Bibliotheksprogrammtechnik am ER 56

Inhaltsverzeichnis

Recheninstitut der Technischen Hochschule Stuttgart

Juni 1964

Bearbeiter: Dirlewanger

Bibliotheksprogrammtechnik

TERMINOLOGIE

Wird für eine vorgesehene Aufgabe ein Programm gefertigt, dann nennt man dieses Programm Gesamtprogramm. Es besteht im allgemeinen aus kleineren zusammengehörenden Befehlsgruppen. Man nennt sie Teilprogramme. Sie sind die Übersetzung eines gewissen Sachverhalts in die Maschinensprache. Man könnte nun die Teilprogramme so aufbauen, daß sie, aneinandergefügt, sofort das Gesamtprogramm ergeben.

Es ist im allgemeinen jedoch zweckmäßiger die Teilprogramme als für sich allein schon selbständig verwendbare Programme aufzubauen und das Zusammenspiel durch ein sogenanntes Hauptprogramm zu steuern. Ein Beispiel ist eine Berechnung von

$$y = \frac{x + \sin x}{x^2 + \sin^2 x}$$

Hier wird man die Berechnung von sin x, $\sin^2 x$ und x^2 Teilprogrammen zuweisen.

Sinnvollerweise wird man nun das Teilprogramm zur Berechnung von sin x so gestalten, daß es auch zur Berechnung von $\sin^2 x$ verwendet werden kann. Das Teilprogramm zur Berechnung von $\sin^2 x$ hat dann nur noch das Quadrat zu bilden.

Das Teilprogramm für sin x ist dann so organisiert, daß es bei fester Stellung im Speicher innerhalb des Gesamtprogrammes mehrfach verwendet werden kann. Ein solches Programm nennt man Unterprogramm.

ABSOLUTE UND RELATIVE ADRESSIERUNG

Übersetzt man ein Problem z.B. einen Algorithmus in die Maschinensprache in ihrer einfachsten Form, dann wird man dies so tun, daß man die Befehle in einer Reihe aufschreibt und dabei dem ersten Befehl und damit auch dem folgenden ganz feste Plätze innerhalb des Speichers zuordnet. Die Daten wird man ebenfalls in bestimmte Zellen schreiben.

Die Befehle des Programms beziehen sich nun auf Zelleninhalte (d.h. Zahlen oder andere Befehle), deren Position immer durch die Adresse der Zelle festgelegt ist.

Ein Beispiel sieht so aus: (der Ausdruck wurde zur Demonstration absichtlich nicht vereinfacht)

Berechne $y_i = \frac{x_i a + x_i b}{x_i^2}$ für alle Werte x_i, solange $y_i \geq Y$ ist

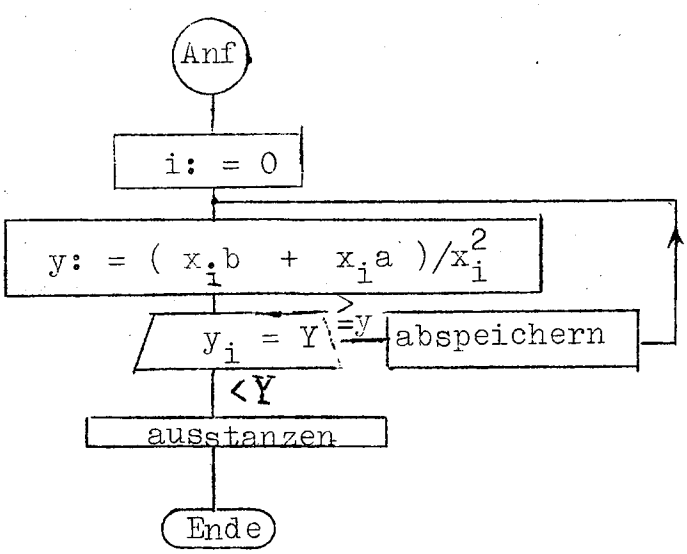
Die Rechnung soll in Gleitkomma durchgeführt werden, die Werte y_i sollen ausgestanzt werden.

Das Programm stehe ab Zelle 100, notwendige Hilfszellen sollen an das Programmende angeschlossen werden. Die Daten seien wie folgt abgespeichert:

(1000) = a
(1002) = b
(1004) = Y
(1006) = x_0
(1008) = x_1
. .
. .

Das Programm sieht dann so aus:

Flußdiagramm



Zelle	Befehl			Bemerkung
0100	2000	0	19	GK
1	0000	1	91	$0 \rightarrow I_1$
2	1000	0	31	
3	1100	1	37	$x_i a \rightarrow Hz$
4	0122	0	32	
5	1002	0	31	
6	1100	1	37	$x_i\ a + x_i\ b \rightarrow Hz$
7	0122	0	35	
8	0122	0	32	
9	1100	0	31	$x_i^2 \rightarrow A$
0110	1100	0	37	
1	0122	0	39	$y_i := (Hz)/ x_i^2$
2	1004	0	28	y_i vergl. Y
3	0115	0	13	Sprung bei $>$
4	0118	0	10	
5	0124	1	32	y_i abspeichern
6	0002	1	93	$\langle I_1 \rangle + 2 \rightarrow I_1$
7	0102	0	10	
8	7000	0	19	EIN $\Rightarrow \omega$A - M
9	0124	1	32	$\omega\omega$ abspeichern
0120	0124	0	69	stanzen
1	9000	0	19	Stop
2	0999	9	99	Hz
3	9999	9	99	Hilfszellen zum Aufbewahren der y_i
				⋮

Liest man nun dieses Programm nicht ab Zelle 100 ein, sondern ab einer anderen Zelle, dann funktioniert es nicht mehr, denn jeder Befehl bezieht sich auf eine ganz bestimmte Adresse, völlig unabhängig davon in welcher Zelle er selbst steht. Man nennt ein solches Programm absolut adressiert.

Wollte man es ab einer anderen Zelle einlesen, dann müßte man die Adreßteile aller Befehle ändern, die sich z.B. auf Hilfszellen beziehen, und auch die Adreßteile der Sprungbefehle, weil ja nun die Hilfszellen und die angesprungenen Zellen ganz andere Adressen haben.

