PDV Berichte

Projekt Prozeßlenkung mit DV-Anlagen

KFK-PDV 100

Programmieranleitung für das ASME 1-PEARL-SUBSET

November 1976

GESELLSCHAFT FÜR KERNFORSCHUNG MBH KARLSRUHE

PDV-Berichte

Die Gesellschaft für Kernforschung mbH koordiniert und betreut im Auftrag des Bundesministers für Forschung und Technologie das im Rahmen der Datenverarbeitungsprogramme der Bundesregierung geförderte Projekt Prozeßlenkung mit Datenverarbeitungsanlagen (PDV). Hierbei arbeitet sie eng mit Unternehmen der gewerblichen Wirtschaft und Einrichtungen der öffentlichen Hand zusammen. Als Projektträger gibt sie die Schriftenreihe PDV-Berichte heraus. Darin werden Entwicklungsunterlagen zur Verfügung gestellt, die einer raschen und breiteren Anwendung der Datenverarbeitung in der Prozeßlenkung dienen sollen.

Der vorliegende Bericht dokumentiert Kenntnisse und Ergebnisse, die im Projekt PDV gewonnen wurden.

Druck und Verbreitung:
Gesellschaft für Kernforschung mbH
7500 Karlsruhe 1, Postfach 3640
Printed in Western-Germany

Projekt Prozeßlenkung mit DV-Anlagen

Forschungsbericht KFK-PDV 100

Programmieranleitung für das ASME-PEARL-SUBSET/1

181 Seiten
2 Abbildungen

November 1976

Verfasser:

Gruber	Firma ESG, München
Inderst	Firma ESG, München
Piche	Firma ESG, München

unter Mitwirkung von:

Alt	IVD, Universität Stuttgart
Bühler	IVD, Universität Stuttgart
Eichenauer	Firma GPP, München
Elzer	Physikalisches Institut III der Universität Erlangen-Nürnberg
Ghassemi	IRP, Universität Stuttgart
Helfert	IRP, Universität Stuttgart
Holleczek	Physikalisches Institut III der Universität Erlangen-Nürnberg
Ißmayer	Firma ESG, München
Lindstedt	Physikalisches Institut III der Universität Erlangen-Nürnberg
Mühlhahn	Firma ESG, München
Pelz	Physikalisches Institut III der Universität Erlangen-Nürnberg
Prester	Physikalisches Institut III der Universität Erlangen-Nürnberg
Rössler	Physikalisches Institut III der Universität Erlangen-Nürnberg
Wiedenmann	IRP, Universität Stuttgart
Winckler	Firma ESG, München
Zeh	IRP, Universität Stuttgart

INHALTSVERZEICHNIS

KAPITEL 1

KAPITEL 2

DATEN

KAPITEL 3

AUSDRÜCKE

KAPITEL 4

ANWEISUNGEN

KAPITEL 5

KAPITEL 8

FEHLERMELDUNGEN

ANHANG I

METASYMBOLE ZUR SYNTAXBESCHREIBUNG

ANHANG II

VERZEICHNIS DER SCHLÜSSELWÖRTER

ANHANG III

ZUSAMMENSTELLUNG ALLER ZUGELASSENEN VEREINBARUNGEN

EINLEITUNG

1. ALLGEMEINES

In den folgenden Kapiteln wird das in der Arbeitsgemeinschaft ASME* entwickelte PEARL**-Subset [1] [2] für Anwender an den Anlagen SIEMENS 306, AEG 60-50 und AEG 60-10 beschrieben.

Diese Beschreibung erhebt nicht den Anspruch, ein Lehrbuch zur PEARL-Programmierung zu sein. Vielmehr wird erwartet, daß sich der Anwender aufgrund seiner Kenntnisse in anderen höheren Programmiersprachen wie ALGOL 60, FORTRAN oder PL/1 in die neuen Begriffe und Sprachmittel von PEARL einarbeiten kann, um sie zur Lösung seiner Realzeit-Probleme effektiv einsetzen zu können.
Anlagenabhängige Unterschiede in Syntax und Semantik werden in einem, jedem Unterpunkt dieser Beschreibung nachgestellten, andersfarbigen Blatt dargestellt.

---

* ASME = Arbeitsgemeinschaft Stuttgart-München-Erlangen

** PEARL = Process and Experiment Automation Realtime Language

[1] PDV-Bericht: KFK-PDV1, PEARL

[2] ASME Bericht: ASME-PEARL-SUBSET/1

2. EINFÜHRUNG IN DIE WICHTIGSTEN SPRACHEIGENSCHAFTEN DES PEARL-SUBSETS

Das Task-Konzept zur Beschreibung von Realzeitvorgängen

Programm-Systeme zur on-line-Steuerung und on-line-Auswertung von Prozessen und Experimenten bestehen meist aus einer Anzahl von Programmteilen, die einzeln betrachtet autonom ablaufen, aber zu gewissen Anlässen koordiniert werden müssen.
PEARL bietet durch das Sprachmittel der "Task" die Möglichkeit zur Programmierung zeitlich parallel ablaufender Vorgänge, wobei mit einer Reihe von Anweisungen die zeitliche Koordinierung der Tasks beschrieben werden kann (sog. "Tasking").
Der Task-Körper besteht aus einer Folge von PEARL-Anweisungen. Als "Task" wird der dynamische Ablauf dieser Folge von Anweisungen unter Kontrolle des Betriebssystems verstanden. Eine Task kann die Ausführung weiterer in diesem Sinne abgeschlossener Anweisungsfolgen vom Betriebssystem fordern (sog. ACTIVATE-Anweisung). Diese Tasks laufen zeitlich parallel zueinander ab, soweit die vorhandenen Betriebsmittel (Zentraleinheit, Kanalwerke, periphere Geräte) dies gestatten. Das Betriebssystem vergibt Betriebsmittel im Konfliktfall gemäß den Prioritäten der anfordernden Tasks. Im PEARL-Subset werden die Prioritäten entweder bei der Task-Aktivierung oder bei der Vereinbarung vergeben.
Da die zeitliche Reihenfolge von Operationen in parallel ablaufenden Tasks in vielen Fällen nicht willkürlich sein kann (z.B. wenn ein "Verbraucher" einen Datenpuffer leeren soll, dann muß zuvor ein "Erzeuger" Daten in diesen Puffer eingebracht haben), sind Mittel vorhanden, um in den parallel laufenden Tasks die gewünschte Reihenfolge solcher Operationen zu erzwingen. Dies geschieht durch Semaphor-Variable und auf sie wirkende Semaphoroperationen im Dijkstraschen Sinne [3].
Für die Prozeßprogrammierung ist es wesentlich, die zeitlich zueinander asynchron ablaufenden Aufgaben zeitlich einplanen zu können; vor allem aber müssen spontan auftretende äußere Ereignisse kurzfristig beantwortet werden können.
Die meisten Tasking-Anweisungen können daher entweder zeitlich absolut oder zeitlich periodisch eingeplant, aber auch abhängig vom Eintritt sporadisch auftretender Ereignisse (Interrupts) eingeleitet werden (sog. "scheduling").

Eingabe/Ausgabe zur Standard- und Prozeß-Peripherie

Der PEARL-Subset enthält Anweisungen zur Datenübertragung von/zur Standard- und Prozeßperipherie sowie Anweisungen zur Datei-Verwaltung. Die Datenübertragung von/zur Standard-Peripherie (Platte, Drucker, Lochkarten-Leser) geschieht konventionell zeichenweise formatiert ähnlich wie in PL/1. Die Übertragung von Daten von/zur Prozeß-Peripherie geschieht element- oder satzweise unformatiert, d.h. Binär. Durch Angabe einer Option ist es möglich, während der Datenübertragung eine Transformation auf den Daten-String auszuführen (z.B. Eich- oder Kontrollprozeduren) oder Steuerinformationen an Geräte weiterzuleiten.

---

[3] E.W. Dijkstra: Cooperating sequential processes, in "programming languages" bei Academic Press London 1968

Graphische Informationsinhalte können mit PEARL-Anweisungen von und zu graphischen Aufzeichnungsgeräten (Bildschirm oder Plotter) transferiert werden.
Mit den Anweisungen zur Datei-Verwaltung können u.a. auf Standard-Externspeichern Dateien eingerichtet werden.

Hardware-Konfigurations-Beschreibung auf Sprachebene

In einem Prozeßprogramm sollte die gesamte Peripherie auf Sprachebene ansprechbar sein. Da es keine allgemeine standardisierte Prozeßperipherie gibt, kann es auch keine allgemeine anlagenunabhängige Beschreibung auf Sprachebene geben.
In PEARL werden deshalb alle anlagenspezifischen Beschreibungen in einer speziellen Notation im sog. "System-Teil" aufgeführt. Alle anlagenunabhängigen Teile, d.i. das eigentliche Prozeßprogramm, werden dagegen im sog. "Problem-Teil" beschrieben. Durch diese strikte Trennung erreicht man eine weitgehende Anlagenunabhängigkeit der Prozeßprogramme.
Im Systemteil beschreibt der Anwender die benötigte Hardware-Konfiguration und belegt diejenigen Prozeßnaht- oder -Endstellen mit selbstgewählten Namen, die er im Problemteil ansprechen will. Derartige Namen werden beispielsweise vergeben für Standard- oder Prozeß-Geräte, für von der Hardware kommende Alarme, sog. "Interrupt-Anschlüsse" oder für Zustandsmeldungen (z.B. Plattenüberlauf, Geräteausfall), sog. "Signal-Anschlüsse" des Gesamtsystems.

Diese Art der Konfigurations-Beschreibung bringt sowohl bezüglich der Dokumentierbarkeit wie auch bezüglich der Verarbeitung Vorteile. Es kann z.B. ein auf das Prozeßprogramm zugeschnittenes optimales Betriebssystem generiert werden. Ebenso ist eine für den Rechner eigene "job control language" überflüssig. Ein weiterer Vorteil ist darin zu sehen, daß in den E/A-Anweisungen nicht mehr wie in anderen Sprachen die umständliche Beschreibung des Datenweges vorgenommen werden muß, da man lediglich den im Systemteil definierten symbolischen Endstellen-Namen anzugeben hat.

Algorithmische Sprachmittel im PEARL-Subset

Im Bereich der konventionellen algorithmischen Sprachmittel bietet das PEARL-Subset die von anderen Sprachen (z.B. ALGOL-60, PL/1) bekannten Elemente, wie z.B. die Grund-Datentypen einschließlich Bit-Daten, die Deklaration von Bezeichnern, die Blockstruktur, die Prozedur- und Funktionsaufrufe, die Ausführung von Ausdrücken ("expression") und eventuell deren Zuweisung an Variable sowie Anweisungen zur Steuerung des sequentiellen Programmablaufs, wie Sprung-Anweisung, Bedingte Anweisung und Schleifen-Anweisung.

Im PEARL-Subset können Bezeichner nicht nur für Variablen und Variablen-Felder sondern auch für Konstanten und Konstanten-Felder deklariert werden (Referenz-Stufe -Ø). Zusätzlichen Komfort bietet die Verfügbarkeit von Zeitdatentypen wie Uhrzeit und Zeitintervall.

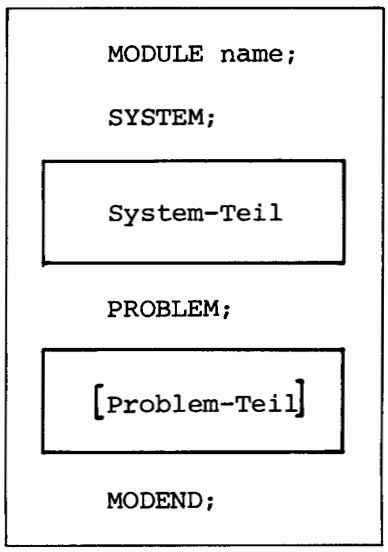
Programm-Struktur

Die PEARL-Programm-Struktur ist eine Modul-Struktur. Ein Modul ist eine Zusammenfassung von Task- und Prozedur-Deklarationen sowie "Common-Größen". Task- und Prozedur-Rümpfe sind in konventionelle Blöcke unterteilt (wie PL/1, ALGOL). Jeder Modul ist einzeln übersetzbar, aber für sich nicht ablauffähig. Übersetzte PEARL-Module bilden zusammen mit den Betriebssystembausteinen ein PEARL-Programm-System. Zum Ablauf wird ein PEARL-Programm-System dadurch gebracht, daß eine dafür vorgesehene Task über die Bedienungseinheit gestartet wird.

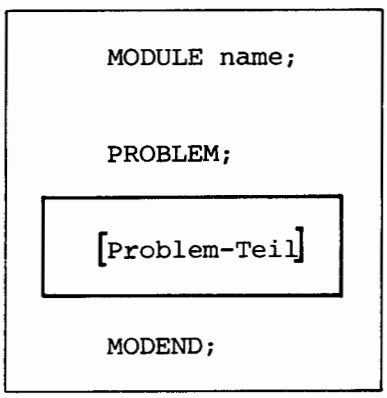
KAPITEL 1

1-1. AUFBAU EINES PROGRAMM-MODULS

Ein Programm-Modul besteht aus einem System-Teil und einem Problem-Teil oder nur aus einem Problemteil, wobei der Problemteil auch leer sein kann und hat folgenden syntaktischen Aufbau:



oder



1-1.1 System-Teil

Die Beschreibung des System-Teils erfolgt getrennt im Kapitel 7.

1-1.2 Problem-Teil

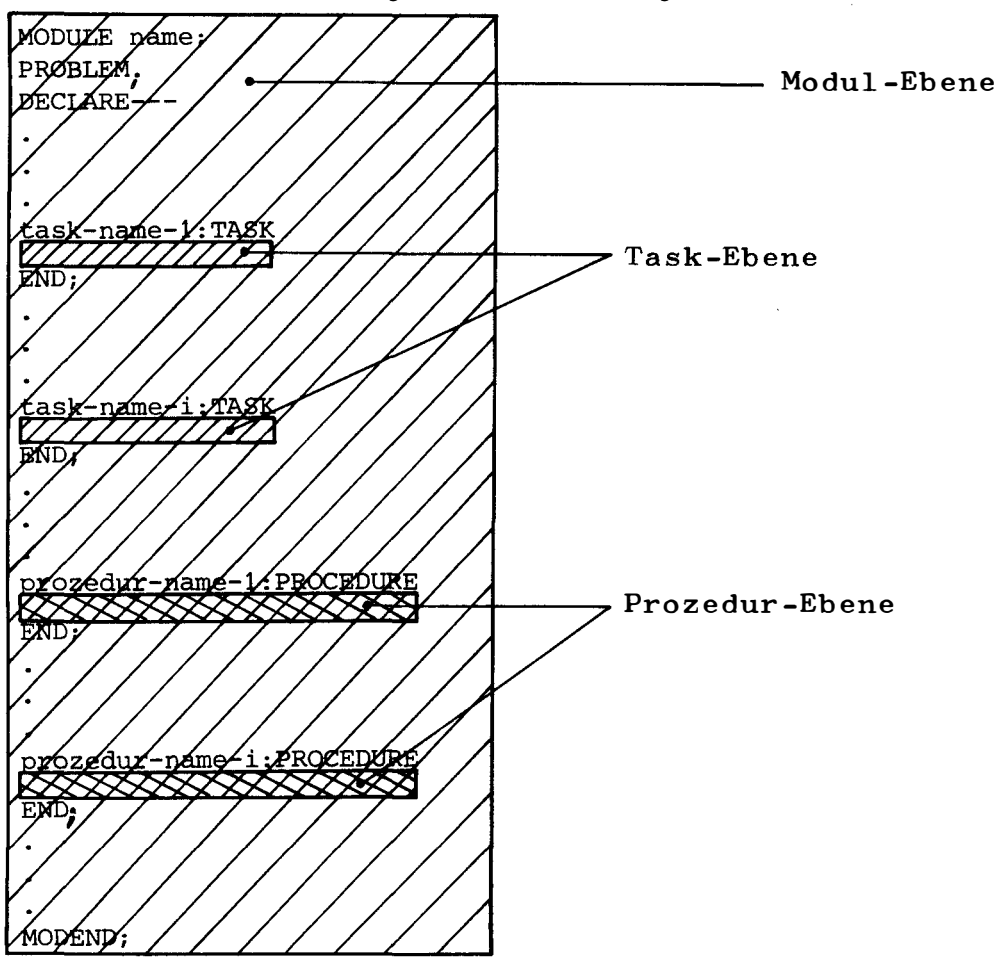
Ein Problem-Teil besteht nur aus Vereinbarungen. Diese Vereinbarungen stehen auf oberster Beschreibungsebene für PEARL-Sprachelemente, der sog. Modul-Ebene (siehe nachfolgende schematische Darstellung). Ausführbare Anweisungen sind nur innerhalb von Task- und Prozeduredeklarationen erlaubt.

1-1.2.1 Vereinbarungen auf Modulebene

Auf Modulebene können im Problemteil nachfolgende Größen deklariert werden (Syntax siehe Kapitel 2, 4):

- Prozeduren
- Tasks
- Variable
- Felder
- Konstanten-Namen
- Konstanten-Felder
- Synchronisations-Variable
- Files

Schematische Darstellung der Blockbildung auf Modulebene



Beispiel der Blockbildung auf Task- und Prozedurebene (siehe auch Kapitel 2, 2-4.3 Beispiele)

```
        V;...
        A;...
B1:     BEGIN;
            V;...
            A;...
B2:         BEGIN;
                V;...
                A;...
B3:             BEGIN;
                    V;...
                    A;...
                END;
            END;
            A;...
B4:         BEGIN;
                V;...
                A;...
            END;
        END;
        A;...
B5:     BEGIN;
            V;...
            A;...
B6:         BEGIN;
                V;...
                A;...
            END;
        END;
```

V bedeutet Vereinbarung
A bedeutet Anweisung

Die Blöcke können in Stufen angeordnet werden.
Stufe 1: B1, B5

Stufe 2: B2, B4, B6

Stufe 3: B3
aus dieser Stufen-Ordnung ergeben sich die Neben- oder Unterordnungen von Blöcken, z.B. ist
B1 und B5 nebengeordnet,
B3 ist B2 und B1 untergeordnet, usw.

1- 1.2.2 Globale Größen

Auf Modul-Ebene deklarierte und spezifizierte Größen (Variable, Tasks, Prozeduren, etc.) sind innerhalb des gesamten Problem-Teils bekannt und dürfen nicht wieder deklariert oder spezifiziert werden. Da ein PEARL-Programm aus mehreren getrennt übersetzten Modulen bestehen kann, ist es notwendig, die auf Modulebene deklarierten Größen zu kennzeichnen, die im Gesamtsystem bekannt sein sollen. Dies geschieht durch Anfügen des Global-Attributes "GLOBAL in der Deklaration. Größen, die auf diese Art in verschiedenen Modulen deklariert worden sind, müssen genau die gleichen Attribute besitzen.

Einführung von globalen Größen

Globale Größen werden auf zwei verschiedene Arten in einem neuen Modul eingeführt, nämlich durch Deklaration oder Spezifikation.

Variable und Felder sowie auch Synchronisations-Variable werden in jedem Modul, in dem sie benutzt werden, durch globale Deklaration eingeführt (Syntax siehe Kapitel 2, 4).

Prozeduren, Tasks und Konstanten-Namen sowie Konstanten-Felder werden nur in einem Modul deklariert (mit Code) und in allen anderen, in denen sie auch benutzt werden, global spezifiziert (Syntax siehe Kapitel 2, 4).

Beispiel:

Problemteil im Modul A:

```
PROBLEM;
DECLARE K FLOAT GLOBAL,
X VAL FLOAT GLOBAL IDENTICAL (5.7);
.
.
.
MODEND;
```

Problemteil im Modul B:

```
PROBLEM;
DECLARE K FIXED GLOBAL,
X VAL FLOAT GLOBAL;
.
.
.
MODEND;
```

Die Einführung der Variablen K in die Module A und B als globale Variable ist fehlerhaft, da die Attribute nicht übereinstimmen. Die Einführung des Konstanten-Namens X in den Modul B mittels Spezifikation ist jedoch richtig.

1 - 1.2.3 Verbindung mit dem System-Teil

Alle frei wählbaren Bezeichner, die im Systemteil eingeführt wurden, sind im ganzen Modul bekannt. Sie besitzen die Referenz-Stufe-Ø (siehe Kapitel 2, 1.1 Referenz-Stufen).
Diese Bezeichner müssen innerhalb des Problemteils mit dem Global-Attribut spezifiziert werden.

Beispiel:

```
MODULE XR;

SYSTEM;
.
.
.
CONSOL: TYPE <-> CPU;
.
.
.
PROBLEM;
.
.
.
DECLARE CONSOL VAL DEVICE GLOBAL;/* CONSOL WIRD ALS SYMBOLISCHER GERAETE-
                                    NAME IN DEN PROBLEMTEIL EINGEFUEHRT*/
.
.
.
MODEND;
```

Beispiel: Programm-Aufbau und Spezifizierung von globalen Größen

```
MODULE TESTØ1;
SYSTEM;
PROBLEM;

/*ES FOLGT EINE GLOBALE KONSTANTEN-FELD-DEKLARATION:*/

DECLARE FIELD (3,3) VAL CHARACTER (1) GLOBAL
        IDENTICAL ('A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I');

/*ES FOLGT EINE GLOBALE PROZEDURSPEZIFIKATION:*/
DECLARE ROUT ENTRY ((2,4) VAL CHARACTER (1), CHARACTER (1),
                     FIXED) GLOBAL REENTRANT;

/*ES FOLGT EINE GLOBALE TASKSPEZIFIKATION, ZU ERKENNEN AN DEM SEMIKOLON
NACH DEN ATTRIBUTEN:*/

DIFFER: TASK GLOBAL RESIDENT;
   .
   .
   .
MODEND;

MODULE TESTØ2;
SYSTEM;
PROBLEM;

/*ES FOLGT EINE GLOBALE KONSTANTEN-FELD-SPEZIFIKATION:*/
DECLARE FIELD (3,3) VAL CHARACTER (1) GLOBAL;

/*ES FOLGT EINE GLOBALE PROZEDURDEKLARATION, MIT PARAMETERSPEZIFIZIERUNG:*/

ROUT: PROCEDURE (AR, X, K) GLOBAL REENTRANT

   /*ES FOLGT DIE PARAMETERSPEZIFIKATION:*/
        DECLARE AR(2,4) VAL CHARACTER (1),
                X CHARACTER (1),
                K FIXED;
           .
           .
        X = AR (K, K+1);
           .
           .
        END;

/*ES FOLGT EINE GLOBALE TASKDEKLARATION:*/

DIFFER: TASK GLOBAL RESIDENT
        DECLARE IR FIXED
        .       Y CHARACTER (1);
        .
        Y = FIELD (1, IR)
        .
        .
        END;
MODEND;
```

1-2. GRUNDELEMENTE DES PEARL-SUBSETS

Die Grundelemente des PEARL-Subsets sind die kleinsten oder simpelsten logisch zusammengehörigen syntaktischen Einheiten dieser Sprache. Grundelemente sind außer dem Zeichensatz auch Schlüsselwörter, Namen, Ersatztrennzeichen oder Kommentare.

1-2.1 Zeichensatz

Der im Subset zugelassene Zeichensatz kann nachfolgender Tabelle entnommen werden.

Bezeichnung	Darstellung
Buchstaben	A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V W|X|Y|Z
Ziffer	Ø|1|2|3|4|5|6|7|8|9
Sonderzeichen	Leerzeichen | +|-|*|/|(|)|,|.|'|:|; | =|>|<
Binär-Ziffer	Ø|1
Oktal-Ziffer	Ø|1|2|3|4|5|6|7
Alphanumerische-Zeichen	Buchstaben|Ziffer
Zeichen	alphanumerische-Zeichen|Sonderzeichen

1-2.2 Schlüsselwörter

Schlüsselwörter sind Aneinanderreihungen von Buchstaben, die in der Sprache eine feste Bedeutung besitzen. Die Schlüsselwörter sind nicht reserviert, d.h. sie können auch als Namen benutzt werden. Aus Übersichtsgründen sollte man jedoch für freiwählbare Namen keine Schlüsselwörter verwenden. Im Anhang III sind alle maschinenunabhängigen Schlüsselwörter aufgeführt.

Die Namen von Standardprozeduren sind keine Schlüsselwörter im obigen Sinne und bleiben daher reserviert. Sie sind zielmaschinenabhängig verschieden.

> Leerzeichen in Schlüsselwörtern sind nicht erlaubt.

Beispiele:

für Schlüsselwörter: GOTO (nicht erlaubt GO TO)
TASK
RETURN

1-2.3 Namen

Namen sind frei wählbare Aneinanderreihungen von alphanumerischen Zeichen mit führenden Buchstaben, die zur Identifizierung von Programmgrößen, Prozeduren, Tasks, etc. dienen.

name:: = buchstabe [alphanumerische-zeichen ...]

Ein Name darf aus beliebig vielen Zeichen bestehen, nur die ersten 6 Zeichen haben unterscheidende Bedeutung.

Leerzeichen sind innerhalb eines Namens nicht erlaubt.

1-2.4 Trennung von Sprachelementen und Verwendung von Leerzeichen.

Um die Sprachelemente, aus denen sich ein Programm zusammensetzt, voneinander unterscheiden zu können, müssen diese gegeneinander abgegrenzt sein.
Als Trennzeichen zwischen Sprachelementen ist mindestens ein Zwischenraum vorgeschrieben. Die Einfügung von Zwischenräumen darf entfallen, wenn ein anderes Sonderzeichen als Ersatztrennzeichen zwischen den Sprachelementen auftritt.

Ersatztrennzeichen: - für Schlüsselwörter, Konstanten und Namen: ( ) * + , - . / : ; = > < - für Operatoren: jedes als Anfangszeichen zugelassene Zeichen von gültigen Operanden (siehe Kapitel 3).

1-2.5 Kommentar

Zwischen Sprachelementen sind in den Programmtext Kommentare einfügbar:

kommentar ::= /* zeichen ... */

In der Zeichenfolge darf die Folge */, durch die ein Kommentar beendet wird, nicht auftreten.

KAPITEL 2: DATEN

2-1. DATENELEMENTE

Im allgemeinen benötigt ein Programm bei seiner Ausführung Daten. Dabei kann es sich um Daten handeln, die

- vom Programm verarbeitet werden, mit denen es rechnet (Problemdaten)
- den Programmablauf steuern (Programm-Steuerungsdaten).

Die kleinste Einheit der Daten wird als Datenelement bezeichnet; sie ist eine skalare Größe.

2-1.1 Referenzstufen

Im PEARL-Subset gibt es zwei Referenzstufen:

- Referenzstufe Ø

  Größen, die die Referenzstufe Ø besitzen, können sich während des Programmablaufs nicht ändern.
  Vereinbarungen für Referenz-Ø-Größen müssen das VAL-Attribut enthalten. Die Zuordnung von Name und Konstante erfolgt mit dem Attribut IDENTICAL (s. 2-4.2).
  Im folgenden werden für die Problemdaten mit der Referenzstufe Ø die Begriffe

  . Konstante
  . Konstantenname
  . Konstantenfeldelement
  . Konstantenfeld

  verwendet.

- Referenzstufe 1

  Größen, die die Referenzstufe 1 besitzen, dürfen während des Programmablaufs nacheinander verschiedene Werte annehmen.
  In der Vereinbarung für Referenz-1-Größen fehlt das VAL-Attribut; Anfangswerte werden mit dem Attribut INITIAL vereinbart (s. 2-4.2).

  Im folgenden werden die Problemdaten mit der Referenzstufe 1 mit den Begriffen

  . Variable
  . Feldelement
  . Feld

  bezeichnet.

2-1.2 Skalare Größen

2-1.2.1 Konstante

Konstante müssen nicht explizit vereinbart werden. Sie sind Programmgrößen, deren Darstellung alle benötigten Beschreibungsinformationen (Typ, Länge, usw.) enthält.
Konstante können sich während des Programmablaufs nicht ändern.

2-1.2.2 Konstantenname

Es besteht die Möglichkeit, für eine Konstante einen Namen zu vereinbaren. Dieser Name kann im Programm anstelle der Konstanten verwendet werden.

Beispiel:
```
/* VEREINBARUNG DES NAMENS PI FÜR DIE KONSTANTE 3.14159*/
DECLARE PI FLOAT IDENTICAL (3.14159);

/* DIE KONSTANTE 3.14159 KANN IN ANWEISUNGEN MIT DEM
NAMEN PI ANGESPROCHEN WERDEN: */
U = 2*PI*R;
CALL P (PI);
```

2-1.2.3 Konstantenfeldelement

Ein Konstantenfeldelement (indizierter Konstantenname) bezeichnet ein Datenelement eines Konstantenfeldes.

2-1.2.4 Variable

Eine Variable bezeichnet ein Datenelement, das während des Programmablaufs nacheinander verschiedene Werte annehmen kann. Alle Werte, die eine Variable annehmen kann, müssen die gleichen Datenattribute (Typ, Länge) besitzen.

2-1.2.5 Feldelement

Ein Feldelement (indizierte Variable) bezeichnet ein Datenelement eines Feldes. Es gelten die unter 2-1.2.4 gemachten Aussagen.

2-1.3 Bereiche

Ein Bereich (auch Feld, Tabelle oder Matrix genannt) ist eine n-dimensionale geordnete Menge von skalaren Datenelementen, die alle die gleichen Datenattribute besitzen.

Im Subset sind folgende Einschränkungen gemacht:

- Es gibt nur statische Bereiche mit konstanten Dimensionsangaben.
- Es gibt - abhängig von der Datenart - ein-, zwei- und dreidimensionale Bereiche.

2-1.3.1 Anordnung von Bereichen im Speicher

Die Elemente eines Bereichs werden in einer aufsteigenden Folge von Speicherplätzen angelegt, und zwar so, daß zuerst der Wertebereich des ersten Index, dann der des zweiten und zuletzt der des dritten durchlaufen wird.

Beispiel: DECLARE F (3,2,2) FIXED;

F + Ø	F(1,1,1)
F + 1	F(2,1,1)
F + 2	F(3,1,1)
F + 3	F(1,2,1)
F + 4	F(2,2,1)
F + 5	F(3,2,1)
F + 6	F(1,1,2)
F + 7	F(2,1,2)
F + 8	F(3,1,2)
F + 9	F(1,2,2)
F + 1Ø	F(2,2,2)
F + 11	F(3,2,2)

2-1.3.2 Konstantenfelder

Die Datenelemente eines Konstantenfeldes enthalten Konstanten. Sie können während des Programmablaufs nicht verändert werden.

2-1.3.3 Felder

Die Datenelemente eines Feldes dürfen während des Programmablaufs nacheinander verschiedene Werte annehmen. Alle Werte müssen dabei die gleichen Datenattribute besitzen.

2-2. NAMEN FÜR PROBLEMDATEN

Mit Namen werden

- Konstante
- Variable
- Konstantenfelder
- Felder

angesprochen.
Ein Element eines Bereichs wird durch den Bereichsnamen und die Angabe seiner Position innerhalb des Bereichs identifiziert. Die Angabe der Position erfolgt in einer Indexliste. Die einzelnen Indices werden durch Kommata getrennt; die Liste wird in Klammern eingeschlossen:

```
name (index[, index][, index])
```

Für die Indices gelten folgende Regeln:

- Es dürfen Ausdrücke der allgemeinen Form nach Kapitel 3/3-1. eingesetzt werden. Es ist allerdings zu beachten, daß das Ergebnis des Indexausdrucks den Typ FIXED besitzt.

- Die Anzahl der Indices muß gleich der Anzahl der Dimensionen des Bereichs sein.

- Der Wert eines Index muß innerhalb der vereinbarten Grenzen liegen.

Beispiele:

```
KONST
VAR
FELD
FELD(1)
F2(KFELD(I)+FKT(K), KONST)
F3(I,J,K)
```
(Die Vereinbarung von Namen s. 2-4.)

2-3. DATENARTEN

Die im Subset erlaubten Datenarten lassen sich wie folgt einteilen:

- Problemdaten
  - Arithmetische Daten
  - Bitketten
  - Zeichenketten
  - Zeitdaten
- Programm-Steuerungsdaten
  - Marken
  - Semaphore
  - Ereignisdaten

Die Charakteristiken von Datenelementen einer bestimmten Datenart werden mit Hilfe von Attributen festgelegt (s. 2-4.).

2-3.1 Problemdaten

2-3.1.1 Arithmetische Daten

Ein arithmetisches Datenelement besitzt einen numerischen Wert. Im Subset kann dieser Wert eine ganze oder eine rationale Zahl sein. Arithmetische Datenbereiche dürfen ein-, zwei- oder dreidimensional sein.

Hinweis: Arithmetische Konstante haben im Subset kein Vorzeichen.

2-3.1.1.1 Ganze Zahlen

Das Attribut FIXED wird ausschließlich ganzen Zahlen zugeordnet. Konstante ganze Zahlen werden in dezimaler Darstellung angegeben. Sie bestehen aus einer Dezimalziffer oder einer Folge von Dezimalziffern:

ganze-zahl::= dezimalziffer ...

Beispiele: 171Ø
Ø
14

Hinweis: "-5" ist ein Ausdruck mit dem monadischen Operator "-" und der ganzen Zahl "5" als Operand.

2-3.1.1.2 Rationale Zahlen

Eine rationale Zahl besitzt einen Wert mit dem Attribut FLOAT. Konstante rationale Zahlen werden in Gleitpunktdarstellung zur Basis 1Ø angegeben. Sie bestehen aus einer Ziffernfolge mit Dezimalpunkt oder einer Ziffernfolge mit nachfolgendem Exponenten oder beidem. Der Exponent besteht aus dem Buchstaben E, dem eine ganze Zahl mit oder ohne Vorzeichen folgt.
Der Wert der Gleitpunktkonstanten ergibt sich aus der Multiplikation der Mantisse mit der durch den Exponenten angegebenen Potenz von 1Ø.

rationale-zahl::={mantisse [exponent]|ganze-zahl exponent}

mantisse::={ [ganze-zahl]. ganze-zahl | ganze zahl. }

exponent::= E{[+]|-} ganze-zahl

Bemerkungen:

- Der Exponent darf wegfallen, wenn er Ø ist und die Zahl einen Dezimalpunkt enthält.

- Enthält die Zahl vor dem Exponenten keinen Dezimalpunkt, so wird er nach ihrer letzten Ziffer angenommen.

Beispiele: .Ø
17.5
17.5E2
17.E+13
Ø.E122
18.E-13
16.ØEØ
136E14 (≙ 136.E14)

Hinweis: "+5.EØ" ist ein Ausdruck mit dem monadischen Operator "+" und der rationalen Zahl "5.EØ" als Operand.

2-3.1.2 Bitketten

Bitketten-Werte besitzen das Attribut BIT(n), wobei die ganzzahlige Konstante n die Anzahl der Binärziffern (nicht die der Oktalziffern!) angibt, aus denen die Bitkette besteht.
Eine Bitkette darf im Subset

- 1 bis 24 Binärziffern
- oder 1 bis 8 Oktalziffern

enthalten. Bitkettenbereiche dürfen ein-, zwei- oder dreidimensional sein.

Bitketten-Konstante bestehen aus einer Folge von einer oder mehreren binären bzw. oktalen Ziffern, die in Hochkommata eingeschlossen sind, gefolgt von der Zeichenfolge B1 bzw. B3:

'binärziffer ...' B1 | 'oktalziffer ...' B3

Eine Oktalziffer entspricht drei Binärziffern:

oktal	binär
Ø	ØØØ
1	ØØ1
2	Ø1Ø
3	Ø11
4	1ØØ
5	1Ø1
6	11Ø
7	111

Beispiele:

binär	oktal	Attribut
1Ø1Ø11'B1	'53'B3	BIT(6)
'1Ø1'B1	'5'B3	BIT(3)
'ØØØ'B1	'Ø'B3	BIT(3)
'ØØ'B1	--	BIT(2)
'1'B1	--	BIT(1)
'111'B1	'7'B3	BIT(3)
'1Ø1Ø'B1	--	BIT(4)

Hinweis: Die Booleschen Wahrheitswerte werden durch eine Bitkette der Länge 1 dargestellt:

'1'B1 entspricht TRUE
'Ø'B1 entspricht FALSE

2-3.1.3 Zeichenketten

Zeichenketten-Werte besitzen das Attribut CHARACTER(n), wobei die ganzzahlige Konstante n die Anzahl der Zeichen angibt, aus denen die Zeichenkette besteht.
Im Subset darf eine Zeichenkette 1 bis 4Ø Zeichen enthalten, Zeichenketten-Bereiche dürfen ein-, zwei- oder dreidimensional sein.
Zeichenketten-Konstante bestehen aus einem oder mehreren Zeichen, die in Hochkommata eingeschlossen sind:

'zeichen ...'

