei ROM-Modulen ist bei einem Auftrag anzugeben, welchen Steckplatz das zweite ROM belegen soll.

Auf alle Umbauten erhalten Sie ein Jahr Garantie!

Mit diesen neuen Speichererweiterungen können Sie den Hauptspeicher Ihres HP-71 bis auf die *siebenfache Kapazität* ausbauen. Da diese Erweiterungen einfach in den Kartenleserport eingesteckt werden, bleiben alle Modulports für Software-Module frei! Einmal eingesteckt, wird der Zusatzspeicher zum festen Bestandteil des Hauptspeichers und steht sofort für die Programmierung zur Verfügung. Die HHP-71 M/M 32k RAM und die HHP-71 M/M 64k RAM haben eine Lithium-Batterie eingebaut, die das Auswechseln dieser Erweiterungen *ohne Datenverlust* erlauben. Die EPROM-Erweiterungen sind eine preiswerte Alternative zu Kundenmodulen, wenn nur kleine Stückzahlen gebraucht werden oder die Software regelmäßig geändert werden muß. EPROMs können neu gebrannt werden und sind schnell und preiswert auszutauschen. Für weitere Informationen rufen Sie uns bitte an!

W&W SOFTWARE PRODUCTS GMBH

W&W Software Products GmbH
Im Ashlemaar 20, Postfach 800 133
5060 Bergisch Gladbach 2, Tel.: 02202/65068