Es gibt aber doch einen Weg ein Programm zu erhalten, das unabhängig, von der Zelle ab der man es einliest, immer funktioniert. Sieht man sich nämlich an wie man die Adreßteile ändern muß, um ab einer beliebigen Zelle n einlesen zu können, dann zeigt sich, daß sie gerade (bei unserem Beispiel) um n-100 geändert werden müssen (ab Zelle 100 steht unser Programm in der absoluten Adressierung). Mit anderen Worten: Für die Befehle des Programms, kommt es nicht darauf an, Zellen bestimmter Adresse zu erreichen, sondern vielmehr z.B. den Inhalt einer Zelle zu erhalten, die etwa 10 Zellen weiter unten steht oder z.B. auf einen Befehl zu springen, der 17 Zellen weiter oben steht usw.

Es spielen also die relativen Abstände eine Rolle und wenn man eine Möglichkeit findet diese relativen Abstände anstatt der absoluten Adressen in die Befehle einzuführen, dann ist das Programm unabhängig davon ab welcher Zelle es eingelesen wird.
Man nennt es dann "relativ" adressiert.

Wir wollen nun unser absolut adressiertes Programm nocheinmal aufschreiben, jedoch in die Adreßteile der Befehle, die sich auf andere Befehle des Programms beziehen (z.B. Sprungbefehle) oder auf Inhalte von Hilfszellen nur die oben besprochenen "relativen Abstände" eintragen. Also für eine Hilfszelle die 7 Zellen unter dem betreffenden Befehl steht 0007, oder wenn sie 13 Zellen weiter oben steht 9987 (=Komplement zu 10000). Dieses Programm funktioniert offensichtlich dann sofort, wenn wir zum Adreßteil aller Befehle, die als Adresse einen relativen Abstand haben, die Adresse der Zelle addieren in der sie stehen. Mit anderen Worten wir müssen an all diesen Befehlen die Adresse modifizieren und zwar mit einem Indexregister, das immer gerade die Adresse desjenigen Befehls enthält, der gerade ausgeführt wird. Am ER 56 existiert ein solches Indexregister: Es ist das Befehlsadressenregister (auch Befehlsfolgezähler genannt), das hier als Indexregister 9 der Programmierung zugänglich ist. Wir müssen also nur die zu relativierenden Befehle mit I_9 indizieren und erhalten damit ein relativ adressiertes Programm,

(Verschiebung der relativen Adressen um -1 siehe unten).

Dieses Verfahren der Relativierung des Programms durch Modifikation der "relativen" Adresse des Befehls mit der Adresse der Zelle in der er steht wird allgemein angewandt, nur die Ausführung (die am ER 56 mit I_9 sehr einfach ist) hängt vom Aufbau und der Struktur des jeweiligen Rechenautomaten ab.

BIBLIOTHEKSPROGRAMME

Die relative Adressierung gibt die Möglichkeit ein Programm herzustellen, das immer funktioniert, ganz gleich an welche Stelle man es im Speicher stellt. Damit kann man eine sogenannte Programmbibliothek aufbauen, d.h., eine Bibliothek schaffen, die universell verwendbare Programme für oft gebrauchte Probleme enthält, z.B.f die Berechnung von Winkelfunktionen. Um die universelle Verwendbarkeit zu garantieren, müssen für alle Bibliotheksprogramme (neben der relativen Adressierung) noch eine Reihe von Vereinbarungen getroffen werden, die es dem Bibliotheksprogramm ermöglichen, die Daten, die es braucht, von einem beliebigen Ort im Kernspeicher vom Hauptprogramm zu übernehmen. Sie sind wie folgt festgelegt:

1) Die Adressen der innerhalb des Bibliotheksprogramms auftretenden Befehle sind (mittels I_9) zu relativieren, um es unabhängig von der Stellung im Speicher zu machen.
2) Der Sprung aus dem übergeordneten Programm in das Bibliotheksprogramm erfolgt auf den ersten Befehl des Bibliotheksprogrammes. Die für das Bibliotheksprogramm notwendigen Daten oder deren Adressen, sind in den auf den Absprungbefehl im Hauptprogramm folgenden Zellen gespeichert. Den Rücksprung ins Oberprogramm organisiert man mittels des Indexregisters 0. Man nützt dabei die Tatsache aus, daß I_0 nach einem Sprungbefehl die um 1 erhöhte Adresse dieses Sprungbefehls enthält. Nach dem Einsprung ins Bibliotheksprogramm enthält also I_0 die um 1 erhöhte Absprungadresse. Man braucht nun I_0 nur um die Zahl der durch die Daten für das Bibliotheksprogramm (oder deren Adressen) belegten Zellen zu erhöhen und hat damit die Adresse auf die der Rücksprung erfolgen muß. (Man speichert diese dann im allgemeinen gleich mit Befehl 89 in den Adreßteil des Rücksprungbefehls ab, der am Ende des Bibliotheksprogramms steht).
3) Man ist übereingekommen, daß Bibliotheksprogramme nur die Indexregister I_6, I_7 und I_8 verwenden. Werden weitere Indexregister verwendet, dann müssen ihre Inhalte am Anfang des Bibliotheksprogramms in Hilfszellen gebracht und am Schluß wieder zurückgebracht werden. (Analog gilt natürlich für die Indexregister I_6, I_7 und I_8 folgendes: Enthalten diese Indexregister beim Einsprung ins Bibliotheksprogramm Informationen, die das Hauptprogramm später wieder braucht, dann müssen diese noch vor dem Einsprung ins Bibliotheksprogramm in Hilfszellen gebracht werden, um sie nach Ablauf des Bibliotheksprogramms noch zur Verfügung zu haben.)