Tritt innerhalb einer Zeichenkette ein Hochkomma auf, so muß diesem zur Kennzeichnung ein weiteres Hochkomma vorausgehen.

Beispiele:

Zeichenkette	Attribut
'MESSWERT␣='	CHARACTER(1Ø)
'Ø'	CHARACTER(1)
'␣␣'	CHARACTER(2)
'AUF␣GEHT"S'	CHARACTER(1Ø)

2-3.1.4 Zeitdaten

Zeitdaten sind

- Uhrzeiten
- Zeitintervalle.

Für beide Arten von Zeitdaten sind im Subset keine Bereiche zugelassen.

2-3.1.4.1 Uhrzeiten

Uhrzeit-Werte besitzen das Attribut CLOCK. Ihre Genauigkeit beträgt 1 Sekunde.
Eine Uhrzeit-Konstante setzt sich aus drei ganzen Zahlen zusammen, die durch zwei Doppelpunkte getrennt sind:

ganze-zahl: ganze-zahl: ganze-zahl

Die Zahlenangaben bedeuten (von links nach rechts):

- Stundenangabe (Zahlenbereich: Ø bis 23)
- Minutenangabe (Zahlenbereich: Ø bis 59)
- Sekundenangabe (Zahlenbereich: Ø bis 59)

Beispiele: Ø:Ø:Ø
12:Ø:Ø2
13:59:1

2-3.1.4.2 Zeitintervalle

Zeitintervall-Werte besitzen das Attribut DURATION.
Eine Zeitintervall-Konstante setzt sich - wenn vorhanden - aus Stunden-, Minuten- und Sekundenangaben zusammen, denen jeweils das entsprechende Schlüsselwort HRS, MIN oder SEC folgt:

```
ganze-zahl HRS [ganze-zahl MIN][ ganze-zahl[.ziffer    ] SEC]|
               ganze-zahl MIN [ ganze-zahl[.ziffer    ] SEC]|
                              ganze-zahl[.ziffer    ] SEC
```

Stunden- und Minutenangaben dürfen maximal vierstellige ganze Zahlen sein.
Die Sekunden können durch vierstellige ganze Zahlen oder durch rationale Zahlen ohne Exponent angegeben werden. Eine rationale Zahl darf dabei vor dem Dezimalpunkt vier Stellen und nach dem Dezimalpunkt eine Stelle besitzen.

Beispiele:
```
2ØØ MIN Ø.4 SEC
1224 HRS 1234 MIN 7654 SEC
 2ØØ HRS Ø.Ø SEC
  2Ø SEC
  2Ø.3 SEC
  12 MIN
   1 HRS Ø MIN
```

2-3.2 Programm-Steuerungsdaten

2-3.2.1 Marken

Mit Marken werden bestimmte Stellen im Programm gekennzeichnet, auf die an anderer Stelle Bezug genommen wird.

Hinweis: Eine Marke darf nicht durch eine weitere Marke gekennzeichnet werden.

Im Subset sind nur Markenkonstanten und Markenfelder, die mit Anweisungsmarkennamen initialisiert werden, zugelassen. Die Markenfelder müssen eindimensional sein und mit dem Attribut LABEL vereinbart werden.
Markenkonstanten bestehen aus einem Markennamen, dem ein Doppelpunkt folgt:

```
markenname:
```

Man unterscheidet drei Arten von Markenkonstanten:

- Anweisungsmarken
  Anweisungsmarken dürfen nur vor Anweisungen stehen.
- Eingangsnamen
  Eingangsnamen dienen der Identifizierung von Tasks und Prozeduren.
- Formatmarken
  Formatmarken identifizieren Formatlisten.

Beispiele:
```
T: TASK
   DECLARE MFELD(3) LABEL INITIAL (M1,M2,M3);
   GOTO MFELD(I);
M1: A=B+C;
   GOTO M3;
M2: D=A*2;
M3: CALL P;
   END; /*T*/
 P: PROCEDURE
 F: FORMAT (SKIP,X,(1Ø)A(3));
   END; /*P*/
```

2-3.2.2 Semaphor-Variable

Semaphor-Variable dienen der Koordinierung von Tasks. Sie bieten sehr anpassungsfähige Möglichkeiten, den Verkehr mehrerer Tasks mit einem Objekt (Gerät, Datei, Task, usw.) zu steuern (s. Kapitel 5/5-4.3.2 Anwendung von Semaphoren).
Semaphor-Variable enthalten immer ganzzahlige Werte. Sie müssen mit einem Attribut SEMA vereinbart werden und können mit vorzeichenlosen ganzen Zahlen initialisiert werden. Semaphor-Variable können nur mit bestimmten Anweisungen (RELEASE, REQUEST) verändert werden.

2-3.2.3 Ereignisdaten

Ereignisdaten können internen (Überlauf, Dateiende, usw.) oder externen (Meldungen der Peripherie, Prozeß-Alarme, usw.) Ursprungs sein. Das Eintreffen eines Ereignisses kann eine Reihe von Maßnahmen hervorrufen, die den Ablauf des Programmsystems ändern. So können aufgrund eines Ereignisses z.B. Tasks gestartet, angehalten, fortgesetzt oder beendet werden.
Im Subset gibt es zwei Arten von Ereignisdaten:

- Interrupts

  Ein Interrupt startet oder setzt eine Task fort, sofern eine solche Reaktion eingeplant ist.

- Signals

  Der weitere Ablauf einer Task kann durch ein Signal gesteuert werden. Dies setzt voraus, daß die Task eine Operation ausgelöst hat, die ein Signal erzeugt.

Die Herkunft der Ereignisdaten wird im Systemteil festgelegt. In den Problemteil werden sie durch Spezifikationen mit den Attributen INTERRUPT bzw. SIGNAL eingeführt (s. 2-4.2).

2-4. VEREINBARUNGEN FÜR BEZEICHNER

Ein Bezeichner in einem PEARL-Programm kann eine Reihe verschiedenartiger Objekte kennzeichnen. Er kann als Name

- einer Variablen
- eines Feldes
- einer Marke
- einer Task
- einer Prozedur
- eines Files
- eines Gerätes
- usw.

vereinbart werden.
Beim Auftreten eines Namens innerhalb eines Programms (Verwendung eines Namens) muß seine Bedeutung bekannt und eindeutig sein.
Die Bedeutung von Namen wird durch Vereinbarungen festgelegt. Die Vereinbarungen enthalten Angaben über die charakteristischen Eigenschaften

- der Objekte, die ein Name kennzeichnet (z.B. Datentyp)
- und des Namens selbst (z.B. Gültigkeitsbereich).

Sämtliche Eigenschaften bilden die Menge der Attribute, die einem Namen zugeordnet werden können.

Beispiele:

Attribut	Bedeutung
CHARACTER(3Ø)	Mit Namen, die dieses Attribut besitzen, können Zeichenketten der Länge 3Ø angesprochen werden.
DEVICE	Namen mit diesem Attribut bezeichnen Geräte.
GLOBAL	Namen, die dieses Attribut besitzen, sind in mehreren Modulen ansprechbar.

Durch Vereinbarungen wird die Bedeutung eines Namens für einen bestimmten Gültigkeitsbereich festgelegt. Vereinbarungen können sowohl Deklarationen, als auch Spezifikationen sein.
Sie können im Modul

- explizit (DECLARE-Vereinbarungen)
- textabhängig (Markenkonstanten, Eingangsnamen)
- implizit (Standardfunktionen)

auftreten.

2-4.1 Gültigkeitsbereich von Vereinbarungen

Wenn in einem Modul eine Vereinbarung über einen Bezeichner getroffen wird, so gibt es einen definierten Programmbereich, in dem der Bezeichner bekannt ist. Dieser Bereich heißt "Gültigkeitsbereich des Namens".
Im PEARL-Subset muß die Vereinbarung stets vor der Verwendung des Namens erfolgen. Ausgenommen von dieser Regel sind die Vereinbarungen von

- Markenkonstanten
- Tasks
- Prozeduren
- Daten auf Modulebene.

Definition des Gültigkeitsbereichs

Ein PEARL-Programm ist strukturiert; es kann folgende Programmbereiche enthalten:

- Modul
- Task
- Prozedur
- Block.

In jedem dieser Programmbereiche müssen bzw. können Vereinbarungen stehen (s. Kapitel 1).

Definition: Der Gültigkeitsbereich einer Vereinbarung für einen Bezeichner ist als der Programmbereich definiert, in dem die Vereinbarung liegt, aber ausschließlich aller in ihm enthaltenen Programmbereiche, in denen eine andere Vereinbarung für den gleichen Bezeichner liegt.

Anmerkung: Als "enthalten in" einem Programmbereich gilt der gesamte Programmtext zwischen Kopfanweisung (MODULE, TASK, PROCEDURE, BEGIN) und der MODEND- bzw. END-Anweisung; also auch weitere geschachtelte Programmbereiche. Nicht enthalten in einem Programmbereich ist dagegen der Eingangsname bzw. die Marke, der bzw. die eine Kopfanweisung kennzeichnet.

Beispiele: siehe 2-4.3.

2-4.2 Daten-Vereinbarungen

Namen für Datenelemente werden mit Ausnahme von Anweisungsmarken und Eingangsnamen durch die DECLARE-Vereinbarung deklariert bzw. spezifiziert.

Allgemeine Form:

DECLARE{{name|(namenliste)} [(feldgrenzen)] [VAL] datenattribut [GLOBAL]
[wertzuordnungsattribute]} [,{name|(namenliste)} usw.]...;

Die DECLARE-Vereinbarung wird als Spezifikation bezeichnet, wenn es sich bei den vereinbarten Größen um

- Interrupts, Signals
- globale Konstantennamen bzw. -felder ohne Wertzuordnungsattribute

handelt.

Die einzelnen Angaben in der DECLARE-Vereinbarung haben folgende Bedeutung:

- name | (namenliste)
  Name der zu vereinbarenden Größe.
  Mehrere Namen dürfen - durch Kommata getrennt - in einer Liste angegeben werden. Allen zugehörigen Größen werden dadurch die gleichen Attribute zugeordnet.

  Hinweis: Es ist zulässig, daß die Liste nur einen Namen enthält.

  Beispiele: ANTON
  (BERTA, B3, XYZ)
  (CAESAR)

- feldgrenzen
  Angabe der unteren und oberen Indexgrenze (getrennt durch Doppelpunkt) für jede Felddimension.
  Im Subset gelten folgende Einschränkungen:

  . Es gibt nur ein-, zwei- und dreidimensionale Felder.
  . Die untere Indexgrenze ist stets 1 und braucht nicht explizit angegeben werden.

  Beispiele: (1:1Ø, 1:2, 1:5)
  (1:2ØØ, 3Ø)
  (45)

- VAL
  Das VALUE-Attribut bezeichnet die Referenzstufe Ø (Konstantennamen, Konstantenfelder).

- datenattribut
  Das Datenattribut gibt den Typ und gegebenenfalls die Länge (Längenattribut) einer Größe an. Erlaubte Attribute sind:

  . FIXED
  . FLOAT
  . BIT[(länge)]
  . CHARACTER[(länge)]
  . DURATION
  . CLOCK
  . LABEL
  . SEMA
  . INTERRUPT
  . SIGNAL

  Hinweis: Fehlende Längenangabe bedeutet die Länge 1.

- GLOBAL
  Größen, die in mehreren Moduln bekannt sein sollen, müssen mit dem Attribut GLOBAL vereinbart werden.

- wertzuordnungsattribute

  Allgemeine Form:

  {INITIAL | IDENTICAL}([{+|-}] konstante [,[{+|-}] konstante]...)

  Mit den Wertzuordnungsattributen kann eine Besetzung von Datenelementen mit (vorzeichenbehafteten) Werten veranlaßt werden.
  Es gibt zwei Arten der Wertzuordnung:

  . Wertzuordnung bei Referenz-Ø-Größen:
    Sie erfolgt mit dem Schlüsselwort IDENTICAL. Der einmal vereinbarte Wert einer Referenz-Ø-Größe kann während des Programmablaufs nicht verändert werden.

  . Vorbesetzung von Referenz-1-Größen:
    Sie erfolgt mit dem Schlüsselwort INITIAL. Die vereinbarten Anfangswerte können während des Programmablaufs überschrieben werden. Bei einem erneuten Eintritt in den Programmbereich, in dem die Vereinbarung liegt, erhalten die Datenelemente auch erneut ihre Anfangswerte.

  Anmerkung:

  . Markenfelder können nicht verändert werden, da im Subset die Zuweisung für Datenelemente vom Typ LABEL nicht zugelassen ist.

  . Semaphor-Variable dürfen nur auf Modulebene vereinbart werden. Nach Veränderungen ihres Wertes ist daher eine erneute Vorbesetzung mit ihren Anfangswerten nicht möglich.

Für die Wertzuordnungen gelten folgende Regeln:

. Der Typ des Werts muß mit dem Datenattribut verträglich sein (s. 2-3.).

. Es ist nicht erlaubt mehr Werte anzugeben, als Datenelemente pro vereinbarter Größe vorhanden sind:

  .. Bei Variablen und Konstantennamen darf nur ein Wert angegeben werden. Er wird allen angegebenen Größen zugeordnet.

  .. Bei Bereichen dürfen maximal so viele Werte angegeben werden, wie Datenelemente pro Bereich vorhanden sind. Es werden die ersten Elemente (s. 2-1.3) aller angegebenen Bereiche mit der ersten Konstanten der Liste vorbesetzt, die zweiten Elemente mit der zweiten Konstanten, usw.
  Sind weniger Werte als Datenelemente vorhanden, so werden sämtliche überzähligen Elemente mit der letzten Konstanten vorbesetzt.

Beispiele:

```
DECLARE (A,B) VAL FIXED IDENTICAL (3),
        S SEMA GLOBAL INITIAL(1);

DECLARE (MF1, MF2) (3) LABEL INITIAL (M1,M2,M3),
        (F1,F2,F3) (3,2) FLOAT INITIAL (-1.Ø, 2.Ø, -3.Ø),
        (BF) (2) BIT(3) INITIAL ('1Ø1'B1),
        CF (3) VAL CHARACTER(4) IDENTICAL
                   ('EINS', 'ZWEI', 'DREI');
```

Datenelemente	Anfangswert
A, B	3
S	1
MF1(1), MF2(1)	M1
MF1(2), MF2(2)	M2
MF1(3), MF2(3)	M3
F1(1,1), F2(1,1), F3(1,1)	-1.Ø
F1(2,1), F2(2,1), F3(2,1)	2.Ø
F1(3,1), F2(3,1), F3(3,1)	-3.Ø
F1(1,2), F2(1,2), F3(1,2)	-3.Ø
F1(2,2), F2(2,2), F3(2,2)	-3.Ø
F1(3,2), F2(3,2), F3(3,2)	-3.Ø
BF(1)	'1Ø1'B1
BF(2)	'1Ø1'B1
CF(1)	'EINS'
CF(2)	'ZWEI'
CF(3)	'DREI'

Fehlerhafte Beispiele	Fehler
DECLARE (A,B,C) FIXED INITIAL (1,2,3);	Bei Variablen darf nur ein Wert angegeben werden.
DECLARE (A,B,C) VAL FIXED IDENTICAL (1.Ø);	Konstantentyp (FLOAT) ist nicht mit Datenattribut (FIXED) verträglich.
DECLARE (A,B) (3) FIXED INITIAL (1,2,3,4,5);	Zahl der angegebenen Werte ist größer als die Anzahl der Datenelemente pro Bereich.

2-4.3 Beispiele (Vereinbarungen, Gültigkeitsbereich)

Die nachfolgenden Beispiele sollen die vorausgegangenen Beschreibungen von Vereinbarungen und Gültigkeitsbereich veranschaulichen.

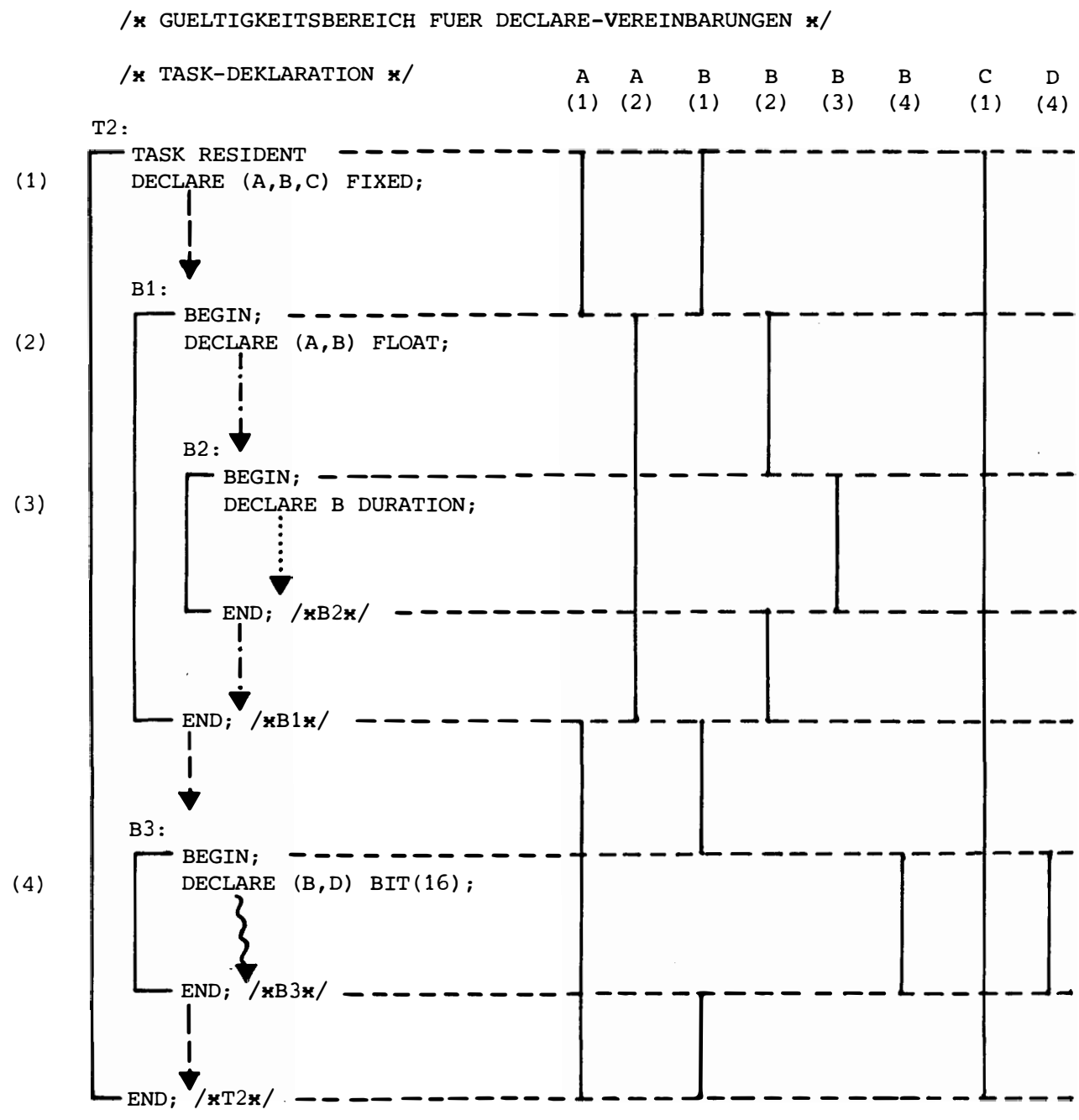
```
MODULE M;
/* BEISPIELE FUER VEREINBARUNGEN */
PROBLEM;

/* PROBLEMDATEN-VARIABLE, -FELDER */
DECLARE V1 FLOAT GLOBAL, F1(3,3,6) FLOAT;
DECLARE (F2,F3) (1Ø) BIT(1Ø);

/* KONSTANTENNAMEN, -FELDER */
/* SPEZIFIKATIONEN */
DECLARE K1 VAL FIXED GLOBAL, KF3 (1:4,5) VAL FIXED GLOBAL;
/* DEKLARATIONEN */
DECLARE K2 VAL DURATION IDENTICAL (1Ø HRS 3 MIN),
       (KF1, KF2) (3) VAL BIT(2) IDENTICAL ('ØØ'B1, 'Ø1'B1, '1Ø'B1);

/* SEMAPHOR-VARIABLE */
DECLARE (S1, S2) SEMA,
        (S3, S4) SEMA GLOBAL INITIAL (Ø);
DECLARE S5 SEMA INITIAL (1);

/* INTERRUPT, SIGNAL-SPEZIFIKATIONEN */
DECLARE INT VAL INTERRUPT GLOBAL,
        (SIG1, SIG2) VAL SIGNAL GLOBAL;
DECLARE SIG3 VAL SIGNAL GLOBAL;
```

Fehlerhafte Beispiele:

Beispiel	Fehler
`DECLARE A,B,C FIXED;`	Namenliste muß eingeklammert werden.
`DECLARE (A,B) FIXED, B FIXED;`	B ist mehrfach vereinbart.

`MODULE M;`
`PROBLEM;`
`DECLARE A VAL INTERRUPT;`
	GLOBAL-Attribut fehlt.

`DECLARE MF(1) LABEL INITIAL`
`(L1);`
`MODEND;`
	Markenfeld-Deklaration auf Modulebene nicht zugelassen.


/* GUELTIGKEITSBEREICH FUER DECLARE-VEREINBARUNGEN */
/* TASK-DEKLARATION */
A (1)
A (2)
B (1)
B (2)
B (3)
B (4)
C (1)
D (4)
T2:
TASK RESIDENT
(1)
DECLARE (A,B,C) FIXED;
B1:
BEGIN;
(2)
DECLARE (A,B) FLOAT;
B2:
BEGIN;
(3)
DECLARE B DURATION;
END; /*B2*/
END; /*B1*/
B3:
BEGIN;
(4)
DECLARE (B,D) BIT(16);
END; /*B3*/
END; /*T2*/

```
/* GUELTIGKEITSBEREICH FUER EINGANGSNAMEN UND ANWEISUNGSMARKEN */

                                        P2  P3  MF  L1  L1  L2  L2  L2  L3
                                       (1) (2) (3) (4) (6) (5) (7) (9) (8)
(1)  P2:
     PROCEDURE REENTRANT
          |
          v
     END; /*P2*/
(2)  P3:
     PROCEDURE
     DECLARE (A,B,C) FIXED;
(3)  DECLARE MF(3) LABEL INITIAL
                   (L1,L2,L3);
          |
          v
(4)  L1: A=B;
          |
          v
(5)  L2:
     BEGIN; /*B1*/
          |
          v
(6)    L1: A=C;
       GOTO L2;
          |
          v
       BEGIN; /*B2*/
          :
          v
(7)    L2: CALL P2;
          :
          v
       END; /*B2*/
          |
          v
       GOTO L1;
     END; /*B1*/
          |
          v
(8)  L3:
     BEGIN; /*B3*/
          |
          v
(9)    L2: A=B;
       GOTO L1;
          |
          v
     END; /*B3*/
          |
          v
     GOTO MF(C);
          |
          v
END; /*P3*/
MODEND;
```

KAPITEL 3: AUSDRÜCKE

3-1. AUFBAU EINES AUSDRUCKS

Ein Ausdruck ist eine Folge von Operanden und Operatoren, die eine Berechnungsvorschrift für einen Wert darstellt. Im ASME-PEARL-Subset (Stufe 1) gibt es nur skalare Ausdrücke. Sie haben folgende allgemeine Form:

[monadischer-operator] operand [{dyadischer-operator operand} ...]

Eine einzelne Konstante oder Variable kann danach bereits ein Ausdruck sein.

3-1.1 Operanden

Ein Ausdruck darf folgende Operanden besitzen:

Operand	Beispiele
Konstante	5, 'Ø1Ø'B1
Konstantenname	K
Konstantenfeldelement	A(5), A(I,K,J*3)
Variable	V
Feldelement (indizierte Variable)	A(3,K)
Aufruf einer Funktions-Prozedur	F(P1,P2)
Ausdruck in Klammern	(A+B(I))

3-1.2 Operatoren

Die Operatoren bestimmen die Art der Verknüpfung von Operanden. In den folgenden Tabellen sind die im Subset zugelassenen Operatoren und ihre Prioritäten zusammengestellt (Priorität 1 = höchste Priorität).

3-1.2.1 Monadische Operatoren

Monadischen Operatoren darf kein Operand oder Operator unmittelbar vorausgehen. Sie können daher nur zu Beginn eines Ausdrucks oder nach einer geöffneten Klammer stehen.

Operator	Art der Verknüpfung	Priorität
+	arithmetisch	1
-		1
NOT	logisch	1

Beispiele: -5
+A(I,K)
B1 AND (NOT C)

fehlerhafte Ausdrücke	Art des Fehlers
-+5	Zwei Operatoren folgen unmittelbar aufeinander
B1 NOT C	Dem monadischen Operator geht ein Operand voraus.

3-1.2.2 Dyadische Operatoren

Ein dyadischer Operator steht immer zwischen zwei Operanden.

Operator	Art der Verknüpfung	Priorität
**	arithmetisch	1
*		2
/		2
+		3
-		3
AND	logisch	6
OR		7
EXOR		7
GT	Vergleich	4
GE		4
LE		4
LT		4
EQ		5
NE		5
//	Verkettung	3
RBIT	Bitoperation	2
LBIT		2
SHIFT		3
CHAR	Zeichenselektion	2

3-1.3 Abarbeitung von Ausdrücken

Die Reihenfolge der Abarbeitung von Verknüpfungen in einem Ausdruck hängt von den Prioritäten der Operatoren ab. Es gelten folgende Regeln:

- Verknüpfung mit höherprioren Operatoren werden zuerst ausgeführt.
- Enthält ein Ausdruck Verknüpfungen mit Operatoren gleicher Priorität der Klassen 2 bis 7, so werden diese von links nach rechts abgearbeitet.

- Enthält ein Ausdruck Verknüpfungen mit Operatoren gleicher Priorität der Klasse 1 (monadische Operatoren, Exponentiation), so werden diese von rechts nach links ausgeführt.

- Klammern ändern die Ausführungs-Reihenfolge der einzelnen Verknüpfungen. Sie haben die in der Arithmetik übliche Bedeutung. Die Klammerstruktur eines Ausdrucks wird aufgelöst, indem nacheinander der jeweils innerste Klammerausdruck nach den obigen Regeln berechnet wird.

  Hinweis: Klammern dürfen alle Operanden nach 3-1.1 umschließen. Sie müssen gesetzt werden, wenn

  . der Nenner eines Bruchs aus mindestens einer Verknüpfung besteht (Ausnahme: Exponentiation im Nenner)
  . ein monadischer Operator innerhalb eines Ausdrucks vorkommt.

Beispiele:

Ausdruck	Abarbeitung
-A+B+C*D/E	-A+B: Zwischenergebnis Z1 C*D: Zwischenergebnis Z2 Z2/E: Zwischenergebnis Z3 Z1+Z3: Ergebnis
-A**B**C	B**C: Zwischenergebnis Z1 A**Z1: Zwischenergebnis Z2 -Z2: Ergebnis
(A*(B+FKT(C))+C/D**2	B+FKT(C): Zwischenergebnis Z1 A*Z1: Zwischenergebnis Z2 D**2: Zwischenergebnis Z3 C/Z3: Zwischenergebnis Z4 Z2+Z4: Ergebnis

Voraussetzung für die Ausführung einer Verknüpfung ist, daß die beteiligten Operanden die dafür zugelassenen Datenattribute besitzen (s. 3-2.1 bis 3-2.6). Eine Typ-Wandlung erfolgt nur bei der Zuweisung (siehe Kapitel 4-4.2), nicht während der Abarbeitung eines Ausdrucks.

Hinweis: Zu den Werten der Operanden einer Verknüpfung wird nicht in einer bestimmten Reihenfolge zugegriffen. Das Ergebnis darf daher nicht von einer bestimmten Reihenfolge des Wertezugriffs abhängen.

3-2. VERKNÜPFUNGEN

3-2.1 Arithmetische Verknüpfung

Allgemeine Form:

$$[\{+|-\}]\ \text{operand}\ [\{\{**\ |\ *\ |\ /\ |\ +\ |\ -\}\ \text{operand}\}\ \cdots]$$

Die folgende Tabelle enthält eine Zusammenstellung der zulässigen Operanden-Typen und des Ergebnis-Typs für arithmetische Verknüpfungen:

Operator	Typ linker Operand A	Typ rechter Operand B	Typ Ergebnis	Verknüpfung
+	-- -- -- --	FIXED FLOAT DURATION CLOCK	FIXED FLOAT DURATION CLOCK	+B
-	-- -- --	FIXED FLOAT DURATION	FIXED FLOAT DURATION	-B
**	FIXED FLOAT	FIXED FLOAT	FIXED FLOAT	A**B
*	FIXED FLOAT DURATION FIXED	FIXED FLOAT FIXED DURATION	FIXED FLOAT DURATION DURATION	A*B
/	FIXED FLOAT	FIXED FLOAT	FIXED FLOAT	A/B (B≠Ø)
+	FIXED FLOAT DURATION DURATION CLOCK	FIXED FLOAT DURATION CLOCK DURATION	FIXED FLOAT DURATION CLOCK CLOCK	A+B
-	FIXED FLOAT DURATION CLOCK CLOCK	FIXED FLOAT DURATION DURATION CLOCK	FIXED FLOAT DURATION CLOCK DURATION	A-B

Besonderheiten:

- Division:
  - Division durch Null ist verboten.
  - Besitzen Zähler und Nenner das Attribut FIXED, so ist das Ergebnis der Division diejenige Zahl, die sich nach dem Abschneiden der Dezimalbrüche des Quotienten ergibt.

- Exponentiation:
  - Besitzen Basis und Exponent das Attribut FIXED, so gelten, wenn der Exponent negativ ist, die gleichen Regeln wie bei der Division (s.o.).

Beispiele:

Verknüpfung	Ergebnis
5/2	2
-5/2	-2
1/3	Ø
1**(-5) ≙ 1/(1**5)	1
5**(-5) ≙ 1/(5**5)	Ø

3-2.2 Logische Verknüpfung

Allgemeine Form:

[NOT] operand [{{ AND | OR | EXOR } operand} ···]

Durch logische Operatoren werden ausschließlich Bitketten gleicher Länge miteinander verknüpft.

Operator	Typ linker Operand A	Typ rechter Operand B	Ergebnis	Verknüpfung
NOT	--	BIT(n)	BIT(n)	NOT B
AND	BIT(n)	BIT(n)	BIT(n)	A AND B
OR				A OR B
EXOR				A EXOR B

Das Ergebnis einer logischen Verknüpfung wird Bitstelle für Bitstelle ermittelt. Jede Bitstelle des Ergebnisses hat einen Wert, der in der folgenden Tabelle festgelegt ist.

BIT1	BIT2	NOT BIT1	NOT BIT2	BIT1 AND BIT2	BIT1 OR BIT2	BIT1 EXOR BIT2
1	1	Ø	Ø	1	1	Ø
1	Ø	Ø	1	Ø	1	1
Ø	1	1	Ø	Ø	1	1
Ø	Ø	1	1	Ø	Ø	Ø

Beispiele: A: 0 1 0 1 1 0

B: 1 1 1 1 1 1

C: 1 0 1 0 0 1

A AND B OR C ergibt: 1 1 1 1 1 1

A AND (B OR C) ergibt: 0 1 0 1 1 0

NOT A EXOR B AND C ergibt: 0 0 0 0 0 0

3-2.3 Vergleich

Allgemeine Form:

operand { { GT | GE | LE | LT | EQ | NE } operand } ...

Abhängig von der Art der Operanden kann man folgende Vergleiche unterscheiden:

- Algebraischer Vergleich:
  Die Operanden sind arithmetische Ausdrücke.
- Vergleich von Zeiten:
  Es können sowohl Uhrzeiten als auch Zeitdauern verglichen werden.
- Vergleich von Bitketten:
  - . Logische Verknüpfungen innerhalb eines Vergleichs müssen eingeklammert werden, wenn die Prioritäten ihrer Operatoren kleiner sind als die der Vergleichsoperatoren.
  - . Zu vergleichende Bitketten müssen die gleiche Länge besitzen.
- Vergleich von Zeichen:
  Zu vergleichende Zeichenketten müssen die gleiche Länge besitzen.

Die für die einzelnen Vergleiche zugelassenen Vergleichsoperatoren sind in der folgenden Tabelle zusammengestellt:

Operator	Typ linker Operand A	Typ rechter Operand B	Ergebnis	Verknüpfung
GT GE LE LT	FIXED FLOAT DURATION CLOCK	FIXED FLOAT DURATION CLOCK	BIT(1)	A GT B A GE B A LE B A LT B
EQ NE	FIXED FLOAT DURATION CLOCK BIT(n) CHARACTER(n)	FIXED FLOAT DURATION CLOCK BIT(n) CHARACTER(n)	BIT(1)	A EQ B A NE B

Das Ergebnis eines Vergleichsausdrucks ist stets eine Bitkette der Länge 1. Ihr Wert ist

- 1, wenn der Vergleich "wahr" ist
- Ø, wenn der Vergleich "falsch" ist.

Dies ist zu berücksichtigen, wenn mehrere Operanden durch Vergleichsoperatoren verknüpft werden. Unter Umständen müssen dann manche Operanden das Attribut BIT(1) besitzen (s. Beispiele).

```
Beispiele:  DECLARE (A,B,I,J,K) FIXED,
                    (C,D) BIT(5),
                    (E,F) BIT(2),
                    G(5,5,5) FIXED;
```

Folgende Vergleichsausdrücke sind erlaubt:

```
A+B GT G(I,J,K) * 3/B
E//F//'1'B1 NE ('1ØØØ1'B1 OR C)
A LE B  EQ  I LT K
A GT B  EQ  '1'B1
```

Nicht erlaubt sind folgende Vergleichsausdrücke:

Beispiel	Fehler
E GT F	GT ist bei Bitketten nicht zugelassen.
E EQ C	E und C haben verschiedene Längen.
E//F//'1'B1 NE '1ØØØ1'B1 OR C	Die OR-Verknüpfung wird nach dem Vergleich durchgeführt. Damit stimmen dann die Längen nicht mehr überein (linker Operand der OR-Verknüpfung hat den Typ BIT(1)).
A GT B EQ '11'B1	Die Längen der Operanden stimmen nicht überein (A GT B ergibt den Typ BIT(1)).
A GT B LT A-3	Die Typen der Operanden stimmen nicht überein (A GT B ergibt den Typ BIT(1), A-3 den Typ FIXED.

3-2.4 Kettung

Allgemeine Form:

operand {// operand} ...

Es können nur Bit- oder Zeichenfolgen verkettet werden.

Operator	Typ linker Operand A	Typ rechter Operand B	Typ Ergebnis	Verknüpfung
//	BIT(n) CHARACTER(n)	BIT(m) CHARACTER(m)	BIT(p) CHARACTER(p)	A//B

Das Ergebnis der Kettung ist eine Bit- bzw. Zeichenfolge, deren Länge gleich der Summe der Längen der beiden Operanden ist (p = m+n). Die Länge p darf bei einer Bitkette maximal 24, bei einer Zeichenkette maximal 4Ø betragen. Die beiden Operanden werden so verbunden, daß das letzte Bit bzw. Zeichen des linken Operanden unmittelbar vor dem ersten Bit bzw. Zeichen des rechten Operanden steht.

Beispiele:

A: | 1 | Ø | 1 | Ø |

B: | 1 | 1 |

C: | W | A | U |

A//B// 'ØØ'B1 ergibt: | 1 | Ø | 1 | Ø | 1 | 1 | Ø | Ø |

C//C ergibt: | W | A | U | W | A | U |

3-2.5 Bitoperation

Allgemeine Form:

operand {{ RBIT | LBIT | SHIFT } operand} ...

Die Bitoperationen gestatten es,

- einzelne Bits innerhalb von Bitketten anzusprechen (Bitselektion)
- Bitketten nach rechts und nach links zu schieben.

Operator	Typ linker Operand A	Typ rechter Operand B	Typ Ergebnis	Verknüpfung
RBIT LBIT	FIXED	BIT(n)	BIT(1)	A RBIT B A LBIT B
SHIFT	BIT(n)	FIXED	BIT(n)	A SHIFT B

Das Ergebnis einer Bitselektion ist der Wert des Bits des rechten Operanden, dessen Stellung innerhalb der Bitkette (von rechts (RBIT) bzw. links (LBIT) gezählt)) durch den linken Operanden angegeben wird. Der linke Operand muß eine ganze positive Zahl ergeben, die maximal gleich der Länge des rechten Operanden ist.

