PROGRAMMIERFIBEL

DIEHL combitron S

mit Sonderprogramm

VORWORT

Die Ausführungen in dieser Programmierfibel erfordern für ihr uneingeschränktes Verständnis vom Leser ein vorausgehendes gründliches Studium der bereits bekannten Bedienungsanleitung DIEHL combitron =S=.

Obgleich das programmierte Rechnen mit DIEHL combitron =S= mit Sonderprogramm im allgemeinen in unserer Bedienungsanleitung hinreichend erklärt wird, können Situationen eintreten, die ein tiefergehendes Eindringen in die Programmiertechnik des elektronischen Rechensystems erforderlich machen.

Diesen Umständen haben wir mit dieser Programmierfibel versucht Rechnung zu tragen, wenngleich es wichtig ist zu wissen, daß damit noch keine Patentrezepte für die Programmierung ganz bestimmter Probleme gegeben sind.
Sie wird jedoch durch ihre problembezogenen Hinweise und programmiertechnischen Kunstgriffe jedem die Hilfsmittel erschliessen helfen, die er benötigt, um erfolgreich programmieren zu können.

Sollten dennoch irgendwelche Fragen, die Programmierung von DIEHL combitron =S= mit Sonderprogramm betreffend, unbeantwortet bleiben, so steht Ihnen unsere Abteilung Produktanwendung jederzeit gerne mit Rat und Tat zur Seite.

Wir wünschen Ihnen viel Erfolg mit Ihrem DIEHL Rechensystem.

DIEHL
Rechenmaschinen

I N H A L T

Seite

I N H A L T

III. KUNSTGRIFFE DER PROGRAMMIERUNG

I. DIE PROGRAMMIERUNG VON DIEHL COMBITRON =S= MIT SONDERPROGRAMM

Vorab noch einige Ausführungen zum Blockschema:

Das Blockschema ist ein "Modell", woran sich der jeweils bei Betätigen einer Taste abspielende Vorgang im Rechensystem DIEHL combitron =S= mit Sonderprogramm praktisch verfolgen läßt. Diese "Modellvorstellung" ist für die Programmierung von DIEHL combitron =S= mit Sonderprogramm eine ausreichende Hilfe.

Einige Gegebenheiten sollen hier nochmals hervorgehoben werden:

1. Die verschiedene Funktionsweise der Befehle S bzw. S (siehe Bedienungsanleitung, Seite 45).

2. Die Unabhängigkeit der Rechenspeicher.
   Deshalb ist es z. B. mit DIEHL combitron =S= mit Sonderprogramm möglich, eine Addition zu unterbrechen und zwischendurch sowohl zu multiplizieren/dividieren und zu radizieren, als auch Umspeicherungen von Zahlen vorzunehmen, ohne daß eine dieser Operationen die andere Operation beeinflußt.
   Eine gewisse Einschränkung gilt für den Ergebnisspeicher (siehe 1.).

3. DIEHL combitron =S= mit Sonderprogramm unterscheidet sich vom Rechensystem DIEHL combitron =S= in folgendem:

   a) DIEHL combitron =S= mit Sonderprogramm besitzt 10 Konstanten- *und* 10 Programmspeicher, wobei für externes Rechnen die 10 Programmspeicher noch als zusätzliche Konstantenspeicher eingesetzt werden können; d. h., es stehen maximal 20 Konstantenspeicher zur Verfügung. Umgekehrt können die 10 Konstantenspeicher *nicht* als Programmspeicher verwendet werden.

b) Bei manueller Rechnung erscheint auf dem Kontrollstreifen bei Wegspeicherung einer negativen Zahl in einen der Programmspeicher P (0 - 9)

Zahl = ↙ (0 - 9) ;

bzw. bei Abruf einer negativen Zahl aus diesen Speichern

Zahl = ↗ (0 - 9)

c) DIEHL combitron =S= mit Sonderprogramm hat eine Befehlskapazität von 10 Befehlen pro Programmspeicher.

d) Die Eingabe einer Zahl in einen Konstantenspeicher erfolgt mittels ↙ (0 - 9), in einen Programmspeicher mittels P ↙ (0 - 9). Entsprechend gilt für den Abruf ↗ (0 - 9) bzw. P ↗ (0 - 9). Insbesondere wird also ein Zifferncode eingegeben über P ↙ (0 - 9) und abgerufen über P ↗ (0 - 9).

4. Der Zifferncode einer Befehlsfolge eines Programmspeichers ist eine 14- oder 15-stellige positive Zahl.

Ein gründliches Studium der Erklärungen der "Kontroll- und Steuerelemente" und des "Blockschemas" in Verbindung mit Beispielen ist für das Erlernen der Programmierung von DIEHL combitron =S= mit Sonderprogramm sehr nützlich.

Bevor man selbst zur Programmierung übergeht, sollte man zunächst in der Lage sein, die Programmierung nachfolgender Beispiele zu verstehen. Dazu ist es vorteilhaft, daß man bei jedem Befehl den Vorgang im Rechensystem anhand des Blockschemas verfolgt.

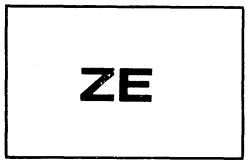
Die Programmierung der Beispiele in I. ist so dargestellt, daß neben den Befehlen alle Speicher, die belegt werden, aufgeführt sind. Nach jedem Befehl sind die jeweiligen Speicherinhalte ersichtlich; die Speicherinhalte, welche sich gegenüber den vorherigen Inhalten ändern, sind farblich deutlich markiert.

Anfangs empfiehlt es sich für den Ungeübten bei der Programmerstellung ebenso zu verfahren.

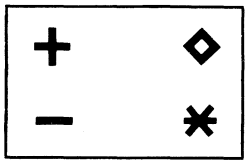
Bei den Beispielen, die der Bedienungsanleitung für DIEHL combitron =S= entnommen sind, wird auf die entsprechende Seite verwiesen.

Erläuterungen zu den Beispielen:

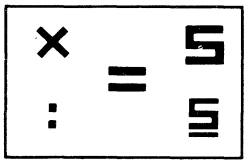
Im Speicherbelegplan bedeutet:



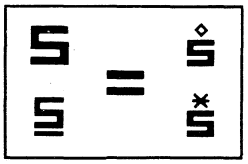
= Zentraleinheit (Druckspeicher)



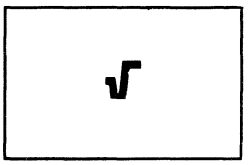
= Saldierspeicher



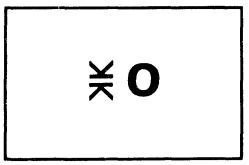
= Multiplikations-/Divisionsspeicher (MD-Speicher)



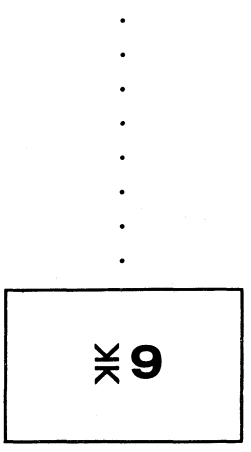
= Ergebnisspeicher



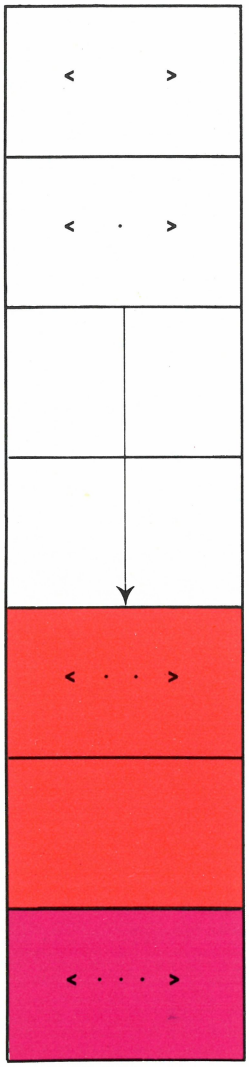
= Wurzelspeicher



= Konstantenspeicher 0



= Konstantenspeicher 9



= Der entsprechende Speicher enthält die Zahl 0

= Der entsprechende Speicher hat als Inhalt < · >

= Der entsprechende Speicher hat als Inhalt < · · >

= Der Inhalt des entsprechenden Speichers ändert sich und enthält als neuen Inhalt < · · >

= Der Inhalt des entsprechenden Speichers ändert sich und enthält als neuen Inhalt die Zahl 0

= a) Der Inhalt der ZE wird in den MD-Speicher übernommen; eine begonnene Multiplikation/Division wird beendet; danach enthält der MD-Speicher die Zahl 0.

b) Der Inhalt der ZE wird in den Wurzelspeicher übernommen, das Ergebnis gebildet und der Speicher mit der Zahl 0 „belegt".

= Befehlsfolge

= Der Befehl < · · · · > hat die Befehlsnummer n (n = 1, 2, . . , 100).

= Der Befehl mit der Befehlsnummer n.
In dieser Form wird ggf. bei Erläuterungen auf den n-ten Befehl Bezug genommen.

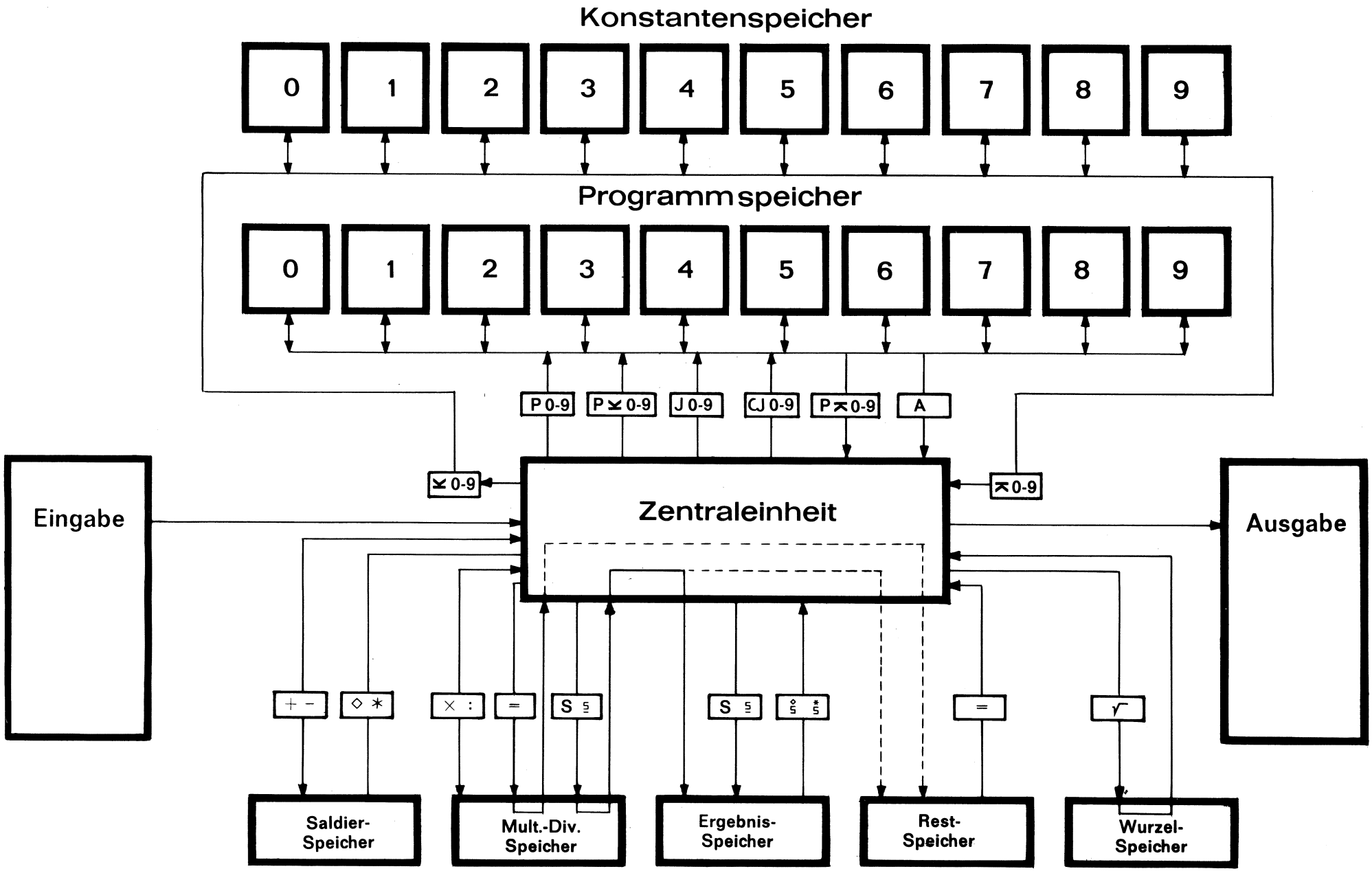
Beispiel 1

(siehe Bedienungsanleitung, Seite 24)

$$\frac{7{,}23 \cdot 8{,}35 \cdot (17{,}83 + 2{,}09)^2}{\sqrt{173{,}88} - 64{,}03} = -471{,}15\ 82\ 19$$

Blockschema

combitron S
mit Sonderprogramm



Beispiel 1

I

	B	Z E	+ −	× : =		√			
1	7,23	7,23							
2	×			7,23 · (?)					
3	8,35	8,35							
4	×			7,23 · 8,35 · (?)					
5	17,83	17,83							
6	+		17,83						
7	2,09	2,09							
8	+		17,83 + 2,09						
9	✱	19,92							
10	×			60,3705 · 19,92 · (?)					
11	:			60,3705 · 19,92² : (?)					
12	173,88	173,88							
13	√	13,186356				173,88			
14	+		13,186356						
15	64,03	64,03							
16	−		13,186356 − 64,03						
17	✱	− 50,843644							
18	=	− 471,158219		$\frac{60{,}3705 \cdot 19{,}92^2}{-\ 50{,}843644}$					

Beispiel 2

(siehe Bedienungsanleitung, Seite 24)

$$\frac{(7{,}6 \cdot 2{,}8)^2}{9{,}81 \cdot \sqrt{9{,}81}} + \frac{7{,}8^2 + 3{,}9 - 4{,}2}{-2{,}831} - 0{,}621 = -7{,}26\ 76\ 25$$

Beispiel 2

I

	B	Z E	+ −	× : =		√	M 7	M 8	M 9
1	7,6	7,6							
2	×	↓		7,6 · (?)					
3	2,8	2,8		↓					
4	=	21,28		7,6 · 2,8 = 21,28					
5	×	↓		21,28 · (?)					
6	:	↓		$21{,}28^2$: (?)					
7	9,81	9,81		↓					
8	:	↓		$\frac{21{,}28^2}{9{,}81}$: (?)					
9	√	3,132091		↓		9,81			
10	M	14,738044		$\frac{21{,}28^2}{9{,}81 \cdot \sqrt{9{,}81}} = 14{,}738044$	14,738044				
11	7,8	7,8			↓				
12	×	↓		7,8 · (?)	↓				
13	=	60,84		$7{,}8^2 = 60{,}84$	↓				
14	+	↓	60,84		↓				
15	3,9	3,9	↓		↓				
16	+	↓	60,84 + 3,9 = 64,74		↓				
17	4,2	4,2	↓		↓				
18	−	↓	64,74 − 4,2 = 60,54		↓				
19	✱	60,54			↓				

Beispiel 2

I

	B	Z E	+ − ◇ ✱	× : = S S	S S ◇ ✱	√	x²	x³	x⁹
20	:	60,54		60,54 : (?)	14,738044				
21	2,831	2,831							
22	(-)	− 2,831							
23	S	− 21,384669		$\frac{60,54}{-2,831} = -21,384669$	14,738044 + (-21,384669)				
24	,621	0,621							
25	S	↓			14,... + (-21,...) - 0,621				
26	✱S	− 7,267625							

Beispiel 3

(siehe Bedienungsanleitung, Seite 34)

GERADENGLEICHUNG

$y = a + b\,x$

Die Programmierung ist hier etwas anders als in der Bedienungsanleitung. Während dort die Speicherung der Konstanten a und b manuell erfolgte, ist diese hier in das Programm einbezogen worden (in P 0).

Die Schleife zur Berechnung von y für verschiedene x - Werte befindet sich in P 1 und P 2. Da es sich hier um die Programmierung für DIEHL combitron =S= mit Sonderprogramm handelt, sind natürlich die Zifferncodes andere als in der Bedienungsanleitung. Dies trifft auf alle anderen internen Programmierungen zu.

Bei sich ändernden Konstanten ist mit J 0 der Programmspeicher P 0 anzuwählen, dann sind a und b in bekannter Weise einzugeben.

Erläuterung:

B 1: Die Löschung des Saldierspeichers muß programmiert werden, da sonst bei verändertem a diese Konstante zu der vorherigen addiert würde.

Beispiel 3

I

	B	Z E	+ −	◇ ✱	× :	=	⊆ ⊆	⊆ ⊆	◇ ⊆ × ⊆	√	✱ 7	✱ 8	✱ 9
	P												
	0												
1	✱	a											
2	(a)	a											
3	#	a (print)											
4	+	↓	a										
5	(b)	b											
6	#	b (print)											
7	✱	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
	9	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	b
	A	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
	P												
	1	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
8	(X)	x	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
9	#	x (print)	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
10	×	↓	↓	↓	x · (?)	↓	↓	↓	↓	↓	↓	↓	↓
11	✱	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
	9	b	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
12	⊆	x · b	↓	↓	x · b	↓	x · b	↓	↓	↓	↓	↓	↓

P_1

Beispiel 3

I

	B	Z E	+ −	◇ ✱	× : =	S S	S* S	√	M 7	M 8	M 9
13	◇	a	a			x · b					b
14	S	↓				x · b + a					
15	S*	a + b x									
16	#	a + b x (print)									
	A										
	P										
	2										
P1 ↑ 17	J										
	1	↓	↓								↓
	A										

Beispiel 4

(siehe Bedienungsanleitung, Seite 36)

MITTELWERT UND STANDARDABWEICHUNG

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

$$S = \sqrt{\frac{1}{n-1}\left[\sum_{i=1}^{n} x_i^2 - \bar{x}\sum_{i=1}^{n} x_i\right]}$$

Erläuterungen:

1. Nebenstehende Programmierung unterscheidet sich in P_1 von der Programmierung in der Bedienungsanleitung, indem die ◇ durch ✱ (B 19) ersetzt und ✱ vor J 0 weggelassen wurde.