Bei der Indizierung mit I_9 ist nun eine Besonderheit dieses Indexregisters zu beachten: Sein Inhalt wird sofort beim Beginn der Durchführung eines Befehls (also vor jeder anderen Teiloperation) um 1 erhöht. Zu dem Zeitpunkt, zu dem (bei indizierten Befehlen, d.h., wenn der Indexteil ≠ 0 ist), die Adressenumrechnung ausgeführt wird, enthält also I_9 bereits die um 1 erhöhte Adresse desjenigen Befehls, der gerade ausgeführt wird. Man berücksichtigt dies, indem man in die Adreßteile der zu relativierenden Befehle nicht genau den relativen Abstand schreibt, sondern: relativer Abstand minus 1. Will man z.B. den Inhalt einer Hilfszelle in den Akkumulator bringen, die 7 Zellen weiter unten steht, dann ist der Befehl 0006931 (und nicht 0007 931) oder will man auf einen Befehl springen (z.B. unbedingter Sprung), der 13 Zellen weiter oben steht, dann ist der Befehl 9986 910 (und 9987 9 10).

Das vorige Programm mit relativer Adressierung hat damit folgende Befehlsliste (die Daten sollen in denselben Zellen wie früher stehen):

rel. Adresse	Befehl	
0000	0000 9 10	}〉HZ〈→I_0
1	0026 0 93	}
2	0017 9 89	〉HZ〈→Adreßteil von Bef. in.rel.Pos.20
3	0020 9 89	〉HZ〈→ " " " " " " 24
4	0020 9 89	〉HZ〈→ " " " " " " 25
5	2000 0 19	
6	0000 1 91	
7	1000 0 31	
8	1100 1 37	
9	0017 9 32	
0010	1002 0 31	
1	1100 1 37	Programm wie früher, jedoch sind
2	0014 9 35	die Adressen der Befehle, die HZ
3	0013 9 32	verwenden und der Sprungbefehle mit
4	1100 0 31	I_9 relativiert.
5	1100 0 37	
6	0010 9 39	
7	1004 0 28	
8	0001 9 13	
9	0003 9 10	
0020	0000 1 32	
1	0002 1 93	
2	9984 9 10	
3	7000 0 19	
4	0000 1 32	
5	0000 0 69	
6	9000 0 19	
7	9999 9 99	}HZ
8	9999 9 99	}
⋮	⋮ ⋮	}Hilfszellen f. y_i

Dieses rel. adressierte Programm ist so entstanden, daß ein zunächst in absoluter Adressierung aufgeschriebenes Programm nachträglich relativiert wurde. Geht man jedoch gleich beim Programmentwurf von rel. Adressierung aus, dann entsteht selbstverständlich eine elegantere Lösung.

Ähnlich wie mit den Indexregistern verfährt man mit den Programmerkern:
Die Programmerker PM6...PM9 bleiben für das Bibliotheksprogramm allgemein reserviert; verwendet es weitere, dann muß es deren Inhalte in Hilfszellen aufbewahren und am Schluß wieder einschreiben.

EINIGE MÖGLICHKEITEN DER PARAMETERÜBERNAHME UND DER ORGANISATION DES RÜCKSPRUNGS.

A) Treten nur 2 Parameter (x,y) auf, die das Bibliotheksprogramm braucht, dann kann man sie im Akkumulator und Multiplikatorregister bereitstellen. Der Rücksprung wird dann folgendermaßen organisiert:

Hauptprogramm

Pos.	Bef.		Bemerkung
⋮			
.	....	...	(A)=x,(M)=y
.	....	10	Sprung zum BP
RSA	....	..	Rücksprung vom BP her
⋮	⋮	⋮	⋮

Bibliotheksprogramm

rel.Pos.	Bef.			Bemerkung
0000	(P_1-1)	9	89	Dieser Befehl transportiert die RSA in den Adreßteil des Rücksprungbefehls in der Position $P_{rü}$
⋮	⋮	⋮	⋮	
$P_{rü}$	0000	0	10	Rücksprung ins Hauptprogramm

B) Allgemeiner Fall mit beliebig vielen Parametern. Man kann dann entweder die Parameter selbst unmittelbar nach dem Absprungbefehl speichern oder die Adressen der Zellen in denen sie stehen. Die verschiedenen Möglichkeiten werden im folgenden am Beispiel von 3 Parametern (x,y,z) erklärt.

I)Die Parameter stehen hinter dem Absprungbefehl im Hauptprogramm. Das Hauptprogramm sieht dann so aus:

Hauptprogramm

Pos.	Bef.			Bemerkung
⋮	⋮	⋮	⋮	⋮
	....	.	10	Sprung ins BP
				} x
				} y
				} z
RSA	....	.	..	Rücksprung vom BP
⋮	⋮	⋮	⋮	

BP

rel. Pos.	Bef.			Bemerkung
0000	0000	9	89	} $\langle I_o \rangle$ =)x(
0001	0000	0	31	} x → P_1
.	(P_1-1)	9	32	
.	0002	0	93	)y(:=)x(+ 2
.	0000	9	89	} y → P_2
.	0000	0	31	
.	(P_2-1)	9	32	
.	0002	0	93	)Z(:=)y(+ 2
.	0000	9	89	} Z → P_3
.	0000	0	31	
.	(P_3-1)	9	32	
.	0002	0	93	RSA :=)Z(+ 2
.	($P_{rü}$-1)	9	89	RSA → Adreßteil des Rücksprungbefehls
⋮	⋮	⋮	⋮	⋮
$P_{rü}$	0000	0	10	Rücksprung ins HP

dieser Text gehört hier her

a) Übernahme der Zahlen ins BP.
Man sieht im Bibliotheksprogramm 3 (Hilfs-)Zellen mit den rel. Positionen P_1,P_2,P_3 vor, in die man die Parameter einspeichert. Der Rücksprung ist ähnlich wie bei A) organisiert.