Als Ergebnis der Verschiebung erhält man die Bitkette, die sich ergibt, wenn man den linken Operanden um die Anzahl von Stellen verschiebt, die durch den rechten Operanden angegeben ist und die nachfolgenden Bitstellen mit Nullen besetzt.
Das Vorzeichen der Verschiebezahl ergibt die Schieberichtung:

- Verschiebezahl Ø: Verschieben nach links
- Verschiebezahl Ø: Verschieben nach rechts
- Verschiebezahl = O: Kein Verschieben

Beispiele: A: 1 Ø 1 Ø 1 Ø
B: 1 1 Ø 1 1 Ø

2 RBIT A	ergibt:	1
4 LBIT (A AND B)	ergibt:	Ø
A SHIFT (-2)	ergibt:	Ø Ø 1 Ø 1 Ø
A//B SHIFT 6	ergibt:	1 1 Ø 1 1 Ø Ø Ø Ø Ø Ø Ø
'11'B1 SHIFT (-3)	ergibt:	Ø Ø

3-2.6 Zeichenselektion

Allgemeine Form:

operand CHAR operand

Durch Zeichenselektion können einzelne Zeichen innerhalb einer Zeichenkette angesprochen werden.

Operator	Typ			Verknüpfung
	linker Operand A	rechter Operand B	Ergebnis	
CHAR	FIXED	CHARACTER(n)	CHARACTER(1)	A CHAR B

Das Ergebnis einer Zeichenselektion ist der Wert des Zeichens des rechten Operanden, dessen Stellung innerhalb der Zeichenkette (von links gezählt) durch den linken Operanden angegeben wird. Der linke Operand muß eine ganze positive Zahl ergeben, die maximal gleich der Länge des rechten Operanden ist.

Beispiele: A: 1 2 3 4 5
B: A B C D E

2 CHAR A	ergibt:	2
3 CHAR B//5 CHAR A	ergibt:	C 5

3-2.7 Kombinationen von Verknüpfungen

Kombinationen sind - wie in den Beispielen zu 3-2.1 bis 3-2.6 bereits angedeutet - in großem Umfang erlaubt. Voraussetzung dafür ist, daß beim Auflösen eines Ausdrucks (Klammerstruktur, Operatorenpriorität (s. 3-1.3)) bis hin zum Ergebnis jede Teil-Verknüpfung (s. 3-2.1 bis 3-2.6) zulässig ist.

Beispiele:

```
DECLARE (I,J,K) FIXED,
        (B1,B2) BIT(1Ø),
        F(3,5,5) FLOAT,
        B3 BIT(5);

  I*3/J RBIT (B1 OR B2) EQ I LBIT B1
  '1'B1 EQ F(I,J,K) GT F(1,1,1)+3.Ø
((B1 AND B2 OR (NOT B1)) SHIFT (-I))//B3
```

KAPITEL 4

ANWEISUNGEN:

Die Anweisungen können in folgende logische Gruppen eingeteilt werden: Blockbildungs-Anweisungen, Zuweisungs-Anweisung, Anweisungen zur Steuerung des sequentiellen Programmablaufs, Anweisungen für die parallele Task-Ablaufsteuerung, Anweisung für die Interruptbehandlung, Anweisung zur Synchronisation von Tasks sowie Anweisungen zur Eingabe/Ausgabe. Allen Anweisungen mit Ausnahme der Format-Anweisung kann eine Markenkonstante vorausgestellt werden. Die Format-Anweisung ist bindend mit einer Format-Marke zu versehen. Jede Anweisung ist mit einem Semikolon abzuschließen.

Allgemeine Form:

anweisung::= [marken-konstante:] nicht-markierte-anweisung;

4-1. BLOCKBILDUNGS-ANWEISUNGEN

Die Blockbildungs-Anweisungen BEGIN und END klammern einen Programm-Block ein. Ein Block kann beliebig viele Deklarationen und Anweisungen enthalten, mindestens jedoch eine Anweisung. Innerhalb des Blockes gelten für die Verwendung von Namen und Marken gewisse Regeln (siehe Kapitel 2, 2-4.1 Gültigkeitsbereich von Vereinbarungen).

Syntax:
block ::= BEGIN ; [deklaration...] {anweisung...} END;

4-2. ZUWEISUNGS-ANWEISUNG

Mit einer Zuweisungs-Anweisung kann der Wert eines Ausdrucks einer oder mehreren "Ziel-Variablen" zugewiesen werden. "Ziel-Variable" können sein:

- Variable
- Feldelemente

Syntax:
ziel-variable, zielvariable, ... = ausdruck;

Beispiele:

```
X = Y + Z/4;
FELD(I) = 'TEXTINHALT=' // TEXT;
AFELD (I, K), BFELD(I, K+1), CFELD(I, K+2) = FUNK(I);
```

4-2.1 Konvertierung bei der Zuweisung

Ist der Resultat-Typ des Ausdrucks rechts vom Gleichheitszeichen unterschiedlich von dem Typ der Ziel-Variablen links vom Gleichheitszeichen, so findet mit Ausnahme der Typen DURATION und CLOCK eine Typkonvertierung statt. Innerhalb des Ausdrucks findet keine Typkonvertierung statt; hier gilt die Regel für den Aufbau eines gültigen Ausdrucks, die besagt, daß jeweils bei der Auflösung eines Ausdrucks jede Teil-Verknüpfung zulässig sein muß (siehe Kapitel 3).
Nicht ausführbare Konvertierungen führen auf Signal-Reaktionen.

Konvertierungs-Tabelle:

In der Konvertierungs-Tabelle sind die erlaubten Konvertierungen mit dem für sie gültigen Konvertierungs-Bereich α und β aufgeführt. Die Konvertierung ist immer vom Typ des Ausdruck-Ergebnisses zum Typ der Ziel-Variablen gerichtet. Die jeweiligen Höchstzahlen, posmax für höchste positive ganze Zahl und negmax für kleinste negative ganze Zahl, sind zielmaschinenabhängig verschieden.

Fall	α → β		Konvertierungsbereich
1	FLOAT	FIXED	negmax $\leq \alpha \leq$ posmax
2	BIT(n)	FIXED	n $\leq$ 24 erstes Bit wird als Vorzeichen interpretiert, negative Ganzzahlen werden im 2er Komplement dargestellt
3	FIXED	FLOAT	
4	BIT(n)	FLOAT	zurückgeführt auf Fall 2 und Fall 3
5	FIXED	BIT(n)	n $\leq$ 24
6	FLOAT	BIT(n)	zurückgeführt auf Fall 1 und Fall 5
7	BIT(n)	BIT(m)	max (n, m) $\leq$ 24
8	CHAR(n)	CHAR(m)	max (n, m) $\leq$ 40

Beschreibung der Konvertierungsfälle:

Fall 1 und 2: Konvertierung in eine FIXED-Größe

Gemeinsam für diese Konvertierungsfälle gilt, daß das Dezimalequivalent des zu konvertierenden Ausdruck-Ergebnisses nicht über den Höchstzahlen liegen darf, sonst kann keine eindeutige Konvertierung mehr stattfinden.

Fall 1: FLOAT → FIXED

Die Konvertierung erfolgt, den Absolutwert der zu konvertierenden Größe betrachtet, immer auf die nächstkleinere ganze Zahl (sog. Abschneiden!). In nachfolgender Tabelle werden die möglichen Konvertierungen und deren Grenzfälle beschrieben.

Beispiel:

Zahl	Konvertierungsbereich	Konvertierungsergebnis
-5E3	-8388608 $\leq$ Bereich $\leq$ -1	-5000
-7E-1 2.1E-2 }	-1 < Bereich < 1	ø
1.341E1	1 $\leq$ Bereich $\leq$ 8388607	13

Fall 2: BIT(n)→ FIXED

Die Konvertierung einer BIT-Kette in eine Ganzzahl ist im Subset bis zu einer Kettenlänge von 24 Bits möglich. Das erste Bit von links gesehen wird als Vorzeichen interpretiert. Demnach werden BIT-Ketten mit führender "Null" in positive Ganzzahlen konvertiert. BIT-Ketten mit führender "EINS" werden im 2er-Komplement interpretiert und in eine negative Ganzzahl konvertiert, wobei sich das Komplement immer auf die vereinbarte Bitlänge n bezieht.

Beispiele:

'Ø111'B1 → +7
'1ØØØ'B1 → -8
'1111'B1 → -1
'1'B1 → -1
'11'B1 → -1
'Ø'B1 → Ø

Fall 3 und 4: Konvertierung in eine FLOAT-Größe

Gemeinsam für diese Fälle gilt, daß die Genauigkeit der Konvertierung eines Ausdruck-Ergebnisses in den Typ FLOAT im wesentlichen von der Intern-Darstellung der FLOAT-Größe abhängt, so daß maschinenunabhängig nur qualitativ auf die Konvertierungsmechanismen eingegangen werden kann.

Fall 3: FIXED → FLOAT

Das Ergebnis vom Typ FIXED wird in die normierte Mantissendarstellung gebracht. Der Exponent berücksichtigt die Dezimalpunktverschiebung.

Beispiel:

132 → 1.32E2
-1 → -1.EØ

Fall 4: BIT → FLOAT

Die Konvertierung wird auf die Fälle 2 und 3 zurückgeführt. Es gelten die dort gemachten Einschränkungen.

Beispiel:

'1000'B1 → (-8) → -8.EØ

Fall 5: FIXED → BIT(n)

Im Subset kann wegen der Längenbegrenzung der BIT-Ketten auf maximal n = 24 nur eine eindeutige Konvertierung von Ganzzahlen in BIT-Ketten innerhalb des Zahlenbereich $-2^n \leq z \leq 2^n-1$ stattfinden.

Beispiel:

512 → 'Ø1ØØØØØØØØØ'B1
-1 → '1'B1 n mal Bit 1

Fall 6: FLOAT → BIT(n)

Die Konvertierung wird auf die Fälle 1 und 5 zurückgeführt. Es gelten die dort gemachten Einschränkungen.

Beispiel:

5.15E2 → (515) → 'Ø1000000011'B1

Fall 7 und 8: Längenwandlung von CHARACTER- und BIT-Grössen
BIT(n) → BIT(m) und CHAR(n) → CHAR(m)

Bestimmend für die Konvertierung ist das Längenattribut der Zielvariablen (m). Das Ergebnis des Ausdrucks wird linksbündig in der Zielvariablen abgelegt. Für n < m werden die freien Stellen bei Bit-Größen mit Nullen, bei Charakter-Größen mit Leerzeichen aufgefüllt. Für n > m werden die nicht mehr faßbaren Binären-Ziffern bei Bit-Größen bzw. Zeichen bei Charakter-Größen rechts abgeschnitten. Die Höchstgrenzen für n und m sind bei Bit-Größen 24, bei Charakter-Größen 40.

4-3. ANWEISUNGEN ZUR STEUERUNG DES SEQUENTIELLEN PROGRAMMABLAUFS

4-3.1 GOTO-Anweisung

Mit der GOTO-Anweisung kann die Reihenfolge der Bearbeitung der im Quelltext stehenden Anweisungen gesteuert werden. Die GOTO-Anweisung bewirkt, daß als nächste Anweisung die mit der GOTO-Anweisung ausgewählte markierte Anweisung ausgeführt wird.

Syntax:

GOTO { markenkonstante | indizierte Markenvariable } ;

Regeln:

- der Index der Markenvariablen kann eine Variable vom Typ FIXED oder eine ganzzahlige Konstante sein
- Mit einer GOTO-Anweisung sind nur Anweisungen erreichbar, die im gleichen Block oder einem dazu äußeren Block liegen.
- Wird ein Block verlassen, so sind alle innerhalb des Blockes definierten Variablen unbekannt und ihr Inhalt undefiniert.
- Der Körper einer Prozedur oder Task kann mit einer GOTO-Anweisung nicht verlassen werden.

Beispiel:

```
DECLARE M(3)  LABEL INITIAL (M1, M2, M3);
              .
              .
              .
            M2:
                    .
                    .
                    .
              I = 2;
              GOTO M(I);
                    .
                    .
                    .
```

4-3.2 IF-Anweisung

Mit der IF-Anweisung kann in Abhängigkeit vom Ergebnis eines Vergleichsausdrucks festgelegt werden, welche Anweisung als nächste ausgeführt werden soll:

Syntax:

```
IF ausdruck
THEN anweisung
[ELSE anweisung]
FI;
```

Regeln:

- Die Bedingung in der IF-Anweisung besteht aus einem Ausdruck, dessen Resultat vom Typ BIT(1) sein muß (siehe Kapitel 3).
- Liefert die Bedingung den Wert '1' B1, so wird der THEN-Zweig ausgeführt, liefert sie den Wert 'Ø' B1, so wird der ELSE-Zweig ausgeführt. Fehlt der ELSE-Zweig und liefert die Bedingung den Wert 'Ø' B1, so wird die nächste Anweisung nach der IF-Anweisung ausgeführt.
- Nach Ausführung des THEN- oder ELSE-Zweigs wird, falls der Zweig nicht durch eine GOTO-Anweisung verlassen wird, die auf die IF-Anweisung folgende Anweisung ausgeführt.
- Bei geschachtelten IF-Anweisungen wirken die Schlüsselwörter IF und FI wie Anweisungsklammerungen (Schachtelung bis Tiefe 15 zugelassen).

Beispiele:

```
DECLARE I FIXED INITIAL (3),
        A BIT(5) INITIAL ('1Ø1Ø1'B1),
       MASKE BIT(5) INITIAL ('11ØØØ'B1);

IF I LBIT A THEN GOTO MARKE1;  FI;
IF I LE 5 THEN
   IF A NE ( NOT MASKE) THEN GOTO MARKE2;
   FI;
          ELSE GOTO MARKE3;
FI;
```

4-3.3 Leer-Anweisung

Syntax:

```
[Leer-Zeichen ...];
```

Die Leer-Anweisung hat keine Wirkung auf die sequentielle Ausführung der Anweisungen.

Beispiele:

```
MARKE:;
IF A GT B THEN A=B; ELSE; FI;
```

4-3.4 Schleifen-Anweisung

Mit der Schleifen-Anweisung kann die einmalige oder mehrfache Ausführung einer Anweisungsfolge bestimmt werden:

Syntax:

	es bedeutet:
[FOR variable]	Zählvariable
[FROM ausdruck]	Anfangswert
[BY ausdruck]	Schrittweite
[TO ausdruck]	Schleifenendwert

```
REPEAT {anweisung ···}
END;
```

Regeln:

- Die Zählvariable muß das Attribut FIXED besitzen.
- Die Zählvariable kann fehlen, es ist aber dann kein Zugriff zum aktuellen Zählerstand mehr möglich.
- Die Resultate der in der Schleifen-Anweisung zugelassenen Ausdrücke für Anfangswert, Schrittweite und Schleifenendwert müssen vom Typ FIXED sein.
- Für fehlenden Anfangswert und Schrittweite werden folgende Ersatzwerte angenommen:

  Anfangswert =1
  Schrittweite =1

- Für den Fall, daß der Schleifenendwert fehlt, kann die Anweisungsfolge unbegrenzt oft durchlaufen werden.
- Enthält die zu wiederholende Anweisungsfolge Sprungziele, so muß sie mit den Blockbildungs-Anweisungen BEGIN; und END; eingeschlossen werden; ein Einspringen in die Schleife ist somit unmöglich.

  Beispiel:

```
FOR I TO 100 REPEAT BEGIN;
                    .
                    .
                    .
                M1:A=A+1;
                    .
                    .
                    .
                GOTO M1;
                    .
                    .
                    .
                END;
            END;
```

- Schachtelung von Schleifen sind bis zu einer Tiefe von 15 zugelassen.

Die Wirkungsweise der Schleifenanweisung kann mit Ausnahme obiger Sonderfälle und Vorschriften im folgenden Flußdiagramm für die Anweisung
FOR I FROM A BY B TO C REPEAT S; ... END; verdeutlicht werden, wobei A, B, C die Resultate von Ausdrücken vom Typ FIXED darstellen und S; ... eine Anweisungsfolge symbolisiert.

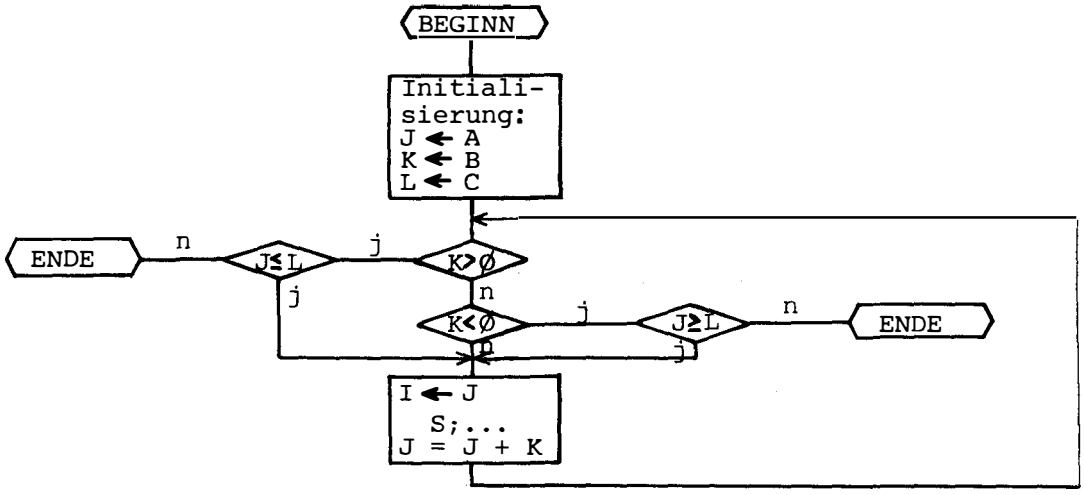


Tabelle zur Fallunterscheidung für positive und negative Schleifenparameter

Fall	FROM A	BY B	TO C	Beschreibung AF = Anweisungsfolge n = Ø, 1, 2 ....
Ø	+	+	+	Die AF wird, solange C ≥ A+n.B ist, (n+1) mal durchlaufen, sog. hochzählen
1	+	+	-	Die AF wird nicht durchlaufen
2	+	-	+	Die AF wird, solange C ≤ A+n·B ist, (n+1) mal durchlaufen, sog. herunterzählen
3	+	-	-	wie Fall 2, herunterzählen
4	-	+	+	wie Fall 1 } , hochzählen
5	-	+	-	wie Fall 1 }
6	-	-	+	Die AF wird nicht durchlaufen
7	-	-	-	wie Fall 2, herunterzählen

Beispiel:

```
FOR I FROM 1 BY 2 TO 20 REPEAT
      MOVE(MESSTL(I)) TO MESSWT(I);
                          END;
```

4-3.5 CALL-Anweisung

Mit der CALL-Anweisung wird die Ausführung einer Prozedur veranlaßt.

Syntax:

CALL name (aktueller-parameter , aktueller parameter ...) ;

Regeln:

- Die Anzahl der aktuellen Parameter muß mit der Anzahl der formalen in der Prozedur-Vereinbarung übereinstimmen. Es sind maximal 31 Parameter zugelassen.
- Die Attribute (Typ, Feldgrenzen) der aktuellen und formalen Parameter müssen übereinstimmen (s. 6-1.).
- Die Referenzstufen der aktuellen Parameter müssen größer oder gleich denen der zugehörigen formalen Parameter sein.
- Als aktuelle Parameter sind zugelassen:

  . Ausdrücke
  . Konstante, Konstantennamen
  . Variable
  . (Konstanten-)Feldelemente
  . (Konstanten-)Felder.

Der Parameter-Übergabemechanismus ist in 6-2.4.2 beschrieben.
- Handhabung von Prozeduren siehe 6-2.

Beispiel:

```
CALL PROZØ1 (2.5,'1Ø1'B1, ALPHA  (3), SWITCH, 'ABCDE', FELD);
```

4-3.6 RETURN-Anweisung

Die RETURN-Anweisung bewirkt die Rückkehr aus einer Prozedur. Die aufrufende Task bzw. Prozedur wird danach an der Stelle unmittelbar nach dem Prozeduraufruf fortgesetzt. Eine Rückkehr aus Prozeduren mit Hilfe der GOTO-Anweisung ist (auch bei Subprozeduren) nicht erlaubt.

RETURN [(variablen-name)] :

Regeln:

- Rückkehr aus Funktions-Prozeduren
  Die RETURN-Anweisung muß einen Variablennamen enthalten.
  Der angegebenen Variablen wird in der Prozedur der Funktionswert zugewiesen.
  Beim Ablauf einer Funktions-Prozedur darf die END-Anweisung nicht erreicht werden.

- Rückkehr aus CALL-Prozeduren
  Die RETURN-Anweisung darf keinen Variablennamen enthalten.
  EINE RETURN-Anweisung unmittelbar vor der END-Anweisung darf weggelassen werden.

- Außerhalb eines Prozedur-Körpers darf keine RETURN-Anweisung stehen.

- Handhabung von Prozeduren siehe 6-2.

Beispiel:

```
RETURN;
RETURN (WERT);
```

4-3.7 ON-Anweisung

Mit der ON-Anweisung kann der Programmablauf innerhalb einer Task von einem externen oder internen Ereignis abhängig gemacht werden. Im Subset sind nur Standardereignisse zugelassen.

Syntax:

```
ON signal-name { goto-anweisung | SYSTEM };
```

Regeln:

- ON-Anweisungen sind nur auf äußerster Task- und Prozedur-Ebene erlaubt.
- Einplanungen der Signal-Reaktionen, die in Prozeduren vorgenommen werden, werden bei Verlassen derselben aufgehoben.
- Einplanungen der Signal-Reaktionen in Tasks werden erst beim Terminieren der Task aufgehoben (siehe Kapitel 5, TASKING).

4-4. ANWEISUNGEN FÜR DIE PARALLELE TASK-ABLAUFSTEUERUNG

Während in den vorhergehenden Anweisungen festgelegt werden kann, in welcher Reihenfolge Anweisungen in Tasks und Prozeduren ausgeführt werden sollen, kann mit diesen Anweisungen ausgedrückt werden, wann die in Tasks angegebenen Anweisungen ausgeführt werden sollen. In einem eigenen Kapitel 5, TASKING wird auf die für das Zusammenspiel der einzelnen Anweisungen wichtige Task-Koordinierung eingegangen.

Die nachfolgend in einem Zustandsdiagramm aufgeführten Task-Anweisungen werden im Kapitel 5, TASKING in Syntax und Semantik ausführlich beschrieben.

4-5. ANWEISUNGEN FÜR DIE INTERRUPTBEHANDLUNG

4-5.1 DISABLE-Anweisung

Mit einer DISABLE-Anweisung kann ein Interrupt gesperrt werden. Das bedeutet, daß keine WHEN-Bedingung erfüllt werden kann solange ein Interrupt gesperrt ist und somit alle mit dieser WHEN-Bedingung verknüpften Anweisungen zum Start bzw. Fortsetzen von Tasks (ACTIVATE-, CONTINUE- und RESUME-Anweisungen) nicht ausgeführt werden können.

Syntax:

```
DISABLE  interrupt-bezeichner;
```

Beispiel:

```
DISABLE INT4;
```

4-5.2 ENABLE-Anweisung

Durch eine ENABLE-Anweisung kann ein mittels einer DISABLE-Anweisung gesperrter Interrupt wieder erlaubt werden. Die diesem Interrupt aktuell zugeordneten Maßnahmen (siehe 2-5.1) werden wieder wirksam.

```
ENABLE  interrupt-bezeichner;
```

Beispiel:

```
ENABLE INT4;
```

4-5.3 TRIGGER-Anweisung

Mit der TRIGGER-Anweisung kann das Auftreten eines Interrupts simuliert werden. Sie ist ein Hilfsmittel zum Austesten von Automationsprogrammen.

```
TRIGGER  interrupt-bezeichner;
```

Beispiel:

```
TRIGGER INT4;
```

4-6. EINGABE/AUSGABE

Die E/A wird beschrieben durch Anweisungen zur Daten-Übertragung und durch Anweisungen zur Datei-Verwaltung.
Die Datenübertragung im PEARL-Subset geschieht auf dreierlei Weise, nämlich durch reihenweise formatierte Datenübertragung im externen Code (Zeichen) (GET/PUT-Anweisung), durch Element- bzw. Satzweise unformatierte Datenübertragung im internen Code (Binär) (MOVE-Anweisung) und durch Übertragung von graphischen Informationsinhalten (SEE/DRAW-Anweisung).
Die Datenübertragung ist immer gerichtet von der Endstelle der Daten-Quelle zur Endstelle der Daten-Senke. Daten-Quelle bzw. Daten-Senke können physikalische Endstellen (Platte, Analog-Digital-Wandler, etc.) oder logische Endstellen (Files, Programmgrößen wie Variable, Felder, etc.) sein.
Alle in Datenübertragungs- und Dateiverwaltungs-Anweisungen benutzten Namen müssen vor ihrer Verwendung deklariert werden. Die Namen für physikalische Endstellen werden im System-Teil eingeführt und müssen im Problem-Teil mit den Attributen VAL DEVICE, VAL SIGNAL, VAL INTERRUPT vereinbart werden. Namen für logische Endstellen werden nur im Problem-Teil mit den Attributen FIXED, FLOAT, BIT, CHARACTER, DURATION, CLOCK und VAL FILE vereinbart.

Beispiel für die Vereinbarung von Enstellen-Namen

```
MODULE TEST;
SYSTEM;
  .
  .
  .
LKEI <- KARTE:;    /* PHYSIKALISCHE ENDSTELLE */
  .
  .
  .
PROBLEM;
DECLARE KARTE VAL DEVICE  GLOBAL,
        TEXT CHARACTER (40),    /* LOGISCHE ENDSTELLEN */
        NUMMER FIXED,           /*       DITO          */
        DATEI VAL FILE;         /*       DITO          */
  .
  .
  .
GET KARTE EDIT (TEXT) (A(40));
PUT DATEI EDIT (NUMMER, TEXT) (F(2), A(40));
  .
  .
  .
MODEND;
```

4-6.1 File-, Geräte-Vereinbarungen

4-6.1.1 File-Deklaration

Ihre Deklaration hat die allgemeine Form:

DECLARE { name | (namenliste) } VAL FILE;

Beispiele:

```
DECLARE (F1, F2, F3) VAL FILE, F4 VAL FILE;
```

4-6.1.2 Geräte-Spezifikation

Geräte-Vereinbarungen erfolgen mit Hilfe von Spezifikationen auf Modulebene. Sie dienen dazu, die im Systemteil deklarierten Größen in den Problemteil einzuführen.
Ihre Spezifikation erfolgt durch die DECLARE-Vereinbarung (siehe 2-4.2 Interrupt, Signal):

DECLARE { name | (namenliste) } [([1:] g_1)] VAL DEVICE GLOBAL;

Beispiele:

```
DECLARE (G1,G2) VAL DEVICE GLOBAL;
DECLARE LKEI VAL DEVICE GLOBAL;
DECLARE (G3,G4) (1ØØ) VAL DEVICE GLOBAL;
```

4-6.2 Das File-Handling

4-6.2.1 Filehandling-Begriffe

Zum Verständnis der bei Filehandlingoperationen innerhalb des PEARL-Programmiersystems ablaufenden Vorgänge sei eine kurze Begriffserklärung gebracht.

'Dataset': Unstrukturierte Menge von Daten auf einem (Massenspeicher bzw.) Gerät (:Massendaten). Datasets werden durch Zeichenketten identifiziert.

'File': Bezug auf einen strukturierten Dataset (strukturierte Menge von Daten) mit bestimmten Eigenschaften bezüglich zugelassener Operationen. Files werden durch Namen identifiziert.

'Eröffnet' der Programmierer einen File über einem Dataset, so erhalten die unstrukturierten Daten eines Datasets

- eine Struktur
  (Aufteilung in Sätze bestimmter Länge)

- weitere Eigenschaften
  (Festlegung von Zugriffs-, Benutzungs- und Darstellungsart).

Dabei werden Files und Datasets derart miteinander verknüpft, daß jeglicher Datenverkehr (E/A-Operationen) zwischen Programmgrößen und Massendaten nur 'über' Files stattfindet (Fig. 1, 2). Das heißt, für einen Programmierer sind Files die externe Endstelle bei einem Datentransfer mit Geräten der Std.-E/A ('File-E/A-Operationen'). Bei Verknüpfung mit einem File erhalten die Daten eines Datasets eine Struktur: Sie werden in eine Folge von Sätzen variabel fester Länge unterteilt, die man sich in irgendeiner Reihenfolge durchnumeriert vorstellen kann. Für jeden File gibt es nun einen Zeiger, der auf einen dieser durchnumerierten Sätze zeigt ('Satzzeiger'). Er gibt z.B. an, mit welchem Satz bei einer (genauer: der nächsten) File-E/A-Operation begonnen werden soll. Der Stand des Zeigers kann vom Programmierer verändert werden.

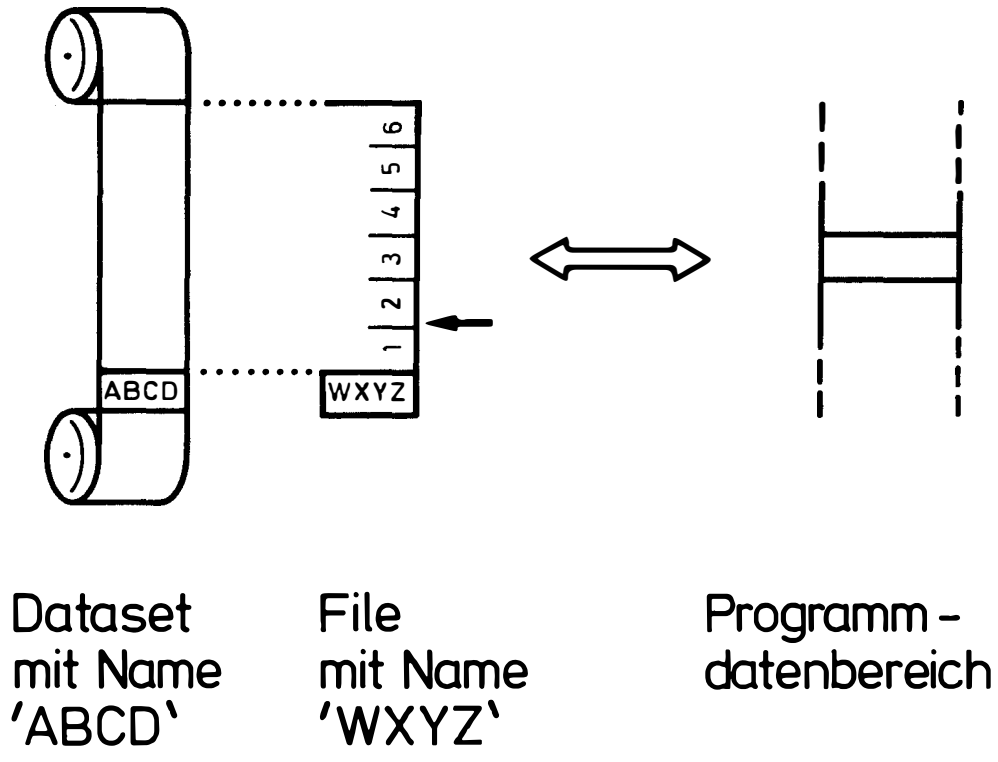


Fig. 1 Schematische Darstellung der durch Eröffnen eines Files über einem Dataset möglichen Verhältnisse: File 'WXYZ' veknüpft mit Dataset 'ABCD'; Daten in 6 Sätze unterteilt; Datenaustausch von und nach Programmdatenbereich erlaubt; der Satzzeiger zeigt auf den zweiten Satz.

Die Art der Verknüpfung zwischen Files und Datasets läßt es zu, daß über einem Dataset mehrere Files eröffnet werden, d.h., daß mehrere Files mit einem Dataset verknüpft sind. Das kann dann von Vorteil sein, wenn man 'gleichzeitig' mehrere Satzzeiger zur Verfügung haben will (, um z.B. Sätze an verschiedenen Stellen des Datasets zugleich bearbeiten zu können - vergl. Fig. 2). Andererseits ist es möglich, daß ein File nacheinander mit verschiedenen Datasets verknüpft wird.

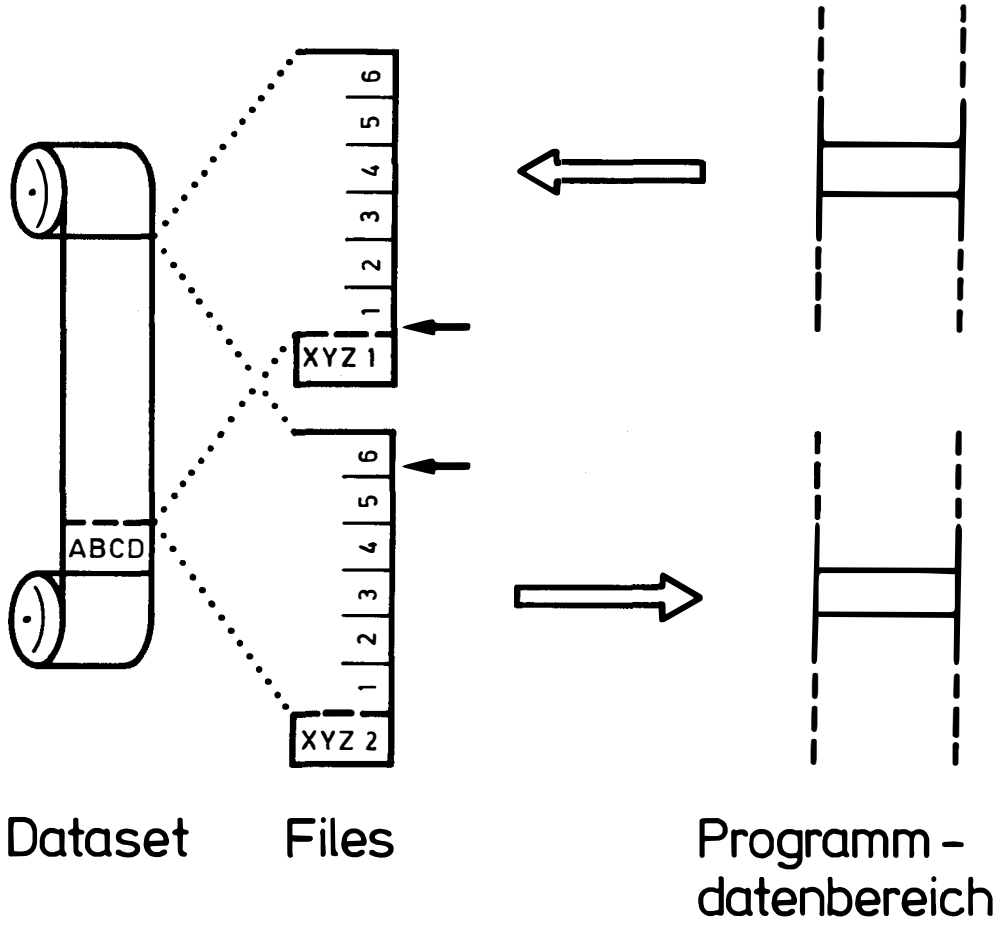


Fig. 2 Schematische Darstellung der durch Eröffnen mehrerer Files über einem Dataset möglichen Verhältnisse:

File 'XYZ1' verknüpft mit Dataset 'ABCD', Daten in 6 Sätze aufgeteilt; Datenaustausch vom Programmbereich zum File möglich; der Satzzeiger zeigt auf den 1. Satz.

File 'XYZ2' verknüpft mit Dataset 'ABCD', Daten in 6 Sätze unterteilt; Datenaustausch vom File zum Programmdatenbereich möglich; der Satzzeiger zeigt auf den 6. Satz.

4-6.2.2 Filehandling-Anweisungen

Anweisungen zur Dataset-Verwaltung existieren zum Einrichten und Löschen von Datasets sowie zum Eröffnen und Schließen von Files auf einem eingerichteten Dataset. Auch bei wiederholtem Eröffnen (nach vorhergehendem Schließen) müssen aus Sicherheitsgründen immer alle filespezifischen Informationen, mit Ausnahme der Längenbeschreibung, angegeben werden.