2. *Vor erstmaliger Programmausführung* sind die Konstantenspeicher ⩤ 8 und ⩤ 7 durch *Eingabe der Zahl 0 manuell zu "löschen"*, da in beiden über den Saldierspeicher je eine Summe gebildet wird.

$$\leftarrow 7 = \sum_{i=1}^{n} x_i$$

$$\leftarrow 8 = n \quad \text{(Anzahl der Meßwerte)}$$

Bei "Summenbildung in Konstantenspeichern" ist immer darauf zu achten, daß die Löschung der entsprechenden Konstantenspeicher vorgenommen wird.

3. In den Programmspeichern P_o, P_1 und P_2 befindet sich das Programm zur Berechnung von

$$\sum_{i=1}^{n} x_i \; ; \; \sum_{i=1}^{n} x_i^2 \quad \text{und} \quad n.$$

Der Mittelwert $\bar{x}$ und die Standardabweichung s werden in P_3, P_4 und P_5 ermittelt. Deshalb ist nach Eingabe aller x_i durch J 3 dieses Programm manuell anzuwählen.

Der Speicherbelegungs - Darstellung liegt am Anfang von P_o die Annahme zugrunde, daß n - 1 x_i - Werte bereits eingegeben seien.

Außerdem werden am Schluß durch die Befehle B 40 bis B 42 die Konstantenspeicher 8 und 7 automatisch gelöscht, womit bei Berechnungen für neue Meßwertreihen die manuelle Löschung entfällt.

Beispiel 4

I

	B	Z E	± ◇	× : =		√	K 7	K 8	K 9
1	1,	1,							
2	K								
	9								1,
3	,	0,							
4	K								
	8							0,	
5	K								
	7						0,		
	P								
	O								
6	(X_n)	x_n			$\sum_{i=1}^{n-1} x_i^2$		$\sum_{i=1}^{n-1} x_i$	n − 1	
7	#	x_n (print)							
8	+		x_n						
9	×			$x_n \cdot (?)$					
10	⊆	x_n^2		x_n^2	$\sum_{i=1}^{n} x_i^2$				
11	K								
	7	$\sum_{i=1}^{n-1} x_i$							
12	+		$\sum_{i=1}^{n} x_i$						
13	✱	$\sum_{i=1}^{n} x_i$							

P_0

Beispiel 4

I

	B	Z E	± ◇ ✱	× : = ∑	∑ ◇	√	M 7	M 8	M 9
	A								
	P								
	1								
14	M								
	7	$\sum_{i=1}^{n} x_i$			$\sum_{i=1}^{n} x_i^2$		$\sum_{i=1}^{n} x_i$	n − 1	1,
15	M								
	8	n − 1							
16	+		n − 1						
17	M	↓							
	9	1,							
18	+	↓	n						
19	✱	n							
	A								
	P								
	2								
20	M	↓						↓	
	8							n	
21	J	↓							
	O	↓			↓		↓	↓	↓

P_0

Beispiel 4

I

	B	Z E	+ −	◇ *	× :	=	Σ Σ	Σ* Σ*	√	x̄ 7	x̄ 8	x̄ 9
	A											
	P											
	3											
22	#	n (print)						$\sum_{i=1}^{n} x_i^2$		$\sum_{i=1}^{n} x_i$	n	1,
23	+		n									
24	x̄											
	7	$\sum_{i=1}^{n} x_i$										
25	:				$\left(\sum_{i=1}^{n} x_i\right)$: (?)							
26	◇	n										
27	=	$\bar{x}$			$\bar{x}$							
28	#	$\bar{x}$ (print)										
29	×				$\bar{x}$ · (?)							
	A											
	P											
	4											
30	x̄											
	7	$\sum_{i=1}^{n} x_i$										
31	Σ	$\bar{x} \cdot \sum_{i=1}^{n} x_i$			$\bar{x} \cdot \sum_{i=1}^{n} x_i$		$\sum_{i=1}^{n} x_i^2 - \bar{x} \cdot \sum_{i=1}^{n} x_i$					
32	Σ*	$\sum_{i=1}^{n} x_i^2 - \bar{x} \cdot \sum_{i=1}^{n} x_i$										

Beispiel 4

I

	B	Z E	+ −	× : =	√	M+ 7	M+ 8	M+ 9
33	:	$\sum_{i=1}^{n} x_i^2 - \bar{x} \cdot \sum_{i=1}^{n} x_i$	n	$\left(\sum_{i=1}^{n} x_i^2 - \bar{x} \cdot \sum_{i=1}^{n} x_i\right) : (?)$		$\sum_{i=1}^{n} x_i$	n	1,
34	M+							
	9	1,						
35	−		n − 1					
36	×	n − 1						
37	=	s^2		$\left(\sum_{i=1}^{n} x_i^2 - \bar{x} \cdot \sum_{i=1}^{n} x_i\right) : (n-1)$				
	A							
	P							
	5							
38	√	s			s^2			
39	#	s (print)						
40	×	0,						
41	M+							
	7					0,		
42	M+							
	8						0,	
P_0 43	J							
	0							
	A							

Beispiel 5:

(siehe Bedienungsanleitung, Seite 38)

TILGUNGSPLAN

Gegeben:

Annuität	:	A	
Zinssatz	:	P ;	(%)
Darlehensbetrag	:	K	(Kapital)

Gesucht:

Fälligkeitstermin	:	t jährlich (monatlich, ...)
Zinsen für das t-te Jahr (Monat,.....)	:	Z_t
Tilgung für das t-te Jahr (Monat,.....)	:	T_t
Restdarlehen nach dem t-ten Jahr (Monat,...):		K_t

I DIE PROGRAMMIERUNG VON DIEHL COMBITRON =S= MIT SONDERPROGRAMM

Formeln:

$$Z_{t+1} = K_t \cdot \frac{p}{100} = K_t \cdot p^* \text{ mit } p^* = \frac{p}{100}$$

$$T_{t+1} = A - Z_{t+1}$$

$$K_{t+1} = K_t + Z_{t+1} - A$$

Erläuterungen:

1. In P_o und P_1 werden die Anfangswerte A, p^* und K weggespeichert. Dann wird nach dem Überlaufprinzip (siehe II. 5 und Bedienungsanleitung, Seite 30) von P_2 nach P_3 übergegangen, wo die Schleife zur Berechnung von t, Z_t, T_t und K_t beginnt; sie endet in P_5 mit dem Rücksprungbefehl J 2 (B 36).

2. Die Speicherbelegungs-Darstellung ist auch hier allgemein gehalten. So findet man bei B 10 schon eine Belegung vor, die aufgrund von (angenommenen) t Schleifendurchläufen bereits vorliegt. Vor dem 1. Durchlauf wäre die Belegung natürlich gleich der am Ende von P_1. Indem man mit t = 0 beginnt, kann man sich allerdings sukzessive den Ablauf jeweils klar machen. Es ist dabei K_o das aufgenommene Darlehen.

3. Ist $K_{t+1} = K_t + Z_{t+1} - A \leq 0$, so heißt dies, daß das noch verbleibende Restdarlehen $K_t + Z_{t+1}$ (ohne Abzug der Annuität) kleiner oder gleich der noch zu zahlenden Annuität ist, was wiederum zur Folge hat, daß durch Zahlung des verbleibenden Restdarlehens das ursprüngliche Darlehen, einschließlich Zinsen, endgültig getilgt ist. Danach soll das Rechensystem mit der Programmausführung stoppen. Die Abfrage $K_t + Z_{t+1} - A \leq 0$ (siehe III. 11, Grundfall 3), erfolgt mit B 28. Bejahendenfalls verfolge man den Ablauf ab B 37 weiter.

Beispiel 5

	B	Z E	+ − ◇ *	× : = ≤ ≥	≤ ≥ ◇ *	√	* 7	* 8	* 9
	P								
	O								
1	(A)	A							
2	#	A (print)							
3	*								
	9								A
4	(p*)	p*							
5	#	p* (print)							
6	*								
	8							p*	
7	(K)	K							
8	#	K (print)							
	A								
	P								
	1								
9	+		K						
	A								

Beispiel 5

I

	B	Z E	+ −	◇ ✱	× :	=	S S	S S	◇ ✱ S	√	✱ 7	✱ 8	✱ 9
	P												
	2												
10	✱												
	7	t		K_t							t	p*	A
11	S							t					
12	✱												
	8	p*											
13	:				p* : (?)								
14	S	1,			$\frac{p^*}{p^*} = 1,$			t + 1					
15	✱ S	t + 1											
16	#	t + 1 (print)											
	A												
	P												
	3												
17	✱												
	7										t + 1		
18	◇	K_t											
19	S							K_t					
20	×				$K_t \cdot$ (?)								

Beispiel 5

I

	B	Z E	+ − ◇ *	× : = S S	S S S*	√	X 7	X 8	X 9
21	X								
	8	p*	K_t	$K_t \cdot (?)$	K_t		t + 1	p*	A
22	S	Z_{t+1}		$K_t \cdot p^* = Z_{t+1}$	$K_t + Z_{t+1}$				
23	#	Z_{t+1} (print)							
	A								
	P								
	4								
24	X								
	9	A							
25	S				$K_t + Z_{t+1} - A = K_{t+1}$				
26	S*	K_{t+1}							
27	(−)	$-K_{t+1}$							
28	CJ								
	6	$-K_{t+1} \geq 0$?							
29	◇	K_t							
30	S				$Z_{t+1} - A = -T_{t+1}$				
31	X*	$-T_{t+1}$ (print)							
	A								

≥0 → P6

Beispiel 5

I

	B	Z E	+ − ◇ ✱	× : = S S	S S ◇ ✱ S	√	✱ 7	✱ 8	✱ 9
	P								
	5								
32	#	$-T_{t+1}$ (print)	K_t				t + 1	p*	A
33	+		$K_t + (-T_{t+1}) = K_{t+1}$						
34	◇	K_{t+1}							
35	#	K_{t+1} (print)							
P_2 36	J								
	2								
	A								
	P								
	6								
P_6 37	✱								
	9	A	$K_t = K_{n-1}$		$K_{t+1} = K_n$		t + 1 = n	p*	A
38	S				$K_n + A = T_n$				
39	✱ S	T_n							
40	(−)	$-T_n$							
41	#	$-T_n$ (print)							
42	✱ S	0,							
43	#	$Z_n = 0$ (print)							

Beispiel 5

I

B	Z E	+ − ◇ ✱	× : = S S	S S ◇ S	√	✱ 7	✱ 8	✱ 9
✱								
7		K_{n-1}				0,	p*	A
A								
P								
7								
✱	K_{n-1}							
J								
O								
A								

P_0

Beispiel 6:

(siehe Bedienungsanleitung, Seite 40)

DIE e - FUNKTION

Gegeben: x

Gesucht: $y = e^x = \sum_{K=0}^{\infty} \frac{x^K}{K!}$

Erläuterungen:

Für dieses Beispiel gilt ähnliches wie für Beispiel 5

1. Mit dem Studium der Programmierung beginne man bei B 6
   In P_o wird x nach ⩗ 8 weggespeichert und $\frac{x^o}{0!} = 1$ in den Ergebnisspeicher gebracht.

2. In P_1 beginnt die Schleife zur Berechnung von $\sum_{K=1}^{\infty} \frac{x^K}{K!}$ und der dargestellten Speicherbelegung liegt die Annahme zugrunde, daß bereits n - 1 Durchläufe stattgefunden haben.

3. Man beachte die zu diesem Beispiel in der Bedienungsanleitung gegebenen Erläuterungen.

Beispiel 6

I

	B	Z E	+ − ◇ ✱	× : = ⊆ ⊇	⊆ ⊇ ⊆✱⊇	√	✱ 7	✱ 8	✱ 9
1	1,	1,							
2	✱								
	9								1,
	P								
	0								
3	✱	n			$e^x = \sum_{K=0}^{n} \frac{x^K}{K!}$			x	1,
4	✱ ⊆	e^x							
5	#	e^x (print)							
6	(X)	x							
7	#	x (print)							
8	✱								
	8							x	
9	✱								
	9	1,							
10	⊆				$1, = \frac{0}{0!}$				
	A								
	P								
	1	$\frac{x^{n-1}}{(n-1)!}$	n − 1		$\sum_{K=0}^{n-1} \frac{x^K}{K!}$				
11	X			$\frac{x^{n-1}}{(n-1)!} \cdot (?)$					

P_0 → 3

P_1 → 11

Beispiel 6

I

	B	Z E	+ −	× : =		√	x↔7	x↔8	x↔9
12	x↔								
	8	x	n − 1	$\frac{x^{n-1}}{(n-1)!} \cdot (?)$	$\sum_{K=0}^{n-1} \frac{x^K}{K!}$			x	1,
13	:			$\frac{x^n}{(n-1)!} : (?)$					
14	x↔								
	9	1,							
15	+		n						
16	◇	n							
17	⊆	$\frac{x^n}{n!}$		$\frac{x^n}{n!}$	$\sum_{K=0}^{n} \frac{x^K}{K!}$				
	A								
	P								
	2								
18	CJ								
P3	3	$\frac{x^n}{n!} \geq 0$?							
19	J								
P1	1	$\frac{x^n}{n!} < 0$							
	A								
	P								
	3								
P3	(−)	$-\frac{x^n}{n!} \leq 0$							

Beispiel 6

I

		B	Z E	+ − ◇ ✱	× : = ⊆ ⊆	⊆ ⊆ ⊆ ⊆	√	✱ 7	✱ 8	✱ 9
P_0 ↑	21	CJ								
		O	$-\frac{x^n}{n!}=0$?	n		$\sum_{K=0}^{n}\frac{x^K}{K!}$			x	1,
	22	(-)	$\frac{x^n}{n!}>0$							
P_1 ↑	23	J								
		1								
		A								

II. WICHTIGE HINWEISE

II. 1. Programmeingabe:

Durch Drücken der P - Taste und eine der Zifferntasten (0 - 9) wird der entsprechende Programmspeicher P_0, P_1, ..., P_9 angerufen und zum Lernen bereitgestellt.

Pro Programmspeicher können maximal 10 Befehle gelernt werden. Wird diese Befehlskapazität versehentlich überschritten, erfolgt auf dem Kontrollstreifen Anzeige durch F - Druck (siehe II. 2.). Der Lernvorgang muß in *jedem* Programmspeicher durch *einmaliges* Betätigen der A - Taste abgeschlossen werden.

Haltbefehle (zur Eingabe von Variablen im Programmablauf) werden bei Programmeingabe über die Funktionstasten durch Eingabe einer positiven Zahl (beispielsweise die Zahl 1) programmiert. Auch die Zahl "0" darf als Haltbefehl eingegeben werden, allerdings *nicht mit der Kommataste allein*, da in diesem Falle die Eingabe ignoriert wird.

Ist die Richtigkeit einer Programmierung durch eine Kontrollrechnung gesichert, ruft man sich die Zifferncodes mit P ↗ (0 - 9) ab und gibt das Programm im Bedarfsfalle als Zifferncode über P ↙ (0 - 9) ein (siehe Bedienungsanleitung Seite 29).

II. 2. Kapazitätsüberschreitung

Kapazitätsüberzug wird durch einen F - Druck auf dem Kontrollstreifen angezeigt und tritt in folgenden Fällen ein:

a) Wenn ein Ergebnis einer Grundrechenoperation über die Stellenkapazität des Rechensystems geht (insbesondere: "Division durch Null"). Das vorher innerhalb der Stellenkapazität liegende Zwischenergebnis einer Addition/Subtraktion bzw. Multiplikation/Division bleibt jedoch erhalten.

b) Nach der "Quadratwurzel aus einer negativen Zahl".

c) Nach "Zifferneingabe - Kommataste - Zifferneingabe - Kommataste" in den Kommapositionen "2 - 4 - 6 - 8".

d) Bei Überschreitung der Befehlskapazität eines Programmspeichers.

e) Wenn der Lernvorgang in einem Programmspeicher nicht mit der A - Taste abgeschlossen und ein neuer Programmspeicher mit P (o - 9) zum Lernen angewählt wird. In diesem Falle ist der falsche Lernvorgang abzuschließen und nochmals, richtig, zu wiederholen.

f) Wenn nach Lernen einer Befehlsfolge über Funktionstasten versucht wird eine Programmausführung mit der A - Taste zu starten (siehe II. 3. f).

Nach F-Druck steht immer die Zahl "0" im Druckspeicher.

Da bei Programmeingabe über Funktionstasten das Rechensystem die zu lernenden Operationen auch durchführt (es wird ja manuell gerechnt, nur, daß die Operationsfolge auch gleichzeitig gelernt wird), kann es mitunter aus den genannten Gründen zu Kapazitätsüberzug kommen. In den Fällen a) und b) ist ein F - Druck ohne Einfluß auf den richtigen Lernvorgang, desgleichen im Falle c), wenn damit Stopp gelernt werden sollte. Im Falle e) sind die über die Befehlskapazität eines Programmspeichers hinausgehenden Befehle im *nächstfolgenden* Programmspeicher zu lernen (siehe Bedienungsanleitung, Seite 30, Überlaufprinzip); ist allerdings der 10. und 11. Befehl einer der "zweistelligen" Befehle ↙ (0 - 9); ↗ (0 - 9), J (0 - 9) oder CJ (0 - 9), so kann, trotz des F - Druckes, vor Abschluß der Lernvorganges mit der A - Taste, immer noch als 10. Befehl ein "einstelliger" gelernt werden (siehe Bedienungsanleitung, Seite 28, sowie III.14).

II. 3. Programmausführung:

Vor Programmausführung sollten folgende Punkte beachtet werden:

a) Die Kommaposition einstellen, mit welcher die Rechnung durchgeführt werden soll.

b) Alle in der Rechnung benötigten Speicher löschen (insbesondere den MD - Speicher durch " 1, = ").

c) Die notwendigen Konstanten in die entsprechenden Speicher eingeben.

d) *Vor erstmaliger Programmausführung* den Programmspeicher, in welchem das Programm beginnt, manuell durch die J - Taste und eine der Zifferntasten (0 - 9) anwählen.

Sollte diese manuelle Anwahl J (0 - 9) vergessen und die Programmausführung mit "Variableneingabe —> A - Taste" bzw. "A - Taste" gestartet werden, kann folgendes eintreten:

Die Programmausführung eines vorher gerechneten Programmes wird fortgesetzt, falls ein Programm über *Zifferncodes* eingegeben wurde (Beispiel 7).