1. Version (man verwendet nur I_o)

2. Version
(man verwendet noch I_7 dazu und kommt dann mit weniger Befehlen aus)

rel. Pos.	Bef.			Bem.
0000	0000	9	89	>x< → I_7
0001	0000	7	91	
.	0000	7	31	x → P_1
.	(P_1-1)	9	32	
.	0002	7	31	y → P_2
.	(P_2-1)	9	32	
.	0004	7	31	z → P_3
.	(P_3-1)	9	32	
.	0006	0	93	RSA:= >x< + 6
.	($P_{rü}$-1)	9	89	RSA→ Adreßteil d. Rücksprungbef.
⋮	⋮	⋮	⋮	⋮
$P_{rü}$	0000	0	10	Rücksprung ins Hauptprogramm.

b) Übernahme der Adressen der Parameter in Adreßstellen von Befehlen im BP.
In den relativen Positionen P_1, P_2, P_3 des BP stehen Befehle, die die Parameter x, y, z verarbeiten sollen und man transportiert die Adressen der Parameter (die Parameter stehen ja in den Zellen nach dem Absprungbefehl) direkt in die Adreßteile der Befehle in P_1, P_2, P_3.

rel. Pos.	Bef.			Bemerkung
0000	(P_1-1)	9	89	>x< -> Adreßteil des Bef. in Pos.1
0001	0002	0	93	>y< := >x< + 2
.	(P_2-1)	9	89	>y< -> Adreßteil des Bef. in Pos.2
.	0002	0	93	>z< := >y< + 2
.	(P_3-1)	9	89	>z< -> Adreßteil des Bef. in Pos.3
.	0002	0	93	RSA := >z< + 2
.	($P_{rü}$-1)	9	89	RSA -> Adreßteil d. Rücksprungbef.
⋮	⋮	⋮	⋮	⋮
$P_{rü}$	0000	0	10	Rücksprung

II) Die Adressen (>x<, >y<, >z<) der im BP zu verarbeitenden Parameter stehen hinter dem Absprungbefehl im Hauptprogramm.
Man gibt in diesem Fall nicht direkt die Parameter nach dem Absprungbefehl an, sondern die Adressen unter denen sie (vom Hauptprogramm her) stehen und zwar jeweils in den ersten 4 der der 7 Stellen der Zellen. Das Hauptprogramm sieht dann so aus:

Pos.	Bef.			Bemerkung
⋮	⋮	⋮	⋮	
.	...	.	10	Sprung ins BP
.	>x<			
.	>y<			
.	>z<			
RSA	...	.	..	Rücksprung vom BP
⋮	⋮	⋮	⋮	⋮

a) Übernahme der Adressen der Parameter in Adressenstellen im BP.

Das Verfahren ist analog dem von B, I, b). In den relativen Postitionen P_1, P_2, P_3 des BP stehen wieder Befehle, die die Parameter x, y, z verarbeiten sollen und man transportiert die Adressen der Parameter in die Adreßteile der sie verarbeitenden Befehle. Das Verfahren benötigt jedoch einige Befehle mehr als in B, I, b), da dort nach dem Einsprung ins BP I_0 bereits die Adresse von x enthält, während hier I_0 nach dem Einsprung die Adresse der Adresse von x enthält.

1. Version (man verwendet nur I_0 und I_8)

rel. Pos.	Befehl			
0000	0000	9	89	Adreßteil d. Bef. in rel.Pos.1:=>>x<<
0001	0000	0	80	< >>x<< >→I_8, d.h. >x< → I_8
.	(P_1-1)	9	86	>x< → Adreßteil d. Bef. in Pos P_1
.	0001	0	93	>>y<< := >>x<< + 1
.	0000	9	89	>y< → Adreßteil d. Bef. in Pos. P_2
.	0000	0	80	
.	(P_2-1)	9	86	
.	0001	0	93	>>z<< := >>y<< + 1
.	0000	9	81	>z< → Adreßteil d. Bef. in Pos. P_3
.	0000	0	80	
.	(P_3-1)	9	86	
.	0001	0	93	RSA := >>z<< + 1
.	($P_{rü}$-1)	9	89	RSA → Adreßteil d. Rücksprungbef.
⋮	⋮	⋮	⋮	⋮
$P_{rü}$	0000	0	10	Rücksprung

2) Version (Man verwendet neben I_0 und I_8 ein weiteres Indexregister; hier z.B. I_7. Man kommt dann mit weniger Befehlen aus).

rel. Pos.	Befehle			Bemerkungen
0000	0000	9	89	⟩⟩x⟨⟨ → Adreßteil d.Bef.i.d.r.Pos.1
0001	0000	7	91	⟩⟩x⟨⟨ → I_7
.	0000	7	80	⟨⟩⟩x⟨⟨⟩ → I_8, d.h. ⟩x⟨ → I_8
.	(P_1-1)	9	86	⟩x⟨ → Adreßteil d.Bef. in rel.Pos.P_1
.	0001	7	80	⟩y⟨ → Adreßteil d.Bef.i.rel.Pos. P_2
.	(P_2-1)	9	86	
.	0002	7	80	⟩Z⟨ → Adreßteil d.Bef. in rel.Pos.P_3
.	(P_3-1)	1	86	
.	0003	0	93	RSA → I_0
.	($P_{rü}$-1)	9	89	RSA → Adreßteil des Rücksprungbef.
.	⋮	⋮	⋮	⋮
$P_{rü}$	0000	0	10	Rücksprung

b) Übernahme der Positionsangaben aus den KS-Zellen in Indexregister.

Bringt man nach dem Einsprung ins Bibliotheksprogramm die Adressen der Parameter, die das Oberprogramm (in den Kernspeicherzellen unterhalb des Absprungbefehls) bereitstellt, gleich in Indexregister, dann kann man die Parameter durch einfaches Indizieren der Befehle, die sie verwenden, erreichen:

rel. Pos.	Befehle			Bemerkungen
0000	0000	9	89	⟩⟩x⟨⟨ → Adreßteil d.Bef.inrel.Pos. 2
0001	0000	0	80	⟨⟩⟩x⟨⟨⟩ → I_8,d.h. ⟩x⟨ → I_8
.	0000	6	99	⟩x⟨ → I_6
.	0001	0	93	⟩y⟨ := ⟩x⟨ +1
.	0000	9	89	⟩y⟨ → I_7
.	0000	0	80	
.	0000	7	99	
.	0001	0	93	⟩Z⟨ := ⟩y⟨ + 1
.	0000	9	89	⟩Z⟨ → I_8
.	0000	0	80	
.	0001	0	93	RSA := ⟩Z⟨ + 1
.	$P_{rü}$-1	9	89	RSA → Adreßteil des Rücksprungbefehls
.	⋮	⋮	⋮	⋮
$P_{rü}$	0000	0	10	Rücksprung