Es wird unterschieden zwischen den 'eigentlichen' Filehandling-Anweisungen (die die Verknüpfung zwischen Files und Datasets regeln und für Festlegen und Zurücknehmen von File-Eigenschaften verantwortlich sind) und den File-E/A-Anweisungen (die den Datenverkehr zwischen Files und Programmgrößen durchführen - s. Kapitel 4-6.2.4).

4-6.2.2.1 Anweisungen zum Eröffnen von Files (und ggfls. Einrichten von Datasets)

Syntax:

```
CREATE  }
OPEN    }  file-name [ TITLE dataset-name ] UPON device-name

          benutzungsart [ (satzanzahl) ] zugriffsart

        { (einheitenanzahl) darstellungsart /
          darstellungsart[([1:] einheitenanzahl)] data-attribut };
```

filename: Name des Files, auf dem in E/A-Anweisungen Bezug genommen werden soll, muß als solcher deklariert sein und daher den Einschränkungen für Bezeichner genügen.

datasetname: Kennzeichnung des Datasets, mit dem der File verknüpft werden soll, in Form eines Characterstrings aus max. 4 Zeichen oder in Form einer entsprechenden Variablen.
Fehlt die TITLE-Option, so wird dem Dataset eine implementationsabhängige Kennzeichnung vergeben.

devicename: Name des Gerätes, auf welchem der Dataset untergebracht ist bzw. werden soll:

z.B. auf der Zielmaschine in Erlangen

PSEA	für Plattenspeicher	
MBEA	für Magnetbandspeicher	noch nicht realisiert

Er muß im Systemteil und Problemteil deklariert bzw. spezifiziert sein.

benutzungsart:

INPUT: wenn Datenaustausch nur in den Programmdatenbereich erlaubt sein soll

OUTPUT: wenn Datenaustausch nur vom Programmdatenbereich erlaubt sein soll

UPDATE: wenn Datenaustausch zwischen Programmdaten und File in beiden Richtungen erlaubt sein soll.

satzanzahl: Die Anzahl der Sätze, in die der Dataset unterteilt werden soll; ist keine satzanzahl angegeben, wird mit einem Dataset ohne bestimmte Länge gearbeitet. Die Länge des Datasets ergibt sich dann durch seine augenblickliche Länge bei einer CLOSE-Anweisung.

zugriffsart:

SEQ: bei einer File-E/A-Anweisung kann eine Relativ-'positionierung' vorgenommen werden

DIR: bei einer File-E/A-Anweisung kann eine Relativ- oder Absolut-'positionierung' vorgenommen werden.

einheitenanzahl: Anzahl der Einheiten, aus denen sich ein Satz zusammensetz.

einheitengröße: Größe einer Einheit eines Satzes, wird angegeben als das Attribut einer Programmdatengröße, die in dieser Einheit Platz finden soll. Ist die Programmdatengröße ein Feld, darf maximal ein eindimensionales Feld angegeben werden.

einheitenanzahl und einheitengröße: ergeben zusammen die Länge des Satzes.

darstellungsart: **ALPHA**: Programmdatengröße soll auf dem File formatiert (in Externdarstellung) untergebracht sein

BASIC: Programmdatengröße soll auf dem File unformatiert (in Interndarstellung) untergebracht sein.

data-attribut: **FIXED**: Interne Programmdatengröße als Binary-Fixed-Zahl (Integer)

FLOAT: Interne Programmdatengröße als Binary-Float-Zahl (Real).

4-6.2.2.1.1 Anweisungen zum Eröffnen von Files - Funktionsweise

Die CREATE-Anweisung erfüllt einen doppelten Zweck. Durch sie soll

- ein Dataset neu errichtet werden
- ein File mit gewünschten Eigenschaften versehen und mit dem angegebenen Dataset verknüpft werden.

'Eröffnung' eines Files

Durch die OPEN-Anweisung soll

- ein File mit gewünschten Eigenschaften versehen und mit dem angegebenen Dataset verknüpft werden.

'Eröffnung' eines Files

CREATE- und OPEN-Anweisung unterscheiden sich also nur dadurch, daß bei der OPEN-Anweisung ein File mit einem bereits verbundenen Dataset verknüpft wird, während bei der CREATE-Anweisung ein Dataset eigens neu eingerichtet wird. Da bei der Verküpfung eines Files mit einem vorhandenen Dataset dieser bereits eine feste Länge hat, kann eine Festlegung der Dataset-Länge nur bei

der CREATE-Anweisung durchgeführt werden. Eine solche Längenfestlegung erfolgt durch die Angabe einer 'satzanzahl'. Entfällt sie, bleibt die Länge zunächst unbestimmt (und kann zu einem späteren Zeitpunkt durch eine CLOSE-Anweisung festgelegt werden).

CREATE- und OPEN-Anweisung bewirken gleichermaßen, daß der Satzzeiger auf den 1. Satz des Files gestellt wird. Die Eröffnung eines Files durch eine CREATE- bzw. OPEN-Anweisung ist außerdem die Bedingung dafür, daß mit E/A-Anweisungen auf ein File zugegriffen werden kann.

4-6.2.2.1.2 Anweisungen zum Eröffnen von Files - Beispiele

Ein Dataset besitzt einen bestimmten logischen Aufbau (Struktur), der durch die Parameter in der CREATE-Anweisung festgelegt wird. Das File repräsentiert ein logisches Raster, das über den Dataset gelegt wird und das mit der Struktur dieses Datasets verträglich sein muß. Das heißt, die Parameter von Dataset und File können verschiedene Werte haben, wobei die Werte der File-Parameter in den Werten der Dataset-Parameter enthalten sein müssen.

```
CREATE  FILE1 TITLE 'DSO7' UPON PSEA
        UPDATE (50) DIR (100) ALPHA;

CREATE  FILE2 TITLE KENVAR UPON MBEA
        UPDATE (90) SEQ BASIC (40) FLOAT;
```

Einrichten eines Datasets auf dem Medium Platte bzw. Band bestehend aus 50 bzw. 90 Datensätzen, wobei in einem Fall der Satz 100 Alphazeichen, im anderen Fall 40 (interne) Float-Variable umfassen soll. Im ersten Fall handelt es sich um einen Dataset, auf den im Update-Mode direkt zugegriffen werden soll, im zweiten Fall um einen Dataset, der sequentiell beschrieben oder gelesen werden kann und zwar mit Platz (d.h. Raum) für 90 Sätze. Läßt man die '(90)' weg, dann ist die Anzahl der Sätze auf diesem Dataset unbegrenzt.

```
OPEN  FILE1 TITLE 'DSO7' UPON PSEA
      UPDATE (2) DIR (100) ALPHA;
```

Eröffnen eines Files für den oben eingerichteten Dataset, der unter der Kennung 'DSO7' auf der Platte steht. Die übrigen Parameter müssen mit denen des Datasets (siehe CREATE-Anweisung) verträglich sein, was im Beispiel durch Identität sichergestellt ist. Verträglichkeit ist aber noch gegeben, wenn UPDATE durch OUTPUT bzw. INPUT und DIR durch SEQ ersetzt würde. Die Satzanzahl für das File darf höchstens kleiner sein als die des Datasets. Das File muß natürlich den Gerätetyp ansprechen, für den der Dataset eingerichtet worden ist.

```
OPEN  FILE2 UPON MBEA INPUT SEQ BASIC (20) FLOAT;
```

Eröffnen eines Files für den oben eingerichteten Dataset. Die Kennzeichnung des Datasets wird nicht geprüft, deswegen entfällt die Title-Option. (Bei Magnetbändern befindet sich normalerweise nur ein Dataset auf einer Bandspule, bei Magnetplatten sind auf einem Plattenstapel normalerweise mehrere Datasets untergebracht. In diesem Falle ist die Dataset-Kennzeichnung unbedingt erforderlich, um den richtigen Dataset auf dem Plattenstapel finden zu können.)

Zu dem Dataset-Parameter UPDATE ist der File-Parameter INPUT verträglich; OUTPUT wäre auch verträglich. Der File-Parameter SEQ ist verträglich zu dem Dataset-Parameter SEQ (und DIR). Eine Satzanzahl ist nicht angegeben, dadurch wird solange gelesen, bis das Ende des Datasets erreicht ist. Bei einem File-Parameter OUTPUT kommt es bei fehlender Angabe Satzanzahl natürlich zu einem Überlauf, wenn der Dataset "voll" geschrieben ist. Die Systemmaßnahmen im Falle von Überlauf sind implementationsabhängig, wobei die Möglichkeiten Endfile-Signal, Fehler-Signal oder Ignore denkbar sind. Die Datenrasterung des Files von 20 Float-Variablen pro Satz ist durchaus verträglich mit der Rasterung von 40 Float-Variablen pro Satz auf dem Dataset. Ein Dataset-Satz entspricht so zwei File-Sätzen. Diese verschiedene Rasterung ist unbedenklich bei sequentiellen Datasets, im Falle von Direct-Datasets würden Fehler in der Satzadressierung die Folge sein.

Im Rahmen einer Überlagerungstechnik ist eine Verträglichkeit zwischen den Parametern FLOAT und FIXED denkbar, setzt aber voraus, daß die rechnerinterne Speicherrasterung für Integer- und Real-Variable kompatibel ist. Eine Verträglichkeit zwischen den Parametern ALPHA und BASIC ist wegen der damit verbundenen vielfältigen Störungsmöglichkeiten nicht statthaft.

4-6.2.2.2 Anweisungen zum Schließen von Files (und ggfls. Löschen von Datasets)

Syntax:

```
CLOSE  }
DELETE }  file-name;
```

file-name: Name des Files auf das die Anweisung wirkt, muß als solcher deklariert sein und daher den Einschränkungen für Bezeichner genügen.

4-6.2.2.2.1 Anweisungen zum Schließen von Files - Funktionsweise

Die CLOSE-Anweisung macht die in einer CREATE- bzw. OPEN-Anweisung erfolgte Eröffnung rückgängig, d.h.

- die Eigenschaften des Files werden gelöscht
- die Verknüpfung des Files mit einem Dataset wird gelöscht.

Wirkt die CLOSE-Anweisung auf ein File, das mit einem Dataset unbestimmter Länge verknüpft war, dann wird hierbei zusätzlich noch die Länge des betreffenden Datasets festgelegt. Sie ergibt sich aus dem augenblicklichen Stand des Satzzeigers.

Die DELETE-Anweisung hat wieder eine mehrfache Wirkung. Es wird

- die Verknüpfung des Files mit dem Dataset gelöst und die Eigenschaften des Files gelöscht
- der mit dem File verknüpfte Dataset gelöscht.

CLOSE- und DELETE-Anweisung haben gleichermaßen zur Folge, daß auf das betreffende File mit E/A-Anweisungen nicht mehr zugegriffen werden kann.

4-6.2.2.2.2 Anweisungen zum Schließen von Files - Beispiele

```
DELETE FILE1;
```

Das File wird abgeschlossen und der zugehörige Dataset, der durch obiges CREATE eingerichtet worden ist, wird gelöscht, d.h. er ist nach Ausführung der Anweisung nicht mehr vorhanden.

```
CLOSE FILE2;
```

Das File wird abgeschlossen und für weitere Zugriffe durch E/A-Anweisungen gesperrt.
Ein DELETE statt CLOSE würde einen erneuten Zugriff (durch OPEN) auf diesem Dataset unmöglich machen.

4-6.2.3 Regeln zur Verwendung von Filehandling-Anweisungen

- Files und Datasets können sich in verschiedenen Zuständen befinden:

  Files können eröffnet sein oder nicht.
  Datasets können auf einem bestimmten Gerät vorhanden sein oder nicht. Sind sie vorhanden, können sie eine feste oder eine unbestimmte Länge haben.

- Zwischen einigen Zuständen sind Übergänge möglich. Erlaubte Übergänge werden durch Filehandling-Anweisungen ausgelöst. Sind Übergänge verboten, werden entsprechende Fehler-'Signals' ausgelöst.

- Vor dem ersten Ablauf eines Programms ist über den Zustand der Datasets nichts bekannt (Dataset existiert "vielleicht"). (Eine Kopplung zwischen File und Dataset findet nur durch CREATE/OPEN statt.) Das File/Datasetsystem befindet sich dann in einer Art Initialisierungszustand. In ihn wird es beispielsweise auch durch CLOSE/DELETE zurückversetzt.

- Beim Einrichten eines Datasets darf der zugeordnete File-name nicht schon für einen anderen existierenden Dataset verwendet sein.

- Für einen noch nicht eingerichteten Dataset kann kein File eröffnet werden.

- Mit dem Einrichten eines Datasets erfolgt gleichzeitig das Eröffnen eines Files für diesen Dataset.

- Das Löschen eines Datasets schließt das Schließen eines zugeordneten Files ein.

- Files dürfen nicht entgegen ihrer Zugriffsart benutzt werden.

- Files dürfen nur innerhalb des durch die zugehörige File-Vereinbarung gegebenen Gültigkeitsbereichs angesprochen werden.

- Es können mehrere Files gleichzeitig dem gleichen Dataset - durch Eröffnungs-Anweisungen - zugeordnet werden.

- Ein bereits eröffnetes File muß vor jeder weiteren Eröffnung geschlossen werden.

- Eine CLOSE-Anweisung auf ein bereits geschlossenes File ist unzulässig.

- Soll mit einer DELETE-Anweisung ein File geschlossen und der zugeordnete Dataset gelöscht werden, so wird das Löschen nur dann wirksam, wenn keine weiteren Files zu diesem Dataset existieren.

4-6.2.4 File-E/A-Anweisungen

E/A-Anweisungen auf Files sind in Analogie zu E/A-Anweisungen auf Geräte zu sehen. Es genügt hier, auf die Unterschiede zur Geräte-E/A einzugehen.

4-6.2.4.1 File-E/A-Anweisungen - Begriffliche Unterschiede zur Geräte-E/A

Wie bei E/A-Anweisungen auf Geräte gibt es bei File-E/A-Anweisungen die Aufteilung zwischen zeichenweiser, formatierter Übertragung (GET/PUT-Anweisung) und binärer, unformatierter Übertragung (MOVE-Anweisung).

Unterschiedlich zu Geräte-E/A-Anweisungen ist:

- Als externe Endstelle ist keine 'device-name' anzugeben, sondern ein 'file-name'. Das bedeutet jedoch nicht, daß auf ein- und dasselbe Gerät entweder mit einer Geräte-E/A-Anweisung oder - nach Einrichten eines Files über diesem Gerät - über eine File-E/A-Anweisung zugegriffen werden kann.

- Die externe Endstelle kann durch eine 'positionierung' genauer bestimmt werden.

  Bemerkung: Positionierung ist nur möglich bei Files für Magnetbandgeräte oder Magnetplattengeräte (sog. magnetische Massenspeicher).

- Die MOVE-Anweisung auf Files darf nicht durch gerätespezifische 'options' modifiziert werden.

Diese Unterschiede schlagen sich natürlich in Schreibweise, Funktionsweise (und Implementierung) der File-E/A-Anweisungen nieder.

4-6.2.4.2 File-E/A-Anweisungen - Schreibweise

formatierte File-E/A:

Eingabe: Übertragung von File nach Programmdatenbereich

```
GET file-name [positionierung] EDIT ( programmdatenliste ) ...
                                  ... { R (markenname)  } ;
                                      { (formatliste)   }
```

Ausgabe: Übertragung von Programmdatenbereich nach File

```
PUT file-name [positionierung] EDIT ( programmdatenliste ) ...
                                  ... { R (markenname)  } ;
                                      { (formatliste)   }
```

unformatierte File-E/A:

Eingabe:
```
MOVE file-name [ positionierung] TO  (programmdatenliste);
```

Ausgabe:
```
MOVE (programmdatenliste) TO file-name [positionierung] ;
```

wobei:

file-name: wie bei Filehandling-Anweisungen

positionierung: { ADV ({ + / - } { integer-Konstante / integer-Variable }) / POS ({ integer-Konstante / integer-Variable }) }

programmdatenliste / formatliste / R (markenname) : wie bei Geräte-E/A

4-6.2.4.3 File-E/A-Anweisungen - Funktionsweise

File-E/A-Anweisungen durchlaufen nacheinander folgende Schritte:

- Zulässigkeitsprüfungen
- [Satz-Zugriff]
- [Formatierung]
- Übertragung

Bei allen diesen Schritten können zur Laufzeit Fehler auftreten, die dem Programmierer durch Signals gemeldet werden.
Diese Signals spezifizieren die Art des Fehlers.

4-6.2.4.3.1 File-E/A-Anweisungen - Zulässigkeitsprüfungen

Bei allen File-E/A-Anweisungen werden ihre verschiedenen Modi auf Verträglichkeit mit den bei der File-Eröffnung (durch CREATE/ OPEN) angegebenen Eigenschaften geprüft. Bei Unverträglichkeit werden entsprechende Signals erzeugt und die Anweisungen abgebrochen.
Insbesondere ist eine File-E/A-Anweisung nur erlaubt, wenn das entsprechende File überhaupt eröffnet ist.
Ist ein File als SEQUENTIAL eingeführt, darf der Satz-Zugriff nicht durch eine absolute 'positionierung' erfolgen.
Bei Files (wie auch bei Geräten) als externe Endstelle setzen GET/PUT-Anweisungen eine externe (d.h. alphanumerische), MOVE-Anweisungen dagegen eine interne (d.h. binäre) Darstellung der Daten auf dem Dataset voraus. Infolgedessen sind GET/PUT-Anweisungen nur auf Files des Typs ALPH, MOVE-Anweisungen nur auf Files des Typs BASIC zugelassen. Verstöße werden durch Fehler-Signals angezeigt.

GET-Anweisungen und MOVE-Anweisungen, deren Quelle ein File ist, bedeuten Eingabe-Anweisungen und sind folglich nicht auf Files des Typs OUTPUT erlaubt (sonst Signal). PUT-Anweisungen und MOVE-Anweisungen, deren Senke ein File ist, bedeuten Ausgabe-Anweisungen und sind nicht auf Files des Typs INPUT erlaubt (sonst Signal). Auf Files des Typs UPDATE sind dagegen Ausgabe- und Eingabe-Anweisungen erlaubt.

4-6.2.4.3.2 File-E/A-Anweisungen - Satz-Zugriff

Nach den Zulässigkeitsprüfungen wird ggfls. der richtige Zugriff auf den zur Übertragung vorgesehenen Satz des Files sichergestellt. Jede Übertragung beginnt mit einem neuen logischen Satz. Ohne 'positionierung' beginnt jede File-E/A-Anweisung mit dem auf den letzten Satz der vorangegangenen Anweisung folgenden Satz. Insbesondere beginnt die erste File-E/A-Anweisung nach dem Eröffnen eines Files (durch CREATE/OPEN) mit dem ersten Satz dieses Files.
Mit 'positionierung' kann diese Regel je nach gewählter Zugriffsart durchbrochen werden.

Wurde die Zugriffsart als SEQUENTIAL angegeben, kann vor der E/A-Anweisung durch ADV (Satznummer) eine relative 'positionierung' vorgenommen werden. Sie bewirkt, daß relativ zu dem Satz, mit dem die Anweisung sonst (s.o.) zu beginnen hätte, die angegebene Anzahl von Sätzen übersprungen wird. Hier wird also der Stand des Satzzeigers als Ausgangswert genommen.

Wurde die Zugriffsart als DIRECT angegeben, kann vor der E/A-Anweisung darüber hinaus durch POS (Satznummer) eine absolute 'positionierung' erfolgen. Durch die angegebene Satznummer wird der Satz, bei dem die Operation beginnen soll, explizit angewählt.

Nach vollzogenem Zugriff wird der Satzzeiger auf den mittlerweile zur Verarbeitung anstehenden Satz nachgestellt. Beim Satz-Zugriff sind verschiedene Fehlerfälle möglich:
Überschreitet die Satznummer innerhalb der 'positionierung' eine sinnvolle Größe (angezeigt durch Signal), wird die 'positionierung' gar nicht erst vorgenommen. Werden durch den Satz-Zugriff die Grenzen des Datasets erreicht (angezeigt durch Signal), wird die 'positionierung' ebenfalls nicht wirksam. Kommt es schließlich zu einem Gerätefehler oder werden die Grenzen des freien Externspeichers erreicht, wird ebenfalls ein Signal erzeugt. In allen diesen Fällen bleibt der Satzzeiger auf dem Stand vor der File-E/A-Anweisung.

4-6.2.4.3.3 File-E/A-Anweisungen - Formatierung

Während MOVE-Anweisungen die Daten in unveränderter (binärer) Form zwischen Programmdatenbereich und File austauschen, bedingen GET/PUT-Anweisungen eine zeichenweise (alphanumerische) Darstellung der Daten auf dem externen Medium. In diesem Fall erfolgt demnach vor der eigentlichen Ausgabeoperation (bzw. nach der Eingabeoperation) eine Umformatierung der in der 'programmdatenliste' stehenden Daten. Diese Umwandlung entspricht derjenigen der Geräte-E/A, insbesondere sind die gleichen Signals möglich.

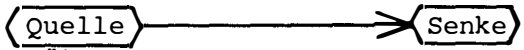
4-6.2.4.3.4 File-E/A-Anweisungen - Übertragung

Entsprechend der formatierten Geräte-E/A werden die angegebenen Daten nicht direkt zwischen dem Programmdatenbereich und dem externen Medium ausgetauscht, sondern in einem Puffer zwischengespeichert. Der Puffer hat im allgemeinen die Länge eines physikalischen Satzes des entsprechenden Gerätes.
Werden beim Datenaustausch zwischen Programmdatenbereich und Puffer dessen Grenzen überschritten, wird der Zwischenpuffer entleert und erneut gefüllt. Werden die Grenzen des (logischen) Satzes überschritten, wird wie bei der Geräte-E/A ein Signal erzeugt.
Nach der Übertragung wird der Satzzeiger um die Anzahl der übertragenen Sätze nachgeführt. Werden während der Übertragung die Grenzen des Datasets überschritten, wird die Übertragung abgebrochen, der Satzzeiger auf den letzten vollständig übertragenen Satz gestellt und ein Signal erzeugt. Werden bei der Eingabe die Grenzen des freien Arbeitsspeicherbereichs, bei der Ausgabe die Grenzen des freien Externspeicherbereichs überschritten oder treten Übertragungs- oder Gerätefehler auf, muß mit entsprechenden Signals gerechnet werden.

4-6.3 Reihenweise formatgesteuerte Daten-Übertragung

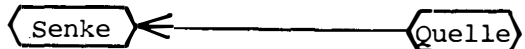
Bei der reihenweisen Datenübertragung wird die Datenmenge als eine fortlaufende Reihe von Zeichen angesehen. Die GET-Anweisung dient dazu, Daten von der Datenmenge (Quelle) zur Programmgröße (Senke) zu übertragen, die PUT-Anweisung dagegen von Programmgrößen (Quelle) zur Datenmenge (Senke). Wegen der unterschiedlichen Datenflußrichtung spricht man im ersten Übertragungsfalle von Eingabe, im zweiten Falle von Ausgabe.

Syntax der formatgebundenen Eingabe:




```
GET {geräte-name  }
    {file-name [position]} EDIT ([datenliste])
    {(formatliste) | (R(marken-name))};
```

Syntax der formatgebundenen Ausgabe:




```
PUT {geräte-name  }
    {file-name [position]} EDIT ([datenliste])
    {(format-liste) | (R(marken-name))};
```

Regeln:

- Mit dem Geräte-Namen werden nur Standard-E/A-Geräte als Quelle bzw. Senke zugelassen. Die Auswahl und die Namen der Geräte sind zielmaschinenabhängig verschieden.

- Der File-Name bezieht sich auf ein File, dessen Zuordnung zu einer Datenmenge (Datei) über eine Eröffnung (siehe Kapitel 4, 4-6.2.2.1) vorgenommen wird.

- Mit Hilfe einer Positionierangabe kann ein bestimmter Satz ausgewählt werden, ab dem die Datenübertragung beginnen soll.

  Syntax: position::= ADV ({[+|-] zahl|variable})|
  POS (zahl|variable)

  - Die Angabe hinter ADV stellt die Anzahl der Sätze dar, um die die gewünschte von der aktuellen Position entfernt ist (relative Positionierung). Positive Angaben bewirken ein vorwärts Positionieren, negative Angaben bewirken ein rückwärts Positionieren.
  - Bei dem mit DIR gekennzeichneten File sind auch absolute Positionierungen mittels POS (zahl/variable) möglich. Die Zahl stellt hierbei unmittelbar die Nummer des Satzes dar, ab dem die Datenübertragung beginnen soll.

- Die Datenlisten für die GET- und PUT-Anweisungen sind unterschiedlich und können auch ganz ausgelassen werden.

  o Bei GET-Anweisung erlaubt durch Kommas getrennte Aufeinanderfolge von:

    Variablen
    Felder
    Feldelementen
    } ⟨Senken⟩

    Beispiel:

    ```
    GET FILE1 EDIT (NR, FELD, FIELD(I), SOLL) ((4)A(7));
    ```

  o Bei PUT-Anweisung erlaubt durch Kommas getrennte Aufeinanderfolge von:

    Ausdrücken
    Felder
    Konstanten-Felder
    } ⟨Quellen⟩

    Beispiel:

    ```
    PUT FILE2 EDIT (-5, A(I)+1, 'TEXT') ((2)F(2), A(4));
    ```

- Die Formatliste kann unmittelbar nach der Datenliste angegeben werden. Es ist aber auch möglich durch Angabe eines Marken-Namens auf die mit dieser Marke versehene Format-Anweisung hinzuweisen (siehe dieses Kapitel, 4-6.3.2).

4-6.3.1 Formatliste

Syntax:

```
formatliste:: [(wiederholungsfaktor)] formatelement
              [{,[(wiederholungsfaktor)] formatelement} ... ]
                   wiederholungsfaktor (formatliste)
```

Allgemeine Regeln:

- Jedem Element der Datenliste muß ein Element der aufgelösten Formatliste zugeordnet sein. Die Zuordnung erfolgt von links nach rechts, wobei Steuerungs-Formatelemente (siehe 4-6.3.3.2) unberücksichtigt bleiben.

- Vor einem Formatelement darf ein positiver Wiederholungsfaktor (ganze Zahl > Ø) angegeben werden.

4-6.3.2 FORMAT-ANWEISUNG

Syntax:

marken-name: FORMAT (formatliste);

Regeln:

- Der Marken-Name vor dem Schlüsselwort FORMAT stellt die Verbindung her zwischen der E/A-Anweisung (R(marken-name)) und der ausgelagerten Formatliste in der Format-Anweisung.

- Für den Gültigkeitsbereich dieses Format-Marken-Namens gelten die gleichen Regeln wie in Kapitel 2, 2-4.1 und 2-4.3 aufgeführt.

4-6.3.3 Formatelemente

Es gibt zwei Arten von Formatelementen:

Datenformatelemente (A, B1, B3, E, F, D, T) und
Steuerungsformatelemente (X, SKIP, PAGE).
Ein Formatelement kann auch wieder eine Formatliste sein.

Syntax:

```
formatelement:: = daten-formatelement|
                  steuerungs-formatelement|
                  (formatliste)
```

4-6.3.3.1 Daten-Formatelemente

Zeichenketten-Formatelement A

Mit dem Formatelement A wird die externe Darstellung einer Kette von Zeichen beschrieben.

Syntax: A(w) w = Feldlänge

Regeln:

- Die externe Darstellung der Zeichenkette wird durch die Feldlänge w festgelegt.
- Die interne Darstellung durch:
  - o die Vereinbarung (z.B. DECLARE KETTE CHARACTER (10);)
  - o die Konstante selbst (z.B. 'REIHE')
  - o das Ausdruckergebnis (z.B. KETTE // 'REIHE')
- Die Daten werden immer linksbündig übertragen. Wenn externe und interne Darstellung voneinander abweichen, erfolgt nach folgender Tabelle ein rechtsseitiges Auffüllen mit Leerzeichen bzw. Abschneiden.

Anweisung \ Längen	extern > intern	extern < intern
GET	abschneiden	auffüllen
PUT	auffüllen	abschneiden

Bitketten-Formatelement B1

Mit dem Formatelement B1 wird die externe Darstellung einer Kette bestehend aus den Binär-Ziffern Ø oder 1 beschrieben.

Syntax: B1(w)

Regeln:

Es gelten die unter dem Formatelement A angegebenen Regeln sinngemäß, das Auffüllen bei GET erfolgt aber mit Nullen.

Oktalketten-Formatelement B3

Mit dem Formatelement B3 wird die externe Darstellung einer Kette bestehend aus den Oktal-Ziffern Ø bis 7 beschrieben.

Syntax: B3(w)

Regeln:

Es gelten die unter dem Formatelement A angegebenen Regeln sinngemaß, das Auffüllen bei GET erfolgt aber mit Nullen.

Festkomma-Formatelement F

Mit dem Formatelement F wird die externe Darstellung einer FLOAT- oder FIXED-Größe mit Vorzeichen beschrieben.

Syntax: F(w[, d]) d = Dezimalstellen nach dem Komma

Mit F(w) wird die externe Darstellung der FIXED-Größe, mit F(w, d) die einer FLOAT-Größe beschrieben.

Regeln:

- Eingabe (GET): Das Datenelement kann irgendwo innerhalb des Feldes mit der angegebenen Feldlänge w stehen. Fehlt der Dezimalpunkt, so wird er nach der d-ten Stelle von rechts im Eingabefeld angenommen.
- Ausgabe (PUT): Die Zahl wird rechtsbündig in dem Datenfeld mit der angegebenen Länge w abgesetzt. Führende Nullen werden durch Zwischenräume ersetzt.

Gleitpunkt-Formatelement E

Mit dem Formatelement E wird die externe Darstellung einer Gleitpunkt-Zahl beschrieben. Die interne Darstellung geschieht durch eine Vereinbarung mit dem Attribut FLOAT.

Syntax: E(w[, d])

Regeln:

Eingabe (GET):

- Das Datenelement kann in nachfolgender Form irgendwo innerhalb des Feldes mit der angegebenen Feldlänge w stehen (siehe auch Kapitel 2. 2-3.1.1.2 Rationale Zahlen).

  [+ | -] festkommazahl [E [+|-] exponent]

  - o Der Exponent ist eine ganze Zahl mit n Ziffern (n ist zielmaschinenabhängig verschieden). Ist er mit dem vorgestellten E nicht vorhanden, so wird ein Exponent Null angenommen.
  - o Enthält die Festkommazahl keinen Dezimalpunkt, so wird ein Dezimalpunkt nach der d-ten Stelle von rechts der Festkommazahl angenommen.

Ausgabe (PUT):

- Bei der Ausgabe wird das Datenelement in folgender Form dargestellt:

  [-] · d-Ziffern E { +|- } exponent

  - o Der Exponent ist eine ganze Zahl mit n Ziffern (n ist zielmaschinenabhängig verschieden).
  - o Für die Berechnung der Feldweite besteht folgende Ungleichung $w \geq d + n + 3$
  - o Die Gleitpunktzahl wird rechtsbündig in normalisierter Form ausgegeben. Die evtl. freibleibenden Stellen werden mit Leerzeichen aufgefüllt.

Dauer-Formatelement D

Mit dem Formatelement D wird die externe Darstellung einer Dauer-Größe beschrieben. Die interne Darstellung ist implementationsabhängig und entweder in vollen Sekunden oder Zehntel-Sekunden.

Syntax: D(w[, d])

Regeln:

- w bezeichnet die Gesamtfeldlänge, d die Dezimalstellen nach dem Komma bei der Sekundenangabe.
  d ist implementationsabhängig Ø oder 1.
- Die Zahlen für die Stunden-, Minuten- und volle Sekundenangabe dürfen nur zweistellig sein.
- Die Dauerkonstante wird bei der Ausgabe rechtsbündig abgesetzt.

Beispiel: Drucken im Format D (24, 1):

```
PUT SD EDIT(ZEITDAUER)(D(24,1));
```

Ausgabe: ␣␣␣9␣HRS␣12␣MIN␣46.7␣SEC

Uhrzeit-Formatelement T

Mit dem Formatelement T wird die externe Darstellung einer Uhrzeit-Größe beschrieben.

Syntax: T(w) w = Gesamtfeldlänge

Regeln:

- Die Zahlen für die Stunden-, Minuten- und Sekundenangaben dürfen nur zweistellig sein.

- Die Uhrzeitkonstante wird rechtsbündig abgesetzt.

Beispiel: Drucken im Format T(10):

```
PUT SD EDIT(UHRZEIT)(T(10));
```

Ausgabe: ␣␣␣9:46:46

4-6.3.3.2 Steuerungs-Formatelemente X, SKIP, PAGE

Das Steuerungs-Formatelement X bewirkt das Überlesen eines Zeichens bzw. die Ausgabe eines Leerzeichens unabhängig davon, was im betreffenden File als Einheit vereinbart ist; falls nötig, wird zur nächsten größeren Einheit (Informations-Einheit → Satz-Einheit) übergegangen.

Beim Steuerungs-Formatelement SKIP wird ein neuer Satz bzw. eine neue Zeile begonnen.
Das Steuerungs-Formatelement PAGE ist im Subset nur für Schnelldrucker-Ausgabe zugelassen, und bewirkt einen Blattvorschub.

4-6.4 Element- und Satzweise unformatierte Datenübertragung (MOVE-Anweisung)

Die nichtformatierte Datenübertragung ist die elementarste Übertragungsform in PEARL. Die Übertragung zwischen den verschiedenen Endstellen im Programm (Geräte, Files, Variable, Felder, etc.) geschieht im internen Code (Binär). Durch die Angabe einer Option (OPT) ist es möglich, während der Datenübertragung eine Transformation auf den Datenstring auszuführen (z.B. Eich- oder Kontrollprozeduren) oder Steuerinformation an Geräte weiterzuleiten. Die Art und Verwendung der Option ist zielmaschinenabhängig verschieden.

Die Beschreibung der MOVE-Anweisung ist in drei Teile untergliedert:

- Beschreibung der datenlosen gerichteten MOVE-Steueranweisung mit obligatorischer Option-Liste (nur für CAMAC-E/A)
- Beschreibung der elementweisen Datenübertragung zwischen Geräteendstellen und sonstigen Programmgrößen (ausgenommen Files) mit Option-Liste
- Beschreibung der satzweisen Datenübertragung zwischen Files und Programmgrößen ohne Option-Liste.

4-6.4.1 Allgemeine Syntax der MOVE-Anweisung

Syntax:

```
MOVE [quelle] TO senke [OPT (option-liste)];
```

wobei die Option-Liste aus einer beliebigen Folge von durch Kommas getrennten Options-Aufrufen besteht.

```
option-liste:: = option-call[{, option-call} ...]
option-call::  = option-name [(o-param[{, o-param}]...)]
o-param::      = ganze-zahl|'binärziffer...'B1
```

4-6.4.2 Beschreibung der datenlosen gerichteten MOVE-Steueranweisung

Syntax:

```
MOVE TO{geräte-name                    } OPT (option-list);
      {geräte-feld [(index)]           }
```

Regeln:

- Da es sich hierbei um eine bestimmte Steueranweisung für die CAMAC-Peripherie handelt, deren Auswahl durch die Option-Liste vorgenommen wird und deren Empfänger die mit dem Geräte-Namen bezeichnete Endstelle ist, ist kein Datentransfer vorgesehen.
- Die explizite Anweisung von Geräten als MOVE-Endstellen ist zielmaschinenabhängig verschieden.
- Die Index-Angabe für das Gerätefeld kann sowohl durch eine Ganze-Zahl, als auch durch eine Variablen-Angabe erfolgen.

Beispiel: MOVE TO CREATE5 OPT (DISABLE);

4-6.4.3 Beschreibung der elementweisen Datenübertragung

Die elementweise Datenübertragung geschieht zwischen den im System-Teil ausgewiesenen und im Problem-Teil durch Vereinbarung eingeführten Geräte-Endstellen und sonstigen im Programm deklarierten Endstellen mit Ausnahme von Files.

Syntax für die Eingabe:

```
      geräte-name                       variable
MOVE {                        } TO(  {  feld-element } ) [OPT (option-liste)] ;
      geräte-feld  [(index)]            feld
```

Syntax für die Ausgabe:

```
       ausdruck            geräte-name
MOVE ({          })  TO  {                        } [OPT (option-liste)] ;
       feld                geräte-feld [(index)]
```

Regeln:

- Bei dem angesprochenen Gerät darf es sich nur um eine sog. MOVE-Endstelle handeln. Die explizite Ausweisung von Geräten als MOVE-Endstellen ist zielmaschinenabhängig verschieden.
- Für die Datenübertragung vom Gerät zur Senke sind als Senken nur Variable, Feldelemente und Felder vom Typ BIT oder FIXED zugelassen.
- Für die Datenübertragung von der Quelle zum Gerät sind als Quelle nur Ausdrücke vom Resultat-Typ BIT oder FIXED sowie Felder vom Typ BIT oder FIXED zugelassen.
- Die Index-Angabe kann sowohl durch eine ganze Zahl als auch durch eine Variablen-Angabe erfolgen.