Wird das Programm über die Funktionstasten eingegeben, erfolgt nach Betätigung der A - Taste F - Druck (siehe II. 2. f).

Erfolgt während der Programmausführung F - Druck, so wird der Programmablauf *sofort nach* dem Befehl, der zum F - Druck führte, *unterbrochen*.

Eine falsch eingegebene Zahl kann durch die "C - Taste" gelöscht werden. Nach Übernahme durch die "A - Taste" in das Programm ist eine Korrektur nur noch durch manuelles oder programmiertes "Rückrechnen" (Korrekturschleife) möglich!

Beispiel 7:

Wir geben zunächst das Programm zur Berechnung von $Z_1 = x \cdot y$ in den Programmspeicher P_o und führen einige Berechnungen durch, dann geben wir das Programm zur Berechnung von $Z_2 = \frac{x}{y}$ als Zifferncode nach P_o und "vergessen" J 0. Durch die A - Taste wird nun nicht der neue Programmablauf gestartet, sondern das vorherige Programm wird noch *einmal* gerechnet.

Dies kann man sich plausibel machen, indem man sich noch einen zusätzlichen "Ausführungsspeicher" vorstellt, in den die jeweils zur Ausführung kommende Befehlsfolge *eines* Programmspeichers transferiert wird, womit diese vorübergehend zweimal im Rechensystem vorhanden ist.

Nach Eingabe des Programmes $Z_2 = \frac{x}{y}$ über Zifferncode, befindet sich dieses zwar in P_o, aber noch nicht im "Auführungsspeicher", dort steht noch das Programm $x \cdot y$, welches durch J 0 gelöscht und durch das Programm $\frac{x}{y}$ ersetzt würde.

Deshalb, *vor erstmaliger Programmausführung* J (0 - 9) *nicht vergessen!*

Anwendungsbeispiel

Archiv-Nr.

Beispiel 7

Programm- und Konstantenspeicher: Belegplan

P ≙ Programm ↙ ≙ Konstante

	0	1	2	3	4	5	6	7	8	9
P	P	P								
K										

Programmierung

V≙ Variable B≙ Befehl

	0		1		2		3		4		5		6		7		8		9	
	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B
		P		P																
		0		0																
1	X	(·)	X	(·)																
2		#		#																
3		:		X																
4	y	(·)	y	(·)																
5		#		#																
6		=		=																
7		#		#																
8		J		J																
9		0		0																
10																				
11		A		A																
12																				

Zifferncode	Kontrollstreifen und Bezeichnung

3 6 4 4 6 4 8, 7 1 6 7 0 9 4 4 P ⌅ 0 } Z_1
3 6 4 4 6 4 8, 7 1 6 7 0 9 4 4 ⌅ 8 }

3 6 4 5 6 7 9, 5 0 8 8 6 0 4 8 P ⌅ 0 } Z_2
3 6 4 5 6 7 9, 5 0 8 8 6 0 4 8 ⌅ 9 }

Programmausführung

Eingabe	
Zifferncode Z_1:	P⌅
	0
	J
	0
x = (6,)	A
y = (3,)	A
x = (27,)	A
y = (3,)	A
Zifferncode Z_2:	P⌅
	0
x = (6,)	A
y = (3,)	A
x = (27,)	A
y = (3,)	A

3 6 4 4 6 4 8, 7 1 6 7 0 9 4 4 ⌅ 8
3 6 4 4 6 4 8, 7 1 6 7 0 9 4 4 P ⌅ 0

6, 0 0 0 0 0 0 0 0 0 # x
3, 0 0 0 0 0 0 0 0 0 # y

2, 0 0 0 0 0 0 0 0 0 A $\frac{x}{y}$

27, 0 0 0 0 0 0 0 0 0 # x
3, 0 0 0 0 0 0 0 0 0 # y

9, 0 0 0 0 0 0 0 0 0 A $\frac{x}{y}$

3 6 4 5 6 7 9, 5 0 8 8 6 0 4 8 ⌅ 9
3 6 4 5 6 7 9, 5 0 8 8 6 0 4 8 P ⌅ 0

6, 0 0 0 0 0 0 0 0 0 # x
3, 0 0 0 0 0 0 0 0 0 # y

2, 0 0 0 0 0 0 0 0 0 A x·y

27, 0 0 0 0 0 0 0 0 0 # x
3, 0 0 0 0 0 0 0 0 0 # y

81, 0 0 0 0 0 0 0 0 0 A x·y

II. 4. Kommaverschiebung:

Wird die Kommaposition mit dem Kommawähler geändert, so bewirkt dies eine entsprechende Kommaverschiebung bei den Zahlen in allen Speichern, mit Ausnahme des MD - Speichers. Hier wird grundsätzlich ein Zwischenergebnis einer Multiplikation/Division in das Zwischenergebnis 1 verwandelt.

Deshalb sind *nach Änderung der Kommaposition* evtl. *vorher von Null verschieden* (≠ 0) *gewesene Speicherinhalte kommarichtig neu einzugeben.*

II. 5. Überlaufprinzip:

Ergänzend zu dem auf Seite 30 der Bedienungsanleitung genannten Überlaufprinzip ($P_o \longrightarrow P_1 \longrightarrow \ldots \longrightarrow P_9 \longrightarrow P_o$) ist zu bemerken, daß dieses Prinzip bei fehlendem Rücksprungbefehl J (0 - 9) am Ende eines Programmes mit Sicherheit nur dann funktioniert, wenn sich in den übrigen *Programmspeichern* 1- bis 13-stellige *positive* Zahlen (einschließlich der Zahl 0) befinden.

Beim Auftreten von 14- oder 15-stelligen positiven Zahlen (Konstanten) in diesen Programmspeichern, muß durch Probieren festgestellt werden, ob diese das Überlaufprinzip bzw. auch den Rechenablauf des gewünschten Programmes störend beeinflussen.

Negative oder *16-stellige positive Zahlen* in den Programmspeichern führen immer zum F-Druck, wenn die interne Abfrage erfolgt, ob sich in dem Programmspeicher ein Programm befindet oder nicht.

II. 6. Decodierprogramm:

Mit diesem Programm kann die Befehlsfolge eines Programmes, das in Form der Zifferncodes vorliegt, ermittelt werden.

Man *achte* darauf, daß *vor* Eingabe des Decodierprogramms, die *Kommaposition 0* eingestellt ist. Nach der Eingabe wird jeweils der zu decodierende Zifferncode *eines* Programmspeichers mit der A - Taste in das Programm übernommen und gleichzeitig der Programmablauf gestartet. Danach druckt das Rechensystem auf dem Kontrollstreifen 10 Zahlen mit "A" - Symbol ab. Bei diesen 10 Zahlen handelt es sich um die Zahlen 1 bis 19, denen jeweils ein bestimmter Befehl zugeordnet ist (siehe Decodierprogramm). Wird nur "A" (ohne Zahl) gedruckt, so bedeutet dies, daß kein Befehl vorhanden ist. Zu *beachten* ist, daß die Ermittlung der Befehlsfolge von "unten nach oben" erfolgt, d. h., daß der erste decodierte Befehl in Wirklichkeit der letzte (zehnte) und der letzte decodierte Befehl in Wirklichkeit der erste Bebefehl ist.

In dieser Weise sukzessive für jeden Zifferncode verfahrend, erhält man schließlich die gesamte Befehlsfolge des Programms. Als Beispiel wurde der Zifferncode des Programmes "Geradengleichung" decodiert (siehe Beispiel 8).

Decodierung des Zifferncodes

(DIEHL combitron =S= mit Sonderprogramm)
Kommaposition 0

Eingabe zur Decodierung

Eingabe:
Zu decodierender Zifferncode eines Programmspeichers

	Op	Sp
364 490 467 049 472	P ↙	0
262 857 924 380 847	P ↙	1
684 174 542 603 424	P ↙	2
32	↙	0
9	↙	1
	J	
	O	
	A	

Anmerkung:
Man beachte, daß die Decodierung der Befehlsfolge eines Programmspeichers von „unten nach oben" erfolgt und daß die Zuordnung besteht:

↙10 ≙ P ↙0;	↗10 ≙ P ↗0
↙11 ≙ P ↙1;	↗11 ≙ P ↗1
↙12 ≙ P ↙2;	↗12 ≙ P ↗2
↙13 ≙ P ↙3,	↗13 ≙ P ↗3
↙14 ≙ P ↙4;	↗14 ≙ P ↗4

Ausgabe	Befehl	Nach den Befehlen
1	J	
2	↗	
3	CJ	J, CJ, ↙↗
4	↙	
5	S̊	0
6	(-)	1
7	Ṡ	2
8	◇	3
9	*	4
10	Stop	5
11	#	6
12	√‾	7
13	+	8
14	-	9
15	:	10
16	S	11
17	S̲	12
18	X	13
19	=	14

Zifferncode	Kontrollstreifen und Bezeichnung

Beispiel 8

3 6 4 4 9 0 4 6 7 0 4 9 4 7 2 P ⌅ 0
2 6 2 8 5 7 9 2 4 3 8 0 8 4 7 P ⌅ 1
6 8 4 1 7 4 5 4 2 6 0 3 4 2 4 P ⌅ 2
3 2 ⌅ 0
9 ⌅ 1

*

* S

#

Programmausführung

Eingabe	
	✱
	✱ S
1	=
	J
	0
Zifferncode von P_0:	
328 046 728 067 072	A
Zifferncode von P_1:	
364 559 458 386 155	A
Zifferncode von P_2:	
417 814 418 554 88	A

3 2 8 0 4 6 7 2 8 0 6 7 0 7 2 #
A
A
14 A 9
4 A ⌅
11 A #
10 A Stop
13 A +
11 A #
10 A Stop
9 A *

3 6 4 5 5 9 4 5 8 3 8 6 1 5 5 #
11 A #
7 A S
16 A S
8 A ◇
16 A S
14 A 9
2 A ⊼
18 A X
11 A #
10 A Stop

4 1 7 8 1 4 4 1 8 5 5 4 8 8 #
A
A
A
A
A
A
A
A
6 A 1
1 A J

Eingabe		Kontrollstreifen und Bezeichnung

III. KUNSTGRIFFE DER PROGRAMMIERUNG

III. 1. Restspeicher

Bei einer Division geht *nicht* der *tatsächliche* Rest R, sondern $R \cdot 10^p$, einschließlich dem Vorzeichen des Dividenden, in den Restspeicher.
(p = 0, 2, 4, 6, 8; eingestellte Kommaposition)
Mitunter kann nun der Restspeicher als Konstantenspeicher eingesetzt werden, nämlich dann, wenn sein Inhalt *vor* einer *Division* abgerufen und gegebenenfalls in einen anderen Speicher gebracht werden kann.

1. Möglichkeit:

Es sei p die Kommaposition (p = 0, 2, 4, 6, 8) mit welcher die Rechnung (das Programm) durchgeführt werden soll.

Man geht auf Kommaposition "0", dividiert $a \cdot 10^p$ ($a \gtrless 0$) durch eine betragsgrößere Zahl (am besten nimmt man als Divisor die Zahl $10^{16}-1$ = 9999 9999 9999 9999) und stellt dann die Kommaposition p ein.

Es empfiehlt sich, die Eingabe einer Konstanten in den Restspeicher den anderen Eingaben voranzustellen. Beim Rechnen bzw. Programmieren ist darauf zu achten, daß der Restspeicherinhalt nur nach abgeschlossener Multiplikation/Division (=, S, S) abgerufen werden kann und dieser bei einem Produkt bzw. Quotienten als Multiplikand bzw. Dividend eingesetzt werden muß.

2. Möglichkeit:

Unabhängig von der Kommaposition p kann eine Zahl a ($a \neq 0$), die sich im Druckspeicher befindet, durch nachstehende Befehlsfolge in den Restspeicher transferiert werden:

a)			b)	
	+	oder		S
	+			S
	+			S
	:			:
	*			*
	=			S
				=

Voraussetzung für diese Methode ist natürlich, daß sowohl der MD - Speicher als auch einer der saldierenden Speicher frei ist.

Mit der Befehlsfolge a) bzw. b) ist es nun möglich die Wegspeicherung von Variablen bzw. Ergebnissen in den Restspeicher zu *programmieren*, sofern diese $\neq 0$ sind.

Die Befehlsfolge ist gleichbedeutend mit der Berechnung

$$\frac{a}{3a} = 0,\underbrace{33.....}_{\text{p-mal}} \quad ; \quad \text{Rest } R = a \cdot 10^{-p}$$

Aus $\frac{a}{3a}$ folgt, daß die Maximalzahl, die dadurch in den Restspeicher gebracht werden kann
$\frac{1}{3}(10^{16-p} - 10^{-p}) = 10^{-p}(\frac{1016-1}{3}) = (3333\ 3333\ 3333\ 3333) \cdot 10^{-p}$ ist.

Es geht immer der *Dividend einschliesslich Vorzeichen* in den Restspeicher.

III. 2. Multiplikation mit einer ganzen Zahl

Bei der Berechnung des ganzzahligen Vielfachen einer Zahl a kann in bestimmten Fällen durch die Additionsmethode

$$\underbrace{a + a + \ldots + a}_{n \text{ - Summanden}} = n \cdot a \; ; \; (n = 2, 3, \ldots)$$

Speicherkapazität gewonnen werden. Voraussetzung ist, daß einer der beiden saldierenden Speicher frei ist. Bei Verwendung des Ergebnisspeichers darf außerdem der MD - Speicher nicht belegt sein.

Beispiele:

Es werde angenommen, daß sich a im Druckspeicher befindet:

1.) 2 a : + + *

2.) 6 a : + + + ◊ + *

3.) 8 a : S S S S ◊ S * S

4.) 12 a : S S S ◊ S S S * S

III. 3. Division durch eine ganze Zahl

Bei Division einer Zahl a durch eine ganze Zahl n (n = 2, 3, ...) kann manchmal folgende Methode von Nutzen sein:

Es werde wieder angenommen, daß sich a im Druckspeicher befindet:

Beispiel 1:

$\frac{a}{2}$: x
:
+
+
*
=

$= \frac{a^2}{2a} = \frac{a}{2}$

Beispiel 2:

$\frac{a}{3}$: x
:
+
+
+
*
=

$= \frac{a^2}{3a} = \frac{a}{3}$

Beispiel 3:

$\frac{a}{12}$: x
:
+
+
+
◊
+
+
+
*
=

$= \frac{a^2}{12a} = \frac{a}{12}$

Voraussetzung für diese Methode ist, daß der MD - und der Saldierspeicher (+,-) nicht belegt sind. Außerdem ist der Wertebereich von a, wegen a^2, im Rahmen der Kapazität eingeengt.

III. 4. Programmierung der Zahl 1

Häufig wird bei Programmen die Zahl 1 benötigt, insbesondere bei Postenzählungen. Auch hier kann gelegentlich Kapazität gewonnen werden, indem man die 1 nicht in einen Konstantenspeicher gibt, sondern an entsprechender Stelle, an welcher die 1 benötigt wird, den jeweiligen Inhalt a des Druckspeichers durch sich selbst dividiert, denn es ist $\frac{a}{a} = 1$.

Dazu ist allerdings erforderlich, daß der MD - und der Restspeicher nicht belegt sind und der Inhalt des Druckspeichers, d. h. $a \neq 0$ ist. Die letzte Bedingung wird aber nur in wenigen Fällen *immer* erfüllt sein. Um nun eine "Division durch Null" auszuschalten, bringt man einen sicherlich von Null verschiedenen Konstantenspeicherinhalt in den Druckspeicher und dividiert diesen durch sich selbst. Sollte kein solcher Konstantenspeicherinhalt ($\neq 0$) vorhanden sein, kann man den Zifferncode von einem der belegten Programmspeicher P_0 bis P_4 nehmen, also

P ≍
(0 - 4)
:
=

Man beachte, daß von den Programmspeichern P_0 bis P_9 nur die Programmspeicher P_0 bis P_4 intern als Konstantenspeicher abgerufen werden können (Näheres dazu siehe III.15).

Anmerkung:

Als Ergänzung zu III. 2, III. 3 und III. 4 sei noch erwähnt, daß sich gelegentlich die Untersuchung lohnt, inwieweit sich mehrere im Programm benötigte Konstanten durch Anwendung der Grundrechenoperationen und der Quadratwurzel durch einige dieser Konstanten aufbauen lassen.

So erhält man z. B. die Konstanten 5, 6,25 und 100 mit Hilfe der Konstanten 2,5; denn es ist

$$5 = 2{,}5 + 2{,}5$$

$$6{,}25 = (2{,}5)^2$$

$$100 = (2{,}5 + 2{,}5 + 2{,}5 + 2{,}5)^2 = 10^2$$

III. 5. Reziprokwert

Zum Reziprokwert a^{-1} einer Zahl a ($a \neq 0$) gelangt man, indem man a in den Druckspeicher transferiert und die Operationen (:), (:), (=) durchführt. Dies ist gleichbedeutend mit

$$\frac{\frac{a}{a}}{a} = \frac{1}{a} = a^{-1}$$

III. 6. Speichern einer Zahl im MD - Speicher

Sind alle Speicher mit Ausnahme des Rest- und des MD - Speichers belegt, so kann man eine Zahl a im MD - Speicher vorübergehend festhalten, indem man a mit der Operationstaste (x) übernimmt. Wird a wieder benötigt, so erhält man a nach den Operationen (:), (=). Durch diese Befehlsfolge wird a mit dem jeweiligen Inhalt b des Druckspeichers multipliziert und wieder durch b dividert.

(Formel: $\frac{a \cdot b}{b} = a$).