III. Die Positionen der im BP zu verarbeitenden Zahlen werden im Hauptprogramm nach Indexregistern gebracht.

Diese Möglichkeit sei nur angedeutet. Es wird im Prinzip das unter B) II. b) aufgeführte Verfahren angewendet, jedoch mit dem Unterschied, daß die Positionen der Parameter im Oberprogramm nicht in Kernspeicherzellen gebracht werden, sondern in Indexregister, aus denen sie sofort vom BP übernommen werden können (durch Indizieren mit diesen Indexregistern). Im BP braucht dann nur noch der Rücksprung organisiert zu werden:

rel.Pos.	Befehle			Bemerkungen
0000	$(P_{rü}-1)$	9	89	RSA → Adreßteil des Rücksprungbefehls
.	.		.	.
.	.		.	.
.	.		.	.
.	.		.	.
.	.		.	.
.	.		.	.
.	.		.	.
.	.		.	.
.	.		.	.
$P_{rü}$	0000	0	10	Rücksprung

Der Rücksprung erfolgt auf die Zelle unterhalb des Absprungbefehls im Hauptprogramm.

EIN BEISPIEL FÜR EIN BIBLIOTHEKSPROGRAMM
(Multiplikation zweier Matrizen)

Es werde die Matrix $\mathfrak{A} = (a_{ik})$ $\quad i=1....m$ (Zeilen), $k=1....n$ (Spalten)

mit der Matrix $\mathfrak{B} = (b_{ik})$ $\quad i=1....n$ (Zeilen), $k=1....p$ (Spalten)

multipliziert. Das Ergebnis sei die Matrix

$$\mathfrak{C} = (c_{ik}) \qquad i=1....m \text{ (Zeilen)}, \; k=1....p \text{ (Spalten)}$$

Die Elemente c_{ik} berechnen sich nach der Formel

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

Man erhält also die ganze Matrix, wenn man für i=1, 2, 3,...m den Index k jeweils die Werte 1...p durchlaufen läßt.
Die Elemente der Matrix $\mathfrak{A}$ seien ab $>a_{11}<$ fortlaufend gespeichert und zwar in folgender Reihenfolge:

$$a_{11}, a_{12},...a_{1n}; a_{21}, a_{22},...a_{2n};.....; a_{m1}, a_{m2},...a_{mn}.$$

Die der Matrix $\mathfrak{B}$ seien analog ab $>b_{11}<$ fortlaufend gespeichert.
Die Matrixelemente der letzten Spalte von $\mathfrak{A}$ und $\mathfrak{B}$ seien mit Q-Markierung versehen. In dem auf $>a_{mn}<$ bzw. $>b_{np}<$ folgenden Speicherplatz ist ein ω -Doppelwort enthalten. Das Bibliotheksprogramm soll die Elemente der Matrix $\mathfrak{C}$ analog ab $>c_{11}<$ abspeichern. Die Rechnung soll in Gleitkomma erfolgen, und die Indexregister 1 bis 5 sollen nach Ablauf des Bibliotheksprogramms den ursprünglichen Inhalt wieder enthalten; nur die Programmerker 6 bis 9 sollen vom BP verwendet werden.
Das Hauptprogramm sehe so aus:

Pos.	Befehl			Bemerkung
⋮		⋮		
	.....	0	10	Sprung ins BP
⋮	$>a_{11}<$	0	00	
⋮	$>b_{11}<$	0	00	
⋮	$>c_{11}<$	0	00	
⋮	2p	0	00	
RSA	.....	.....		Rücksprung vom BP
⋮	⋮	⋮	⋮	⋮

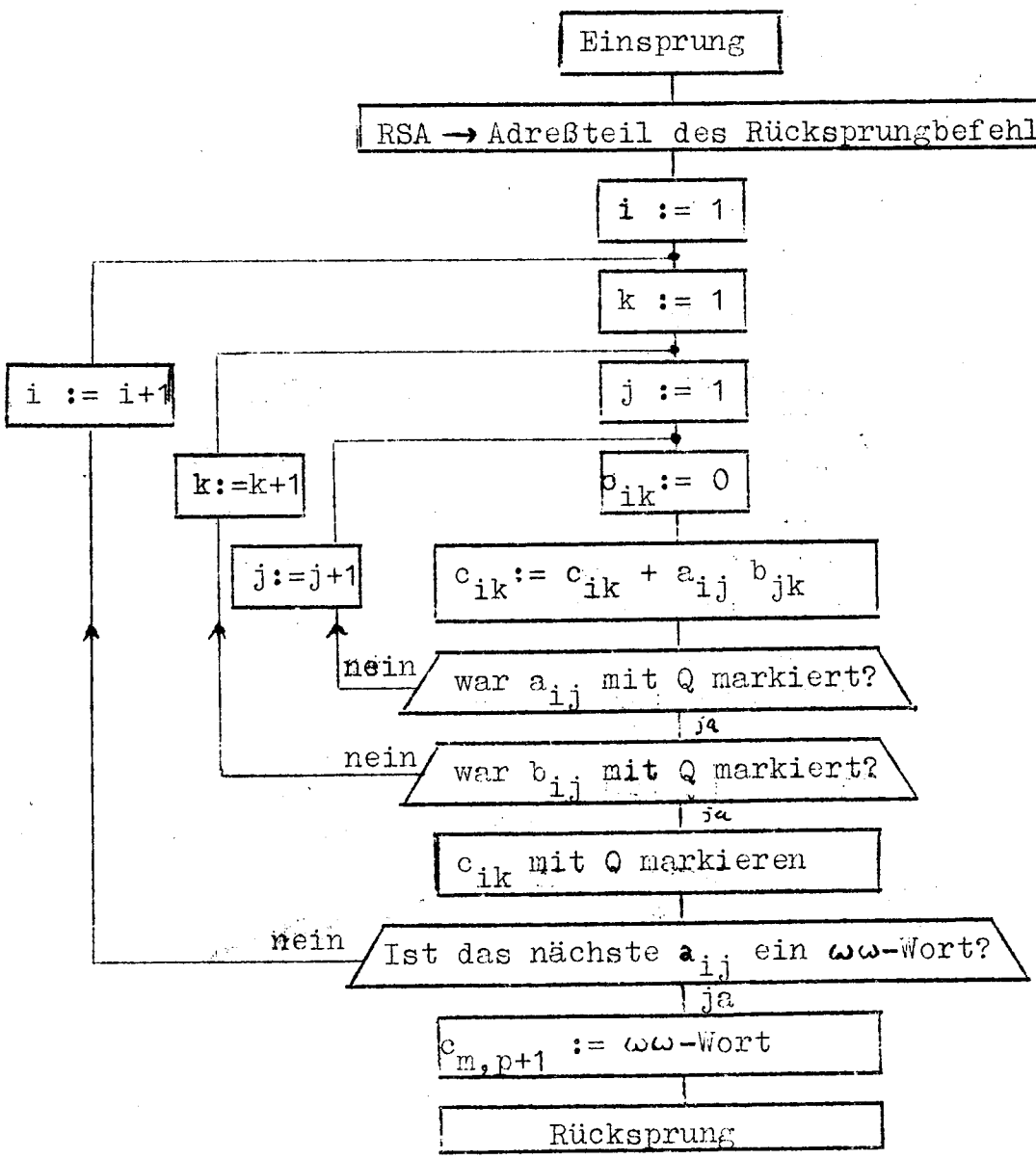
Nach dem Absprung ins BP sind die Adressen $>a_{11}<$, $>b_{11}<$ und $>c_{11}<$ angegeben und auch der Wert 2p. Aus 2p und den Q-Markierungen der letzten Spalte von $\mathfrak{A}$ bzw. $\mathfrak{B}$ und dem ωω-Wort hinter den Elementen kann das BP die Größe der zu multiplizierenden Matrizen und damit die der Matrix $\mathfrak{C}$ erkennen.