Beispiele:

```
MOVE STDEV1 (5) TO(FELD(5)) OPT (GAUGE(NORM));
MOVE ANALOG TO (MESSW) OPT (MESSWA);
MOVE (FELD(2)) TO STDEV5(2);
MOVE (MESSW) TO ANALOG;
```

4-6.4.4 Beschreibung der satzweisen unformatierten Datenübertragung

Größere Mengen von unformatierten Daten, die satzweise geordnet sind, können mit nachfolgender Anweisung zu oder von einem File übertragen werden.

Syntax für die Eingabe:

```
MOVE file-name [position] TO (datenliste);
```

Syntax für die Ausgabe:

```
MOVE (datenliste) TO file-name [position];
```

Regeln:

- Für die Datenübertragung vom File zur Datenliste sind als Elemente der Datenliste nur Variable, Feldelemente und ganze Felder erlaubt.

- Für die Datenübertragung von der Quelle zum File sind als Elemente der Datenliste zugelassen:
  - o Ausdrücke
  - o Felder
  - o Konstanten-Felder.

- Die Positionierung geschieht wie unter 4-6.3 beschrieben

```
position:: = ADV({[+|-] zahl | variable})|
             POS(zahl|variable)
```

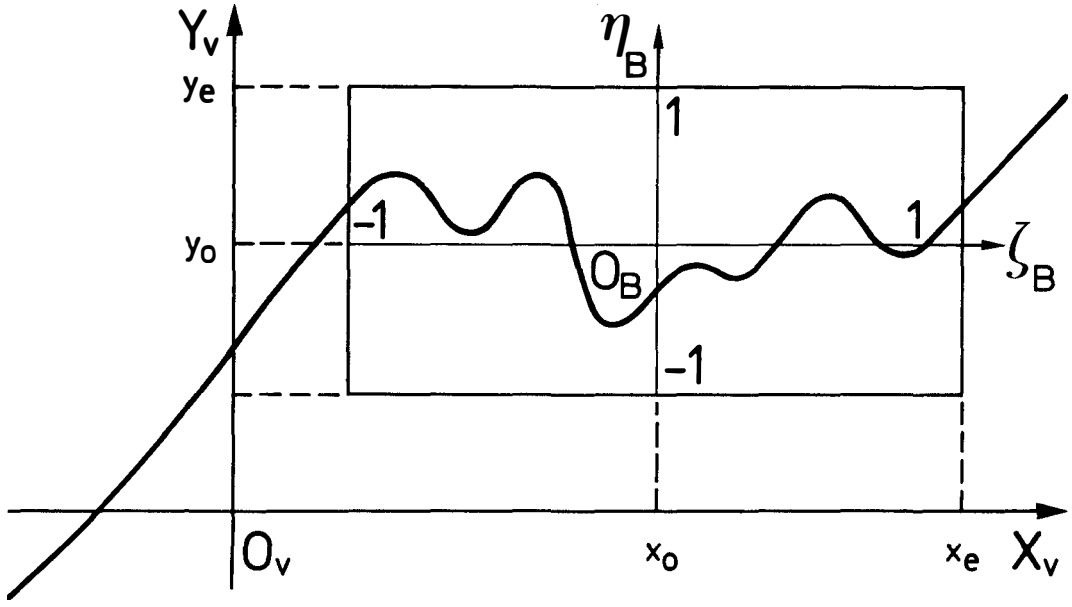
Beispiele:

```
MOVE FILE  POS(12)  TO (FELD(1), AFELD, NUMMER);
MOVE (FELD(2), BFELD, ZEILE) TO FILE  ADV(+2) ;
```

4-6.5 Formatgesteuerte graphische Datenübertragung

Die Möglichkeit mit PEARL-Anweisungen eine Datenübertragung von und zu sog. graphischen E/A-Geräten (Bildschirm, Plotter) zu betreiben, ist für diejenigen Anwendungsgebiete, in denen ein schneller Zugriff zu einer großen Anzahl von Graphen jeder Art nötig ist, von großem Wert.

Bei der graphischen Ausgabe wird die vorliegende Datenmenge als virtuelles Bild aufgefaßt, das bei technischen und physikalischen Anwendungen meistens in rechtwinkeligen Koordinaten vorliegt. Dieses virtuelle Bild bzw. ein Ausschnitt davon soll auf dem graph. Ausgabegerät abgebildet werden. Damit der Programmierer bei dieser Abbildung nicht zu sehr die Hardwaremöglichkeiten des graph. Ausgabegerätes, z.B. die Anzahl der Bildpunkte, die Länge von darstellbaren Vektoren usw. beachten muß, wird zentral in den Bildschirm ein normiertes Koordinatensystem K_B gelegt, dessen Lage im Koordinatensystem K_V des virtuellen Bildes festgelegt werden muß. Zu dieser Festlegung stellt man sich das Koordinatensystem K_V (X_V, Y_V) mit dem virtuellen Bild wie einen großen Plan aufgelegt vor und legt um den Ausschnitt, der auf dem Bildschirm abgebildet werden soll, einen Rahmen parallel zu den Koordinatenachsen X_V, Y_V.



In diesem Rahmen liegt das Koordinatensystem K_B (ξ_B, η_B) mit dem Ursprung O_B im geometrischen Mittelpunkt des Rahmens. Die Koordinaten dieses Punktes O_B (x_o, y_o) und die Länge der positiven Halbachsen ($x_e - x_o$) der ξ_B-Halbachse und ($y_e - y_o$) der η_B-Halbachse in Einheiten des Koordinatensystems K_V genügen zur Beschreibung der Transformation T : $K_V \rightarrow K_B$. Für einen Punkt P mit den Koordinaten (x, y) im System K_V berechnen sich dann die K_B-Koordinaten (ξ,η) des Punktes P' zu

$$\xi = (x - x_o) * S_x \quad , \text{ mit } S_x = |1 /(x_e - x_o)| \text{ und}$$
$$\eta = (y - y_o) * S_y \quad , \text{ mit } S_y = |1 /(y_e - y_o)|.$$

Die Abbildungsmaßstäbe S_x und S_y sind die Beträge der reziproken Längen der positiven Halbachsen.

Für die Punkte P' die auf dem Bildschirm als Bildpunkte ausgegeben werden, gilt dann $|\xi| \leq 1$ und $|\eta| \leq 1$.

Ist für P': $|\xi| > 1$ und/oder $|\eta| > 1$, so liegt P' außerhalb des festlegten Rahmens und erscheint nicht mehr auf dem Bildschirm.

Der Programmierer hat nun, nachdem er die Koordinaten x_o, y_o und die Maßstäbe S_x und S_y festgelegt hat, die Möglichkeit, das Bild durch die Ausgabe von Bildelementen (Punkte und Striche) aufzubauen.
Generell wird ein Bildelement aus einem Koordinatenpaar (x, y) des virtuellen Koordinatensystems K_V aufgebaut, daraus wird die Lage eines Bildpunktes berechnet. Dieser Bildpunkt kann in zwei Helligkeitsstufen gezeichnet werden (Punktausgabe) oder er wird mit dem zuletzt ausgegebenen Bildpunkt durch eine durchgehende oder gestrichelte Linie verbunden (Vektorausgabe). Die Ausgabe von x_o, y_o, S_x und S_y und die Punkt- und Vektorausgabe erfolgt mit Hilfe der in 6.5.3.3 beschriebenen Formate.
Der Programmierer kann die Bildelemente ohne Rücksicht auf den gewünschten Bildausschnitt ausgeben, da von den Ausgaberoutinen mit Hilfe der beschriebenen Transformation T die entsprechende Restriktion erreicht wird. Dadurch kann nur durch Änderung von x_o, y_o, S_x und S_y immer wieder ein anderer Bildausschnitt auf dem Bildschirm abgebildet werden, ohne daß die Folge der Ausgabestatements oder die sonstigen Daten geändert werden müssen.

Bei der graphischen Eingabe markiert der Benutzer einen Punkt auf dem Bildschirm mit der Rollkugelmarke. Nach dem Drücken der Koordinateneingabetaste werden die (ξ,η) Koordinaten des K_B Koordinatensystems mit der Umkehrtransformation T' = $K_B \rightarrow K_V$ umgerechnet. Die dem Programm übergebenen Eingabekoordinaten (x, y) berechnen sich demnach zu

$$x = x_o + \xi / S_x \quad \text{und}$$
$$y = y_o + \eta / S_y.$$

4-6.5.1 Anweisungen für die graphische E/A

4-6.5.1.1 Graphische Ausgabe

```
graphic-c-s::=
     DRAW device-identifier
     EDIT ( [data-source-list])
     ( { g-format-list/remote-g-format-list})

remote-g-format-list::=
     R(format-label-constant)

data-source-list::=
     {,'{expression/array-reference-one/value-array}...}

g-format-list::=
     {,'{[multiplier] g-format-element}...}

g-format-element::=
     g-executive-control/
     g-mapping-control/
     (g-format-list)

multiplier::=
     (decimal-integer-constant-denotation)
```

4-6.5.1.2 Graphische Eingabe

```
graphic-c-1-s::=
     SEE device-identifier
     EDIT ( [data-sink-list] )
     ( {g-format-1-list/remote-g-format-1-list} )

remote-g-format-1-list::=
     R(format-label-constant)

data-sink-list::=
     {,' { reference-one/array-reference-one } ...}

g-format-1-list::=
     {,' { [multiplier] g-format-1-element } ...}

g-format-1-element::=
     absolute-coordinate-control/
     g-executive-control/
     (g-format-1-list)

multiplier::=
     (decimal-integer-constant-denotation)
```

4-6.5.2 Regeln zur graphischen-E/A

Regeln:

- Mit dem Geräte-Namen (device-identifier) können nur graphische E/A-Geräte (Bildschirm, Plotter) angesprochen werden. Die Auswahl und die Namen der Geräte ist zielmaschinenabhängig verschieden.

- Die Abarbeitung der Daten- und Formatliste erfolgt gekoppelt sequentiell in fast gleicher Art wie bei der Std.-E/A. Benötigt ein Formatelement ein Datum, so wird das jeweils nächste aus der Datenliste verarbeitet.

- Die Datenliste kann Konstante (nur bei Ausgabe), Variable, Arrays und Ausdrücke der Typen Fixed und Float enthalten.

- Bei einer Ausgabe-Anweisung kann die Formatliste alle Formatschlüssel, die in der Syntax aufgeführt sind, in beliebiger Zahl und Reihenfolge enthalten. Schachtelungen von Formaten durch Klammern sind bis zur Tiefe 3 erlaubt.

- Bei einer Eingabe-Anweisung sind nur die Formatschlüssel XA und/oder YA und MAP zugelassen.

- Vor den Formatschlüsseln bzw. vor Klammerungen sind konstante Multiplikatoren zugelassen. Der Zahlenraum dieser Multiplier ist zielmaschinenabhängig.

- Ist die Datenliste länger als die Formatliste, so wird der letzte, durch Klammern eingeschlossene Teil der Formatliste wiederholt abgearbeitet bis das Ende der Datenliste erreicht ist.

- Ist die Formatliste länger als die Datenliste - dies ist auch bei fehlender Datenliste der Fall -, so wird die Formatliste nur soweit abgearbeitet wie die Datenliste reicht. D.h. in der Formatliste wird vorgeschritten bis zu einem ein Datum forderndes Formatelement, dessen Datum außerhalb der Datenliste liegt, also nicht existiert. Alle folgenden Formatelemente, auch Formatsteuerungselemente (die kein Datum benötigen) bleiben unberücksichtigt.

4-6.5.3.1 Wirkung der graphischen Formatelemente

Trotz der formalen Ähnlichkeit zwischen den Anweisungen für die Std.-E/A und die Graphic-E/A, besteht ein wesentlicher Unterschied zwischen diesen beiden E/A-Klassen in der Wirkungsweise der Formatelemente. Während bei der Std.-E/A die Formatelemente

eine einmalige Aktion bewirken (Konversion eines Datums), liefern die Formatelemente der Graphic-E/A Vorgabewerte, die solange (be)stehen bleiben bis sie durch entsprechende weitere Formatanweisungen neu bestimmt werden.

D.h. die Formatanweisungen liefern statische Basiswerte (z.B. Parameter für Koordinatenverschiebung), die nur durch Überschreiben mittels neuer Formatanweisungen neu gesetzt werden können. Diese Vorgabewerte werden in sog. Register-Speicher gehalten, die dann das E/A-Gerät (Bildschirm) mit den entsprechenden Vorgaben versorgen.

4-6.5.3.2 Formatelemente für die graphische E/A

```
g-mapping-control::=
     g-linear-control/
     coordinate-control/
     point-layout-control/
     OM

g-linear-control::=
     g-l-c-element/
     FRAME

coordinate-control::=
     absolute-coordinate-control/
     relative-coordinate-control

g-l-c-element::=
     origin-control/
     scale-control/
     increment-control/
     IP

absolute-coordinate-control::=
     XA/YA

relative-coordinate-control::=
     XR/YR

origin-control::=
     XO/YO

scale-control::=
     XS/YS

increment-control::=
     XI

g-executive-control::=
     MAP

point-layout-control::=
     BR
```

4-6.5.3.3 Beschreibung der graphischen Formatelemente

Regel: - Für die meisten Formatelemente sind Datenelemente vom Typ Fixed oder Float zugelassen, nur bei den Formatelementen 'XI', 'BR', 'IP' wird für das Datum der Typ Fixed verlangt. Bei einem Verstoß meldet sich das Betriebssystem.

Register-Speicher enthalten die letzten Werte von;		Startwert
Xo-Register	XO	0
Yo-Register	YO	0
Xs-Register	XS	1
Ys-Register	YS	1
Xa-Register	XA	0
Ya-Register	YA	0
Xr-Register	XR	0
Yr-Register	YR	0
Xi-Register	XI	0
BR-Register	BR	0
IP-Register	IP	0

Koordinaten-Steuerungs-Elemente: XA, YA

Die Datenelemente zu XA bzw. YA beinhalten die absoluten Koordinaten eines abzubildenden Punktes (in Koordinateneinheiten des virtuellen Bildes).

Beispiel: Positionieren des Bildpunktes (Strahl, Stift) auf den Punkt (17,24).

```
DRAW DISPLAY EDIT  (17,24) (XA, YA, MAP);
```

Eingabe eines Koordinatenpaares:

```
SEE  SCHIRM  EDIT  (VX,VY) (XA, YA, MAP);
```

Koordinaten-Steuerungs-Elemente: XR, YR

Die Datenelemente zu XR bzw. YR beinhalten die relativen Koordinaten (Abstände) eines abzubildenden Punktes zu dem zuletzt abgebildeten Punkt (in Koordinateneinheiten des virtuellen Bildes).

Beispiel: Neuer Bildpunkt im Abstand (5, -3) vom vorherigen

```
DRAW DISPLAY EDIT (5, -3) (XR, YR, MAP);
DRAW DISPLAY EDIT (5) (XR, MAP);
```

Die Koordinaten des neuen Bildpunktes belegen die XR/YR-Register-Speicher.

Koordinaten-Ursprung-Steuerungs-Elemente: XO, YO

Die Datenelemente zu XO bzw. YO beinhalten die Koordinaten des Bildmittelpunktes (in Koordinateneinheiten des virtuellen Bildes).

Beispiel: Positionieren des Bildmittelpunktes auf die Werte von X/Y-ORIGIN.

```
DRAW DISPLAY EDIT (XORIG, YORIG) (XO, YO);
```

Skalierungs-Steuerungs-Elemente: XS, YS

Die Datenelemente zu XS bzw. YS beinhalten die Maßstabsfaktoren für die X- bzw. Y-Richtung, die zur Umrechnung vom Koordinatensystem des virtuellen Bildes in das Koordinatensystem des realen Bildes benötigt werden.

Beispiel: Aus dem virtuellen Bild soll der Ausschnitt x c -0.1, +0.1 , y c -100, +100 auf dem Bildschirm abgebildet werden.

```
DRAW DISPLAY EDIT (10, 0.01) (XS, YS);
```

Zuwachs-Steuerungs-Element: XI

Das Datenelement zu XI beinhaltet das Inkrement in X-Richtung, für jeden folgenden Bildpunkt (in Hardware-Schritten). Findet Anwendung bei Ein-/Ausgabe eines Arrays.

Beispiel: Über der X-Koordinate soll ein Array von 512 Meßpunkten aufgezeichnet sein, wobei der Abstand der Punkte in X-Richtung 1 Inkrementalschritt beträgt.

```
DRAW DISPLAY EDIT (1, ARRAY) (XI,(512) (YA, MAP));
```

Der Xi-Register-Speicher wird durch die Formate XA, XR auf Null zurückgesetzt.

Auslassungs-Element: OM

Das zu OM gehörende Datenelement wird übergangen.

Intensitäts-Steuerungs-Element: BR

Das zu BR gehörende Datenelement steuert die Intensität des Bildelementes. Die Bedeutung der Werte, die das Datum annehmen kann, ist zielmaschinenabhängig verschieden, z.B.:

0 = Dunkelausgabe der Bildelemente
1 = Hellausgabe der Bildelemente
2 = Dickpunktausgabe bei Punkten

Interpolations-Steuerungs-Element: IP

Interpolation bedeutet Verbindung der Bildpunkte. Der Wert des zugehörigen Datums bestimmt die Art der Interpolation. Wertebereich und Bedeutung der Einzelwerte ist zielmaschinenabhängig verschieden, z. B.:

O = keine Verbindung der Bildpunkte
1 = lineare Verbindung der Bildpunkte
2 = quadratisch interpolierte Verbindungslinie usw.

-1, -2, ...: Verbindungslinie gestrichelt.

Rahmen-Steuerungs-Element: FRAME

Durch FRAME wird ein neues Bild gestartet (bedeutet beim Bildschirm gleichzeitig Löschen des vorhandenen Bildes).

Alle folgenden graphischen-Ausgabe-Anweisungen innerhalb der laufenden Task werden in das neue Bild gezeichnet bis dieses durch ein nächstes FRAME beendet wird.

Ein eventuell (zielmaschinenabhängig) zugehörendes Datum steuert Breite und Höhe des neuen Bildes.

Sämtliche Register-Speicher bleiben unverändert.

Ausführungs-Steuerungs-Element: MAP

Dem Format-Element MAP ist kein Datum zugeordnet.

Eingabe: Nach Drücken der Eingabetaste werden die Koordinaten der Rollkugelmarke übernommen.

Ausgabe: Aus den Werten aller Register-Speicher wird das nächste Bildelement (Punkt oder Vektor) berechnet und auf dem Display ausgegeben.

4-6.5.4.1 Beispiel zur graphischen Ausgabe

Bei den folgenden Beispielen werden Deklarationen und Spezifikationen soweit wie möglich weggelassen. Die DRAW-Anweisungen beziehen sich auf einen Standard-Bildschirm.

Dieses Beispiel zeigt verschiedene Ausgabemöglichkeiten eines Arrays.

SPEC (512) ist ein Array mit Integerwerten von 1 bis 100 000. Dieser Array soll bildschirmfüllend ausgegeben werden.

```
XORIG  = 256;
YORIG  = 50 000;
XSCALE = 2./512.;
YSCALE = 2./100 000.;

DRAW DISPLAY EDIT (XORIG, YORIG, XSCALE, YSCALE) (XO, YO, XS, YS);
```

Mit diesem Statement wird der Ursprung und der Maßstab so eingestellt, daß alle Werte des Arrays abgebildet werden.

```
     .
     .
     .
DRAW DISPLAY EDIT () (FRAME);
```

Löschen des Bildschirms, Beginn eines neuen Bildes

```
     .
     .
     .
DRAW DISPLAY EDIT (Ø) (IP);
```

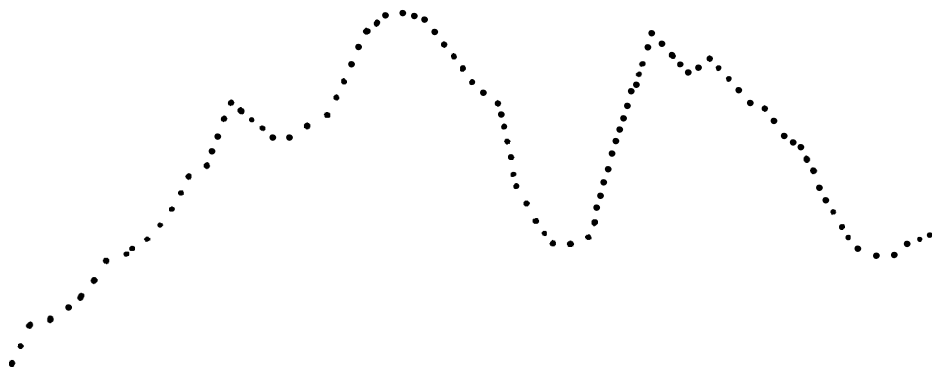
Das Bild soll im Punkt-Mode ausgegeben werden

```
     .
     .
     .
DRAW DISPLAY EDIT (Ø, Ø, Ø) (BR, XA, YA, MAP);
```

Der Kathodenstrahl wird auf den Ursprung positioniert

```
     .
     .
     .
DRAW DISPLAY EDIT (1,1, SPEC) (BR, XR, (512)(YA,MAP));
```

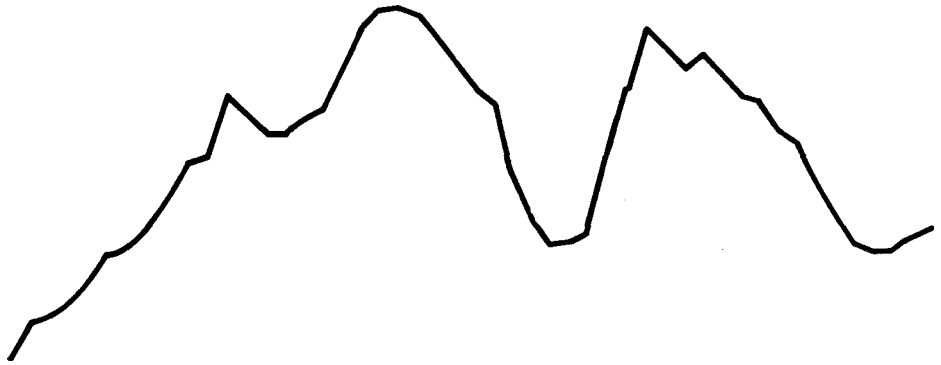
Das Spektrum wird mit maximaller Helligkeit als Punktfolge ausgegeben, wobei bei jeder Punktausgabe die Abszisse X um 1 weitergeschaltet wird.



Ausgabe des Spektrums als durchgehende Kurve
DRAW DISPLAY EDIT (1, Ø, Ø, Ø) (FRAME, IP, BR, XA, YA, MAP);

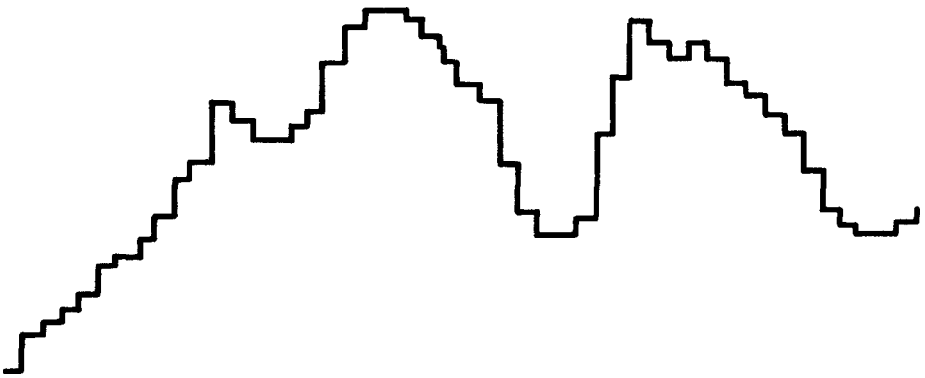
Die dem IP-Format zugeordnete Konstante 1 gibt an, daß aufeinanderfolgende Ausgabepunkte durch eine gerade Linie verbunden werden (Vektor-Mode).

```
DRAW DISPLAY EDIT (1, 1, SPEC) (XR, BR, (512)(YA, MAP));
```



Ausgabe des Spektrums als Treppenfunktion
DRAW DISPLAY EDIT (1, Ø, Ø, Ø, 1) (FRAME, IP, BR, XA, YA, MAP, BR);

```
FOR I FROM 1 TO 512 REPEAT;
DRAW DISPLAY EDIT (SPEC (I), Ø, 1) (YA, XR, MAP, XR, MAP);
END;
```



4-6.5.4.2 Beispiel zur graphischen Eingabe

Vergrößerung eines Bildausschnitts mittels graphischer Eingabe. In einem auszugebenden Bild wird ein Rechteck markiert; dieses Rechteck wird dann bildschirmerfüllend ausgegeben.

```
DRAW DISPLAY EDIT (256,50 000,2./512.,2./100 000.) (XO,YO,XS,YS);
Einstellen des Ursprungs und der Maßstäbe

M1:
DRAW DISPLAY EDIT (1, Ø, Ø, Ø) (FRAME, IP, BR, XA, YA, MAP);
Löschen des Bildes und dunkelpositionieren des Ursprungs

DRAW DISPLAY EDIT (1,1, SPEC) (XR, BR, (512)(YA, MAP));
Ausgabe des Spektrums als Linienzug

SEE DISPLAY EDIT (X1, Y1, X2, Y2)((2)(XA, YA, MAP));

Zwei Koordinatenpaare werden von Rollkugel oder Lichtgriffel
eingelesen. Diese Bildpunkte sind die diagonalen Eckpunkte des
zu vergrößernden Bildausschnitts. Wird zweimal der gleiche Bild-
punkt eingegeben, so wird im Programm fortgefahren.

IF (X1 NE X2) AND (Y1 NE Y2) THEN
  BEGIN;
      XORIG  = X2 - (X2 - X1)/2.;
      YORIG  = Y2 - (Y2 - Y1)/2.;
      XSCALE = ABS (2./(X1 - X2));
      YSCALE = ABS (2./(Y1 - X2));
      DRAW DISPLAY EDIT (XORIG, YORIG, XSCALE, YSCALE)
                        (XO, YO, XS, YS);
      GO TO M1;
      END;
FI;
```

KAPITEL 5: TASKING

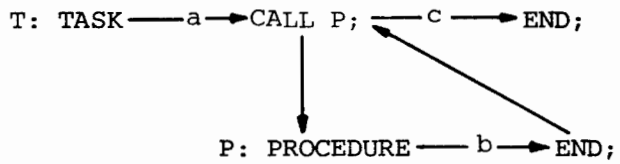
In Prozeß- und Automatisierungssystemen spielen sich zahlreiche Vorgänge zeitlich parallel ab. Der Zeitpunkt, zu dem Ereignisse auftreten, ist in der Regel nicht vorhersagbar. Programmsysteme zur Überwachung und Steuerung von Prozessen müssen diese Eigenschaften berücksichtigen. Mit einer Programmierung, die davon ausgeht, daß alle Programmteile zeitlich nacheinander ablaufen (synchrone Verarbeitung), lassen sich Echtzeitaufgaben kaum lösen. Das Programmsystem muß dem Prozeßgeschehen entsprechen. Dies führt zur asynchronen Datenverarbeitung: Programmteile, die voneinander unabhängig sind, können zeitlich parallel ablaufen.
PEARL ermöglicht die asynchrone Datenverarbeitung durch das Tasking. Es erlaubt dem Programmierer

- die Verwaltung von Betriebsmitteln (Zeit, Interrupts, Geräte, usw.) dem Betriebssystem zu übertragen

- den Zentralprozessor optimal auszulasten.

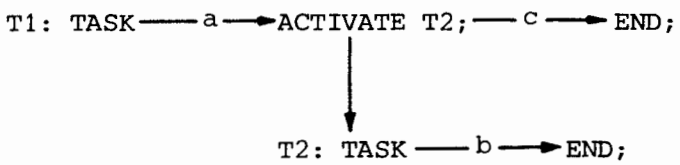
Im folgenden werden synchrone and asynchrone Datenverarbeitung kurz erläutert.

- Synchrone Verarbeitung



  Die Prozedur P läuft als Teil der Task T ab. Die Abschnitte a, b und c werden zeitlich nacheinander ausgeführt.

- Asynchrone Verarbeitung

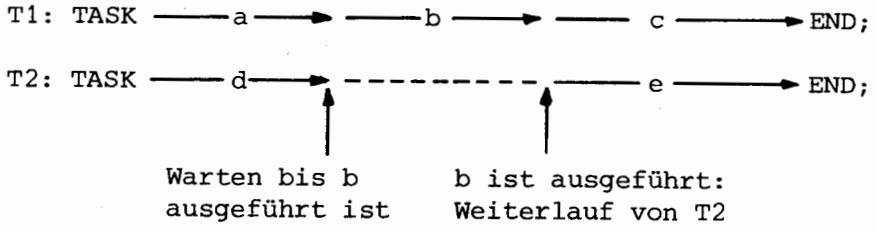


  Die Task T2 läuft parallel zur Task T1. Dabei ist es nicht unbedingt notwendig, daß die Abschnitte b und c simultan ausgeführt werden.

- Synchronisieren asynchroner Abläufe

  Die Tasks in einem Echtzeit-Programmsystem sind nur in beschränktem Umfang unabhängig voneinander. Tritt zu einem Zeitpunkt eine Abhängigkeit verschiedener Tasks auf (z.B.: Die Bearbeitung von Daten in Task T2 kann erst

erfolgen, wenn die Task T1 die Daten zur Verfügung stellt), müssen die asynchronen Abläufe synchronisiert werden.



Für das Synchronisieren asynchroner Abläufe stellt das PEARL-Subset/1

- Task-Anweisungen zum
  - Starten und Beenden von Tasks
  - Anhalten und Fortsetzen von Tasks
- Semaphor-Operationen

zur Verfügung.

Der Anstoß eines PEARL-Tasksystems erfolgt durch das Starten einer (Anlauf-) Task über das Bedienungsprogramm.

5-1. TASK-VEREINBARUNG

Tasks dürfen nur auf Modulebene deklariert bzw. spezifiziert werden (keine Subtasks).

5-1.1 Task-Deklaration

```
eingangsname: TASK [GLOBAL][RESIDENT][PRIORITY  zahl]
              task-segment
              END;
```

Die einzelnen Angaben bedeuten:

- eingangsname
  Name der Task

- GLOBAL
  Die Task kann außer in dem Modul, der die Deklaration enthält, auch in jedem Modul angesprochen werden, in dem sie spezifiziert ist.

- RESIDENT
  Residente Tasks belegen permanent den Kernspeicher; auch wenn sie nicht aktiviert sind.

- PRIORITY zahl
  Die Priorität einer Task ist eine vorzeichenlose ganze Zahl. Im PEARL-Subset/1 besitzt jede Task eine feste Priorität. Dynamischer Prioritätswechsel ist nicht möglich.
  Die Priorität beschreibt die Wichtigkeit einer Task. Bewerben sich mehrere Tasks um ein Betriebsmittel (Zentralprozessor, E/A-Gerät, Datei, usw.), so teilt das Betriebssystem dieses der höchstprioren Task zu.
  Die folgende Regel für die Vergabe von Prioritäten bewirkt eine flüssige Abwicklung aller Tasks:

  . Tasks mit viel E/A-Verkehr: hohe Priorität
  . Rechenintensive Tasks: niedrige Priorität.

  E/A-intensive Tasks müssen oft auf die Geräte warten und benötigen in dieser Zeit das Rechenwerk nicht. Es steht dann den recheninten-siven Tasks zur Verfügung.

  Anmerkung: Prioritäten schützen nicht vor dem unzulässigen Zugriff einer Task zu einem Betriebsmittel. Eine Task hoher Priorität ist machtlos gegenüber Tasks mit niedriger Priorität, wenn sie aus irgendeinem Grund angehalten oder wartend ist. Prioritäten sind ungeeignete Mittel für eine Task-Koordinierung.

- task-segment
  Das Task-Segment enthält in der angegebenen Reihenfolge:

  . Deklaration der Task-Größen (s. 2-4.2)
  . Anweisungen.

  Während die Deklarationen optionell sind, muß mindestens eine Anweisung vorhanden sein.

- END;
  Ende der Task.

  Hinweis: Läuft eine Task auf eine END-Anweisung, so wird sie beendet.

Beispiele:

```
T1:  TASK
     ACTIVATE T2;
     END; /*T1*/

T2:  TASK GLOBAL RESIDENT PRIORITY Ø
     DECLARE A FIXED;
     A = 3;
     TERMINATE;
     END; /*T2*/

T3: TASK PRIORITY 1ØØ
    DECLARE A FLOAT;
    CALL P (A);
    END; /*T3*/
```

5-1.2 Globale Task-Spezifikation

eingangsname: TASK GLOBAL [RESIDENT];

Die einzelnen Angaben haben die gleiche Bedeutung wie bei der Task-Deklaration (s. 5-1.1).
Eine Prioritätsangabe ist nicht erlaubt.

Beispiele:

```
T1:  TASK GLOBAL;
T2:  TASK GLOBAL RESIDENT;
```

5-2. ANWEISUNGEN FÜR DIE PARALLELE TASK-ABLAUFSTEUERUNG

5-2.1 ACTIVATE-Anweisung

Mit der ACTIVATE-Anweisung kann die Ausführung einer Task gestartet werden.

Syntax:

[bedingung] ACTIVATE name [PRIORITY zahl];

Regeln:

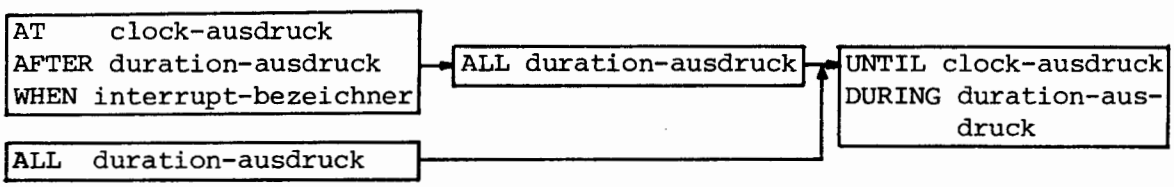
- Die ACTIVATE-Anweisung ohne Bedingungsangabe bewirkt, daß unmittelbar die durch den Namen bezeichnete Task im Betriebssystem als ablauffähig notiert wird. Diejenige Task, die ablauffähig ist und die höchste Priorität besitzt, wird ausgeführt.

- Die ACTIVATE-Anweisung ohne Bedingungsangabe für eine bereits aktive Task hat die Wirkung einer Leeranweisung.

- Durch die Prioritätsangabe wird die Wichtigkeit einer Task in Relation zur Wichtigkeit anderer im System aktiver Tasks gesetzt. Je kleiner die Prioritätsangabe ist, desto wichtiger ist die Ausführung der Task.
  Pro Task kann nur eine Prioritätszahl vergeben werden.
  Die zugelassenen Prioritätszahlen und ihre Bewertung durch das Betriebssystem ist zielmaschinenabhängig verschieden.

ACTIVATE-Anweisung mit Ausführungs-Bedingung

Syntaxdiagram für Bedingungen:



Regeln:

- Die ACTIVATE-Anweisung mit Bedingungsangabe bewirkt, daß die ACTIVATE-Anweisung erst dann ausgeführt wird, wenn die angegebene Bedingung erfüllt ist.

- Tasks können in verschiedenen Teilen des Programms mit unterschiedlichen Bedingungen aktiviert werden, es gilt dann jeweils die zuletzt wirksame Bedingung.

- Wird eine Task nur zyklisch gestartet (ALL), dann wird die Task sofort einmal gestartet und wiederholt nach Ablauf der angegebenen Zeitdauer gestartet.

- Soll die Task ab einem bestimmten Zeitpunkt zyklisch gestartet werden (AT-ALL-Bedingung), so wird die Task erstmalig zum eingeplanten Zeitpunkt gestartet und wiederholt nach Ablauf der angegebenen Zeitdauer gestartet.