(a) im Druckspeicher

x

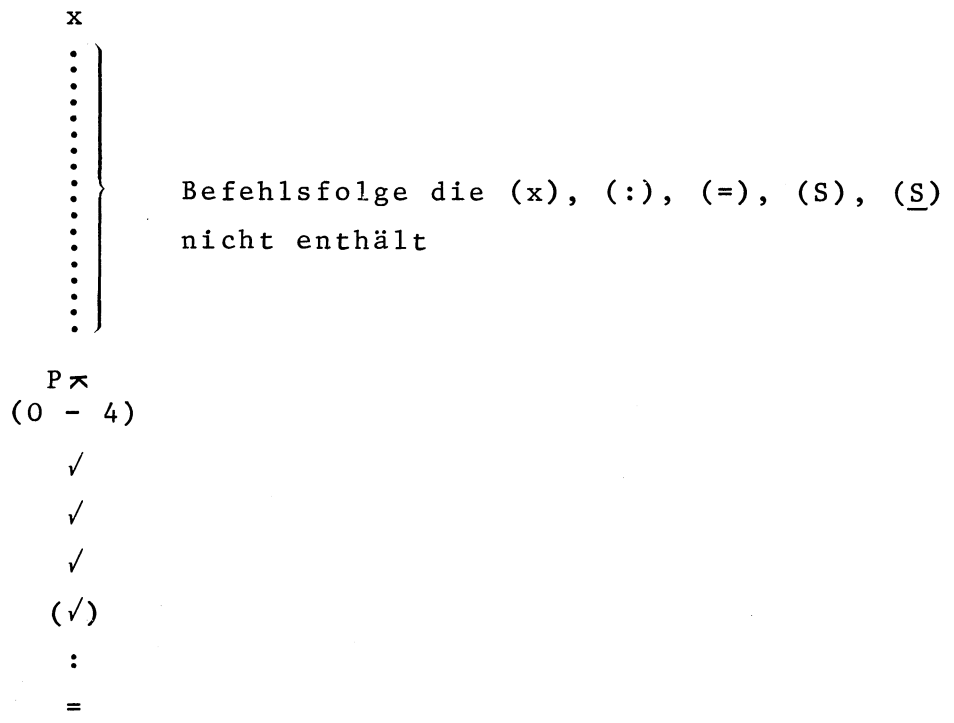
. } Befehlsfolge, die (x), (:), (=), (S), (S) nicht enthält
.
.
.
.
.
.

:

=

Eine mögliche "Division durch Null" kann ähnlich wie bei der "Programmierung der Zahl 1" umgangen werden, indem die Erweiterung und Kürzung mit einem der Zifferncodes von P_o bis P_4 erfolgt. Da aber die Multiplikation bzw. Division einer Zahl a mit einem 14- oder 15-stelligen Zifferncode sehr schnell zu einem falschen Ergebnis oder F - Druck führen kann, ist es zweckmäßig, die Stellenzahl des Zifferncodes zu reduzieren. In den meisten Fällen ist hierfür drei- bis viermaliges Quadratwurzelziehen ausreichend.

(a) im Druckspeicher



III. 7. Programmierte Rundung

DIEHL combitron =S= mit Sonderprogramm rundet bei der Multiplikation, sie rundet *nicht* bei der Division.

Es liege nun folgendes Problem vor:

Es sei p (p = 2 - 4 - 6 - 8) die Kommaposition, in welcher die Rechnung durchgeführt wird, und es sei K (K = 0, 1, 2, ..., 7; K < p) die K-te Nachkommastelle in welcher das Ergebnis gerundet werden soll.

Wie kann diese Rundung nun vollzogen (programmiert) werden?

Man *multipliziert* zunächst das zu rundende Ergebnis mit dem Rundungsfaktor $R = 10^{K-p}$ (K < p) und *dividiert* sodann durch R. Das Ergebnis ist dann in der K-ten Nachkommastelle gerundet.

Für die programmierte Rundung muß selbstverständlich R gespeichert werden.

Beispiel 9:

$2{,}5955\ 5500 \cdot 2{,}99 = 7{,}76070945$

$= 7{,}761$ (aufgerundet)

$p = 8$

$K = 3$

$R = 10^{3-8} = 10^{-5} = 0{,}00001$

Beispiel 10:

$2{,}595500 \cdot 2{,}99 = 7{,}760545$

$= 7{,}76$ (abgerundet)

$p = 6$

$K = 2$

$R = 10^{2-6} = 10^{-4} = 0{,}0001$

Lösungen:

Beispiel 9:

2,59 55 55	x	2 5 9 5 5 5 5 0 0 ×
		2 9 9 0 0 0 0 0 0 =
2,99	=	7 7 6 0 7 0 9 4 5 *
	x	7 7 6 0 7 0 9 4 5 ×
		0 0 0 0 0 1 0 0 0 :
0,00001	:	0 0 0 0 0 1 0 0 0 =
		7 7 6 1 0 0 0 0 0 *
	=	

Beispiel 10:

2,59 55	x	2 5 9 5 5 0 0 ×
		2 9 9 0 0 0 0 ×
2,99	x	0 0 0 1 0 0 :
		0 0 0 1 0 0 =
0,0001	:	7 7 6 0 0 0 0 *
	=	

III. 8. Abspalten der Vor- und Nachkommastellen

Es handelt sich um folgendes Problem:

Eine Zahl a, die sich im Druckspeicher befindet, soll in ihre Vor- und Nachkommastellen, jeweils einschließlich dem Vorzeichen der Zahl a zerlegt werden.

Anders formuliert:

Gesucht werden V und N in der Zerlegung a = V + N

(V = Vorkommastellen)
(N = Nachkommastellen)

Es gibt zwei Lösungsmethoden:

1. Methode:

Es bedeute p wieder die eingestellte Kommaposition:

(a)	S
	:
10^p	x
	S

(Danach steht V im Druckspeicher)

	*
	S

(Danach steht N im Druckspeicher)

2. Methode:

(a) :

10^P x

=

(Danach steht V im Druckspeicher)

=

:

10^P =

(Danach steht N im Druckspeicher)

Anmerkung:

Wie für die Kommaposition p = 8 zu verfahren ist, wird aus den Beispielen 12) und 13) ersichtlich. Man achte darauf, daß in Beispiel 13) (16 - stellige Zahl) der Abspaltungsfaktor anders gewählt ist als in Beispiel 12)(15-stellige Zahl). Bei 16-stelligen Zahlen kann für p = 8 nur die 1. Methode in einer modifizierten Form angewandt werden.

Beispiel 11:

a = 2345,678901 → V = 2345,
p = 6 → N = 0,678901
$10^p = 10^6 = 1\ 000\ 000$

Beispiel 12:

a = 9876678,54321012 → V = 9876678,
p = 8 → N = 0,54321012
$10^p = 10^8$ $10^8 - 10^{-8} = 9999\ 9999,9999\ 9999$

Beispiel 13:

a = -9999 9999,8888 8888 → V = -9999 9999,
p = 8 → N = -0,8888 8888
$10^p = 10^8 = 10^4 \cdot 10^4$

Lösungen:

Nach der 1. Methode

Beispiel 11

```
     2345,678901  ≤ a
     2345,678901 :
1000000,0000000 ×
1000000,0000000  ≦
     2345,000000 *  V

        0,678901 * ≤ N
```

Nach der 2. Methode

Beispiel 11

```
     2345,678901 : a
1000000,0000000 ×
1000000,0000000 =
     2345,000000 * V

  678901,000000 #
  678901,000000 :
1000000,0000000 =
        0,678901 * N
```

Beispiel 12

```
  98766678,54321012  ≤ a
  98766678,54321012 :
99999999,99999999 ×
99999999,99999999  ≦
  98766678,00000000 *  V

        0,54321012 * ≤ N
```

Beispiel 12

```
  98766678,54321012 : a
99999999,99999999 ×
99999999,99999999 =
  98766678,00000000 * V

54321012,09876678 #
54321012,09876678 :
99999999,99999999 =
        0,54321012 * N
```

Beispiel 13

```
99999999,88888888 - ≤ a
99999999,88888888 : -
     10000,00000000 :
     10000,00000000 ×
     10000,00000000 ×
     10000,00000000  ≦
99999999,00000000 * - V

        0,88888888 * ≦ N
```

III. 9. Speichersplittung

Speichersplittung bedeutet parallele Speicherung *zweier* (ggf. auch mehrerer) Ergebnisse bzw. auch Konstanten in ein und demselben Speicher. Was darunter zu verstehen ist und wie diese Speichersplittung vorgenommen wird, dürfte nach eingehendem Studium umseitiger Beispiele klar werden.

Darauf hingewiesen werden muß, daß diese Methode bei bestimmten Problemen den Vorteil bringt, daß weniger Speicherkapazität benötigt wird und der Programmablauf schneller ist, (vgl. Beispiel 14 und 14a). Wegen der Gefahr des Überlaufs der parallel gespeicherten Ergebnisse, ist der Wertebereich eingeengt, daher muß bei Anwendung der Speichersplittung die Größenordnung der Ergebnisse bekannt sein. Dementsprechend hat man auch den Erweiterungsfaktor E zu wählen. Es ist außerdem noch darauf zu achten, daß in ein und demselben Speicher nur Ergebnisse und Konstanten *gleichen Vorzeichens* untergebracht werden dürfen.

Als Grundsätzliches zu den Beispielen werde vorausgeschickt:

Es sollen beispielsweise die beiden Zahlen 3589 und 10256 im Ergebnisspeicher in der gesplitteten Form

3589 0000 000 10256

gespeichert werden:

Dazu geht man auf Kommaposition 0, multipliziert 3589 mit $E = 10^{12}$, bringt dieses Ergebnis in den Ergebnisspeicher und addiert 10256, also:

3589	X
10^{12}	S
10256	S
	◊
	S

Möchte man, umgekehrt, die Splittung wieder rückgängig machen, so gibt es hierfür zwei Methoden:

1. Methode:

◊
S
:
10^{12} =

(3589 im Druckspeicher)

x
10^{12} S
*
S

(10256 im Druckspeicher)

2. Methode:

*
S
:
10^{12} =

(3589 im Druckspeicher)

=

(10256 im Druckspeicher)

Beispiel 14:

Gegeben:

i	x_i	y_i
1	2	5
2	3	6
3	4	7

Gesucht:

$$\sum_{i=1}^{3} x_i = (9);\quad \sum_{i=1}^{3} x_i^2 = (29);\quad \sum_{i=1}^{3} y_i = (18);\quad \sum_{i=1}^{3} y_i^2 = (110)$$

Anmerkung:

Es empfiehlt sich, des besseren Verständnisses wegen, dieses Beispiel manuell durchzurechnen, indem man, nach Eingabe der Zahl 10^{10} in den Konstantenspeicher 9, die Befehlsfolge der Programmspeicher P_0, P_1 und P_2 (JO weglassen!) 3-mal durchläuft. Danach wird die Berechnung gemäß den Befehlen in P_3, P_4 und P_5 fortgesetzt.

In P_0, P_1 und P_2 befindet sich das Programm zur Berechnung der Summen, in P_3, P_4 und P_5 werden die Speichersplittungen ("Plus-Minus-Speicher" und "Ergebnisspeicher") wieder rückgängig gemacht und die Ergebnisse gedruckt.

Im "Plus-Minus-Speicher" befinden sich Σx_i (erweitert mit 10^{10}) und Σy_i, im "Ergebnisspeicher" Σx_i^2 (erweitert mit 10^{10}) und Σy_i^2.

Anwendungsbeispiel

Archiv-Nr.

Beispiel 14:

Programm- und Konstantenspeicher: Belegplan

P ≙ Programm ↙ ≙ Konstante

	0	1	2	3	4	5	6	7	8	9
P	P	P	P	P	P	P				
K										↙
										« 10^10 »

Programmierung

V≙ Variable B≙ Befehl

	0		1		2		3		4		5		6		7		8		9	
	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B
		P		P		P		P		P		P								
		0		1		2		3		4		5								
1	X;	(·)		=		⊆		◇		−		⊆								
2		#		×		J		:		⊆		*								
3		×		×		0		⊼		:		#								
4		⊼		⊼				g		⊼		⊆								
5		g		g				=		g		#								
6		=		⊆				#		=		J								
7		+	Y;					×		#		0								
8		:		#				⊼		×										
9		⊼		+				g		⊼										
10		g		×				=		g										
11		A		A		A		A		A		A								
12																				

Zifferncode	Kontrollstreifen und Bezeichnung

3 6 4 5 5 9 4 6 1 6 9 4 5 4 2 P ↙ 0

6 8 8 9 1 5 3 8 8 6 4 0 6 9 0 P ↙ 1

5 6 4 2 2 1 2 6 3 7 4 0 9 2 8 P ↙ 2

2 9 8 0 5 2 0 5 2 6 5 0 4 5 1 P ↙ 3

4 9 8 5 9 6 8 0 1 0 0 5 6 4 6 P ↙ 4

6 0 8 4 1 5 7 7 3 7 8 6 1 1 2 P ↙ 5

Programmausführung

Eingabe	
	*
	* S
1	=
10 000 000 000	↙
	g
	J
	0
2	A
5	A
3	A
6	A
4	A
7	A
	J
	3

```
                *
                * S
                #


1 0 0 0 0 0 0 0 0 0 0 0   ↙ 9

                2 # x1
                5 # y1
                3 # x2
                6 # y2
                4 # x3
                7 # y3


                9   A Σ xi
               29   A Σ xi²
               18   A Σ yi
              110   A Σ yi²
```

Anwendungsbeispiel

Archiv-Nr.

Beispiel 14 a:

Programm- und Konstantenspeicher: Belegplan

P ≙ Programm ↙ ≙ Konstante

	0	1	2	3	4	5	6	7	8	9
P	P	P	P	P	P	P	P			
K							↙	↙	↙	↙
							$\langle \Sigma y^2 \rangle$	$\langle \Sigma y \rangle$	$\langle \Sigma x^2 \rangle$	$\langle \Sigma x \rangle$

Programmierung

V≙ Variable B≙ Befehl

	0		1		2		3		4		5		6		7		8		9	
	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B
		P		P		P		P		P		P		P						
		0		1		2		3		4		5		6						
1	x_i	(·)		5		×		↙		↗		↗		↙						
2		#		*		5		7		9		6		6						
3		+		↙		↗		*5		#		#		J						
4		×		9		7		↙		↗		*		0						
5		5		*5		+		6		8		↙								
6		↗		↙		↗		J		#		9								
7		9		8		6		0		↗		↙								
8		+	y_i			5				7		8								
9		↗		#		*				#		↙								
10		8		+								7								
11		A		A		A		A		A		A		A						
12																				

Zifferncode	Kontrollstreifen und Bezeichnung

3 6 4 4 0 4 8,9 2 1 8 5 6 7 7 P π 0

5 7 2 9 9 8 2,6 8 9 2 2 2 2 1 P π 1

6 5 0 9 9 2 9,2 6 7 0 3 9 0 4 P π 2

1 5 4 1 7 6 8,1 1 3 3 5 6 8 0 P π 3

8 6 1 4 2 4,5 9 3 9 2 3 5 2 P π 4

8 2 8 5 1 1,4 1 9 1 3 9 8 0 P π 5

1 5 2 8 7 1 8,4 4 7 0 8 3 5 2 P π 6

Programmausführung

Eingabe	
	*
	5*
1,	=
0,	↙
	9
	↙
	8
	↙
	7
	↙
	6
	J
	0

0, 0 0 0 0 0 0 0 0 0 *

0, 0 0 0 0 0 0 0 0 0 * 5

0, 0 0 0 0 0 0 0 0 0 #

0, 0 0 0 0 0 0 0 0 0 ↙ 9
0, 0 0 0 0 0 0 0 0 0 ↙ 8
0, 0 0 0 0 0 0 0 0 0 ↙ 7
0, 0 0 0 0 0 0 0 0 0 ↙ 6

Eingabe		Kontrollstreifen und Bezeichnung
2,	A	
5,	A	
3,	A	
6,	A	
4,	A	
7,	A	
	J	
	4	

2,0 0 0 0 0 0 0 0 # X
5,0 0 0 0 0 0 0 0 # Y
3,0 0 0 0 0 0 0 0 # X
6,0 0 0 0 0 0 0 0 # Y
4,0 0 0 0 0 0 0 0 # X
7,0 0 0 0 0 0 0 0 # Y

9,0 0 0 0 0 0 0 0 A Σx_i
29,0 0 0 0 0 0 0 0 A Σx_i^2
18,0 0 0 0 0 0 0 0 A Σy_i
110,0 0 0 0 0 0 0 0 A Σy_i^2

Beispiel 15:

Es handelt sich um die gleiche Aufgabe wie in Beispiel 1, allerdings mit anderen Zahlen:

i	x_i	y_i
1	2,34	5,67
2	3,45	6,78
3	4,56	7,89

Ergebnisse:

$\Sigma x_i^2 = 38,1717$; $\Sigma y_i^2 = 140,3694$

$\Sigma x_i = 10,35$; $\Sigma y_i = 20,34$

Dieses Beispiel soll zeigen,

1.) wie verfahren werden kann, wenn mit Nachkommastellen gerechnet werden soll.

2.) wie die Programmierung vereinfacht werden kann, wenn die Reihenfolge der ausgedruckten Ergebnisse beliebig sein darf.

Anwendungsbeispiel

Archiv-Nr.

Beispiel 15:

Programm- und Konstantenspeicher: Belegplan

P ≙ Programm ↙ ≙ Konstante

	0	1	2	3	4	5	6	7	8	9
P	P	P	P	P	P					
K										↙
										« 10 »

Programmierung

V≙ Variable B≙ Befehl

	0		1		2		3		4		5		6		7		8		9	
	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B
		P		P		P		P		P										
		0		1		2		3		4										
1	X_i	(·)	Y_i	(·)		*		√		:										
2		#		#		:		=		↗										
3		+		S		↗		#		g										
4		×		×		g		S		√										
5		×		×		=		:		=										
6		↗		↗		#		↗		#										
7		g		g		=		g		J										
8		=		S		:		=		0										
9		+		J		↗		#												
10				0		g		=												
11		A		A		A		A		A										
12																				

Zifferncode	Kontrollstreifen und Bezeichnung

3 6 4 4 0 4 9 5 9 3 0 1 0 2 4 P ↘ 0

3 6 4 5 0 8 0 3 8 5 1 2 6 7 7 P ↘ 1

3 3 3 2 3 6 4 2 4 7 8 4 9 7 4 P ↘ 2

4 4 3 4 8 9 1 6 5 2 0 0 7 5 5 P ↘ 3

5 3 0 4 5 9 1 7 4 9 3 3 5 0 4 P ↘ 4

Programmausführung

Eingabe	
	*
	*S
1,	=
100 000 000,	↘
	9
	J
	0
2,34	A
5,67	A
3,45	A
6,78	A
4,56	A
7,89	A
	J
	2

0,0000 *

0,0000 *S

0,0000 #

100000000,0000 ↘9

2,3400 # x
5,6700 # y
3,4500 # x
6,7800 # y
4,5600 # x
7,8900 # y

38,1717 A Σx_i^2
10,3500 A Σx_i
140,3694 A Σy_i^2
20,3400 A Σy_i

III. 10. Fehlende Befehlskapazität

Es ist nützlich, zu wissen, wie man sich unter Umständen helfen kann, wenn ein Programm um wenige Befehle über die Befehlskapazität des Rechensystems geht.