Es wurden hier also die Parameter zum Teil gemäß Abschnitt II (von Seite 10) und zum Teil gemäß Abschnitt I (von Seite 8) bereitgestellt. Sie werden auch nicht einheitlich übernommen. Wie wir später sehen werden, wird nämlich

$>a_{11}<$ nach I_8,
$>b_{11}<$ in die Hilfszelle 2,
$>c_{11}<$ nach I_7
und 2p in die Hilfszelle 3 übernommen.

Diese gemischte Technik erlaubt es hier Befehle einzusparen - ein Vorteil, den man bei Bibliotheksprogrammen immer wahrnimmt.

Das sich aus der Formel für die c_{ik} ergebende Flußdiagramm sieht so aus:



Das Funktionieren des Flußdiagramms mache man sich am besten an einem Zahlenbeispiel klar.

Da nun im Hauptprogramm die Elemente nicht auf einer "Ebene mit zwei Koordinaten" gegeben werden (≙ Zeilen und Spalten) sondern alle nacheinander wie an einer Schnur aufgereiht angegeben werden, kann man nicht mit Doppelindizes rechnen. Man muß vielmehr die gemäß S. 14 aufgereihten Elemente, beginnend mit a_{11} (bzw. b_{11}, bzw. c_{11}) durchnumerieren und den Zusammenhang mit den Doppelindizes suchen. α sei der Index der Elemente von $\mathcal{A}$

β " " " " " von $\mathcal{B}$

und γ " " " " " von $\mathcal{C}$

Aus a_{ij} wird damit a_α mit $\alpha = (i-1)\cdot n+1$

$i = m$, $j = n+1$ } ist das $\omega\omega$-Wort am Ende der Reihe der Elemente von $\mathcal{A}$

Aus b_{ij} wird b_β mit $\beta = (i-1)\cdot p+1$

$i = n$, $j = p+1$ } ergibt das ωw-Wort

Aus c_{ik} wird c_γ mit $\gamma = (i-1)\cdot p+k$

$i = m$, $k = p+1$ } ergibt das ωw-Wort

Damit lautet die Formel für die Elemente

$$\underbrace{c_{[(i-1)p+k]}}_{=\gamma} = \sum_{j=1}^{n} \underbrace{(a_{[(i-1)n+j]}}_{=\alpha} \underbrace{(b_{[(j-1)p+k]})}_{=\beta}$$

Die ganze Matrix ergibt sich , wenn man für i=1,2,3...m den Index k die Werte 1...p durchlaufen läßt. Da wir nur noch mit 3 Indizes (α, β und γ) arbeiten, müßen wir die Anweisung