Beispiele:

```
AT 5:Ø:Ø ACTIVATE STATUSBERICHT PRIORITY 5;
AT 12:Ø:Ø ALL 5 MIN DURING 3 HRS ACTIVATE HKØ1;
AFTER 1O MIN ALL 1 SEC UNTIL 12:Ø:Ø ACTIVATE LESE;
WHEN INTRP ACTIVATE ALARM PRIORITY 1;
ALL 1O SEC UNTIL 12:Ø:Ø ACTIVATE SWITCH;
```

5-2.2 SUSPEND-Anweisung

Mit der SUSPEND-Anweisung kann die Ausführung einer Task unterbrochen werden:

Syntax:

```
SUSPEND [ task-name];
```

Regeln:

- Die Angabe des Task-Namens darf unterbleiben, wenn die Task, die die SUSPEND-Anweisung enthält, zurückgestellt werden soll.
  Steht eine SUSPEND-Anweisung ohne Namen in einer Prozedur, so bezieht sie sich auf die jeweils aufrufende Task.

- Die SUSPEND-Anweisung wirkt wie eine Leeranweisung, wenn die angesprochene Task schon suspendiert oder noch nicht gestartet ist.

Beispiele:

```
SUSPEND;
SUSPEND TASK1;
```

5-2.3 CONTINUE-Anweisung

Eine durch SUSPEND oder RESUME angehaltene Task kann durch Ausführen einer CONTINUE-Anweisung fortgesetzt werden.

Syntax: (Bedingung ist optional!)

```
[ { AFTER duration-ausdruck     } ]  CONTINUE task-name;
  { AT    clock-ausdruck        }
  { WHEN  interrupt-bezeichner  }
```

Regeln:

- Eine CONTINUE-Angabe mit Bedingung wird erst dann ausgeführt, wenn die angegebne Bedingung erfüllt ist.

- Die CONTINUE-Anweisung wirkt wie eine Leeranweisung, wenn die angesprochene Task noch nicht mit der ACTIVATE-Anweisung gestartet worden ist oder bereits als ablauffähig dem System bekannt ist.

- "CONTINUE" wirkt nur auf eine durch "SUSPEND" oder "RESUME" angehaltene Task.

- Ein erneutes Auftreten einer CONTINUE- oder RESUME-Bedingung bewirkt die vollständige Ersetzung der alten Bedingung.

Beispiele:

```
AT 7:14:25 CONTINUE TASK5;
WHEN INT5 CONTINUE OFEN;
CONTINUE PUMPE;
```

5-2.4 RESUME-Anweisung

Durch eine RESUME-Anweisung kann der Ablauf einer Task für eine bestimmte Zeitspanne unterbrochen werden.

Syntax: (Bedingung ist obligatorisch)

```
{AFTER duration-ausdruck    }
{AT    clock-ausdruck       } RESUME;
{WHEN  interrupt-bezeichner }
```

Regeln:

- Die RESUME-Anweisung entspricht in ihrer Wirkung einer sofortigen Suspendierung der Task und einer bedingten Fortsetzung mit CONTINUE.

- Ein erneutes Auftreten einer CONTINUE- oder RESUME-Bedingung bewirkt die vollständige Ersetzung der alten Bedingung.

- Steht eine RESUME-Anweisung in einer Prozedur, so bezieht sie sich auf die jeweils aufrufende Task.

5-2.5 PREVENT-Anweisung

Mit dieser Anweisung können alle Bedingungen vor ACTIVATE-, CONTINUE- oder RESUME-Anweisungen aufgehoben werden. Die PREVENT-Anweisung ohne Task-Name wirkt nur auf die eigene bzw. (wenn sie in einer Prozedur steht) auf die jeweils aufrufende Task.

Syntax:

```
PREVENT [task-name];
```

5-2.6 TERMINATE-Anweisung

Mit der TERMINATE-Anweisung kann die Ausführung einer Task beendet werden.

Syntax:

```
TERMINATE [task-name];
```

Regeln:

- Die Angabe des Task-Namens kann entfallen, wenn die Task, unter deren Kontrolle die TERMINATE-Anweisung ausgeführt wird, beendet werden soll.
  Steht eine TERMINATE-Anweisung ohne Namen in einer Prozedur, so bezieht sie sich auf die jeweils aufrufende Task.

- Die TERMINATE-Anweisung wirkt wie eine Leeranweisung, wenn die angegebene Task ruhend ist (s. 5-4.1 Task-Zustände).

- Wird während des Ablaufs einer Task das physikalische Ende des Taskblocks erreicht (END;), so wird eine TERMINATE-Anweisung für die Task impliziert.

Beispiele:

```
TERMINATE;
TERMINATE TASK7;
```

5-3. SEMAPHOR-OPERATIONEN

5-3.1 REQUEST-Anweisung

Mit einer REQUEST-Anweisung kann die Fortsetzung einer Task vom Wert einer Semaphor-Variablen abhängig gemacht werden.

Syntax:

REQUEST semaphor-bezeichner;

Regeln:

- Ist der Wert der Semaphor-Variablen bei der Ausführung der REQUEST-Anweisung $> \emptyset$, so wird die Semaphor-Variable um 1 erniedrigt und das Programm mit der auf die REQUEST-Anweisung folgenden Anweisung fortgesetzt.

- Ist der Wert der Semaphor-Variablen Null, so unterbleibt die Erniedrigung des Semaphor-Werts und die Task wird zurückgestellt. Die Fortsetzung der zurückgestellten Task erfolgt, wenn die REQUEST-Anweisung ausführbar wird. Die Blockierung der Task durch die Semaphore wird somit aufgehoben.

Beispiel:

```
REQUEST SYNCHR;
```

5-3.2 RELEASE-Anweisung

Die RELEASE-Anweisung bewirkt eine Erhöhung der Semaphor-Variablen um 1. Damit kann evtl. die Erniedrigung des Semaphor-Werts durch eine vorherige REQUEST-Anweisung (s. 2. Regel bei 5-3.1) wieder ausführbar und die zurückgestellte höchstpriore Task fortgesetzt werden.

Syntax:

RELEASE semaphor-bezeichner;

Beispiel:

```
RELEASE SYNCHR;
```

5-4. TASK-KOORDINIERUNG

5-4.1 Task-Zustände

In einem Echtzeit-Programmsystem, das mehrere Tasks enthält, kann eine Task folgende Zustände annehmen:

- ruhend
  Eine Task befindet sich im Zustand "ruhend", wenn sie zwar im System vorhanden ist, aber keine Betriebsmittel - außer dem beim Laden zugeteilten Extern- bzw. Kernspeicherplatz - belegt.

- ablauffähig
  Eine Task befindet sich im Zustand "ablauffähig",

  . wenn ihr der Zentralprozessor zugeteilt ist ("laufende" Task). Die laufende Task besitzt entweder die höchste Priorität aller auf die Zuteilung des Zentralprozessors wartenden Tasks, oder ist - bei gleichen Prioritäten - als erste aktiviert worden.

  . wenn sie auf ein Betriebsmittel wartet, das vom Betriebssystem (ohne Einflußnahme des Anwender-Tasksystems) vergeben wird (z.B. Laufbereiche für externspeicherresidente Tasks, E/A-Geräte, usw.).

  . oder wenn sie ein vom Betriebssystem zugeteiltes Betriebsmittel benutzt.

  Der Zustand "ablauffähig" besagt, daß der Ablauf einer Task nicht aufgrund einer Anweisung in einer Anwender-Task blockiert ist.

- angehalten
  Der Zustand "angehalten" kann durch entsprechende Anweisungen in Anwender-Tasks erreicht werden. Er besagt, daß eine Task auch dann nicht ab- bzw. weiterläuft, wenn ihr der Zentralprozessor zugeteilt werden könnte.
  Eine Task kann sowohl sich selbst als auch andere Tasks anhalten. Letzteres bedeutet, daß eine ablauffähige Task - von ihr selbst nicht immer beeinflußbar - an einer beliebigen Stelle im Programmablauf angehalten werden kann.
  Der Ab- bzw. Weiterlauf einer angehaltenen Task erfolgt durch den Anstoß einer anderen Task oder nach dem Eintreffen eines erwarteten Ereignisses. Solche Ereignisse können sein:

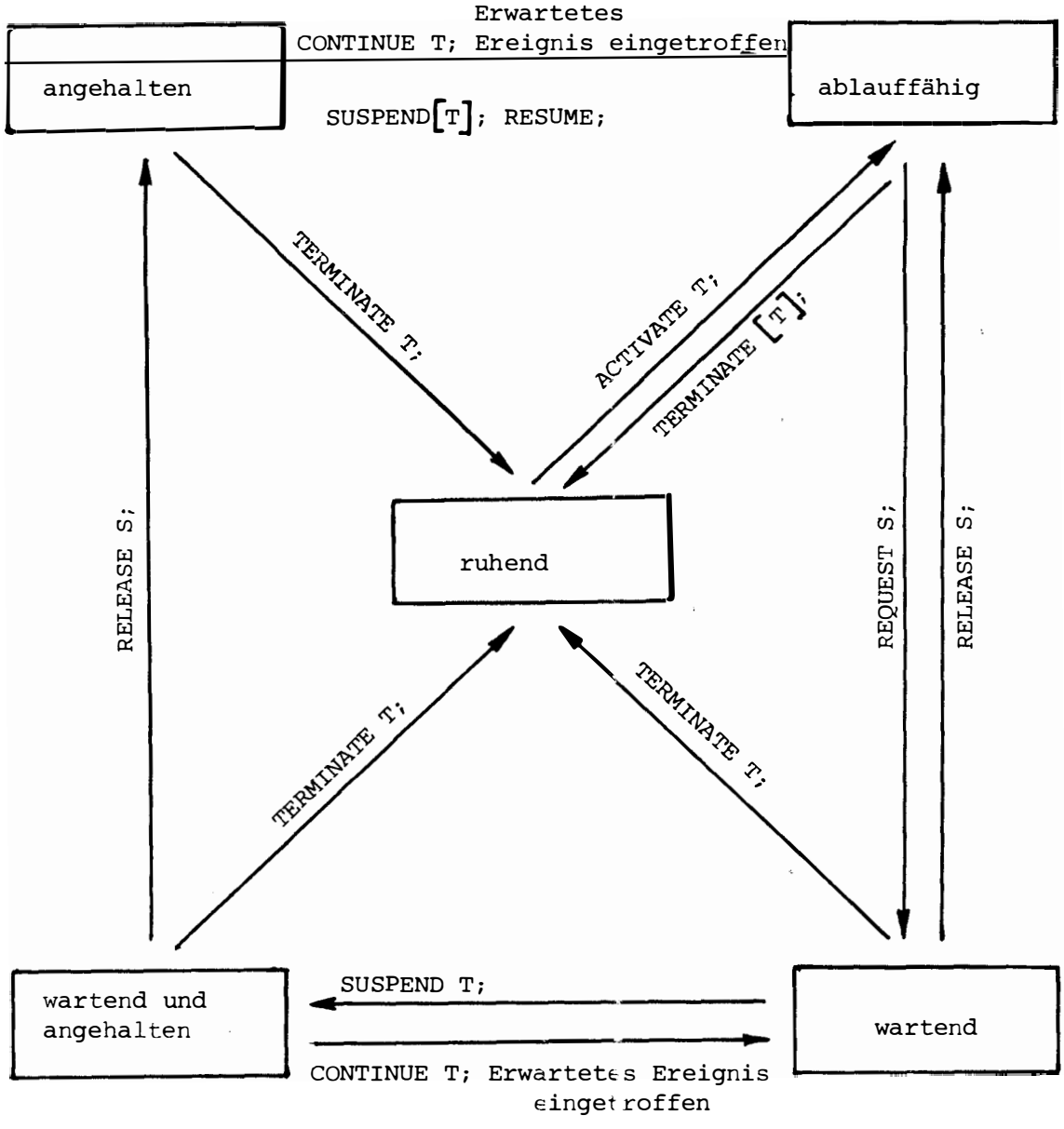
  . Eintreffen eines Interrupts
  . Ablauf einer vom Programm vorgegebenen Zeit

- wartend
  Eine Task kann den Zustand "wartend" annehmen, wenn ihr Ablauf über Semaphore gesteuert wird. Eine Task kann nur durch eigene Initiative wartend werden. Daher ist die Stelle im Programm, an der gewartet wird, stets definiert.
  Der Weiterlauf einer wartenden Task erfolgt, wenn die Fortsetzbedingung durch eine Semaphore erfüllt ist.

- wartend und angehalten
  Eine Task nimmt den Zustand "wartend und angehalten" an, wenn eine wartende Task durch eine zweite angehalten wird. Ihr Weiterlauf erfolgt erst, wenn sowohl die Warte- als auch die Anhalt-Bedingung aufgehoben ist.

5-4.2 Zustandswechsel

Ein Zustandswechsel ist ein Übergang von einem der fünf oben angegebenen Zustände in einen anderen. Die möglichen Zustandswechsel und ihre Ursachen sind im folgenden Schaubild dargestellt.
Eine Zustandsänderung kann eine Umorganisation der ablauffähigen Tasks und einen Wechsel der laufenden Task zur Folge haben.



5-4.3 Möglichkeiten der Task-Koordinierung

5-4.3.1 Anwendung der Task-Anweisungen

Die für die Koordinierung von Tasks zur Verfügung stehenden Anweisungen kann man in folgende Gruppen einteilen:

- Starten und Beenden von Tasks
  - ACTIVATE
  - TERMINATE
- Anhalten und Fortsetzen von Tasks
  - SUSPEND
  - CONTINUE
  - RESUME
- Aufheben von Bedingungen
  - PREVENT

Die Anweisungen ACTIVATE und CONTINUE können, RESUME muß Bedingungen enthalten, die festlegen, wann sie wirksam werden sollen. Bedingungen können sein:
- Angabe eines Zeitpunktes
- Angabe einer Zeitdauer
- Angabe eines Interrupts

Beispiele:
```
AT 1:3Ø:Ø ACTIVATE T;
AFTER 3 SEC CONTINUE T;
WHEN INT1 RESUME;
```

Die Bedingungen können durch die PREVENT-Anweisung aufgehoben werden:
```
PREVENT T;
```
Eine ausführliche Beschreibung der einzelnen Anweisungen ist unter 5-2. zu finden.
Die folgende Tabelle enthält eine Zusammenstellung der Zustände, die eine Task annehmen kann, wenn eine der obigen Anweisungen ausgeführt wird.

Anweisung	Zustand der Task vor Ausführung der Anweisung				
	ruhend	ablauffähig	angehalten	wartend	wartend und angehalten
ACTIVATE	ablauffähig	ablauffähig	angehalten	wartend	wartend und angehalten
TERMINATE	ruhend	ruhend	ruhend	ruhend	ruhend
SUSPEND	ruhend	angehalten	angehalten	wartend und angehalten	wartend und angehalten
CONTINUE	ruhend	ablauffähig	ablauffähig	wartend	wartend
RESUME	nicht möglich	angehalten	nicht möglich	nicht möglich	nicht möglich

Anmerkung: Die PREVENT-Anweisung bewirkt keine Zustandsänderung einer Task; sie verhindert, daß "vorgeplante" Anweisungen wirksam werden.

Beispiele:

```
R: TASK
(1)  ACTIVATE Q;
(2)  SUSPEND;
(3)  SUSPEND Q;
(4)  WHEN INT5 RESUME;
(5)  TERMINATE Q;
(6)  TERMINATE;
     END; /* R */
```

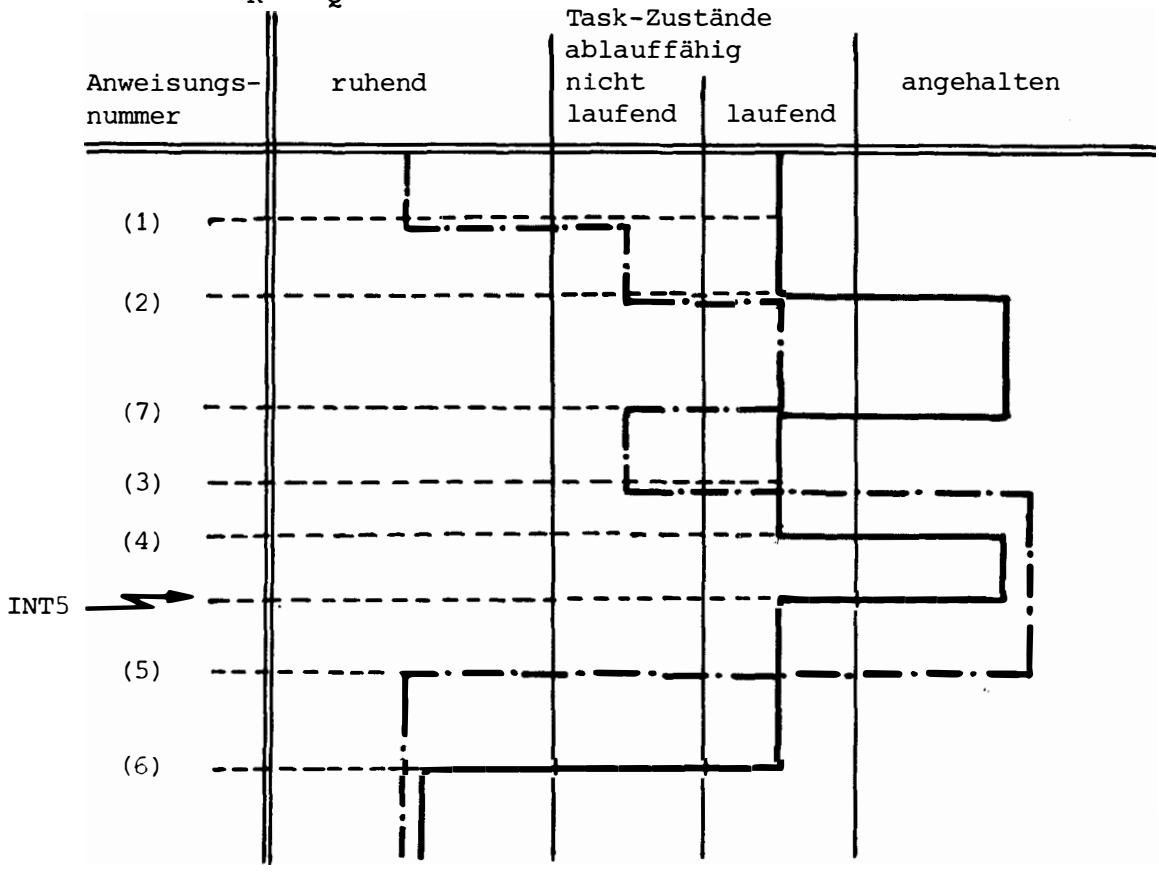
```
Q: TASK
(7)  CONTINUE R;
(8)  TERMINATE;
     END; /* Q */
```

Abängig von den Prioritäten P_R und P_Q ergeben sich zwei unterschiedliche Abläufe des Programmsystems. Beide Abläufe werden in den folgenden Zustandsdiagrammen dargestellt:

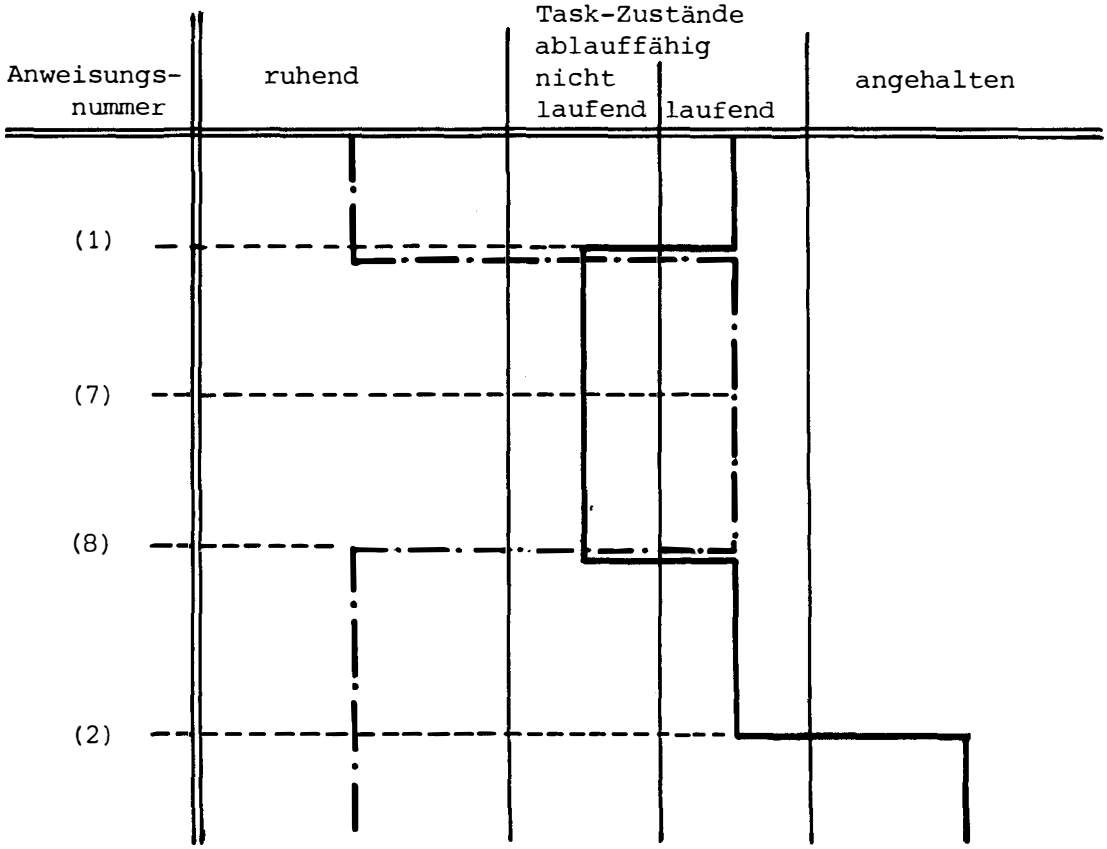
R

Q

Fall 1: $P_R \geq P_Q$



Fall 2: $P_R < P_Q$



Die folgenden Beispiele zeigen einige Anweisungsfolgen, deren Ablauf auf unerwünschte Weise erfolgen kann.

```
                    T: TASK
                      ≀
                      ▼
Fall 1:     (1)   AFTER 5 SEC CONTINUE T;
            (2)   SUSPEND;
                      ≀
                      ▼
Fall 2:     (3)   AT 12:Ø:Ø ACTIVATE T;
            (4)   TERMINATE;
                      ≀
                      ▼
Fall 3:     (5)   WHEN INT RESUME;
                      ≀
                      ▼
```

Fall 1 und Fall 2: In beiden Fällen tritt die gleiche Problematik auf: Es kann zu einem unbeabsichtigten Programmablauf kommen, wenn dieser zwischen den Anweisungen (1) und (2) bzw. (3) und (4) unterbrochen wird. Die Unterbrechung kann durch einen Interrupt verursacht werden, der die Aktivierung höherpriorer Tasks zur Folge hat. Die Task T bleibt daher im Zustand "ablauffähig"; da sie aber nicht "läuft", nimmt sie auch die durch (2) bzw. (4) gewünschten Zustände "angehalten" bzw. "ruhend" nicht an. Werden die in (1) bzw. (3) angegebenen Bedingungen in dieser Situation erfüllt, so laufen die Anweisungen (2) bzw. (4) nach der CONTINUE- bzw. ACTIVATE-Anweisung ab. Die beabsichtigte Aufhebung der durch (2) bzw. (4) erzeugten Zustände durch (1) bzw. (2) ist nicht mehr möglich. Die Task bleibt evtl. für immer angehalten bzw. ruhend.

Hinweis: Im Fall 1 kann das geschilderte Problem gelöst werden. (1) und (2) können ersetzt werden durch:
```
AFTER 5 SEC RESUME;
```

Fall 3: Es ist beabsichtigt, daß die Task fortgesetzt werden soll, wenn der Interrupt INT eintrifft.
Hierbei ist zu beachten, das die Task T auch fortgesetzt wird, wenn in einer anderen Task die Anweisung
```
CONTINUE T;
```
durchlaufen wird. Auf das Eintreffen des Interrupts wird dann nicht mehr gewartet.

5-4.3.2 Anwendung von Semaphoren

Semaphore sind leistungsfähige Hilfsmittel für die Koordinierung simultan laufender Tasks, die gemeinsame Betriebsmittel (Geräte, Dateien, Tasks, usw.) benutzen.
Eine Semaphore wird im Subset auf Modulebene (lokal oder global) vereinbart und steht damit allen Tasks, die sich über sie koordinieren wollen, zur Verfügung. Ihre Handhabung erfolgt mit den in 5-3. ausführlich beschriebenen Anweisungen
- REQUEST
- RELEASE

Die Anwendung von Semaphoren soll durch zwei Beispiele erläutert werden.

Beispiel 1: Zwei Tasks koordinieren sich in der Benutzung einer dritten Task.

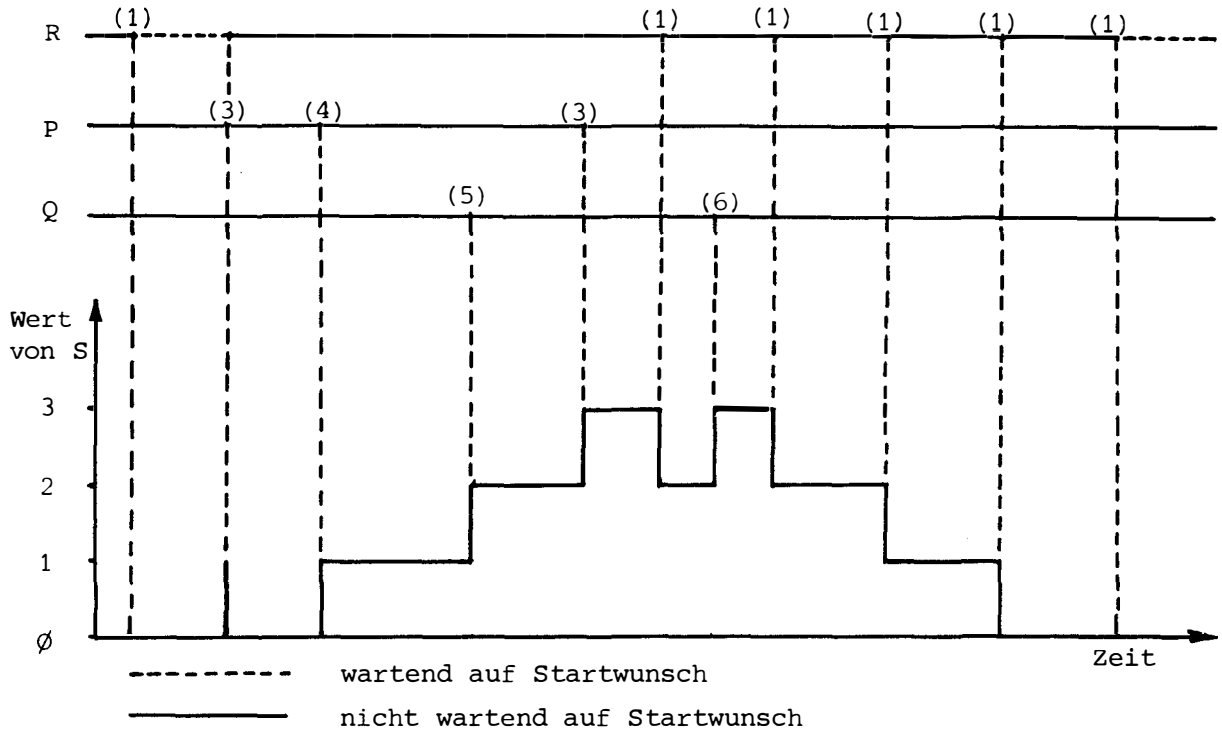
Problem: Zwei Tasks P und Q wollen mehrmals die Task R aufrufen. Dabei soll gesichert sein, daß die Aufrufe nacheinander in der Reihenfolge, in der sie zeitlich auftreten, abgearbeitet werden. Kein Aufruf soll verloren gehen.

```
                    MODULE M;
                    PROBLEM;
              DECLARE S SEMA INITIAL (Ø);

    P: TASK              R: TASK              Q: TASK
                   (1)   ANF: REQUEST S;
(3) RELEASE S;                                (5) RELEASE S;
                         GOTO ANF;
(4) RELEASE S;     (2)   TERMINATE;           (6) RELEASE S;
    TERMINATE;           END; /*R*/           TERMINATE;
    END; /*P*/           T: TASK              END; /*Q*/
                         ACTIVATE R;
                         TERMINATE
                         END; /*T*/

                         MODEND;
```

Anmerkungen:

- Die Task R wird zu Beginn nur ein einziges Mal von der Task T mit der ACTIVATE-Anweisung gestartet. Dabei wird die Anweisung (1) durchlaufen und die Task in den Zustand "wartend" versetzt.
- Die Ende-Anweisung (2) wird nie durchlaufen.
- Die Tasks P und Q können mehrmals von anderen Tasks aktiviert werden, so daß die Startwünsche für die Task R ((3),(4),(5),(6)) entsprechend oft auftreten.

In der folgenden Abbildung wird eine mögliche Situation des Task-Systems dargestellt.

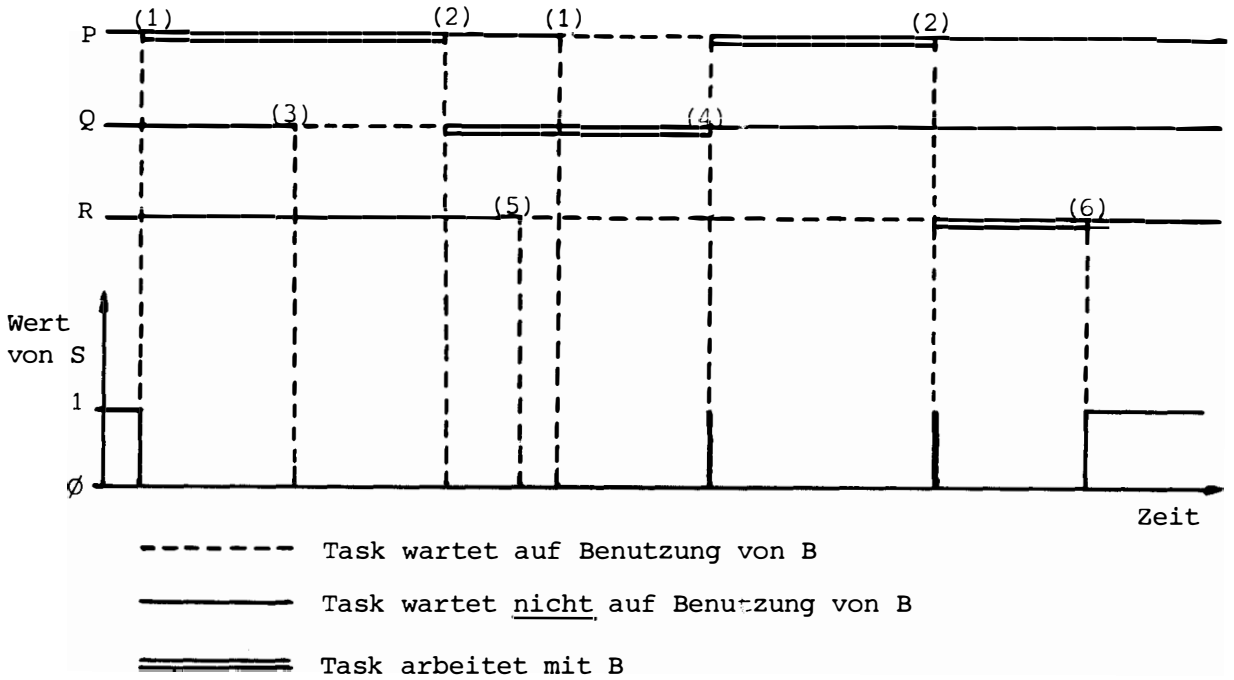


Beispiel 2: Drei Tasks koordinieren sich in der Benutzung eines Betriebsmittels.

Problem: Drei Tasks P, Q, R arbeiten mit einem gemeinsam benutzbaren Betriebsmittel B (z.B. Datei). Es soll jedoch gewährleistet sein, daß zu jedem Zeitpunkt höchstens eine Task B benutzt.

```
                    MODULE M;
                    PROBLEM;
          DECLARE S SEMA INITIAL (1);

     P: TASK              Q: TASK              R: TASK
(1)  REQUEST S;      (3). REQUEST S;      (5)  REQUEST S;
     Arbeiten mit B       Arbeiten mit B       Arbeiten mit B
(2)  RELEASE S;      (4)  RELEASE S;      (6)  RELEASE S;
     TERMINATE;           TERMINATE;           TERMINATE;
     END; /*P*/           END; /*Q*/           END; /*R*/

                    MODEND;
```

Die folgende Abbildung zeigt eine mögliche Situation des Task-Systems. (Prioritäten der Task: $P_P > P_Q > P_R$)



5-5. DATENAUSTAUSCH ZWISCHEN TASKS

Datenaustausch zwischen Tasks ist möglich durch

- Zugriff zu externen Daten (Dateien)
- Vereinbarung von Daten auf Modulebene.

Eine Datenübergabe per Parameter ist nicht möglich.

KAPITEL 6: PROZEDUREN

Zwei Gründe sprechen dafür, eine Folge von Anweisungen zu einer Prozedur zusammenzufassen.

- Gleiche Abläufe, die mehrmals vorkommen, brauchen nur einmal programmiert zu werden. Als Prozedur kann der Ablauf von beliebiger Stelle (vorausgesetzt der Prozedurname ist gültig) innerhalb einer Task bzw. einer anderen Prozedur aufgerufen werden. Damit wird dieselbe Wirkung erzielt, als wenn die betreffende Anweisungsfolge an der jeweiligen Aufrufstelle explizit vorhanden wäre.

- Selbst wenn ein Ablauf nur einmal vorkommt, ist es zweckmäßig, logisch zusammenhängende Programmteile als Prozeduren zu formulieren. Dadurch gewinnt ein Programmsystem an Flexibilität und Übersichtlichkeit.

6-1. PROZEDUR-VEREINBARUNG

Prozedur-Deklarationen sind im Subset/1 auf Modulebene und - als Deklarationen von Subprozeduren - auch auf Task-, Prozedur- und Blockebene zugelassen.
Prozedur-Spezifikationen dürfen nur auf Modulebene stehen.

6-1.1 Prozedur-Deklaration

```
eingangsname: PROCEDURE [(parameterliste)] [ rückkehrattribute] [GLOBAL]
              [REENTRANT]
               prozedurkörper
               END;
```

Die einzelnen Angaben bedeuten:

- eingangsname
  Name der Prozedur

- parameterliste
  Die Parameterliste enthält die Namen der formalen Parameter (durch Kommata getrennt).
  Es sind maximal 31 formale Parameter zugelassen.

- rückkehrattribute

  Allgemeine Form:

  ```
  RETURNS (ergebnisattribut)
  ```

  Die Rückkehrattribute kennzeichnen eine Funktionsprozedur (s.6-2.3). Diese liefert an die Stelle ihres Aufrufs einen Ergebniswert, der den angegebenen Typ besitzt. Als Ergebnisattribute sind zugelassen:

  . FIXED
  . FLOAT
  . BIT[(länge)]
  . CHARACTER[(länge)]
  . DURATION
  . CLOCK

  (Fehlende Längenangabe bedeutet die Länge 1)

- GLOBAL
  Die Prozedur kann außer in den Modulen, in denen sie deklariert ist, auch in allen Modulen aufgerufen werden, in denen sie spezifiziert ist.

- REENTRANT
  In einem Programmsystem können mehrere Tasks eine Prozedur benutzen. Diese muß reentrant sein, wenn sie während ihres Ablaufs jederzeit unterbrochen und noch vor ihrer Beendigung erneut aufgerufen werden kann (s. Beispiele).

  Hinweis: Die Deklaration von Subprozeduren darf die Attribute GLOBAL und REENTRANT nicht enthalten.

- prozedurkörper
  Der Prozedurkörper enthält optionell in der angegebenen Reihenfolge:
  . Spezifikation der formalen Parameter
    Die in der Parameterliste angegebenen Namen müssen unmittelbar nach dem Prozedurkopf mit DECLARE-Vereinbarungen (s. 2-4.2) erklärt werden. Eine Vorbesetzung mit Anfangswerten ist dabei nicht zulässig. Die formalen Parameter dürfen folgende Typen erhalten:
    .. FIXED
    .. FLOAT
    .. BIT [(länge)]
    .. CHARACTER [(länge)]
    .. DURATION
    .. CLOCK
    .. FILE

  . Deklaration der Prozedur-Größen (s. 2-4.2)

  . Anweisungen
    Programmtext der Prozedur

- END;

Ende der Prozedur

Hinweis: Bei einer CALL-Prozedur hat die END-Anweisung die gleiche Wirkung wie eine RETURN-Anweisung. Es erfolgt die Rückkehr in die aufrufende Task oder Prozedur.
Bei einer Funktionsprozedur darf die END-Anweisung nicht überlaufen werden (s. 6-2.2 und 6-2.3).