Man läßt die Druckbefehle (#) für den automatischen Abdruck der eingegebenen Variablen im Programm weg und betätigt bei Programmausführung, nach Eingabe einer jeden Variablen, die [#] - Taste. Dadurch wird die Variable auf dem Konstrollstreifen gedruckt und kann danach durch die A - Taste in den Programmablauf einbezogen werden.

Dieses Verfahren hat überdies den Vorteil, daß nach Abdruck auf dem Kontrollstreifen, aber *vor* Übernahme mit der A - Taste, die eingetastete Zahl noch *kontrolliert* und *korrigiert* werden kann.

III. 11. Der bedingte Sprung

Wie aus der Bedienungsanleitung "DIEHL combitron =S=" bekannt ist (siehe Seite 32), erfolgt bei Programmausführung nach C J (0 - 9) der Sprung an den *Anfang* des Programmspeichers P (0 - 9) nur, falls die Zahl, die sich im Druckspeicher befindet, ≥ 0 ist. Andernfalls setzt DIEHL combitron =S= die Programmausführung linear fort.

Diese Möglichkeit, bedingte Verzweigungen im Programmablauf durchführen zu können, ist für viele Rechenprobleme der Praxis **von** außerordentlich großer Bedeutung. Die Vielfalt der möglichen Fälle läßt sich auf 4 Grundfälle zurückführen:

Grundfall 1:

a) Ist $x \geq 0$, so ist Rechenweg 1 zu durchlaufen ($y_1 = f_1(x)$ zu berechnen)

b) Ist $x < 0$, so ist Rechenweg 2 zu durchlaufen ($y_2 = f_2(x)$ zu berechnen)

Grundfall 2:

a) Ist $x > 0$, so ist Rechenweg 1 zu durchlaufen ($y_1 = f_1(x)$ zu berechnen)

b) Ist $x < 0$, so ist Rechenweg 2 zu durchlaufen ($y_2 = f_2(x)$ zu berechnen)

c) Ist $x = 0$, so ist Rechenweg 3 zu durchlaufen ($y_3 = f_3(x)$ zu berechnen)

Grundfall 3:

a) Ist $x > 0$, so ist Rechenweg 1 zu durchlaufen ($y_1 = f_1(x)$ zu berechnen)

b) Ist $x \leq 0$, so ist Rechenweg 2 zu durchlaufen ($y_2 = f_2(x)$ zu berechnen)

Grundfall 4:

a) Ist $x \neq 0$, so ist Rechenweg 1 zu durchlaufen ($y_1 = f_1(x)$ zu berechnen)

b) Ist $x = 0$, so ist Rechenweg 2 zu durchlaufen ($y_2 = f_2(x)$ zu berechnen)

Das Prinzip der Programmierung dieser vier "Grundfälle" wird auf den nächsten Seiten an graphischen Skizzen und einfachen Rechenbeispielen verdeutlicht.

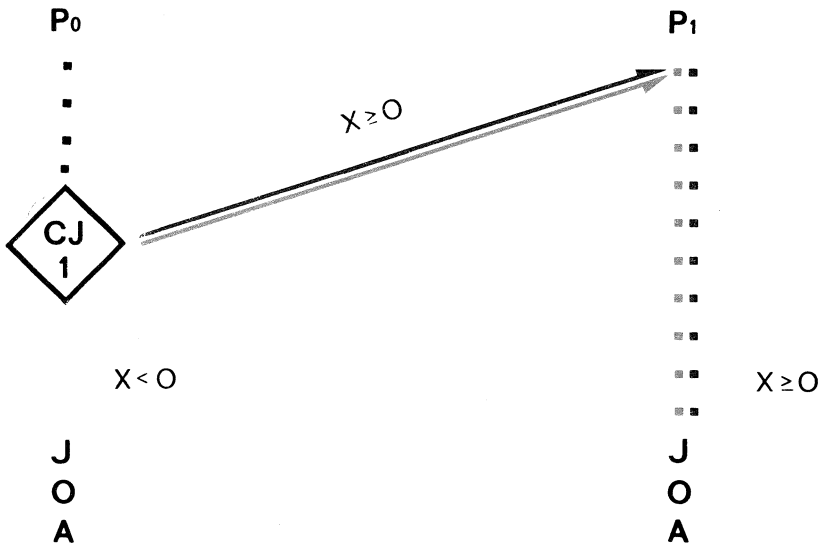
Der allgemeine Bedingungsfall $x \gtrless a$ läßt sich durch Bilden der Differenz $x - a \gtrless 0$ auf die Grundfälle zurückführen. (Vgl. hierzu Beispiel 23 und 24, in denen überdies einige Kunstgriffe angewandt wurden).

Tabellarische Übersicht:

Grundfall	Rechenweg 1 ($y_1 = f_1(x)$)	Rechenweg 2 ($y_2 = f_2(x)$)	Rechenweg 3 ($y_3 = f_3(x)$)
1	$x \geq 0$	$x < 0$	entfällt
2	$x > 0$	$x < 0$	$x = 0$
3	$x > 0$	$x \leq 0$	entfällt
4	$x \neq 0$	$x = 0$	entfällt

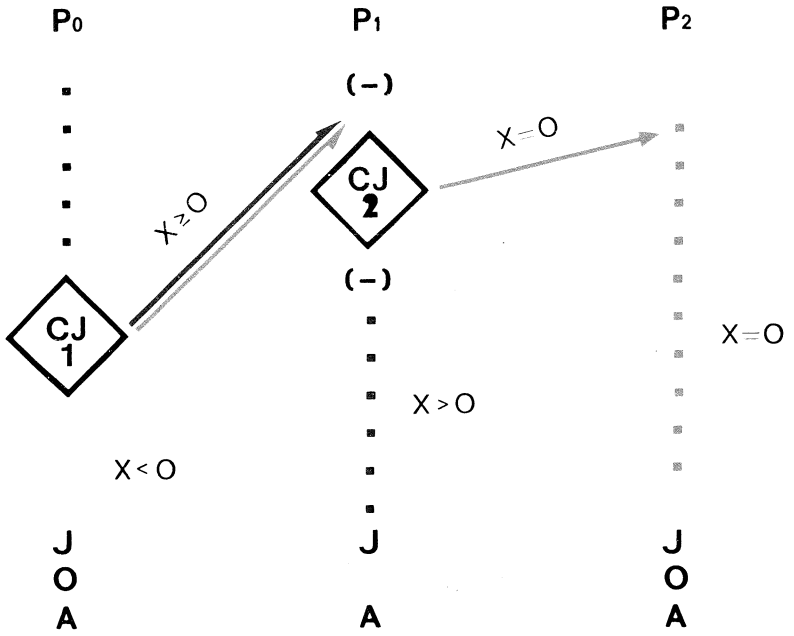
Grundfall 1:

(X≥O; X<O)



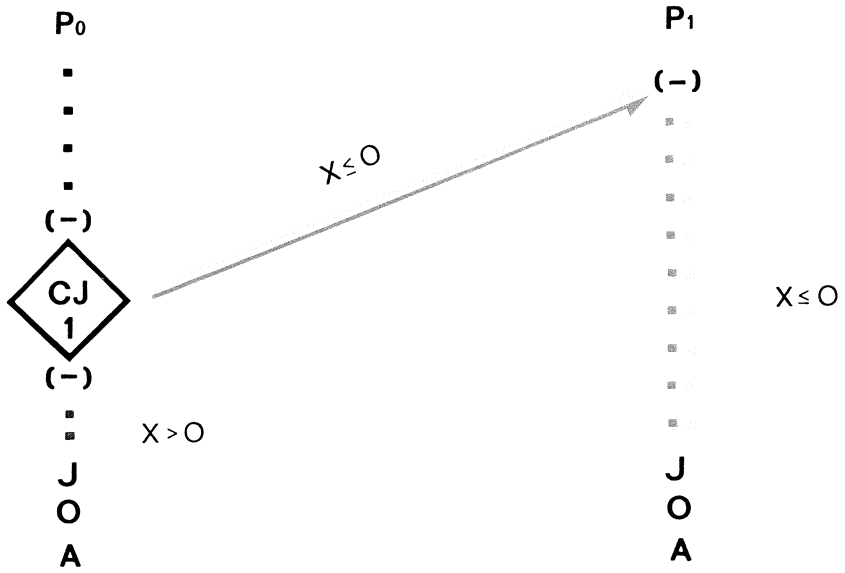
Grundfall 2:

(X>O; X<O; X=O)



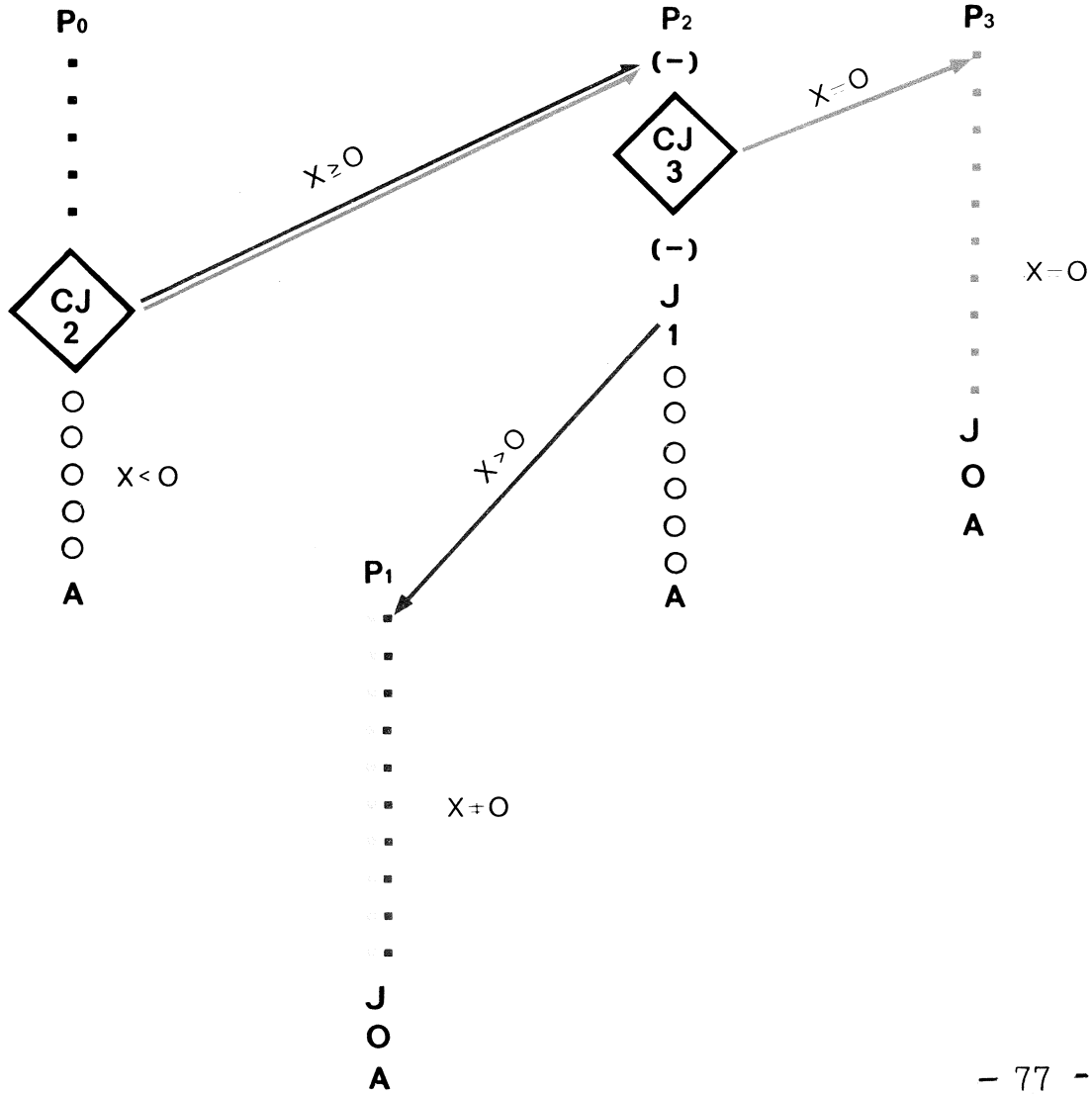
Grundfall: 3

(X>O; X≤O)



Grundfall 4:

(X=O; X=O)



Beispiel 16 :

Gegeben: x

Gesucht: $y_1 = x^2 \quad ; \quad (x < 0)$

$y_2 = \sqrt{x} \quad ; \quad (x \geq 0)$

Anwendungsbeispiel

Archiv-Nr.

Beispiel 16 :

Programm- und Konstantenspeicher: Belegplan

P ≙ Programm ↙ ≙ Konstante

	0	1	2	3	4	5	6	7	8	9
P	P	P								
K										

Programmierung

V≙ Variable B≙ Befehl

	0		1		2		3		4		5		6		7		8		9	
	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B
		P		P																
		0		1																
1	×	(·)		√																
2		#		#																
3		CJ		J																
4		1		0																
5		×																		
6		=																		
7		#																		
8		J																		
9		0																		
10																				
11		A		A																
12																				

Zifferncode	Kontrollstreifen und Bezeichnung

3 6 4 0 4 8 4 9 4 7 2,4 2 5 6 P = 0

4 3 4 3 4 6 8 2 1 4 1,9 0 0 0 P = 1

Programmausführung

Eingabe	
	✱
	✱ S
1,	=
	J
	0
4,	A
0,	A
-4,	A

0,0000 ✱

0,0000 ✱ S

0,0000 #

4,0000 # ×
2,0000 A $\sqrt{x}$

0,0000 # ×
0,0000 A $\sqrt{x}$

4,0000 # − ×
16,0000 A x^2

Beispiel 17 :

Gegeben: x

Gesucht:

$y_1 = x^2 \quad ; \quad (x < 0)$

$y_2 = \sqrt{x} \quad ; \quad (x > 0)$

$y_3 = 100 \quad ; \quad (x = 0)$

Anwendungsbeispiel

Archiv-Nr.

Beispiel 17 :

Programm- und Konstantenspeicher: Belegplan

P ≙ Programm ⌟ ≙ Konstante

	0	1	2	3	4	5	6	7	8	9
P	P	P	P							
K										⌟
										« 100 »

Programmierung

V≙ Variable B≙ Befehl

	0		1		2		3		4		5		6		7		8		9	
	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B
		P		P		P														
		0		1		2														
1	×	(·)		(–)		⌅														
2		#		C		9														
3		C		2		#														
4		1		(–)		J														
5		×		√		0														
6		=		#																
7		#		J																
8		J		0																
9		0																		
10																				
11		A		A		A														
12																				

Ziffercode	Kontrollstreifen und Bezeichnung

3 6 4 0 4 8 4 9 4 7 2,4 2 5 6 P ⨯ 0

2 1 4 6 5 2 1 4 2 2 6,1 2 4 8 P ⨯ 1

8 6 1 4 1 1 0 5 6 0,2 5 6 0 P ⨯ 2

Programmausführung

Eingabe	
	✱
	✱ ≤
1,	=
100,	⨯
	g
	J
	0
625,	A
0,	A
-625,	A

0,0000 ✱

0,0000 ✱ ≤

0,0000 #

100,0000 ⨯ 9

625,0000 # x
25,0000 A $\sqrt{x}$

0,0000 # x
100,0000 A

625,0000 # - x
390625,0000 A x^2

Beispiel 18:

Gegeben: x

Gesucht: $y_1 = x^2 + 100 \quad ; \quad (x \leq 0)$

$y_2 = \sqrt{x} \quad ; \quad (x > 0)$

Anwendungsbeispiel

Archiv-Nr.

Beispiel 18 :

Programm- und Konstantenspeicher: Belegplan

P ≙ Programm ↙ ≙ Konstante

	0	1	2	3	4	5	6	7	8	9
P	P	P								
K										↙
										« 100 »

Programmierung

V≙ Variable B≙ Befehl

	0		1		2		3		4		5		6		7		8		9	
	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B
		P		P																
		0		1																
1	X	(·)		X																
2		#		S																
3		(—)		⊼																
4		(J		g																
5		1		S																
6		(—)		s*																
7		√		#																
8		#		J																
9		J		0																
10		0																		
11		A		A																
12																				

Zifferncode	Kontrollstreifen und Bezeichnung

3 6 4 1 4 7 9 3 6 4 7,2 1 0 1 P × 0

6 5 0 9 9 5 1 8 0 0 7,8 2 4 0 P × 1

Programmausführung

Eingabe	
	✱
	✱ S
1,	=
100,	×
	g
	J
	0
16,	A
0,	A
-16,	A

0,0000 ✱

0,0000 ✱ S

0,0000 #

1000000 × 9

16,0000 # ×
4,0000 A $\sqrt{x}$

0,0000 # ×
1000000 A $x^2 + 100$

16,0000 # − ×
3560000 A $x^2 + 100$

Beispiel 19 :

Gegeben: x

Gesucht: $y_1 = \sqrt{|x|}$; ($x \neq 0$)

$y_2 = 100$; ($x = 0$)

Anwendungsbeispiel

Archiv-Nr.

Beispiel 19 :

Programm- und Konstantenspeicher: Belegplan

P ≙ Programm ⩗ ≙ Konstante

	0	1	2	3	4	5	6	7	8	9
P	P	P	P							
K										⩗
										« 100 »

Programmierung

V≙ Variable B≙ Befehl

	0		1		2		3		4		5		6		7		8		9	
	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B
		P		P		P														
		0		1		2														
1	×	(-)		(—)		⩗														
2		#		(]		9														
3		(]		2		#														
4		1		(—)		]														
5		(—)		√		0														
6				#																
7				]																
8				0																
9																				
10																				
11		A		A		A														
12																				

Zifferncode	Kontrollstreifen und Bezeichnung

3 6 4 0 4 8 0 7 1 7 8,6 4 9 6 P ≍ 0

2 1 4 6 5 2 1 4 2 2 6,1 2 4 8 P ≍ 1

8 6 1 4 1 1 0 5 6 0,2 5 6 0 P ≍ 2

Programmausführung

Eingabe	
	✱
	✱ ≤
1,	=
100,	≍
	9
	J
	0
16,	A
0,	A
-16,	A

0,0000 ✱

0,0000 ✱ ≤

0,0000 #

100,0000 ≍ 9

16,0000 # X
4,0000 A $\sqrt{x}$

0,0000 # X
100,0000 A

16,0000 # - X
4,0000 A $\sqrt{-x}$

Beispiel 20 :

Gegeben: x

Gesucht:

$$y_1 = x^2 \quad ; \quad (x \leq 0)$$

$$y_2 = \sqrt{x} \quad ; \quad (0 < x < 50)$$

$$y_3 = 2 + 0{,}5x \quad ; \quad (50 \leq x \leq 100)$$

$$y_4 = \frac{1}{x} \quad ; \quad (x > 100)$$

Anwendungsbeispiel

Archiv-Nr.