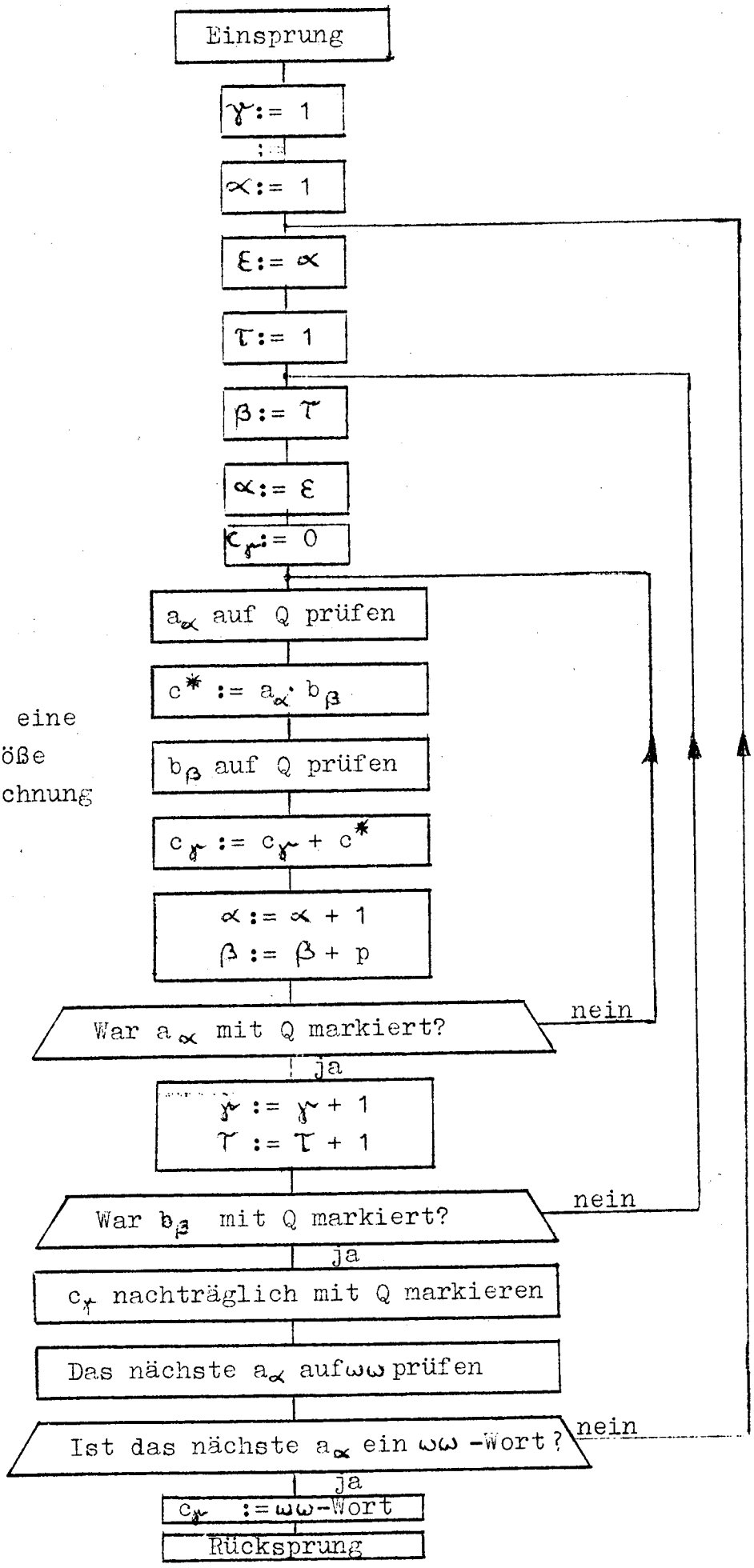
j:= j+1 durch die zwei gleichzeitig

auszuführenden Anweisungen

$\alpha = \alpha+1$ und $\beta := \beta+p$ ersetzen.

Mit k:= k+1 und k:= i+1 könnte man genauso verfahren. Im vorliegenden Beispiel werden jedoch dafür noch die Indizes ε und τ dazugenommen. ε dient zur Numerierung der Zeilen von $\mathcal{A}$, und τ zur Numerierung der Spalten in $\mathcal{B}$.

Das Flußdiagramm sieht dann so aus:



Anmerkung:
c^* ist nur eine Zwischengröße bei der Rechnung

Zur Speicherung der 5 Indizes werden in diesem Programm 4 Indexregister($I_5 \ldots I_8$) und eine Hilfszelle verwendet. Da der Inhalt von I_5 durch den Ablauf des Bibliotheksprogramms nicht verloren gehen darf, muß er in einer Hilfszelle aufbewahrt werden. Im Programm rechnet man nun selbstverständlich nicht mit Indizes, die erst in die Adressen umgerechnet werden müssen, sondern kann (wegen des linearen Zusammenhangs zwischen Index und Adresse) direkt mit den Adressen der Parameter rechnen. Sie sind wie folgt gespeichert:

$\alpha \mathrel{\hat{=}}$ Adresse der fortlaufend gespeicherten Elemente von $\mathcal{A}$ in I_8

$\beta \mathrel{\hat{=}}$ Adresse der fortlaufend gespeichten " von $\mathcal{B}$ in I_5

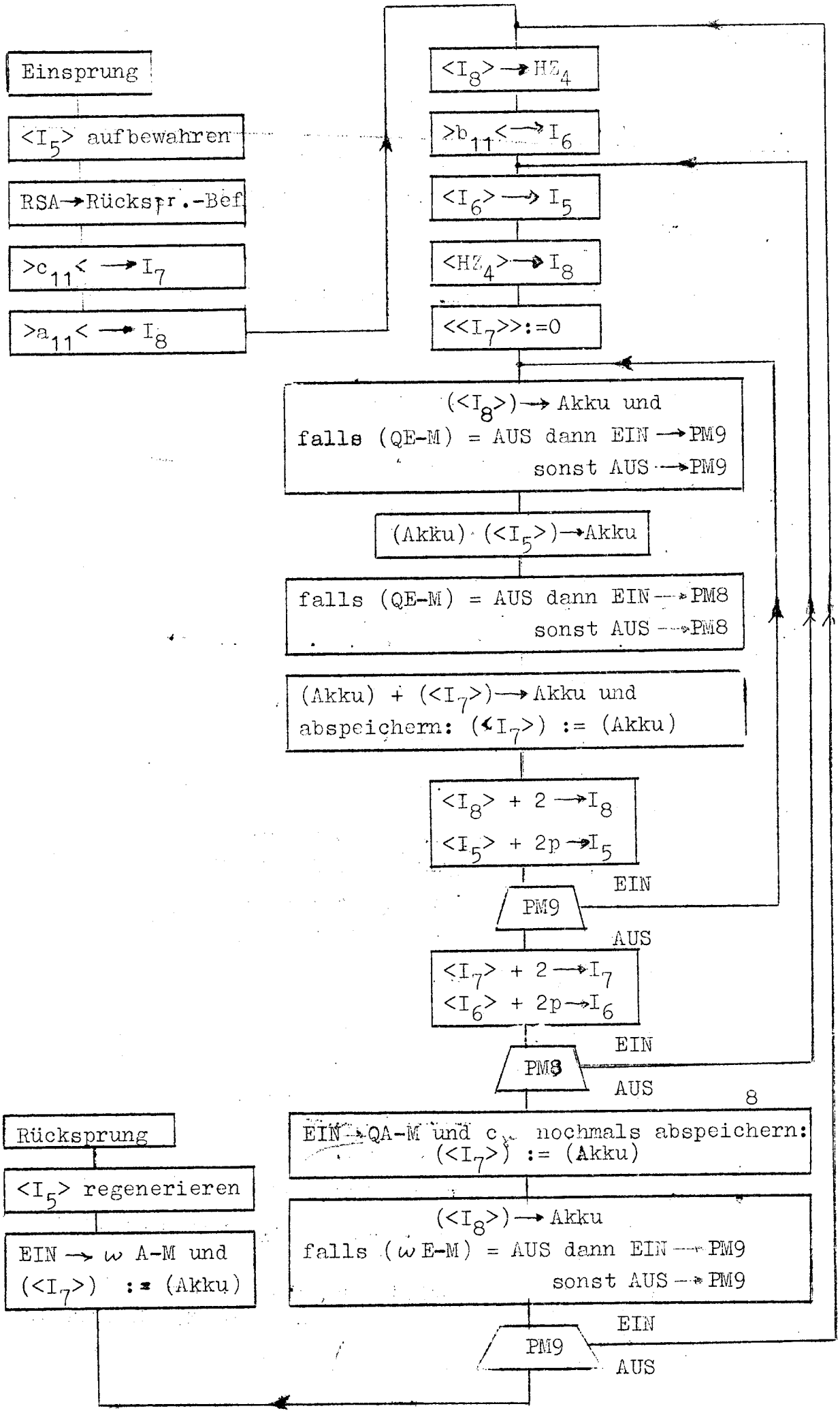
$\gamma \mathrel{\hat{=}}$ " " " " " von $\mathcal{L}$ in I_7