Beispiele: siehe auch 6-3.

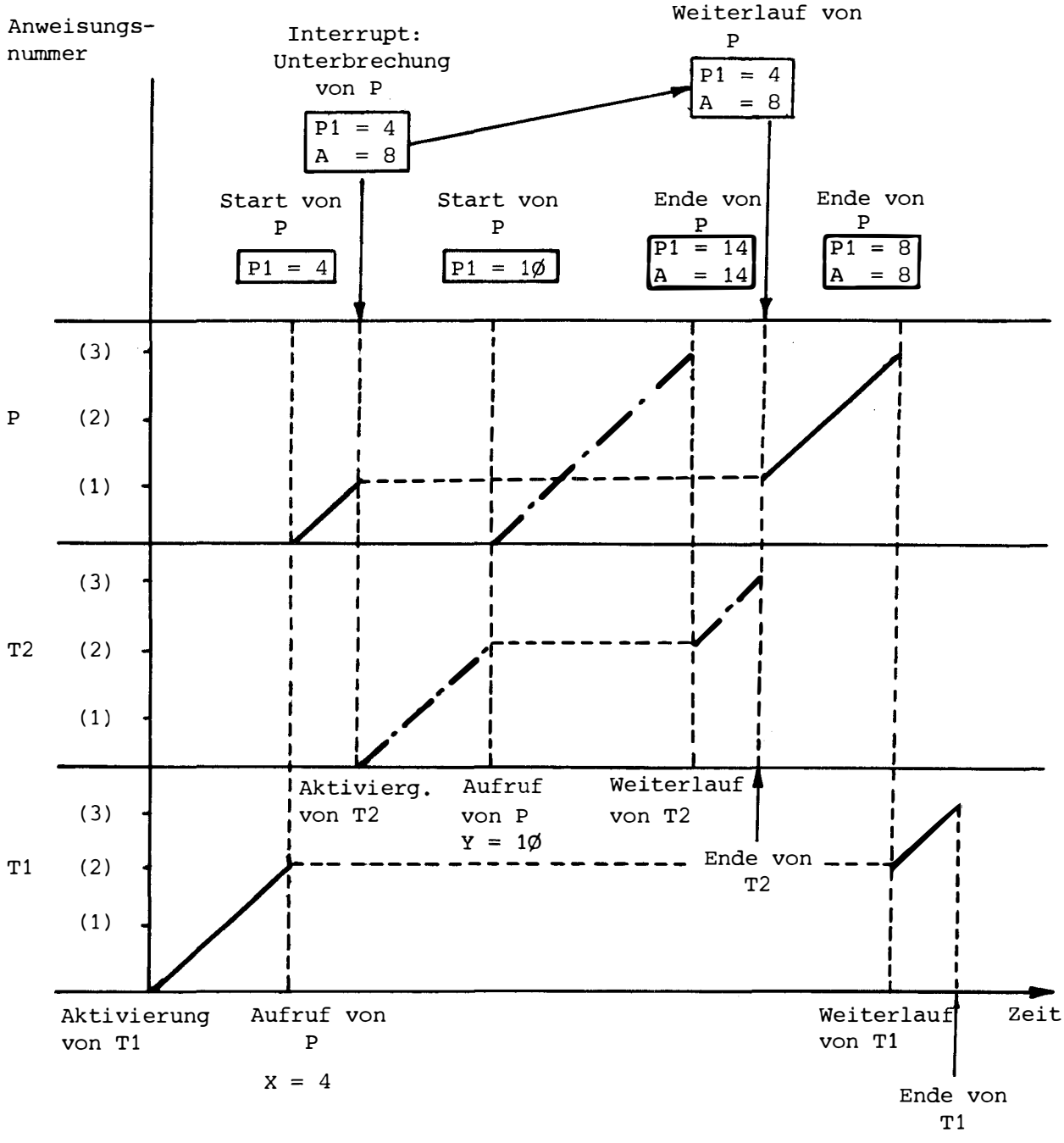
```
        /*CALL-PROZEDUR*/
PROZ1:  PROCEDURE (P1)
          DECLARE P1 FIXED; /* SPEZ. D. FORMALEN PARAMETERS*/
          END;/* PROZ*/

        /*FUNKTIONS-PROZEDUR*/
FUNK:   PROCEDURE RETURNS (FIXED) GLOBAL
          DECLARE A FIXED;
          /*SUBPROZEDUR*/
SUBPR:    PROCEDURE (P)
            DECLARE P FIXED;
            IF (A GT Ø) THEN RETURN; FI;
            P = 2;
            END; /*SUBPR*/
            A = 3;
            CALL SUBPR (A);
            RETURN(A);
          END; /*FUNK*/
```

Das folgende Beispiel soll die Verwendung einer reentranten Prozedur erläutern.
Problem: Zwei Tasks T1 und T2 benutzen eine Prozedur P.

```
                MODULE TEST;
                PROBLEM;
        P:      PROCEDURE (P) REENTRANT
                DECLARE P FIXED;
                DECLARE A FIXED;
(1)             A = P+4;
(2)             P = A;
(3)             RETURN;
                END; /*P*/
        T1:     TASK
                DECLARE X FIXED;
(1)             X = 4;
(2)             CALL P(X);
(3)             TERMINATE;
                END; /*T1*/
        T2:     TASK
                DECLARE Y FIXED;
(1)             Y = 1Ø;
(2)             CALL P(Y);
(3)             TERMINATE;
                END; /*T2*/
                MODEND;
```

Ausgangszustand: . T1 ist aktiviert, T2 ruhend.
. T2 ist höherprior als T1 und kann durch einen Interrupt aktiviert werden.

Ablaufdiagramm:



6-1.2 Globale Prozedur-Spezifikation

DECLARE {{name | (namenliste)} ENTRY-attribut [rückkehrattribute] GLOBAL
[REENTRANT]} {,{name | (namenliste)} usw.} ...;

Die einzelnen Angaben bedeuten:

- name|(namenliste)
  Eingangsname der zu spezifizierenden Prozedur.
  Mehrere Namen dürfen - durch Kommata getrennt - in einer Liste angegeben werden.

  Hinweis: Es ist zulässig, daß die Liste nur einen Namen enthält.

- ENTRY-attribut

  ENTRY [([(feldgrenzen)] [VAL] datenattribut[, [(feldgrenzen)] [VAL]
  datenattribut]...)]

  . feldgrenzen
  Untere und obere Indexgrenze (getrennt durch Doppelpunkt) für jede Felddimension des formalen Feldes. Die Untergrenze ist im Subset stets 1 und kann entfallen.

  . VAL
  Das VALUE-Attribut bezeichnet die Referenzstufe Ø (s. 6-2.4.2 Parameter-Übergabemechanismus).

  . datenattribut
  Typ des formalen Parameters
  Als Datenattribute sind zugelassen:

  .. FIXED
  .. FLOAT
  .. BIT[(länge)]
  .. CHARACTER[(länge)]
  .. CLOCK
  .. DURATION
  .. FILE
  (Fehlende Längenangabe bedeutet die Länge 1)

- rückkehrattribute
  siehe 6-1.1

- GLOBAL
  siehe 6-1.1
  Bei der Prozedur-Spezifikation muß das GLOBAL-Attribut stets angegeben werden.

- REENTRANT
  siehe 6-1.1

Beispiele:
```
DECLARE PROZ2 ENTRY GLOBAL;
DECLARE (F1,F2) ENTRY ((2,3) FIXED, VAL FLOAT, VAL FIXED)
        RETURNS (BIT(3)) GLOBAL REENTRANT;
```

6-1.3 Standard-Prozeduren

Standard-Prozeduren sind häufig benötigte Prozeduren (z.B. mathematische Funktionen, wie SINUS, TANGENS, usw.), die dem Programmierer in einer Bibliothek zur Verfügung stehen. Sie dürfen in Tasks und Prozeduren aufgerufen werden, ohne daß sie im Modul deklariert oder spezifiziert wurden. Ihr Name darf auf Modulebene in Vereinbarungen nicht benutzt werden.

6-2. HANDHABUNG VON PROZEDUREN

6-2.1 Prozeduraufrufe

An jeder Stelle einer Task oder Prozedur, an der ein Eingangsname einer Prozedur bekannt ist, darf diese Prozedur aufgerufen werden. Der Prozeduraufruf hat die Form:

eingangsname [(aktueller-parameter [,aktueller-parameter] ...)]

Regeln:

- Die Anzahl der aktuellen Parameter muß mit der Anzahl der formalen in der Prozedur-Vereinbarung übereinstimmen. Es sind maximal 31 Parameter zugelassen.
- Die Attribute (Typ, Feldgrenzen) der aktuellen und formalen Parameter müssen übereinstimmen (s. 6-1.).
- Die Referenzstufen der aktuellen Parameter müssen größer oder gleich denen der zugehörigen formalen Parameter sein.
- Als aktuelle Parameter sind zugelassen:
  - . Ausdrücke
  - . Konstante, Konstantennamen
  - . Variable
  - . (Konstanten-)Feldelemente
  - . (Konstanten-)Felder.

  Der Parameter-Übergabemechanismus ist in 6-2.4.2 beschrieben.

Abängig von der Art der Prozedur wird der oben beschriebene Prozeduraufruf wie folgt verwendet (s. auch 6-2.3):

- Aufruf von Funktions-Prozeduren
  Der Prozeduraufruf tritt als Operand in einem Ausdruck auf (s. 3-1.).
- Aufruf von CALL-Prozeduren
  Dem Prozeduraufruf geht das Schlüsselwort CALL voran (CALL-Anweisung, s. 4-3.5).

6-2.2 Rückkehr aus Prozeduren

Die RETURN-Anweisung (s. auch 4-3.6)

RETURN [(variablen-name)];

bewirkt die Rückkehr aus einer Prozedur. Die aufrufende Task bzw. Prozedur wird danach an der Stelle unmittelbar nach dem Prozeduraufruf fortgesetzt. Eine Rückkehr aus Prozeduren mit Hilfe der GOTO-Anweisung ist (auch bei Subprozeduren) nicht erlaubt.

- Rückkehr aus Funktions-Prozeduren
  Die RETURN-Anweisung muß einen Variablennamen enthalten.
  Der angegebenen Variablen wird in der Prozedur der Funktionswert zugewiesen.
  Beim Ablauf einer Funktions-Prozedur darf die END-Anweisung nicht erreicht werden.
- Rückkehr aus CALL-Prozeduren
  Die RETURN-Anweisung darf keinen Variablennamen enthalten.
  Eine RETURN-Anweisung unmittelbar vor der END-Anweisung darf weggelassen werden.

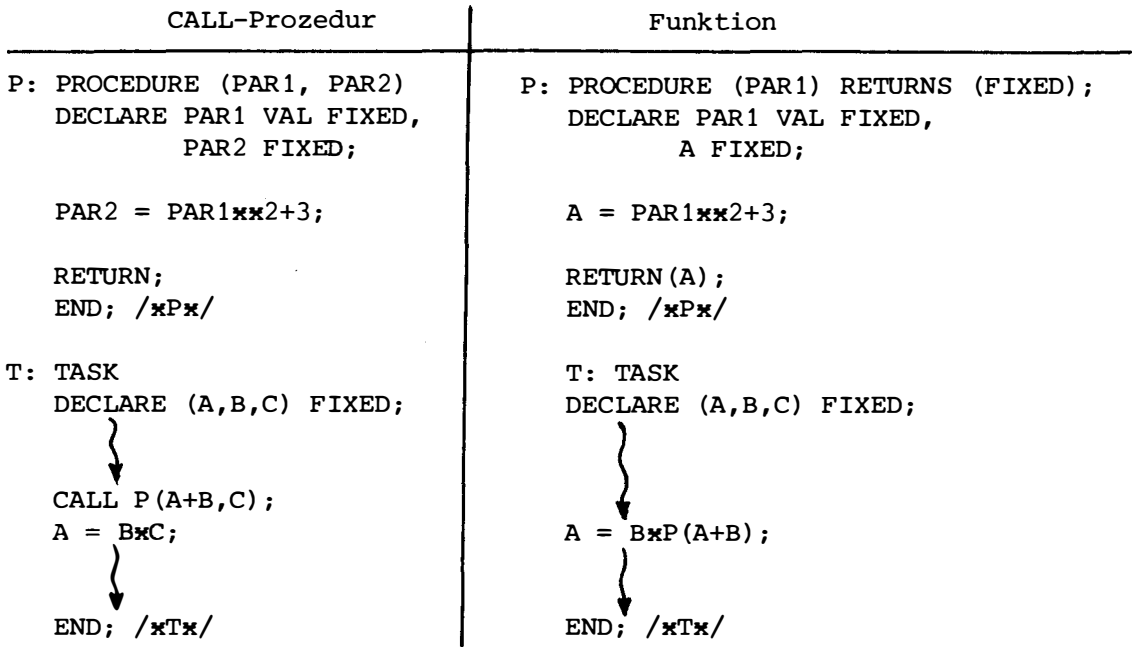
6-2.3 CALL- und Funktions-Prozeduren

Im Subset gibt es zwei Klassen von Prozeduren, die sich durch ihre Vereinbarung, ihre Rückkehr-Anweisung und ihre Verwendung unterscheiden.

- CALL-Prozeduren
  - . Vereinbarung ohne RETURNS-Attribute
  - . Aufruf mit CALL-Anweisung
  - . RETURN-Anweisung ohne Variablenname

- Funktions-Prozeduren bzw. Funktionen
  - . Vereinbarung mit RETURNS-Attributen
  - . Aufruf ohne das Schlüsselwort CALL
    Eine Funktion besitzt stets einen Ergebniswert, der an die Stelle des Aufrufs zurückgegeben wird.
  - . RETURN-Anweisung mit Variablenname

Beispiel: Im folgenden Beispiel soll der Unterschied in der Verwendung zwischen CALL-Prozedur und Funktion verdeutlicht werden:



6-2.4 Datenübergabe

6-2.4.1 Möglichkeiten der Datenübergabe

Es gibt folgende Möglichkeiten für die Datenübergabe zwischen aufrufender Task bzw. Prozedur und der aufgerufenen Prozedur:

- Zugriff zu externen Daten (Dateien)

- Vereinbarung der Daten auf Modulebene
  Datenelemente und Felder für die auf Modulebene Namen vereinbart werden, sind in sämtlichen Tasks und Prozeduren des Moduls bekannt und können dort verwendet werden.

- Datenübergabe durch Parameter
  Ein Prozeduraufruf kann eine Liste mit Parametern (maximal 31) enthalten, die der Prozedur übergeben werden. Diese Parameter heißen aktuelle Parameter oder Argumente. Ihnen entsprechen die in der Prozedur-Vereinbarung (s. 6-1.) definierten formalen Parameter. Beim Prozeduraufruf wird zwischen beiden eine Beziehung hergestellt.

6-2.4.2 Aktuelle und formale Parameter (Übergabemechanismus)

Für das Zusammenspiel von Prozeduraufruf und -vereinbarung gelten folgende Regeln:

- Die Anzahl der aktuellen und formalen Parameter muß gleich sein. Die Zuordnung von aktuellen zu formalen Parametern erfolgt bezüglich ihrer Reihenfolge in der entsprechenden Liste des Aufrufs bzw. der Vereinbarung. Im Subset sind maximal 31 Parameter zugelassen.

- Die aktuellen Parameter müssen die gleichen Datenattribute wie die ihnen zugeordneten formalen Parameter besitzen.

- Aktuelle und formale Parameter müssen, wenn es sich um Felder handelt, gleiche Anzahl der Dimensionen und gleiche Indexgrenzen besitzen.

Die Parameterübergabe kann auf zweierlei Weise geschehen:

- Übergabe per Wert
  Als aktuelle Parameter können übergeben werden:
  - Ausdrücke der allgemeinen Form nach Kapitel 3/3-1.
  - Felder
  - Konstanten-Felder.

  Bei der Datenübergabe per Wert ist ein Überschreiben des aktuellen Parameters nicht möglich, da der formale Parameter nicht verändert werden kann.

- Übergabe per Adresse
  Als aktuelle Parameter können übergeben werden:
  - Variable
  - Feldelemente
  - Felder.

  Bei der Datenübergabe per Adresse erhalten die aktuellen Parameter den Wert des zugehörigen formalen Parameters (Rückgabeparameter!).

Die Art der Parameterübergabe hängt von der Referenz-Vereinbarung der formalen Parameter ab:

- Referenzstufe Ø: Übergabe per Wert
- Referenzstufe 1: Übergabe per Adresse.

Allgemein gilt die Regel, daß die Referenzstufe des aktuellen Parameters größer oder gleich der des formalen sein muß.

Die folgende Tabelle gibt eine Übersicht über die zulässigen und verbotenen Parameterübergaben.

Aktueller Parameter	Formaler Parameter	
	Referenzstufe Ø	Referenzstufe 1
Variable	Wertübergabe	Adressübergabe
Feldelement	Wertübergabe	Adressübergabe
Feld	Wertübergabe	Adressübergabe
Konstante	Wertübergabe	Fehler
Konstantenname	Wertübergabe	Fehler
Konstantenfeldelement	Wertübergabe	Fehler
Konstantenfeld	Wertübergabe	Fehler
Ausdruck mit Operatoren	Wertübergabe	Fehler
Funktions-Prozedur	Wertübergabe	Fehler

Beispiele:

```
MODULE PARUEB;
PROBLEM;
/* DEKLARATION EINER CALL-PROZEDUR */
P: PROCEDURE (P1,P2,P3,P4,P5,P6,P7,P8,P9,P1Ø,P11,P12)
   DECLARE (P1,P2) FIXED,
            P3(1Ø,3) FIXED,
            P4(2,2) VAL FIXED,
           (P5,P6,P7,P8,P9,P1Ø,P11,P12) VAL FIXED;

   END; /*P*/

/* SPEZIFIKATION EINER FUNKTIONSPROZEDUR */
   DECLARE FUNK ENTRY (FIXED) RETURNS (FIXED) GLOBAL;

/* TASK-DEKLARATION */
T: TASK
   DECLARE (A,B,C,I,K) FIXED,
           KON VAL FIXED IDENTICAL (3),
           F1(1Ø,3) FIXED,
           KF(2,2) VAL FIXED IDENTICAL (1,2,3,4),
           (D,E,F) FLOAT,
           F2(1Ø) FIXED;

/* RICHTIGER AUFRUF */
(1)  CALL P (A,F1(I,K),F1,KF,3Ø,KON,KF(2,2),A+B/C,FUNK(A),
             B,F1(2,3),F2(I));

/* FEHLERHAFTER AUFRUF */
(2)  CALL P (D,(E+F)*D,F2,KON);
END; /*T*/
MODEND;
```

Parameterübergabe bei Aufruf (1):

aktueller Parameter	Übergabe per
A	Adresse
F1(I,K)	Adresse
F1	Adresse
KF	Wert
3Ø	Wert
KON	Wert
KF(2,2)	Wert
A+B/C	Wert
FUNK(A)	Wert
B	Wert
F1(2,3)	Wert
F2(I)	Wert

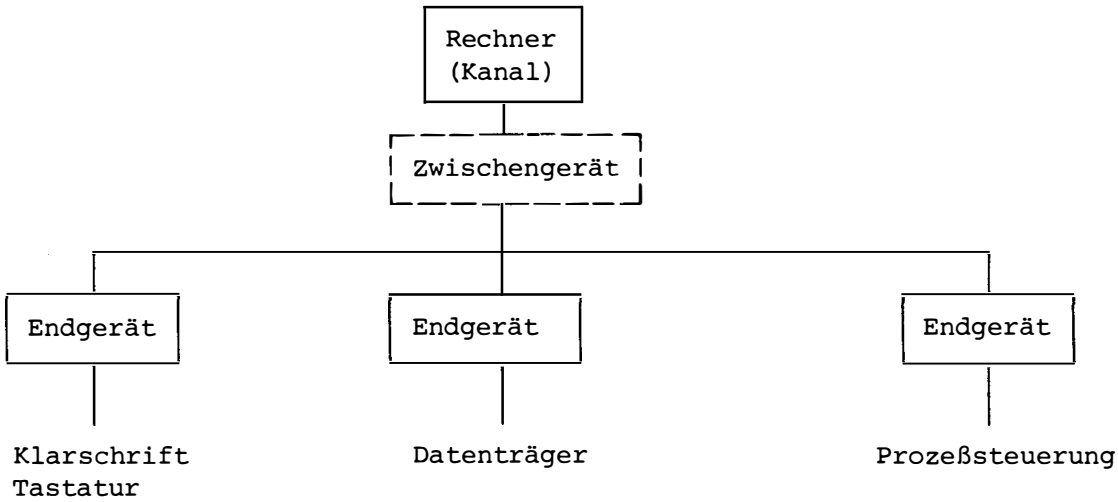
Fehler beim Aufruf(2):

- Anzahl der aktuellen und formalen Parameter stimmt nicht überein.
- Datenattribute bei den ersten beiden Parametern stimmen nicht überein.
- Anzahl der Felddimensionen stimmt bei den letzten beiden Parametern nicht überein.
- 2. Parameter: Ausdruck darf nicht per Adresse übergeben werden, da der formale Parameter die Referenzstufe Ø besitzt.

KAPITEL 7

SYSTEMTEIL

7-1. Im Systemteil gibt der Programmierer an, welche Geräte ihm im Problemteil zur Verfügung stehen müssen und wie diese am Rechner angeschlossen sind. Im Problemteil werden diese Geräte vom Programmierer angesprochen mittels geeigneter Operationen, nämlich Ein-/Ausgabe-Anweisungen. Es sind dies die Anweisungen für Charakter-E/A, graphische-E/A, Prozeß-E/A, File-Handling, Interrupt und Signal-Reaktionen, mit denen entsprechende Geräte aus der Konfiguration externer Geräte am Rechner angesprochen werden.



Für jeden Rechner gibt es eine individuelle Anordnung von externen Geräten. Diese an einen Rechner angeschlossenen externen Geräte sind für einzelne Prozeßrechenanlagen in Funktionsweise und Anzahl verschieden. Um das PEARL-Programm auf Plausibilität prüfen zu können, muß dem Übersetzer die vorhandene Anschlußstruktur der externen Geräte mitgeteilt werden. Für diese systemspezifische programmabhängige Information sieht PEARL den Systemteil vor, während der Problemteil das Programm enthält, das die Lösung des Automatisierungsproblems darstellt. Da die systemabhängige Information des Programms im Systemteil lokalisiert ist, läßt sie sich leicht abändern bzw. austauschen und damit bei der Übertragung des Programms auf ein anderes System den neuen Gegebenheiten anpassen.

Im Systemteil des PEARL-Programms wird die Anschlußstruktur der vom Programm beanspruchten externen Geräte an den Rechner beschrieben. Die einzelnen Geräte können vom Programmierer mit freiwählbaren symbolischen Namen belegt werden. Da im Problemteil die externen Geräte nur über diese symbolischen Namen aus dem Systemteil angesprochen werden, stellen diese Namen die logische Verbindung zwischen Systemteil und Problemteil her.

Die Anschlußstruktur des Prozeßrechensystems wird als Netzwerk aus Computer, Zwischengeräten und Endgeräten betrachtet. Im ASME-PEARL-Subset-1 kann im Systemteil nur eine in "Baumstruktur" angeordnete Peripherie beschrieben werden. Da bei den in Frage kommenden Zielmaschinen die Verbindung der einzelnen "Aste" des "Anschlußbaumes" hardwaremäßig einheitlich und starr (d.h. unveränderlich) ist, führt man die einzelnen "Aste" getrennt auf, wobei diese "Teilbäume" durch veränderbare systemabhängig feste Bezeichner (Schlüsselworte) identifiziert werden. Die Anschlußwege führen vom Rechner (ggfls. über Zwischengeräte) bis zu den Endstellengeräten, wobei diese vom Programmierer mittels symbolischer Namen benannt werden. Der Ursprung, Zwischenstellen und Endstellen in dieser Baumstruktur lassen sich als Knoten der einzelnen Wege für E/A-Operationen betrachten, die Verbindung von zwei Knoten als ein "Wegabschnitt".

Die Programmierung eines Systemteils ist so aufgeteilt, daß für jeden Wegabschnitt eine Systemteil-Anweisung geschrieben wird. Eine solche Anweisung beschreibt den Ursprungsknoten, die Abzweigungsstelle an diesem Knoten, die Datenflußrichtung und den Zielknoten eines Wegabschnitts. Die Knoten werden durch variable systemabhängig feste Bezeichner (Schlüsselworte) repräsentiert. Die Gesamtheit der Wege im Anschlußbaum wird durch die Aufzählung aller einzelnen Wegabschnitte beschrieben, wobei die Endstellen mit symbolischen Namen zu belegen sind.

Beispiel für Systemteil-Anweisung:

Bezeichner-n * Anschlußnummer ⟶ { Bezeichner-n+1 / symb. Name

das ist

Bezeichner für übergeordnetes Gerät	Geräte Anschluß-klemme	Datenfluß-richtung	Bezeichner für nachgeordnetes Gerät

Zusammenfassung:

Für die Anwendungsprogrammierung können folgende allgemeine Punkte als Systemteil-Charakteristika genannt werden:

- Zusammenfassung der Spezifikation der benötigten Systemkomponenten durch den Programmierer.

- Dadurch höchstmögliche Anlagen-unabhängige Programmierung im Problemteil.

- Angaben (soweit frei wählbar) über die Verbindung zwischen den benötigten Systemkomponenten, wobei die gewünschte Datenflußrichtung Teil dieser Angaben ist.

- Vergabe von symbolischen Namen an die spezifizierten Komponenten durch den Programmierer. Dadurch leichtes Ansprechen im Problemteil (statt über Dezimalzahlen als Kanalnummern wie es z.B. in Fortran erforderlich ist).

- Geräte dürfen im Problemteil in E/A-Anweisungen nur so (d.h. als Quelle oder Senke) benutzt werden, wie sie im Systemteil vereinbart bzw. "angeschlossen" wurden.

Erläuternde Beispiele:

Das folgende Beispiel zeigt wie die Systemabhängigkeit im Systemteil konzentriert ist und wie an Geräte freiwählbare Namen vergeben werden.

```
MODULE ... ;
```

für die eine Anlage

```
SYSTEM;
  .
  .
  .
  KONSOL: BSEA ... ;
  .
  .
  .
```

für die andere Rechenanlage

```
SYSTEM;
  .
  .
  .
  KONSOL: TTY47 ... ;
  .
  .
```

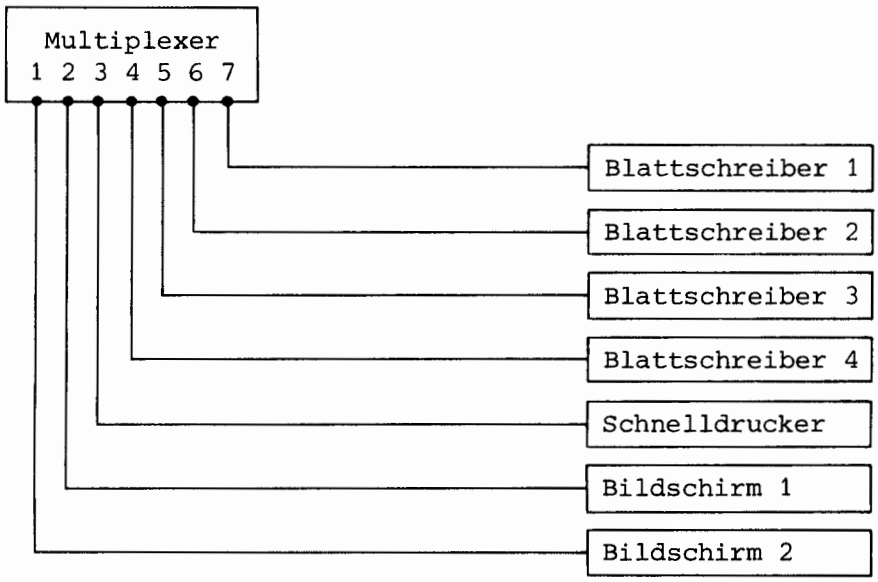
gemeinsamer Problemteil

```
PROBLEM;
  DCL KONSOL VAL DEVICE GLOBAL;
    .
    .
    .
  PUT KONSOL EDIT ( ... ) ( ... );
    .
    .
    .

MODEND;
```

Im Systemteil wird dem Gerät 'BSEA' (Schlüsselwort für Blattschreiber-Ein-Ausgabe) der Programmierername 'KONSOL' zugeordnet. Jede Bezugnahme auf dieses Gerät im Problemteil geschieht über den Namen 'KONSOL', der so die logische Verbindung zwischen Systemteil und Problemteil herstellt. Im Problemteil muß 'KONSOL' als Name für ein Device (Gerät) deklariert werden, damit der Name verwendbar ist. (Die logische Bedeutung von 'KONSOL' ist durch die Anweisung im Systemteil beschrieben.)

Bei Änderungen im System oder bei Übergang auf ein anderes System braucht das PEARL-Programm nur im Systemteil entsprechend geändert bzw. angepaßt werden, im Problemteil ändert sich nichts, weil das Gerät ja über den zugeordneten Namen 'KONSOL' angesprochen wird und nicht etwa über das systemspezifische Schlüsselwort 'BSEA'.

Das folgende Beispiel zeigt, wie Geräteanschlüsse variiert werden, soweit es sich um freiwählbare Anschlüsse handelt.



Es gibt Systeme, in denen gewisse Geräteanschlüsse keinen festen Platz haben, sondern vom Anwender gewechselt werden können, z.B. kann ein Rechner einen einzigen Multiplexer mit sieben Ausgängen besitzen, an die jeweils durch Steckverbindung irgendein beliebiges Gerät aus einer gewissen Gerätemenge angeschlossen werden kann (etwa 1 Schnelldrucker, 4 Blattschreiber und 2 graphische Bildschirme).
Der Programmierer könnte z.B. eine andere, nämlich folgende Anschlußreihenfolge wünschen, wobei

```
MPX  für den Multiplexer,
SDAU für den Schnelldrucker,
BSEA für den Blattschreiber,
GRAF für den Bildschirm mit Rullkugel-Koordinateneingabe,
     die systemspezifischen Gerätebezeichner wären.

SYSTEM;
   MPX * 1  ------>   KONSL1: BSEA;
   MPX * 2  ------>   DRUCK:  SDAU;
   MPX * 3  ------>   KONSL2: BSEA;
   MPX * 4  <------   KONSL3: BSEA;
   MPX * 5  <------>  KONSL4: BSEA;
   MPX * 6  <------   SHIRM1: GRAF;
   MPX * 7  <------   SHIRM2: GRAF;
```

In einer Systemteilanweisung wird immer ein Anschluß (eines Gerätes an ein anderes) beschrieben. Das Beispiel zeigt außerdem, daß dabei u.U. Systemkomponenten angesprochen werden müssen, die zwar für Programmaktionen im Problemteil uninteressant sind, aber trotzdem zur Spezifikation der Anschlußstruktur des aktuellen Systemausschnitts (also nur im Systemteil) zur Verfügung stehen müssen. (Im Beispiel wurde die Stellung des Multiplexers in der Anschlußstruktur der Systemkomponenten als nicht veränderlich - und somit bekannt - vorausgesetzt.)
Das Beispiel zeigt auch (für den Multiplexer), wie Anschlußpunkte an Geräten angesprochen werden, die mehr als einen Ausgang haben.

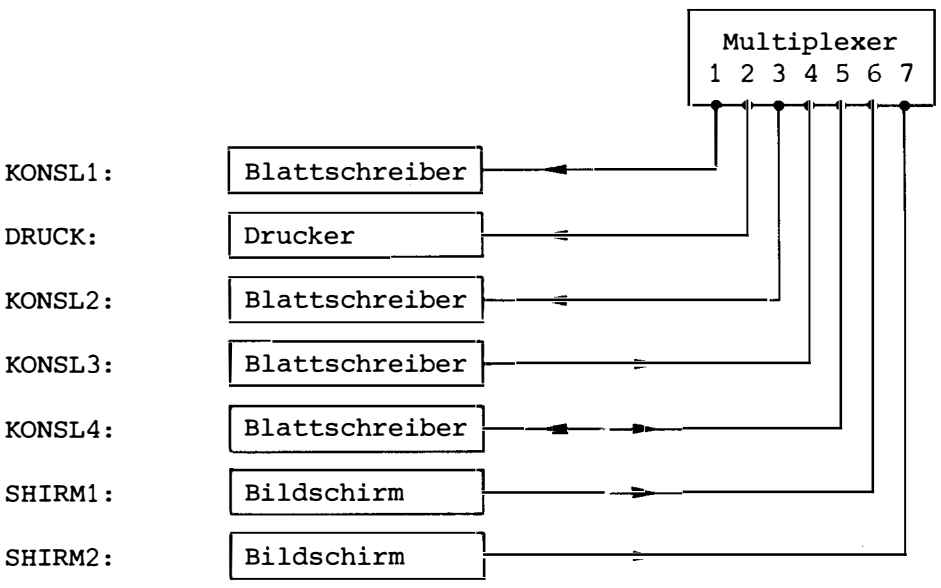
Die Pfeile im Systemteil-Beispiel zeigen die Datenflußrichtung an, die der Programmierer wünscht bzw. erlaubt. Es werden verwendet

- die Blattschreiber 'KONSL1' und 'KONSL2' nur zur Ausgabe
- der Blattschreiber 'KONSL3' nur zur Eingabe
- der Blattschreiber 'KONSL4' zur Ein- und Ausgabe
- die Bildschirme 'SHIRM1' und 'SHIRM2' nur zur Eingabe
  (von Koordinaten, die über Rollkugel eingegeben werden)
- der Drucker 'DRUCK' nur zur Ausgabe (naturgemäß).

Somit sind folgende Datenflußrichtungen festgelegt:

- für 'KONSL1', 'KONSL2' und 'DRUCK' nur hin zu den so benannten Geräten
- für 'KONSL3' 'SHIRM1' und 'SHIRM2' nur weg vom Gerät und
- für 'KONSL4' sowohl hin zum Gerät als auch weg davon.

Bildlich dargestellt:



Damit ist es dem Compiler möglich, bei der Übersetzung eines PEARL-Programmes nachzuprüfen, ob der Programmierer auf einem Gerät im Problemteil unzulässige Operationen ausführen will, wie etwa

```
      .
      .
      .
   PROBLEM;
      .
      .
      .
      DCL KONSL3 VAL DEVICE GLOBAL;
      .
      .
      .
      PUT KONSL3 EDIT (...) (...);
      .
      .
      .
```

zu einer Fehlermeldung des Compilers führen wird, da bei dieser Ausgabe Daten über den Multiplexer zum Gerät transportiert werden müßten, dieses Gerät aber so angefordert wurde, daß nur Eingabe möglich sein sollte.

Ebenfalls würde als Fehler erkannt

```
      .
      .
      .
   PROBLEM;
      .
      .
      .
      DCL SHIRM1 VAL DEVICE GLOBAL;
      .
      .
      .
      GET SHIRM1 EDIT (...) (...);
      .
      .
      .
```

da im Systemteil für die Komponenten mit den Namen 'SHIRM1' und 'SHIRM2' nur die graphische E/A-Operationen (SEE, DRAW) zugelassen sind (angezeigt durch das systemabhängige Schlüsselwort 'GRAF'). Zwar stimmt die Datenflußrichtung der Operation GET mit der im Systemteil spezifizierten Richtung überein, jedoch ist die GET-Anweisung eine unzulässige Operation auf das Gerät 'GRAF'.

7-2. SYSTEM-KONFIGURATION FÜR SUBSET-1

Die Hardware-Konfiguration des Rechners, nämlich die Verbindung mit Std. E/A-Geräten, Prozeßendstellen, Interrupts und Signals einschließlich der Datenflußrichtung in den Verbindungen, wird im Systemteil beschrieben, wobei diesen Hardware-Schnittstellen symb. Namen zugeordnet sind.

Durch die Niederschrift des Systemteils entfallen die bei Universalrechnern notwendigen Angaben der Zuordnung externer Geräte in der Job-Control-Sprache.

Der Systemteil braucht nicht die gesamte Konfiguration der Rechnerperipherie beschreiben, sondern nur die für das jeweilige Anwenderprogramm relevanten Punkte.