Beispiel 20 :

Programm- und Konstantenspeicher: Belegplan

P ≙ Programm ⩗ ≙ Konstante

	0	1	2	3	4	5	6	7	8	9
P	P	P	P	P	P	P	P			
K										⩗
										« 50»

Programmierung

V≙ Variable B≙ Befehl

	0		1		2		3		4		5		6		7		8		9	
	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B
		P		P		P		P		P		P		P						
		0		1		2		3		4		5		6						
1		#		S		*		π		J		◇		J						
2	×			ṡ		×		9		0		:		0						
3		#		(–)		=		S				S								
4		+		CJ		J		ṡ				S								
5		(–)		3		0		CJ				*								
6		CJ		ṡ				5				:								
7		2		*				*				ṡ								
8		S		√				:				S								
9		π		J				:				ṡ								
10		9		0				=												
11		A		A		A		A		A		A		A						
12																				

Ziffercode	Kontrollstreifen und Bezeichnung

3 9 8 4 1 5 3 2 9 7 3,8 8 3 0 P ⨯ 0

5 6 8 6 5 7 1 6 7 2 9,8 5 9 7 P ⨯ 1

3 3 7 1 0 4 6 3 4 6 4,2 4 3 2 P ⨯ 2

8 6 3 1 9 2 9 0 4 3,3 0 1 1 P ⨯ 3

4 0 6 8 1 9 3 0 2 2,7 7 1 2 P ⨯ 4

2 9 8 5 3 4 9 0 4 7 0,9 3 4 4 P ⨯ 5

4 0 6 8 1 9 3 0 2 2,7 7 1 2 P ⨯ 6

Programmausführung

Eingabe	
	*
	s*
1,	=
50,	⨯
	g
	J
	0
-10,	A
0,	A
16,	A
49,99999999	A
50,	A
100,	A
1000,	A

0,000000000 *

0,000000000 * 5

0,000000000 #

50,000000000 ⨯ 9

50,000000000 A

10,000000000 # − x
100,000000000 A x^2

0,000000000 # x
0,000000000 A x^2

16,000000000 # x
4,000000000 A $\sqrt{x}$

49,999999999 # x
7,071067811 A $\sqrt{x}$

50,000000000 # x
27,000000000 A $2 + 0{,}5x$

100,000000000 # x
52,000000000 A $2 + 0{,}5x$

1000,000000000 # x
0,001000000 A $\frac{1}{x}$

Beispiel 21 :

Gegeben: x

Gesucht:

$y_1 = x^2$; $(x < 0)$

y_2 = nicht definiert; drucke F; $(x = 0)$

$y_3 = \sqrt{x} + \frac{x}{4}$; $(0 < x \leq 4)$

$y_4 = \frac{x}{2} + 1$; $(x \geq 4)$

Anwendungsbeispiel

Archiv-Nr.

Beispiel 21 :

Programm- und Konstantenspeicher: Belegplan

P ≙ Programm ⌟ ≙ Konstante

	0	1	2	3	4	5	6	7	8	9
P	P	P	P	P	P	P	P	P		
K										⌟
										« 4, »

Programmierung

V≙ Variable B≙ Befehl

	0		1		2		3		4		5		6		7		8		9	
	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B
		P		P		P		P		P		P		P		P				
		0		1		2		3		4		5		6		7				
1	×	(,)		(—)		CJ		*		⁂		J		⌅		J				
2		#		CJ		4		#		:		0		9		0				
3		CJ		6		⁂		J		⌅				(—)						
4		1		—		:		0		9				√						
5		×		⊆		√				√				#						
6		=		⌅		+				⊆				CJ						
7		#		9		⌅				:				1						
8		J		—		9				⊆				×						
9		0		*		=				⁂				=						
10						+				#				#						
11		A		A		A		A		A		A		A		A				
12																				

Ziffercode	Kontrollstreifen und Bezeichnung
3 6 4 0 4 8 4, 9 4 7 2 4 2 5 6 P ⩵ 0 2 1 4 7 9 8 3, 2 9 9 1 9 7 7 6 P ⩵ 1 1 1 5 7 0 5 7, 6 1 5 7 7 5 8 1 P ⩵ 2 3 2 8 7 9 3 7, 0 5 1 5 2 5 1 2 P ⩵ 3 2 6 2 8 6 7 4, 5 0 8 3 9 2 7 5 P ⩵ 4 4 0 6 8 1 9, 3 0 2 2 7 7 1 2 P ⩵ 5 8 5 9 8 1 3, 2 2 7 5 8 7 6 3 P ⩵ 6 4 0 6 8 1 9, 3 0 2 2 7 7 1 2 P ⩵ 7	

Programmausführung

Eingabe	
	*
	*5
1,	=
4,	⩵
	9
	J
	0
- 10,	A
0,	A
4,	A
10,	A

0, 0 0 0 0 0 0 0 0 *

0, 0 0 0 0 0 0 0 0 * 5

0, 0 0 0 0 0 0 0 0 #

4, 0 0 0 0 0 0 0 0 ⩵ 9

10, 0 0 0 0 0 0 0 0 # − x
100, 0 0 0 0 0 0 0 0 A x^2

0, 0 0 0 0 0 0 0 0 # x
0, 0 0 0 0 0 0 0 0 F n.def.

4, 0 0 0 0 0 0 0 0 # x
3, 0 0 0 0 0 0 0 0 A $\sqrt{x} + \frac{x}{4}$

10, 0 0 0 0 0 0 0 0 # x
6, 0 0 0 0 0 0 0 0 A $\frac{x}{2} + 1$

III. 12. Stoppen eines Programmablaufs

A. Manuell

Ein Programmablauf läßt sich manuell durch Drücken der [#] - Taste unmittelbar unterbrechen. Der Ablauf kann auf Wunsch an dieser Stelle durch die A - Taste fortgesetzt werden.

Vor neuer Programmausführung sind die erforderlichen Speicher zu löschen und der Programmablauf ist mit J (0 - 9) zu starten.

B. Automatisch

Viel wichtiger jedoch ist der programmierte automatische Stopp.
Hierfür gibt es mehrere Möglichkeiten, von denen aber nur die vier gebräuchlichsten angeführt werden sollen:

1. Der normale Haltbefehl (vgl. II. 1. Abs. 3)

   Eine neue Programmausführung ist mit J (0 - 9) zu starten, wenn der Haltbefehl aus Kapazitätsgründen am Ende des Programms anstelle von J (0 - 9) programmiert wurde.

2. Division durch Null

   Diese Möglichkeit des programmierten Stopps wird vornehmlich bei zyklischen Programmen (Schleifen, die keine Eingabe von Variablen enthalten) angewandt, bei denen durch eine "Nullbedingung" der Programmablauf beendet werden kann, so z. B.

a) wenn die Anzahl n (n = 2, 3, ...) der Durchläufe im vorhinein bekannt ist. Man subtrahiert von n nach jedem Durchlauf die Zahl 1 und dividiert anschliessend die verbleibende Anzahl der Durchläufe durch sich selbst. Für n = 0 führt dies zu F - Druck und Stopp.

b) Bei Iterationsprogrammen $x_{n+1} = f(x_n)$, indem man $\frac{x_{n+1} - x_n}{x_{n+1} - x_n}$ untersucht. (x_{n+1} = (n+1)-te Näherung ; x_n = n-te Näherung). Stopp tritt ein, wenn

$x_{n+1} - x_n = 0$, d. h. $x_{n+1} = x_n$ ist.

c) Bei unendlichen, konvergenten Reihenentwicklungen $f(x) = \sum_{n=o}^{\infty} a_n x^n$, indem man das jeweils errechnete Glied $a_n x^n$ der Reihe durch sich selbst dividiert.

Erfolgt F - Druck und Stopp, so bedeutet dies, daß

$a_n x^n = 0$ und das Ergebnis f (x) im Rahmen der Stellenkapazität des Rechensystems erreicht ist.

Bei dieser Methode des F - Stoppes muß bei zyklischen Programmen das Endergebnis manuell aus dem entsprechenden Speicher abgerufen werden. Außerdem ist darauf zu achten, daß vor einer neuen Berechnung (auch bei nichtzyklischen Programmen und manueller Berechnung) der MD - Speicher durch 1,= *gelöscht* und die neue Programmausführung mit J (0 - 9) gestartet wird.

3. Quadratwurzel aus einer negativen Zahl

Eine neue Programmausführung ist mit J (0 - 9) zu starten.

4. Der bedingte Sprung

Der programmierte Stopp mittels CJ (0 - 9) ist die flexibelste der genannten Methoden, weshalb sie auch am meisten Anwendung findet, und zwar vornehmlich bei zyklischen Programmabläufen, die bedingungsabhängig (häufig: 0 - Bedingung) beendet werden können. Außer dem Vorteil, daß hier der automatische Stopp nicht an *eine* Bedingung gebunden ist, bietet der CJ (0 - 9) noch den Vorzug, daß nach automatischer Beendigung eines zyklischen Programmablaufs auch ein *automatischer Wechsel* in ein *anderes* Programm mit *gleichzeitigem Start* der *Programmausführung* möglich ist, was für die Einschaltung zyklischer Unterprogramme in ein Hauptprogramm von außerordentlich großer Bedeutung ist.

Und hier noch ein Hinweis:

Soll wieder (vgl. III. 12 / B. 2a)die Nullentscheidung für n $\longrightarrow$ 0 (lies: n gegen 0) getroffen werden, ist es zweckmäßiger -n $\longrightarrow$ 0 zu programmieren, da diese Entscheidung dann durch *einen* CJ (o - 9) erfaßt wird, im Gegensatz zu n $\longrightarrow$ 0, wofür bekanntlich *zwei* bedingte Sprungbefehle und mehr Speicherkapazität erforderlich wären.

III. 13. Ermittlung des Zifferncodes

1. Automatisch

Obwohl die automatische Zifferncodeermittlung bereits hinreichend bekannt ist, soll diese der Vollständigkeit wegen nochmals genannt werden.

Lernt man in bekannter Weise eine Befehlsfolge über Funktionstasten in einem Programmspeicher, so erstellt DIEHL combitron =S= mit Sonderprogramm hierfür automatisch einen Zifferncode. Dieser kann nach Abschluß des Lernvorgangs mit P ⩛ (0 - 9) abgerufen werden (siehe Beispiel 22).

2. Manuell

Der Zifferncode einer Befehlsfolge eines Programmspeichers läßt sich manuell nach der Formel

$$\text{(F 1):} \qquad \text{Zifferncode} = \sum_{p=1}^{10} a_p \cdot 32^{10-p} \cdot 10^{-K} \; ; \; (K = 0, 2, .., 8)$$

mit p = Position des Befehls im jeweiligen Programmspeicher

a_p = Kennzahl des p-ten Befehls (Kennzahl: siehe Decodierprogramm, Seite 44)

K = eingestellte Kommaposition

berechnen (siehe Beispiel 22). Dazu muß man auf dem Rechensystem die Kommaposition K = 0 einstellen!

Weit wichtiger und interessanter jedoch sind die drei Folgerungen, die sich aus (F 1) ergeben.

1. Die Möglichkeit der Splittung der Doppelbefehle ⩡ (0 - 9), ⩠ (0 - 9), J (0 - 9) und CJ (0 - 9). (Siehe III. 14 und Bedienungsanleitung, Seite 28 und 30.)

2. Die Möglichkeit die Programmspeicher P_0 bis P_4 im internen Programmablauf als Konstantenspeicher einzusetzen (siehe III. 15).

3. Die Möglichkeit der Adressenmodifikation (siehe III. 16).

Beispiel 22:

Programmiert man die Befehlsfolge der Geradengleichung (siehe Beispiele 3 und 8) über die Funktionstasten, so erhält man die Zifferncodes:

3 2 8 0 4 6 7.2 8 0 6 7 0 7 2 P ↑ 0

3 6 4 5 5 9 4.5 8 3 8 6 1 5 5 P ↑ 1

4 1 7 8 1 4.4 1 8 5 5 4 8 8 P ↑ 2

Beispiel 23:

Wir führen jetzt zur Kontrolle die Berechnung der Zifferncodes für das Programm "Geradengleichung" nach (F 1) durch:

P 0

✱	=	$9 \cdot 32^9$	=	316 659 348 799 488	
Stopp	=	$10 \cdot 32^8$	=	10 995 116 277 760	
#	=	$11 \cdot 32^7$	=	377 957 122 048	
+	=	$13 \cdot 32^6$	=	139 586 643 712	
Stopp	=	$10 \cdot 32^5$	=	335 544 320	
#	=	$11 \cdot 32^4$	=	11 534 336	
⩗	=	$4 \cdot 32^3$	=	131 072	
9	=	$14 \cdot 32^2$	=	14 336	
A			Σ =	328 046 728 067 072	P ⩗ 0

P 1

Stopp	=	$10 \cdot 32^9$	=	351 843 720 888 320	
#	=	$11 \cdot 32^8$	=	12 094 627 905 536	
x	=	$18 \cdot 32^7$	=	618 475 290 624	
⩙	=	$2 \cdot 32^6$	=	2 147 483 648	
9	=	$14 \cdot 32^5$	=	469 762 048	
S	=	$16 \cdot 32^4$	=	16 777 216	
◊	=	$8 \cdot 32^3$	=	262 144	
S	=	$16 \cdot 32^2$	=	16 384	
✱S	=	$7 \cdot 32^1$	=	224	
#	=	$11 \cdot 32^0$	=	11	
A			Σ =	364 559 458 386 155	P ⩗ 1

P 2

J	=	$1 \cdot 32^9$	=	35 184 372 088 832	
l	=	$6 \cdot 32^8$	=	6 597 069 766 656	
A			Σ =	41 781 441 855 488	P ⩗ 2

III. 14. Das Splitten der Doppelbefehle

Bekanntlich können die Befehle ↙, ↗, J und CJ bei DIEHL combitron =S= mit Sonderprogramm nicht als 10. Befehl in einem Programmspeicher über die entsprechende Funktionstaste gelernt werden. Um dies zu erreichen, muß man die Zifferncode - Darstellung einer Befehlsfolge zu Hilfe nehmen.

Grundsätzlich könnte man den Zifferncode nach (F 1) ausrechnen. Praktischer jedoch ist folgende Methode:

Man programmiert zunächst die ersten 9 Befehle, ruft den automatisch erstellten Zifferncode ab und addiert eine "Korrekturzahl". Den neuen Zifferncode speichert man wieder im entsprechenden Programmspeicher ab (Beispiele 24 und 25).

Die "Korrekturzahlen" sind:

J	≙	1
↗	≙	2
CJ	≙	3
↙	≙	4

d. h., die "Korrekturzahlen" sind gerade die den Befehlen zugeordneten Kennzahlen, was sofort aus (F 1) folgt, da für p = 10

(F 2): $a_{1o} \cdot 32^{1o-1o} = a_{1o} \cdot 32^{o} = a_{10}$ ist.

Zu beachten ist,

1. daß diese "Korrekturzahlen" für die Addition *ohne Kommataste* einzugeben sind.
2. daß die Befehle $\leq$, $\geq$, J und CJ auch mit dieser Methode am *Ende von P_9 nicht gelernt* werden können.

Die zu diesen Befehlen jeweils erforderliche Adresse (0 - 9) wird am Anfang des *nächstfolgenden* Programmspeichers programmiert, indem *nicht die Adresse als solche, sondern der dieser Adresse zugeordnete Befehl* gelernt wird.

Es bestehen die Zuordnungen:

Adresse	——>	Befehl	
0	——>	$\mathring{S}$	
1	——>	(-)	
2	——>	$\dot{S}$	
3	——>	◇	
4	——>	*	
5	——>	Stop	(Zahleneingabe)
6	——>	#	
7	——>	$\sqrt{\ }$	
8	——>	+	
9	——>	-	
10	——>	:	
11	——>	S	
12	——>	$\underline{S}$	
13	——>	X	
14	——>	=	

Was die Adressen 10 bis 14 bedeuten wird in III. 15 erklärt. Der Vorteil dieser Möglichkeit der Splittung der Doppelbefehle, d. h., Befehl und Adresse in verschiedenen Programmspeichern lernen zu können, ist offensichtlich. Man verliert keine Befehlskapazität, wenn an letzter Stelle in einem Programmspeicher ein Doppelbefehl gelernt werden muß (die Splittung ist allerdings nur dann sinnvoll, wenn dadurch die Programmierung weniger Programmspeicher erfordert). Darüberhinaus ergibt sich manchmal noch ein anderer Vorteil.

Wird nämlich der Anfang des Programmspeichers, in welchem die Adresse (0 - 9) in Form des zugeordneten Befehls gelernt wurde, durch J (0 - 9) oder CJ (0 - 9) angesprungen, so liest das Rechensystem den programmierten Befehl. Nur wenn das Rechensystem nach dem Überlaufprinzip vom *vorangehenden* Programmspeicher an den Befehl kommt, faßt es diesen als die zu ↙, ↗, J oder CJ gehörige Adresse auf.

Mitunter kommt es nun vor, daß für zwei verschiedene Rechenabläufe ein Teil einer Befehlsfolge der gleiche ist. Einmal *mit* dem Befehl, das andere Mal *ausschließlich* dem Befehl, aber mit dem Befehl als Adresse. Was damit im Einzelnen gemeint ist, wird nach dem Studium der Beispiele 24 und 25 verständlich werden.

Beispiel 24:

Dieses Programm ist ein Beispiel für das Splitten der Doppelbefehle (siehe Programmierung am Ende von P_5 bzw. am Anfang von P_6). Kommt das Rechensystem von P_5 an den Anfang von P_6, so wird die * als Adresse 4 gelesen und J 4 durchgeführt. Erfolgt dagegen durch CJ 6 (in P_4) der Sprung an den Anfang von P_6, so werden der Befehl * und alle nachfolgenden Befehle in P_6 durchgeführt.