$\tau \mathrel{\hat{=}}$ Adresse der obersten Elemente in den Spalten von $\mathcal{B}$ in I_6

$\varepsilon \mathrel{\hat{=}}$ Adresse der Elemente ganz links in den Zeilen von $\mathcal{A}$ in HZ_4

Entsprechend der Rechnung mit Doppelworten müßen die Indexregister immer um 2 erhöht werden, wenn man zum nächsten Element will. Will man in $\mathcal{B}$ zu dem eine Zeile tiefer liegenden Element, dann muß man um 2p erhöhen (in I_5). Demgemäß bedeutet z.B. $\alpha := 1$ das Einspeichern von $>a_{11}<$ nach I_8 (und analog für β, γ usw.).

Damit ergibt sich folgendes Flußdiagramm:



Befehlsliste

rel.Pos	Bef.	Bemerkung
0000	2000 0 19	Betriebsart Gleitkomma
1	0000 5 81	Aufbewahren von $\langle I_5 \rangle$ in HZ_1^*
2	0057 9 86	
3	0000 9 89	
4	0000 5 91	RSA→Adreßteil des Rücksprungbefehls
5	0004 0 93	
6	0034 9 89	
7	0001 5 80	$>b_{11}< \longrightarrow HZ_2^*$
8	0006 9 86	
9	0002 5 80	$>c_{11}< \longrightarrow I_7$
10	0000 7 99	
0011	0003 5 80	$2p \longrightarrow HZ_3^*$
2	0015 9 86	
3	0000 5 80	$>a_{11}< \longrightarrow I_8$
4	0003 9 86	$\langle I_8 \rangle \longrightarrow HZ_4^*$
(HZ_2)5	0000 6 91	$\langle HZ_2^* \rangle \longrightarrow I_6$, d.h. $>b_{11}< \longrightarrow I_6$
6	0000 6 81	$\langle I_6 \rangle \longrightarrow I_5$
7	0000 5 99	
(HZ_4)8	0000 8 91	$\langle HZ_4^* \rangle \longrightarrow I_8$
9	0022 9 31	$c_\gamma := 0$
0020	0000 7 32	
1	0000 8 31	$a_\alpha \longrightarrow$Akku und QE-M setzen
2	6009 0 18	(QE-M) nach PM9 übertragen
3	0000 5 37	$c^* := a_\alpha \cdot b_\beta$ und QE-M setzen
4	6008 0 18	(QE-M) nach PM8 übertragen
5	0000 7 35	$c_\gamma := c_\gamma + c^*$
6	0000 7 32	
7	0002 8 93	$\langle I_8 \rangle + 2 \longrightarrow I_8$
(HZ_3)8	0000 5 93	$\langle I_5 \rangle + 2p \longrightarrow I_5$
9	9991 9 09	Sprung wenn (PM9) = EIN
0030	0002 7 93	$\langle I_7 \rangle + 2 \longrightarrow I_5$
1	0002 6 93	$\langle I_6 \rangle + 2 \longrightarrow I_6$
2	9983 9 08	Sprung wenn (PM8) = EIN
3	6000 0 19	EIN→QA-M und c_γ nochmals abspeichern
4	9998 7 32	
5	0000 8 31	nächstes $a_\alpha \longrightarrow$Akku, QE-M setzen
6	7009 0 19	und ω E-M nach PM8 übertragen
7	9976 9 09	Sprung wenn (PM9) = EIN

rel.Pos.	Bef.	Bemerkung
8	7000 0 19	EIN→ ωA-M
9	0000 7 32	c_r := ωω-Wort
HZ_1 0040	0000 5 91	$<I_5>$ regenerieren
1	0000 9 91	Rücksprung
HZ_5 2	1000 0 00	Hilfszelle 5, enthält die Null
3	0000 0 00	
4	9999 9 99	
5	9999 9 99	

Anmerkung: HZ_q^* = erste 4 Stellen der Hilfszelle$_q$ (=Adreßteil des Befehls in dieser Zelle

Speicherbedarf für das BP : 46 Zellen

Speicherbedarf für Rechengrößen: 2(mn + np + mp) + 6 Zellen

Verwendete Indexregister : I_5, I_6, I_7, I_8, I_9 (I_5 wird am Programmende regeneriert!)

Verwendete Programmerker : PM8, PM9, (beide am Programmende auf AUS)

Zeitbedar : (4,65 mnp + 1,65 mp + 1,50m + 3,97) millisek.