Anmerkung:

Hier wurde durch die Splittung von J 4 ein Programmspeicher eingespart (vgl. Beispiel 24 a).
Man achte ferner auf die Schreibweise für die Splittung von Doppelbefehlen, da sich diese gegebenenfalls in unseren Archivprogrammen wieder findet.

Anwendungsbeispiel

Archiv-Nr.

Beispiel 24

$$y = a^{b}$$

Programm- und Konstantenspeicher: Belegplan

P ≙ Programm ↙ ≙ Konstante

	0	1	2	3	4	5	6	7	8	9
P	P	P	P	P	P	P	P			
K								↙	↙	↙
								« 0,5 »		

Programmierung

V≙ Variable B≙ Befehl

	0		1		2		3		4		5		6		7		8		9	
	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B
		P		P		P		P		P		P		P						
		0		1		2		3		4		5		6						
1	a	(·)		CJ		ŝ		S		(—)		⊼		[4] *						
2		#		4		—		⊼		CJ		9		*S						
3		↙		—		◇		8		6		×		⊼						
4		8		⊼		CJ		√		—		⊼		9						
5		:		8		5		↙		J		8		#						
6		S		:		*S		8		2		=		J						
7		↙		:		+		J				↙		0						
8		9		=		×		2				9								
9	b	(·)		↙		⊼						*								
10		#		8		7						[J]								
11		A		A		A		A		A		A		A						
12																				

Zifferncode	Kontrollstreifen und Bezeichnung

```
3 6 4 0 9 0 2,6 6 6 3 0 4 7 5 P π 0

1 1 5 9 3 2 3,5 7 1 8 4 6 5 3 P π 1

1 9 1 5 9 3 4,6 5 6 9 4 2 8 4 P π 2

5 6 5 6 0 8 6,8 6 0 6 6 6 8 8 P π 3

2 1 4 7 9 7 7,9 7 8 1 8 3 6 8 P π 4

  8 6 3 8 2 9,8 6 0 1 7 0 5 6 P π 5
  8 6 3 8 2 9,8 6 0 1 7 0 5 6   +
          0,0 0 0 0 0 0 0 1   +
  8 6 3 8 2 9,8 6 0 1 7 0 5 7 *

  8 6 3 8 2 9,8 6 0 1 7 0 5 7 P ↙ 5

3 2 4 4 4 0 0,5 2 3 6 7 3 6 0 P π 6
```

Programmausführung

Eingabe	
	*
	s*
1,	=
0,5	↙
	7
	J
	0
a = 2,	A
b = 0,5	A
a = 2,	A
b = 0	A
a = 2,	A
b = -0,5	A

```
0,0 0 0 0 0 0 0 0 *

0,0 0 0 0 0 0 0 0 * S

0,0 0 0 0 0 0 0 0 #
0,5 0 0 0 0 0 0 0   ↙ 7
2,0 0 0 0 0 0 0 0 # a
0,5 0 0 0 0 0 0 0 # b
1,4 1 4 2 1 3 5 6   A
2,0 0 0 0 0 0 0 0 # a
0,0 0 0 0 0 0 0 0 # b
1,0 0 0 0 0 0 0 0   A
2,0 0 0 0 0 0 0 0 # a
0,5 0 0 0 0 0 0 0 # -b
0,7 0 7 1 0 6 7 8   A
```

Anwendungsbeispiel	$y = a^b$
Archiv-Nr.	
Beispiel 24 a	

Programm- und Konstantenspeicher: Belegplan

P ≙ Programm ↙ ≙ Konstante

	0	1	2	3	4	5	6	7	8	9
P	P	P	P	P	P	P	P	P		
K								↙	↙	↙
								« 0,5 »		

Programmierung

V≙ Variable B≙ Befehl

	0		1		2		3		4		5		6		7		8		9	
	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B
		P		P		P		P		P		P		P		P				
		0		1		2		3		4		5		6		7				
1	a	(·)		CJ		S		S		(—)		⊼		J		*				
2		#		4		—		⊼		CJ		9		4		S				
3		↙		—		◇		8		7		×				⊼				
4		8		⊼		CJ		√		—		⊼				9				
5		:		8		5		↙		J		8				#				
6		S		:		S		8		2		=				J				
7		↙		:		+		J				↙				0				
8		9		=		×		2				9								
9	b	(·)		↙		⊼						*								
10		#		8		7														
11		A		A		A		A		A		A		A		A				
12																				

Zifferncode	Kontrollstreifen und Bezeichnung

3 6 4 0 9 0 2.6 6 6 3 0 4 7 5 P π 0

1 1 5 9 3 2 3.5 7 1 8 4 6 5 3 P π 1

1 9 1 5 9 3 4.6 5 6 9 4 2 8 4 P π 2

5 6 5 6 0 8 6.8 6 0 6 6 6 8 8 P π 3

2 1 4 8 3 2 1.5 7 5 5 6 7 3 6 P π 4

8 6 3 8 2 9.8 6 0 1 7 0 5 6 P π 5

4 5 0 7 9 9.7 6 7 3 8 8 1 6 P π 6

3 2 4 4 4 0 0.5 2 3 6 7 3 6 0 P π 7

Programmausführung

Eingabe	
	*
	*S
I,	=
0,5	↙
	7
	J
	0
a = 2,	A
b = 0,5	A
a = 2,	A
b = 0	A
a = 2,	A
b = -0,5	A

0.0 0 0 0 0 0 0 0 *

0.0 0 0 0 0 0 0 0 * S

0.0 0 0 0 0 0 0 0 #
0.5 0 0 0 0 0 0 0 ↙ 7

2.0 0 0 0 0 0 0 0 # a
0.5 0 0 0 0 0 0 0 # b
1.4 1 4 2 1 3 5 6 A
2.0 0 0 0 0 0 0 0 # a
0.0 0 0 0 0 0 0 0 # b
1.0 0 0 0 0 0 0 0 A
2.0 0 0 0 0 0 0 0 # a
0.5 0 0 0 0 0 0 0 # - b
0.7 0 7 1 0 6 7 8 A

Beispiel 25:

$$Z = \sqrt{|x^2 - y|}$$

Die Programmierung dieses einfachen Beispiels soll zeigen, wie manchmal durch *"vorzeitige"* Splittung eines Doppelbefehls der gewünschte Effekt, möglichst wenig Programmspeicher für eine Programmierung zu verwenden, gleichfalls erzielt werden kann.

Erklärung:

Wir sprechen von *"vorzeitiger"* Splittung eines Doppelbefehls,

1. wenn *innerhalb* eines Programmspeichers einer der Befehle ↙ , ↗ , J oder CJ an p-ter Stelle steht (p = 1, 2, ..., 9), und *in diesem* Programmspeicher *kein anderer* Befehl mehr *folgt*.

2. wenn die jeweils zu diesen Befehlen gehörige Adresse (0 - 14) in Form des ihr zugeordneten Befehls am *Anfang* des *nächstfolgenden* Programmspeichers gelernt wird.

Bei unserem gewählten Beispiel gewinnt man durch die vorzeitige Splittung des Befehls CJ 1 gegenüber der üblichen Programmierung (vgl. Beispiel 25 a) *einen* Programmspeicher.

Den Zifferncode für die Programmierung in P_o erhält man, indem man wieder zunächst die ersten 8 Befehle in bekannter Weise lernt, den hierfür automatisch erstellten Zifferncode abruft ($P \nearrow 0$) und die "Korrekturzahl"

$$3 \cdot 32^1 = 96$$

addiert. Dies ergibt den richtigen Zifferncode für P_o.

Anwendungsbeispiel

Archiv-Nr.

Beispiel 25

$$Z = \sqrt{|x^2 - y|}$$

Programm- und Konstantenspeicher: Belegplan

P ≙ Programm ↙ ≙ Konstante

	0	1	2	3	4	5	6	7	8	9
P	P	P								
K										

Programmierung

V≙ Variable B≙ Befehl

	0		1		2		3		4		5		6		7		8		9	
	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B
		P		P																
		0		1																
1	X	(·)		[1] (-)																
2		#		(—)																
3		×		√																
4		S		#																
5	Y	(·)		J																
6		#		0																
7		S																		
8		S*																		
9		[CJ]																		
10																				
11		A		A																
12																				

Zifferncode	Kontrollstreifen und Bezeichnung

```
3 6 4 5 7 4 3,5 1 5 9 6 5 4 4 P ≍ 0
3 6 4 5 7 4 3,5 1 5 9 6 5 4 4   +
          0,0 0 0 0 0 0 9 6   +
3 6 4 5 7 4 3,5 1 5 9 6 6 4 0 *

3 6 4 5 7 4 3,5 1 5 9 6 6 4 0 P ≍ 0

2 1 8 1 2 7 4,6 9 1 1 7 4 4 0 P ≍ 1
```

Programmausführung

Eingabe	
	*
	*5
1,	=
	J
	0
x = 5,	A
y = 9,	A
x = 4,	A
y = 25,	A

```
0,0 0 0 0 0 0 0 0 *

0,0 0 0 0 0 0 0 0 * 5

0,0 0 0 0 0 0 0 0 #


5,0 0 0 0 0 0 0 0 # x
9,0 0 0 0 0 0 0 0 # y
4,0 0 0 0 0 0 0 0   A
4,0 0 0 0 0 0 0 0 # x
25,0 0 0 0 0 0 0 0 # y
3,0 0 0 0 0 0 0 0   A
```

Anwendungsbeispiel

Archiv-Nr.

Beispiel 25 a

$$z = \sqrt{|x^2 - y|}$$

Programm- und Konstantenspeicher: Belegplan

P ≙ Programm K ≙ Konstante

	0	1	2	3	4	5	6	7	8	9
P	P	P	P							
K										

Programmierung

V≙ Variable B≙ Befehl

	0		1		2		3		4		5		6		7		8		9	
	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B
		P		P		P														
		0		1		2														
1	X	(·)		(−)		√														
2		#				#														
3		×				J														
4		S				0														
5	y	(·)																		
6		#																		
7		S																		
8		S*																		
9		CJ																		
10		2																		
11		A		A		A														
12																				

Zifferncode	Kontrollstreifen und Bezeichnung

3 6 4 5 7 4 3,5 1 5 9 6 6 4 7 P ≍ 0

2 1 1 1 0 6 2,3 2 5 3 2 9 9 2 P ≍ 1

4 3 4 3 4 6 8,2 1 4 1 9 0 0 8 P ≍ 2

Programmausführung

Eingabe	
	*
	*S
1,	=
	J
	0
x = 5,	A
y = 9,	A
x = 4,	A
y = 25,	A

0,0 0 0 0 0 0 0 0 *

0,0 0 0 0 0 0 0 0 * S

0,0 0 0 0 0 0 0 0 #

5,0 0 0 0 0 0 0 0 # X
9,0 0 0 0 0 0 0 0 # Y
4,0 0 0 0 0 0 0 0 A
4,0 0 0 0 0 0 0 0 # X
25,0 0 0 0 0 0 0 0 # Y
3,0 0 0 0 0 0 0 0 A

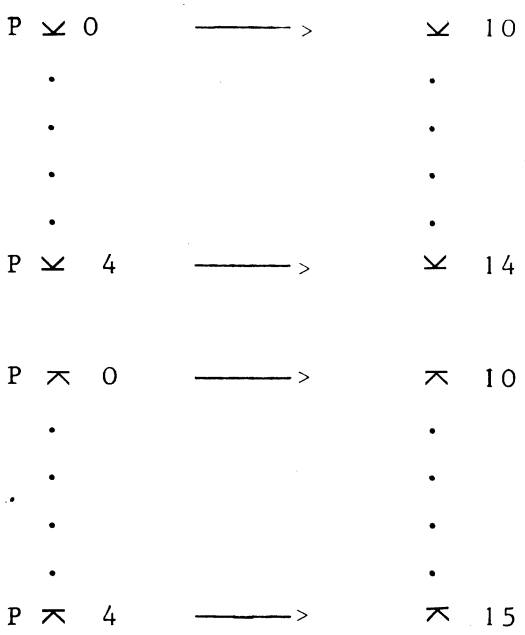
III. 15. Die Verwendung der Programmspeicher P_o bis P_4 als Konstantenspeicher im internen Programmablauf.

Im *internen* Programmablauf können *nur die Programmspeicher* P_o bis P_4 wie die Konstantenspeicher ⩡ (0 - 9) eingesetzt werden, d. h., in diese 5 Programmspeicher können mit internem Programm Zahlen eingespeichert oder Zahlen, die sich in den Programmspeichern befinden, abgerufen werden.

Mit der Programmeingabe über Funktionstasten können P ⩡ (0 - 4) bzw. P ⩠ (0 - 4) wohl *programmiert*, aber *nicht gelernt* werden. DIEHL combitron =S= mit Sonderprogramm lernt statt P⩡(0 - 4) nur ⩡ (0 - 4) bzw. statt P⩠(0 - 4) nur ⩠ (0 - 4). Um P⩡(0-4) bzw. P ⩠ (0-4) lernen zu können, muß man sich wieder der Zifferncode - Darstellung einer Befehlsfolge eines Programmspeichers bedienen.

Um zu erklären, wie die Programmierung zu geschehen hat, führen wir zunächst folgende Zuordnungen (——>) ein.

Programmspeicher	0 ——>	Konstantenspeicher	10
"	1 ——>	"	11
"	2 ——>	"	12
"	3 ——>	"	13
"	4 ——>	"	14



Praktisch verfährt man nun wie folgt:

Man programmiert an Stelle von ↙ (10 - 14) bzw. ↗ (10 - 14) nur ↙ (0 - 4) bzw. ↗ (0 - 4), ruft dann den automatisch erstellten Zifferncode ab und addiert eine oder auch mehrere "Korrekturzahlen", je nachdem, ob in der Befehlsfolge eines Programmspeichers ein oder mehrere Befehle ↙ (10 - 14) bzw. ↗ (10 - 14) enthalten sind. Den neuen Zifferncode bringt man wieder in den entsprechenden Programmspeicher.

Die "Korrekturzahl" ist abhängig von der Position p der zu modifizierenden Adresse, an welcher diese in der Befehlsfolge eines Programmspeichers steht (siehe III. 16)

Die "Korrekturzahlen" sind:

Position p	Korrekturzahl
1	351 843 720 888 320
2	10 995 116 277 760
3	343 597 383 680
4	10 737 418 240
5	335 544 320
6	10 485 760
7	327 680
8	10 240
9	320
10	10

Für den mathematischen interessierten Leser sei hier noch gesagt, daß sich die eben angegebenen "Korrekturzahlen", wegen der Zuordnungen P ≮ 0 —> ≮ 10 usw., wieder aus (F 1) ergeben:

Berechnet man den Zifferncode nach (F 1), so ist für die Adresse (10 - 14) in der Position p die Zahl

$$(a_p + 10) \cdot 32^{10-p}$$

zu addieren, wobei a_p die Kennzahl der an p-ter Stelle stehenden Adresse (0 - 4) ist. Da aber durch Eingabe der Adresse (0 - 4) über Funktionstasten $a_p \cdot 32^{10-p}$ bereits automatisch berechnet worden ist, muß als Korrekturzahl noch

$$10 \cdot 32^{10-p}$$

manuell addiert werden.

Für *alle* zu modifizierenden Adressen *eines* Programmspeichers ergibt sich somit die

(F 3): $$\text{Korrekturzahl} = \sum_p 10 \cdot 32^{10-p}$$ (p = Position der zu modifizierenden Adresse innerhalb eines Programmspeichers)

Beispiel 26:

Es soll etwa der Inhalt von P_4 gedruckt werden:

Man programmiert zunächst:

P 0:

(·) = Stopp, damit der Programmablauf automatisch stoppt

↑ anstelle von $P\uparrow_4 = \uparrow_{14}$
4
#
J
0

A

Sodann addiert man zu dem zugeordneten Zifferncode, wegen p = 3, die

Korrekturzahl = 343 597 383 680.

Der neue Zifferncode stellt das gewünschte Programm dar. Wir testen dies, indem wir beispielsweise manuell die Zahl 10 in ↙ 4, und -200 in P ↙ 4 eingeben.

Zifferncode	Kontrollstreifen und Bezeichnung
3 5 4 3 6 3 8 3 1 7 4 6 5 6 0 P ⌅ 0 3 5 4 3 6 3 8 3 1 7 4 6 5 6 0 + 3 4 3 5 9 7 3 8 3 6 8 0 + 3 5 4 7 0 7 4 2 9 1 3 0 2 4 0 *	
3 5 4 7 0 7 4 2 9 1 3 0 2 4 0 P ⌄ 0	

Programmausführung

Eingabe	
	*
	* 5
1,	=
	J
	0
-200,	P⌄
	4
10,	⌄
	4
	A

0,0 0 0 0 0 0 0 0 *

0,0 0 0 0 0 0 0 0 * ≤

0,0 0 0 0 0 0 0 0 #
2 0 0,0 0 0 0 0 0 0 0 = ⌄ 4
1 0,0 0 0 0 0 0 0 0 ⌄ 4
2 0 0,0 0 0 0 0 0 0 - A

III. 16. Adressenmodifikation

Unter Adressenmodifikation wollen wir die Änderung einer Adresse nach den Befehlen ⩡ , ⩠ , J, CJ durch interne Programmierung verstehen.

Beispiel 27:

Wir betrachten zunächst den einfachsten Fall, daß die zu modifizierende Adresse an 10-ter Stelle des Programmspeichers i (i = 0, ..., 4) steht.

Es soll $a^2 + b$ wahlweise nach ↙ 0, ..., ↙ 9 weggespeichert und die Konstantenspeicheradresse mit A - Symbol gedruckt werden.

Erläuterungen:

1. Bei Programmeingabe über Funktionstasten wird in P_1 an letzter Stelle ↙ 0 gelernt.

2. Soll bei Programmausführung ↙ j (j = 0, ..., 9) durchgeführt werden, so muß nach (F 1) zum Zifferncode von P_1 als "Korrekturzahl" $j \cdot 10^{-k}$ (K = 0, 2, ..., 8, ; eingestellte Kommaposition) addiert und der neue Zifferncode nach P_1 weggespeichert werden. Diese Addition und Wegspeicherung ist in P_o programmiert.

3. Wird die Adresse j in der Kommaposition K *ohne Kommataste* eingegeben, so ist dies gleichbedeutend mit $j \cdot 10^{-K}$.

Anwendungsbeispiel

Archiv-Nr.

Beispiel 27

Programm- und Konstantenspeicher: Belegplan

P ≙ Programm ↙ ≙ Konstante

	0	1	2	3	4	5	6	7	8	9
P	P	P	P	↙						
				Zi Co v. P_1						
K										

Programmierung

V≙ Variable B≙ Befehl

	0		1		2		3		4		5		6		7		8		9	
	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B
		P		P		P														
		0		1		2														
1	j+0 K(·)		a	(·)		J														
2		+		#		0														
3		#		×																
4		⊼		S																
5		13	b	(·)																
6		+		#																
7		*		S																
8		↙		S^*																
9		11		↙																
10				(j)																
11		A		A		A														
12																				

Ziffercode	Kontrollstreifen und Bezeichnung
3 6 6 5 1 7 7 5 9 0 2,1 2 4 8 P ⌅ 0 3 6 6 5 1 7 7 5 9 0 2,1 2 4 8 S 3 3 5 5 4,4 6 4 0 ✱ 3 3 5 5 4,4 6 4 0 S 3 6 6 5 1 8 0 9 4 5 6,5 8 8 8 ✱ S 3 6 6 5 1 8 0 9 4 5 6,5 8 8 8 P ⌄ 0 3 6 4 5 7 4 3 5 1 5 6,3 9 1 7 P ⌅ 1 4 0 6 8 1 9 3 0 2 2,7 7 1 2 P ⌅ 2 3 6 4 5 7 4 3 5 1 5 6,3 9 0 9 P ⌅ 3	

Programmausführung

Eingabe		
		✱
		S ✱
1,		=
		J
		0
Adresse	9	A
a =	5,	A
b =	12,	A
Adresse	4	A
a =	2,8	A
b =	7,2	A

0,0000 ✱

0,0000 ✱ S

0,0000 #

0,0009 A
5,0000 #
12,0000 #

0,0004 A
2,8000 #
7,2000 #

Eingabe			Kontrollstreifen und Bezeichnung
Adresse	0	A	
a =	5,04	A	0,0000 A 5,0400 # 6,0600 #
b =	6,06	A	
Adresse	7	A	
a =	3,8	A	0,0007 A 3,8000 # 6,2000 # -
b =	-6,2	A	
Adresse	5	A	0,0005 A 6,5700 # 100,0000 # -
a =	6,57	A	
b =	-100,	A	
		π	
		0	
		π	
		4	
			31,4616 π 0
		π	15,0400 π 4
		5	56,8351 - π 5
			8,2400 π 7
		π	37,0000 π 9
		7	
		π	
		9	

Der allgemeine Fall, daß die von l in j zu modifizierende Adresse (l, j = 0, ..., 9) an p-ter Stelle (p = 1, ..., 10) in einem Programmspeicher steht, wird analog dem Spezialfall p = 10 behandelt, und die Formel für die jeweilige Korrekturzahl lautet:

(F 4): $\text{Korrekturzahl} = (j_p - l_p) \cdot 32^{10-p} \cdot 10^{-K}$

Hierin bedeutet:

p = Stelle eines Befehls in einem Programmspeicher

K = eingestellte Kommaposition

j_p = Die an p-ter Stelle stehende, *modifizierte* Adresse (j_p = 0, ..., 14)

l_p = Die an p-ter Stelle stehende, *zu modifizierende* Adresse (l_p = 0, ..., 14)

(Vgl. hierzu Beispiel 28)

Beispiel 28:

Eine Zahl a soll wahlweise nach $\measuredangle 0$, $\measuredangle 1$, ..., $\measuredangle 9$ weggespeichert werden.

Erläuterungen:

1. Die zu modifizierende Adresse 0 steht in P_1 an 4. Stelle, folglich erhält man für die Adresse j_4 die Korrekturzahl

$$j \cdot 32^6 \cdot 10^{-K}$$

2. In diesem Beispiel ist die Adresse j mit Komma einzugeben, da $32^6 \cdot 10^{-K}$ im Saldierspeicher gespeichert wurde.

3. Im Restspeicher befindet sich der Zifferncode von P_1 (siehe III. 1).

Anwendungsbeispiel

Archiv-Nr.

Beispiel 28

Programm- und Konstantenspeicher: Belegplan

P ≙ Programm ⩡ ≙ Konstante

	0	1	2	3	4	5	6	7	8	9
P	P	P								
K										

Programmierung

V ≙ Variable B ≙ Befehl

	0		1		2		3		4		5		6		7		8		9	
	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B
		P		P																
		0		1																
1	;	(·)	a	(·)																
2		×		#																
3		#		⩡																
4		◇		(;)																
5		S		J																
6		=		0																
7		S																		
8		S*																		
9		⩡																		
10		11																		
11		A		A																
12																				

Zifferncode	Kontrollstreifen und Bezeichnung

```
3 7 2 0 2 2 0 3 4 5 7,0 3 7 4 P ≺ 0
3 7 2 0 2 2 0 3 4 5 7,0 3 7 4   +
                    0,0 0 1 0   +
3 7 2 0 2 2 0 3 4 5 7,0 3 8 4 *

3 7 2 0 2 2 0 3 4 5 7,0 3 8 4 P ≻ 0

3 6 4 0 8 1 1 9 5 2 5,3 7 6 0 P ≺ 1
```

Programmausführung

Eingabe	
	*
	*S
1,	=
107374,1824	+
36408226899,5584	P≺
	1
	S
	S
	S
	:
	*S
	=
	J
	0

```
                    0,0 0 0 0 *

                    0,0 0 0 0 * S

                    0,0 0 0 0 #
          1 0 7 3 7 4,1 8 2 4   +
3 6 4 0 8 2 2 6 8 9 9,5 5 8 4 P ≺ 1
3 6 4 0 8 2 2 6 8 9 9,5 5 8 4   S
3 6 4 0 8 2 2 6 8 9 9,5 5 8 4   S
3 6 4 0 8 2 2 6 8 9 9,5 5 8 4   S
3 6 4 0 8 2 2 6 8 9 9,5 5 8 4 :
1 0 9 2 2 4 6 8 0 6 9 8,6 7 5 2 * S

1 0 9 2 2 4 6 8 0 6 9 8,6 7 5 2 =
                    0,3 3 3 3 *
```

Eingabe		Kontrollstreifen und Bezeichnung
Adresse = 8,	A	8,0000 A 800,0000# -
a = -800,	A	
Adresse = 4,	A	4,0000 A 400,0000# -
a = -400,	A	
Adresse = 1,	A	1,0000 A 100,0000# -
a = -100,	A	
Adresse = 6,	A	6,0000 A 600,0000# -
a = -600,	A	
	π	
	1	
	π	
	4	100,0000 - π 1 400,0000 - π 4 600,0000 - π 6 800,0000 - π 8
	π	
	6	
	π	
	8	

Beispiel 29

Dieses Beispiel soll zeigen, wie man mit Hilfe der Adressenmodifikation bei DIEHL combitron =S= mit Sonderprogramm 10 "einfache" Summen bilden kann. Es handelt sich um die Aufgabe

$$\sum_{i=1}^{n} x_i^{(j)} \quad , \quad (j = 0, 1, \ldots, 9)$$

Anmerkung:

Es sei nochmals darauf hingewiesen, daß man bei "Summenbildung in einem Konstantenspeicher" v o r Programmausführung diesen Konstantenspeicher löschen, d. h., mit der Zahl 0 belegen muß!

Anwendungsbeispiel

Archiv-Nr.

Beispiel 29

Programm- und Konstantenspeicher: Belegplan

P ≙ Programm ↙ ≙ Konstante

	0	1	2	3	4	5	6	7	8	9
P	P	P	P							
K	↙	↙	↙	↙	↙	↙	↙	↙	↙	↙
	$<\sum x_i^{(0)}>$	$<\sum x_i^{(1)}>$	$<\sum x_i^{(2)}>$	$<\sum x_i^{(3)}>$	$<\sum x_i^{(4)}>$	$<\sum x_i^{(5)}>$	$<\sum x_i^{(6)}>$	$<\sum x_i^{(7)}>$	$<\sum x_i^{(8)}>$	$<\sum x_i^{(9)}>$

Programmierung

V≙ Variable B≙ Befehl

	0		1		2		3		4		5		6		7		8		9	
	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B
		P		P		P														
		0		1		2														
1	j	(·)	$x_i^{(j)}$	(·)		J														
2		#		#		0														
3		×		S																
4		◇		↖																
5		S		(j)																
6		=		S																
7		S		S^*																
8		S^*		#																
9		↙		↙																
10		11		(j)																
11		A		A		A														
12																				

Ziffercode	Kontrollstreifen und Bezeichnung

```
3 6 4 5 6 5 9 7 1 3 4,4 5 1 8 P π 0
3 6 4 5 6 5 9 7 1 3 4,4 5 1 8   +
                  0,0 0 1 0   +
3 6 4 5 6 5 9 7 1 3 4,4 5 2 8 ×

3 6 4 5 6 5 9 7 1 3 4,4 5 2 8 P ⌅ 0

3 6 4 4 9 0 4 3 6 8 8,1 5 4 1 P π 1

  4 0 6 8 1 9 3 0 2 2,7 7 1 2 P π 2
```

Programmausführung

Eingabe	
	✱
	S
1,	=
36449043688,1541	P π
	1
	+
	+
	+
	:
	✱
	=
3355,4433	+
	J
	0

```
                  0,0 0 0 0 ✱

                  0,0 0 0 0 ✱ S

                  0,0 0 0 0 #
  3 6 4 4 9 0 4 3 6 8 8,1 5 4 1 P π 1
  3 6 4 4 9 0 4 3 6 8 8,1 5 4 1   +
  3 6 4 4 9 0 4 3 6 8 8,1 5 4 1   +
  3 6 4 4 9 0 4 3 6 8 8,1 5 4 1   +
  3 6 4 4 9 0 4 3 6 8 8,1 5 4 1 :
1 0 9 3 4 7 1 3 1 0 6 4,4 6 2 3 ✱

1 0 9 3 4 7 1 3 1 0 6 4,4 6 2 3 =
                  0,3 3 3 3 ✱

            3 3 5 5,4 4 3 3   +
```

Eingabe		Kontrollstreifen und Bezeichnung
0	K	
	0	
	K	
	1	
	K	
	2	
	K	
	3	
	K	
	4	
	K	
	5	
	K	
	6	
	K	
	7	
	K	
	8	
	K	
	9	

0,0 0 0 0 K 0
0,0 0 0 0 K K 1
0,0 0 0 0 K K 2
0,0 0 0 0 K K 3
0,0 0 0 0 K K 4
0,0 0 0 0 K K 5
0,0 0 0 0 K K 6
0,0 0 0 0 K K 7
0,0 0 0 0 K K 8
0,0 0 0 0 K K 9

Eingabe		Kontrollstreifen und Bezeichnung
Adresse = 7,	A	
$x_1^{(7)}$ = 7,35	A	
Adresse = 4,	A	
$x_1^{(4)}$ = -0,58	A	
Adresse = 0	A	
$x_1^{(0)}$ = 37,45	A	
Adresse = 4,	A	
$x_2^{(4)}$ = 2,7805	A	
Adresse = 4,	A	
$x_3^{(4)}$ = 6,05	A	
Adresse = 0	A	
$x_2^{(0)}$ = -23,4628	A	
Adresse = 7,	A	
$x_2^{(7)}$ = 6,66	A	
Adresse = 0	A	
$x_3^{(0)}$ = -4,23	A	
Adresse = 7,	A	
$x_3^{(7)}$ = -4,32	A	
	π	
	0	
	π	
	4	
	π	
	7	

```
 7,0000#
 7,3500#
 7,3500  A
 4,0000#
 0,5800#-
 0,5800- A
 0,0000#
37,4500#
37,4500  A
 4,0000#
 2,7805#
 2,2005  A
 4,0000#
 6,0500#
 8,2505  A
 0,0000#
23,4628#-
13,9872  A
 7,0000#
 6,6600#
14,0100  A
 0,0000#
 4,2300#-
 9,7572  A
 7,0000#
 4,3200#-
 9,6900  A
```

```
 9,7572  π 0
 8,2505  π 4
 9,6900  π 7
```

Beispiel 30:

Saldieren in 20 Gruppen:

$$S_m = \sum_{i=1}^{n} x_i^{(m)} \qquad (x_i \geq 0 \text{ ; ganzzahlig}) \qquad (m = 0, 1, \ldots, 19)$$

Die Programmierung dieses Problems enthält einige Kunstgriffe der Programmierung und ist als "Selbsttest" für den Programmierer von DIEHL combitron =S= mit Sonderprogramm gedacht. Deshalb ist auf eine Analyse der Programmierung verzichtet worden.

Hinweis:

Das Programm muß in der Kommaposition 8 gerechnet werden.

Anwendungsbeispiel

Archiv-Nr.

Beispiel 3o

Programm- und Konstantenspeicher: Belegplan

P ≙ Programm ⩡ ≙ Konstante

	0	1	2	3	4	5	6	7	8	9
P	P	P	⩡	⩡	⩡	P	P	P	P	P
			«10^4»	Zi-Co P_1	const.					
K	⩡	⩡	⩡	⩡	⩡	⩡	⩡	⩡	⩡	⩡
	S_0/S_1	S_2/S_3	S_4/S_5	S_6/S_7	S_8/S_9	S_{10}/S_{11}	S_{12}/S_{13}	S_{14}/S_{15}	S_{16}/S_{17}	S_{18}/S_{19}

Programmierung

V≙ Variable B≙ Befehl

	0		1		2		3		4		5		6		7		8		9	
	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B	V	B
		P		P								P		P		P		P		P
		0		1								5		6		7		8		9
1		=		S							m	(·)		=		×		J	X_i	(·)
2		(—)		⊼								#		+		⊼		0		#
3		CJ		(↓)								:		+		14				J
4		9		S								⊼		$\overset{*}{S}$		S				1
5	$X_i^{(m)}$	(·)		$\overset{*}{S}$								12		:		⊼				
6		#		#								:		*		13				
7		:		⩡								+		×		S				
8		⊼		(↓)								S		⊼		$\overset{*}{S}$				
9		12		J								*		12		⩡				
10		:		5								:		×		11				
11		A		A								A		A		A		A		A
12																				

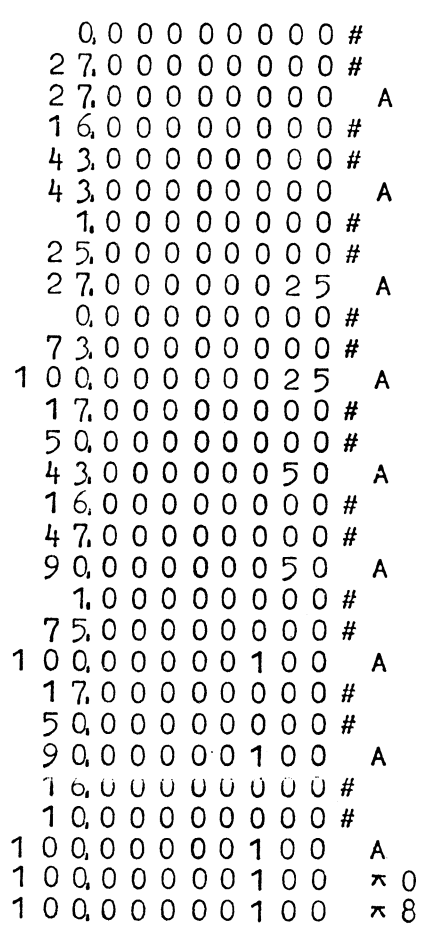
Ziffercode	Kontrollstreifen und Bezeichnung
6 7 5 2 1 8 5,9 8 6 2 7 8 8 7 P ↑ 0	
5 6 5 3 3 8 2,0 1 7 8 9 4 8 2 P ↑ 1	
5 6 5 3 3 8 2,0 1 7 8 9 4 8 2 P ↓ 3	
3 6 4 4 5 6 4,7 8 9 4 9 6 7 9 P ↑ 5	
6 8 3 2 5 1 4,2 6 9 8 6 5 4 6 P ↑ 6	
6 3 6 1 8 7 8,2 2 2 6 7 5 3 6 P ↑ 7	
4 0 6 8 1 9,3 0 2 2 7 7 1 2 P ↑ 8	
3 6 3 9 7 9 1,5 0 9 8 3 1 6 8 P ↑ 9	

Programmausführung

Eingabe	
	*
	S*
1,	=
0	↓
	0
	.
	.
	.
	↓
	9
10000,	P↓
	2
343,59739392	P↓
	4
	J
	5

```
        0,0 0 0 0 0 0 0 0 *
        0,0 0 0 0 0 0 0 0 * S
        0,0 0 0 0 0 0 0 0 #
        0,0 0 0 0 0 0 0 0   ↓ 0
        0,0 0 0 0 0 0 0 0   ↓ 1
        0,0 0 0 0 0 0 0 0   ↓ 2
        0,0 0 0 0 0 0 0 0   ↓ 3
        0,0 0 0 0 0 0 0 0   ↓ 4
        0,0 0 0 0 0 0 0 0   ↓ 5
        0,0 0 0 0 0 0 0 0   ↓ 6
        0,0 0 0 0 0 0 0 0   ↓ 7
        0,0 0 0 0 0 0 0 0   ↓ 8
        0,0 0 0 0 0 0 0 0   ↓ 9
1 0 0 0 0,0 0 0 0 0 0 0 0 P ↓ 2
    3 4 3,5 9 7 3 9 3 9 2 P ↓ 4
```

Eingabe		Kontrollstreifen und Bezeichnung
$m = 0$	A	
$x_1^{(0)} = 27,$	A	
$m = 16,$	A	
$x_1^{(16)} = 43,$	A	
$m = 1,$	A	
$x_1^{(1)} = 25,$	A	
$m = 0$	A	
$x_2^{(0)} = 73,$	A	
$m = 17,$	A	
$x_1^{(17)} = 50,$	A	
$m = 16,$	A	
$x_2^{(16)} = 47,$	A	
$m = 1,$	A	
$x_2^{(1)} = 75,$	A	
$m = 17,$	A	
$x_2^{(17)} = 50,$	A	
$m = 16,$	A	
$x_3^{(16)} = 10,$	A	
	ϰ	
	0	
	ϰ	
	8	



Zifferncode	Kontrollstreifen und Bezeichnung

Programmausführung

Eingabe	