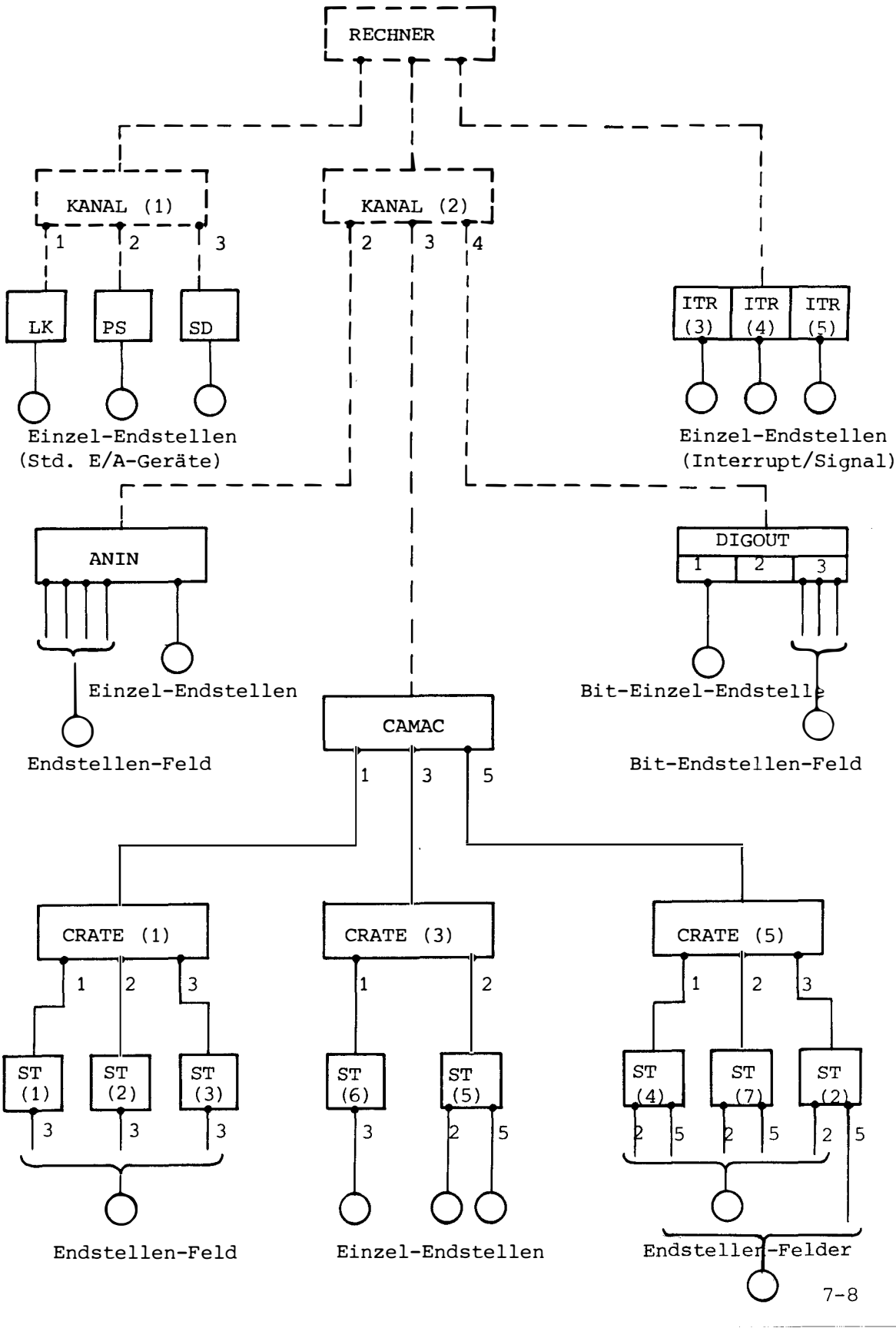
Der Systemteil des PEARL-Subsets beschreibt nur den Teil der Anschlußkonfiguration, der variabel ist. Die unveränderlichen Anschlüsse (z. B. von Zwischengeräten) werden nicht aufgeführt.

Die im Systemteil eingeführten symbolischen Namen der Endstellen sind im ganzen PEARL-Modul bekannt. Im Problemteil müssen diese Namen vor ihrer Verwendung spezifiziert werden, d.h. es muß dem Problemteil mitgeteilt werden, daß es sich hierbei um Namen handelt, die im Systemteil definiert und mit Attributen versehen worden sind.
Diese im Systemteil festgelegten Attribute beschreiben den Anschlußort und die Anschlußart.

Bei Änderungen in der Hardware-Konfiguration und bei Übergang auf einen anderen Rechner braucht nur der Systemteil entsprechend geändert zu werden, um das Problemprogramm der neuen Situation anzupassen.

Es ist jedoch immer erforderlich, daß die Hardware den vom Problemteil geforderten Möglichkeiten genügt.

Beispiel von Anschlüssen externer Geräte an einen Rechner in einer einfachen Baumstruktur.



Die an den Rechner angeschlossenen ext. Geräte, nämlich Std. E/A-Geräte, Prozeßsteuerungsgeräte (Digital, Analog), Camac-Gerät, führen auf die sog. Endstellen des Rechners. Diese Endstellen können vom Namen zugeordnet werden, unter denen das Programm die Endstellen anspricht.

Std. E/A-Geräte führen auf jeweils eine Endstelle, z.B. Lochkartenleser, Schnelldrucker. Prozeßsteuerungsgeräte und Camac-Gerät verfügen über mehrere Ein/Ausgänge, wobei diese Ein/Ausgänge einzeln oder gruppenweise (mehrere Anschlüsse log. zusammen) auf je eine Endstelle führen.

Eine Einzel-Endstelle ist an einen einzelen Ein/Ausgang angeschlossen. Man spricht von einem Einzel-Anschluß.

Ein Endstellen-Feld ist an mehrere Ein/Ausgänge zusammen angeschlossen, wobei man

a) eine äquidistante Folge von Anschlüssen als Anschluß-Feld
b) eine beliebige Folge von Anschlüssen als Anschluß-Gruppe

bezeichnet. Die Elemente eines Endstellen-Feldes sind also Einzelanschlüsse.

Ebenso spricht man bei mehreren Geräten gleicher Art (z.B. Untergeräte von Camac) von

a) einem Geräte-Feld, falls die Indizes der Geräte-Bezeichner eine fortlaufende und lückenlsoe Folge von Nummern bilden.

b) einer Geräte-Gruppe, falls die Indizes der Geräte-Bezeichner eine irgendwie gestreute Folge von Nummern bilden.

Einzel-Anschlüsse oder Anschluß-Felder sind gebildet aus den Elementen

a) Kanal-Anschluß (ein Datenwort parallel) oder

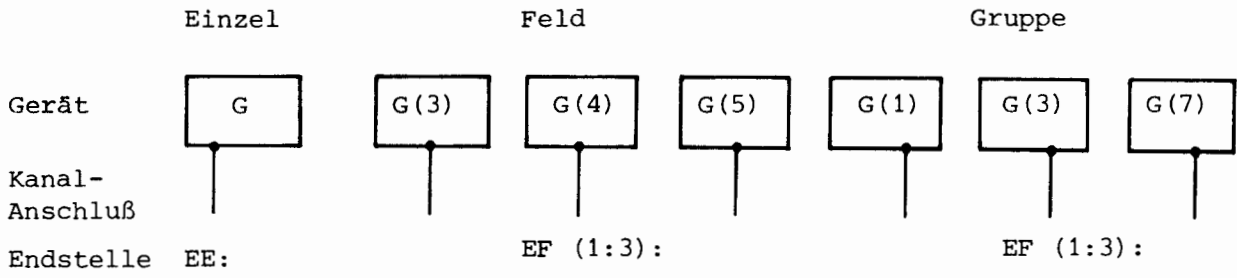
b) Bit-Anschluß (eine Einzel-Bit-Leitung oder eine Mehr-Bit-Leitung bis zu einem Wort), wobei dieser Bit-Anschluß eine Teilmenge aus der zugehörigen Kanal-Leitung darstellt.

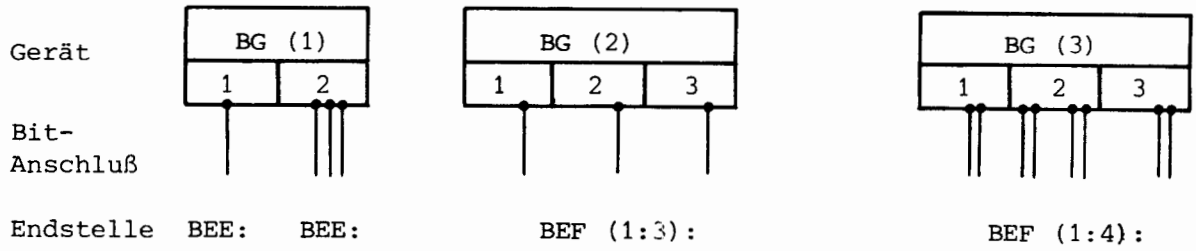
An das Camac-Gerät sind in einer Baumstruktur weitere Untergeräte angeschlossen, deren Ausgänge dann in gleicher Art auf Einzel-Endstellen oder Endstellen-Felder führen.

In diesem PEARL-Subset werden die Datenkanäle für die Anschlußwege an den Rechner nicht beschrieben, denn sie sind zielmaschinenabhängig fest. Die angeschlossenen Geräte weisen sich zielmaschinenabhängig durch ihren Geräte-Bezeichner (Schlüsselwort) aus.

Die Anschlüsse der Einzel-Endstellen und Endstellen-Felder an die Geräte bzw. Untergeräte im Falle des Camac-Gerätes und die Anschlüsse der Untergeräte an das Camac-Gerät sind variabel und damit im Systemteil des PEARL-Programmes zu beschreiben.

Beispiele für Anschluß-Arten:

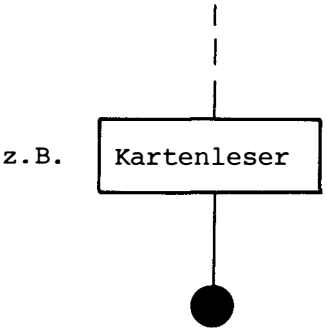




7-3. ANSCHLUSS VON GERÄTEN

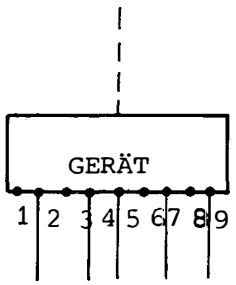
Es können verschiedene Arten von Geräten an den Rechner angeschlossen sein. Die Anschluß-Adresse gibt an, an welchem der durchnummerierten Ausgänge die Endstelle oder ein weiteres Gerät angeschlossen ist.

7-3.1 Standard-E/A-Geräte



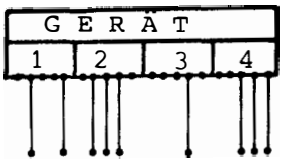
Die Anschlußleitung (Kanal) transportiert sequentiell eine konstante oder variable Anzahl von Datenworten.

7-3.2 Gerät mit einstufiger Anschluß-Adresse



Die Anschlußleitung transportiert ein komplettes Datenwort.

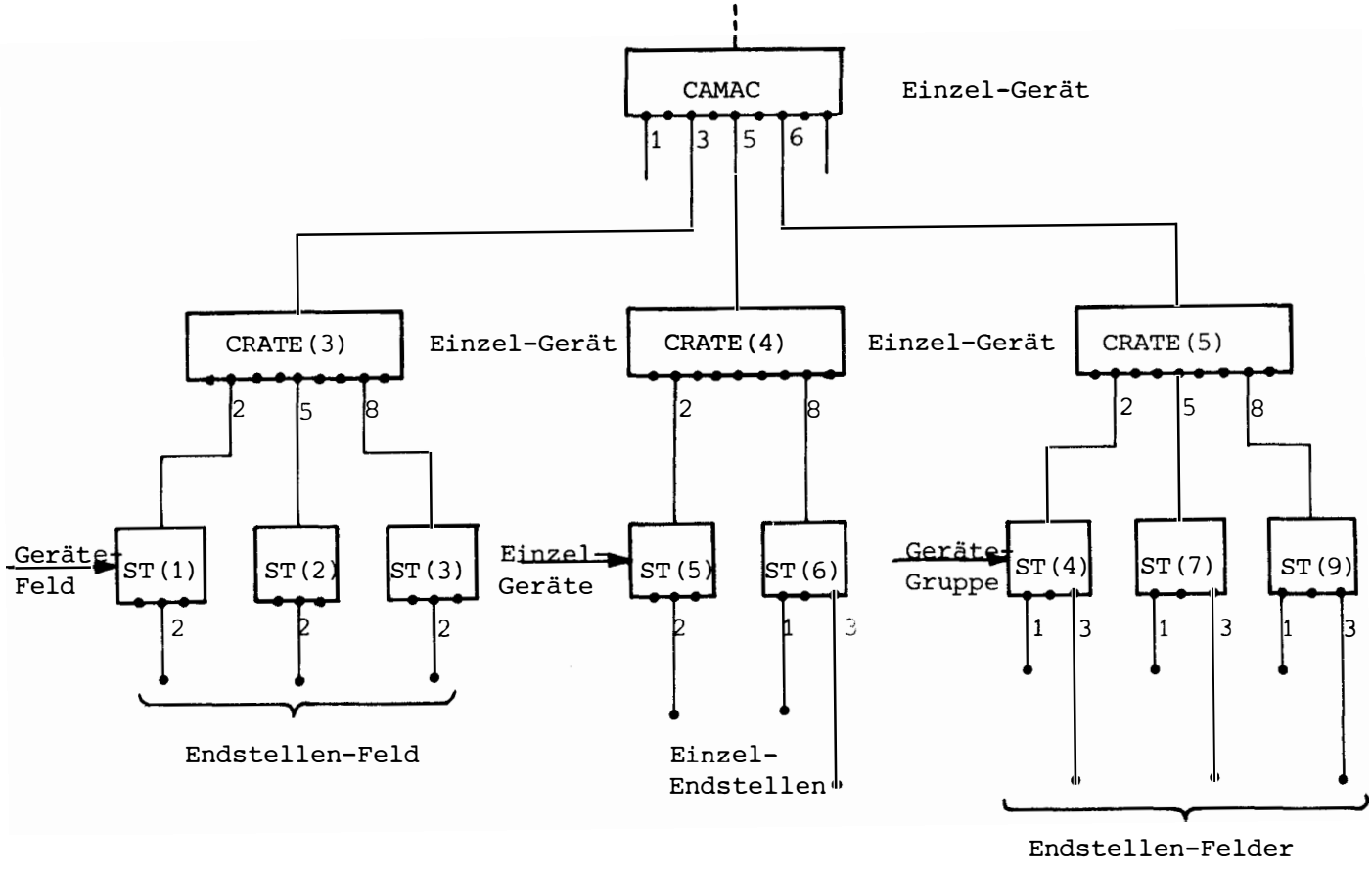
7-3.3 Gerät mit zweistufiger Anschluß-Adresse



Eingang
Geräte-Bezeichnung
Kanal-Ausgänge (Kanal-Adresse)
Bit-Ausgänge der Kanäle

Die Anschlußleitung der Bit-Ausgänge transportiert ein ein-Bit-Datum.

7-3.4 Gerät mit Untergeräten (einstufige Anschluß-Adressen)



Prozeß-Endstellen

Der Weg vom Camac-Gerät zu einer Prozeßendstelle ist bestimmt durch die Verzweigungen auf allen Ebenen. Aus den Anschluß-Adressen (an den Ausgängen) der einzelnen benutzten Geräte ergibt sich die (Gesamt-) Adresse der Endstelle vom Steuergerät aus gesehen.

7-4. VERWENDUNG VON NAMEN IM SYSTEMTEIL

Die Geräte und die Endstellen (Std. Endstellen und Prozeß-Endstellen) werden durch Namen im Sinne von PEARL bezeichnet, wobei man zwischen Einzel-Namen und Feld-Namen (einfach indiziert) unterscheidet.

```
name: = { geräte-bezeichnung [index]  }
        { endstellen-name    [index]: }
```

geräte-bezeichnung: = schlüssel-wort — Für jeden Zielrechner wird ein Satz von Schlüsselworten verbindlich festgelegt, die die an den Rechner angeschlossenen Geräte bezeichnen.

endstellen-name: = symbolischer Name — Vom Programmierer frei wählbar zur Identifikation der Endstelle im Problemteil. Diese symbolischen Namen sind im ganzen Modul bekannt.

```
index: = { (ganze Zahl)              }
         { (ganze Zahl: ganze Zahl)  }
```

Element-Angabe oder Intervall-Angabe aus einem Feld.

Beispiele:

```
ANTON
BERTA(3)
CAESAR(1:3)
DORA(2:5)
```

7-5. SYSTEMTEIL-ANWEISUNG

Der Anschlußweg von einem Gerät zu einem anderen oder von einem Gerät (oder Geräte-Feld oder Geräte-Gruppe) zu einer mit symbolischem Namen belegten Endstelle wird durch eine Systemteil-Anweisung beschrieben.

anweisung: =

geräte-bezeichnung-1 anschluß-adresse richtung { geräte-bezeichnung-2 | endstellen-name : } ;

Wenn in der Anweisung die Seite links von der Richtung mehrmals durch '+'- Zeichen verbunden auftaucht, handelt es sich um die Beschreibung einer Geräte-Gruppe, die rechts von der Richtung immer eine Feldangabe (Geräte-Feld oder Endstellen-Feld) verlangt.

anweisung: = { + { geräte-bezeichnung-1 anschluß-adresse } ... }

richtung { geräte-bezeichnung-2 [index] | endstellen-name [index]: } ;

Bemerkung: geräte-bezeichnung-1 immer ungleich geräte-bezeichnung-2

anschluß-adresse: = kanal-adresse [bit-adresse]

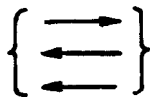
Die Anschluß-Adresse bezeichnet einen oder mehrere von den durchnummerierten Geräte-Ausgängen.

kanal-adresse: = { * symb. Adresse | * ganze Zahl | * (untergrenze: obergrenze [/ schrittweite]) }

bit-adresse: = { * ganze Zahl, bit-anzahl | * (untergrenze: obergrenze [/ schrittweite]),bit-anzahl }

untergrenze:	= ganze Zahl	Anfangswert im benutzten Intervall
obergrenze:	= ganze Zahl	Endwert im benutzten Intervall
schrittweise:	= ganze Zahl	Inkrement zum nächsten Wert
bit-anzahl:	= ganze Zahl	Anzahl der fortlaufend benutzten Bit-anschlüsse
symb. Adresse:	= Schlüsselwort	Im Falle Camac-Gerät für bestimmte Kanal-Anschlüsse am Crate-Register

Im Falle der Schrittweite 1 kann die Angabe der Schrittweite entfallen.

richtung : =  Richtung des Datenflusses

Beispiele: SWORT*3 ⟶ NAME1:;

SWORT*(3:6) ⟵ NAME2(1:4):;

SWORT(2)*7 ⟶ SWORT1(3);
SWORT1(3)*4 ⟶ NAME3:;

SWORT(2)*(7:9) ⟵ SWORT2(3:5);
SWORT2(3:5)* ⟵ NAME4(1:3):;

7-6. BESCHREIBUNG VON GERÄTEN AUF MEHREREN EBENEN

Das Camac-Gerät stellt mit seinen Untergeräten eine 3-stufige Hierarchie dar, nämlich Camac-Steuerung, Crate-Register und Stationen.

Alle Elemente der obersten Ebene (Camac) werden nacheinander aufgeführt. Unmittelbar nach jedem von diesen Elementen werden die daran angeschlossenen, für die Hierarchie tiefer stehenden Elemente aufgeführt wie folgt:

a) Alle Elemente der zweiten Ebene (Crate) und dann alle Elemente der dritten Ebene (Station).

b) Alle Elemente der zweiten Ebene (Crate und unmittelbar nach jedem Crate-Element die daran angeschlossenen Stations-Elemente der dritten Ebene.

c) In einer Rückform der beiden vorgenannten Möglichkeiten ist es auch erlaubt, nach einigen Crate-Elementen sofort daran angeschlossene Stations-Elemente zu schreiben. Zum Schluß können dann die bis dahin noch nicht genannten Stations-Unteranschlüsse von verschiedenen Crate-Elementen aufgeführt werden.

(Siehe auch in der Beispielsammlung für Camac-Anschluß.)

Prinzipielles Beispiel:

Ebene	Elemente	
A	A(1), A(2), A(3), ...	
B	B(1), B(2), B(3), ...	
C	C(1), C(2), C(3), ...	
.		
.		
.		
E	E(1), E(2), E(3), ...	Endstellen!

```
A(1) ──► B(1)
   B(1,1) ──► E(1,1)
   B(1,2) ──► E(1,2)
   B(1,3) ──► E(1,3)
A(2) ──► B(2)
   B(2,1) ──► C(2,1)
      C(2,1,1) ──► D(2,1,1)
      C(2,1,2) ──► D(2,1,2)
      C(2,1,3) ──► D(2,1,3)
   B(2,2) ──► C(2,2)
      C(2,2,1) ──► D(2,2,1)
      C(2,2,2) ──► D(2,2,2)
      C(2,2,3) ──► D(2,2,3)
   B(2,3) ──► C(2,3)
      C(2,3,1) ──► D(2,3,1)
      C(2,3,2) ──► D(2,3,2)
      C(2,3,3) ──► D(2,3,3)
A(3) ──► B(3)
  .
  .
  .
A(5) ──► B(5)
   B(5,1) ──► C(5,1)
   B(5,2) ──► C(5,2)
   B(5,3) ──► C(5,3)
      C(5,3,1) ──► E(5,3,1)
      C(5,3,2) ──► E(5,3,2)
   B(5,4) ──► C(5,4)
      C(5,4,1) ──► E(5,4,1)
      C(5,4,2) ──► E(5,4,2)
      C(5,1,1) ──► E(5,1,1)
      C(5,1,2) ──► E(5,1,2)
      C(5,1,3) ──► E(5,1,3)
      C(5,2,1) ──► E(5,2,1)
A(6)
  .
  .
  .
usw.
```

7-7. KONKRETE FORM DES SYSTEMTEILS AUF DER ZIELMASCHINE IN STUTTGART

7-7.1 Vorgegebene Gerätebezeichnungen

	Gerät	Bezeichnung
Std. E/A-Geräte:	Konsol-Schreibmaschine	IBM
	Tele-Type	TT
	Lochkarten-Eingabe	HKL
	Lochkarten-Ausgabe	HKS
	Lochstreifen-Eingabe	LSL
	Lochstreifen-Ausgabe	LSS
	Zeilendrucker-Ausgabe	ZD
	Plattenspeicher	DISK
	Plotter-Ausgabe	PLOT
		EXIN
		EXOUT
	Bedienungs-Fernschreiber	BFS
Prozeßendstellen:	Digital-Eingabe (Spannung)	DIGINS
	Digital-Ausgabe (Spannung)	DIGOUTS
	Digital-Eingabe (Relais)	DIGINR
	Digital-Ausgabe (Relais)	DIGOUTR
	Analog-Eingabe	ANIN
	Analog-Ausgabe	ANOUT
Unterbrechungen:	Interrupt-Eingabe	ITR
	Signale	SIG

7-7.2 Dimensionierung der Geräte-Ausgänge (Adreß-Bereiche)

Geräte-Bezeichnung	Kanäle	Bitausgänge pro Kanal	Bemerkungen
IBM	1		
TT	1		
HKL	1		
HKS	1		
LSL	1		
LSS	1		
ZD	1		
DISK	1		
PLOT	1		
EXIN	1		
EXOUT	1		
BFS	1...7		
DIGINS	1...8	1...24	
DIGOUTS	1...8	1...12	
DIGINR	1...6	1...24	
DIGOUTR	1...10	1...12	
ANIN	1...31		
ANOUT	1...32		
ITR		1	
SIG		1	

7-7.3 SAMMLUNG TYPISCHER EINZELBEISPIELE ZUM SYSTEMTEIL (Stuttgart)

7-7.3.1 Reihenfolge der Beschreibung der Geräte

Im PEARL-Subset-Systemteil ist für die Beschreibung der Endstellen eine gewisse Reihenfolge vorgegeben:

1. Std. E/A-Geräte in beliebiger Reihenfolge
2. Bedienungs-Fernschreiber und dann Prozeß-Endstellen-Geräte in beliebiger Reihenfolge
3. Interrupt-Eingänge
4. Signale.

7-7.3.2 Standard E/A-Geräte, Interrupts und Signals

```
BEISPIELE STD.-E/A:

      TT   <->  TELTYP: ;
      IBM  <->  KONSOLE: ;
      HKL  <-   LESER: ;
      HKS   ->  STANZER: ;
      LSL  <-   STREINGABE: ;
      LSS   ->  STRAUSGABE: ;
      ZD    ->  DRUCKER: ;
      PLOT  ->  PLOTTER: ;
      DISK <->  PLATTE: ;


BEISPIELE BEDIENUNGSFERNSCHREIBER:

      BFS*1 <-> FSCHR1: ;
      BFS*2 <-> FSCHR2: ;
      BFS*7 <-> FSCHR3: ;


BEISPIELE INTERRUPT:

      ITR(1) <- EREIG1: ;
      ITR(2) <- EREIG2: ;
      ITR(3) <- EREIG3: ;


BEISPIELE SIGNAL:

      SIG(1) <- OVERFLOW: ;
      SIG(2) <- ENDFILE: ;
      SIG(3) <- ERROR: ;
```

7- 7.3.3 Prozeß-Endstellen-Geräte

BEISPIEL 1:

```
DIGINR*1*1,1 <- BITEIN: ;
```

BEISPIEL 2:

```
DIGINR*(2:5)*5,16 <- BITAR(1:4): ;
```

BEISPIEL 3:

```
DIGINS*1*3,1 <- BITIN1: ;
DIGINS*1*6,3 <- BITIN2: ;
DIGINS*4*2,2 <- BITIN3: ;
```

BEISPIEL 4:

```
DIGOUTR*(1:7)*3,1 -> BITAR1(1:7): ;

DIGOUTR*(8:10)*3,3 -> BITAR2(1:3): ;

DIGOUTR*11*(1:9),1 -> BITAR3(1:9): ;

DIGOUTR*12*(1:9/3),2 -> BITAR4(1:3): ;

DIGOUTR*(13:16)*(1:9/3),2 -> BITAR5(1:12): ;
```

BEISPIEL 5:

```
DIGOUTS*2*3,3 +
DIGOUTS*3*3,3 +
DIGOUTS*4*3,3 -> BITAR1(1:3): ;
DIGOUTS*1*4,3 +
DIGOUTS*2*5,3 +
DIGOUTS*3*4,3 -> BITAR2(1:3): ;
```

BEISPIEL 6:

```
DIGOUTS*1*(5:8),1 +
DIGOUTS*3*(7:8),1 +
DIGOUTS*3*5,1     -> BITAR(1:7): ;
```

7-7.3.3 Prozeß-Endstellen-Geräte

BEISPIEL 7:

```
DIGOUTS*1*(3:16/8),4 +
DIGOUTS*2*6,4        +
DIGOUTS*3*(1:16/5),4 +
DIGOUTS*4*5,4        -> BITAR(1:7): ;
```

BEISPIEL 8:

```
ANIN*1   <-  DIGEL: ;

ANIN*(2:4)  <-  DIGAR1(1:3): ;

ANIN*6       +
ANIN*12      +
ANIN*24      +
ANIN*(26:27)  <-  DIGAR2(1:5): ;
```

BEISPIEL 9:

```
ANOUT*3   ->  DIGEL: :

ANOUT*(5:8)   ->  DIGAR1(1:4): ;

ANOUT*1      +
ANOUT*2      +
ANOUT*4      +
ANOUT*(9:12)  ->  DIGAR2(1:7): ;
```

7-7.4 BEISPIEL GESAMTER SYSTEMTEIL (Stuttgart)

```
MODULE EASTGT ;
SYSTEM :
       IBM <->  KONSOLE: ;
       HKL <-   LESER: ;
       HKS  ->  STANZER: ;
       ZD   ->  DRUCKER: ;
       DISK <-> PLATTE: ;

       BFS*1 <-> FSCHR1: ;
       BFS*2 <-> FSCHR2: ;
       BFS*7 <-> FSCHR3: ;

          DIGINR*1*1,1 <- BITEIN: ;
          DIGINR*(2:5)*5,16 <- BITAR0(1:4): ;

          DIGOUTR*(1:7)*3,1 -> BITAR1(1:7): ;
          DIGOUTR*(8:10)*3,3 -> BITAR2(1:3): ;
          DIGOUTR*12*(1:9/3),2 -> BITAR3(1:3): ;

          ANIN*1  <-  DIGEL1: ;
          ANIN*(2:4)  <-  DIGAR1(1:3): ;
          ANIN*6 + ANIN*12 + ANIN*24 + ANIN*(26:27) <-  DIGAR2(1:5): ;

          ANOUT*3  ->  DIGEL2: ;
          ANOUT*(5:8)  ->  DIGAR3(1:4): ;
          ANOUT*1 + ANOUT*2 + ANOUT*4 + ANOUT*(9:12)  -> DIGAR4(1:7): ;

       ITR(1) <- EREIG1: ;
       ITR(2) <- EREIG2: ;
       ITR(3) <- EREIG3: ;

       SIG(1) <- OVERFLOW: ;
       SIG(2) <- ENDFILE: ;
       SIG(3) <- ERROR: ;

PROBLEM ;
DECLARE (KONSOLE,LESER,DRUCKER,STANZER)  VAL DEVICE GLOBAL ;
DECLARE (FSCHR1,FSCHR2,FSCHR3) VAL DEVICE GLOBAL ;

DECLARE  BITEIN      VAL DEVICE GLOBAL ,
         BITAR0(4)   VAL DEVICE GLOBAL ,
         BITAR1(7)   VAL DEVICE GLOBAL ,
         BITAR2(3)   VAL DEVICE GLOBAL ,
         BITAR3(3)   VAL DEVICE GLOBAL ;

DECLARE  DIGEL1      VAL DEVICE GLOBAL ,
         DIGEL2      VAL DEVICE GLOBAL ,
         DIGAR1(3)   VAL DEVICE GLOBAL ,
         DIGAR2(5)   VAL DEVICE GLOBAL ,
         DIGAR3(4)   VAL DEVICE GLOBAL ,
         DIGAR4(7)   VAL DEVICE GLOBAL ;

DECLARE (EREIG1,EREIG2,EREIG3)  VAL INTERRUPT GLOBAL ;
DECLARE (ERROR,ENDFILE,OVERFLOW)  VAL SIGNAL GLOBAL ;
```

```
   DECLARE  STRING    CHARACTER(39) ;
   DECLARE  TEXT      CHARACTER(18) ;
   DECLARE (A,B,C,D)  FIXED ;
   DECLARE  I(3)      FIXED ,
            J(5)      FIXED ,
            K(7)      FIXED ;

   DECLARE  BSTR1        BIT(1) ,
            BFELD1(7)    BIT(1) ,
            BFELD2(3)    BIT(3) ,
            BFELD3(4)    BIT(16);

TO:TASK

   GET LESER    EDIT (A,B,C)   ((3)F(8)) ;
   PUT DRUCKER  EDIT (A,B,C)   ((3)F(8)) ;
   PUT STANZER  EDIT (A,B,C)   ((3)F(8)) ;

   PUT  FSCHR1  EDIT (TEXT)     (A(18)) ;
   GET  FSCHR2  EDIT (A,B,C)   ((3)F(6)) ;
   PUT  FSCHR3  EDIT     (D)    (F(7)) ;

   GET KONSOLE EDIT (STRING) (A(39)) ;
   PUT KONSOLE EDIT (TEXT)   (A(18)) ;

   MOVE  BITEIN      TO  (BSTR1) ;
   MOVE  BITAR0      TO  (BFELD3) ;
   MOVE  BITAR0      TO  (BFELD3(1)) ;
   MOVE  BITAR0(4)   TO  (BFELD3(2)) ;

   MOVE  (BFELD1(2))   TO  BITAR1(2)  OPT(ANTON) ;
   MOVE  (BFELD2(2))   TO  BITAR2(2)  OPT(ANTON) ;
   MOVE  (BFELD2(1))   TO  BITAR2     OPT(BERTA('1011'B1,'1101'B1)) ;
   MOVE  (BFELD2)      TO  BITAR3     OPT(ANTON,BERTA(1,12,123)) ;

   MOVE  DIGEL1      TO  (A)      OPT(LISTE) ;
   MOVE  DIGAR1      TO  (I)      OPT(LISTE) ;
   MOVE  DIGAR2(4)   TO  (J(4))   OPT(LISTE) ;
   MOVE  DIGEL1      TO  (K(5))   OPT(LISTE) ;

   MOVE  (B)      TO  DIGEL2      OPT(LISTE) ;
   MOVE  (I(2))   TO  DIGAR3(3)   OPT(LISTE) ;
   MOVE  (K)      TO  DIGAR4      OPT(LISTE) ;
   MOVE  (K(3))   TO  DIGEL2      OPT(LISTE) ;

   END ;

T1:TASK
   DECLARE  MF(10)  LABEL INITIAL (L1) ,  I  FIXED ;

      ON OVERFLOW GOTO L1 ;
      ON ERROR GOTO MF(I) ;
      ON ENDFILE SYSTEM ;
      DISABLE  EREIG1 ;
      ENABLE   EREIG2 ;
      TRIGGER  EREIG3 ;

      WHEN  EREIG2  ACTIVATE  TO ;
L1: ;
   END ;
   MODEND ;
```

7-8. KONKRETE FORM DES SYSTEMTEILS AUF DER ZIELMASCHINE ERLANGEN

7-8.1 VORGEGEBENE GERÄTEBEZEICHNUNGEN (in Erlangen)

	Gerät	Bezeichnung
Std. E/A-Geräte:	Konsol-Schreibmaschine	BSEA
	Lochkarten-Eingabe	LKEI
	Lochkarten-Ausgabe	LKAU
	Lochstreifen-Eingabe	LSEI
	Lochstreifen-Ausgabe	LSAU
	Schnelldrucker-Ausgabe	SDAU
	Magnetband-Ein/Ausgabe	MBEA
	Plattenspeicher-Ein/Ausgabe	PSEA
Bildschirm-Gerät:	Alpham.-Ein/Ausgabe-Statisch	SALP
	Graphische-Ein/Ausgabe-Statisch	SGRA
	Graphische-Ausgabe-Dynamisch	DGRA
	Graph.- und Alpham.-E/A-Statisch	SPIC
CAMAC-Geräte:	Camac-Controller	CAMC
	Crate	CRT
	Station	ST
Unterbrechungen:	Interrupt-Eingabe	ITRP
	Signale	SIGN

7-8.2 DIMENSIONIERUNG DER GERÄTE-AUSGÄNGE (Adreß-Bereiche)

Geräte-Bezeichnung	Kanäle	Bitausgänge pro Kanal	Bemerkungen
BSEA	1		
LKEI	1		
LKAU	1		
LSEI	1		
LSAU	1		
SDAU	1		
MBEA	1		
PSEA	1		
DGRA	1		
SALP	1		
SGRA	1		
SPIC	1		
CAMC	1... 6		
CRT	1...23, GNZ, ..., RGL		
ST	0...15		
ITRP		1	
SIGN		1	

7-8.3 SAMMLUNG TYPISCHER EINZELBEISPIELE ZUM SYSTEMTEIL

7-8.3.1 Reihenfolge der Beschreibung der Geräte

Im PEARL-Subset-Systemteil ist für die Beschreibung der Endstellen eine gewisse Reihenfolge vorgegeben.

1. Std. E/A-Geräte in beliebiger Reihenfolge
2. Eines von den Bildschirm-Geräten
3. CAMAC-Geräte-Konfiguration
4. Interrupt-Eingänge und Signale in beliebiger Reihenfolge.

7-8.3.2 Standard E/A-Geräte, Interrupts und Signals

```
BEISPIELE STD.-E/A:

        BSEA  <->  KONSOLE: ;
        LKEI  <-   LESER: ;
        LKAU   ->  STANZER: ;
        LSEI  <-   STREINGABE: ;
        LSAU   ->  STRAUSGABE: ;
        MBEA  <->  BAND: ;
        PSEA  <->  PLATTE: ;
        SDAU   ->  DRUCKER: ;


BEISPIELE BILDSCHIRMGERAET:

        SALP  <->  BILDS: ;


BEISPIELE INTERRUPT:

        ITRP(1) <- EREIG1: ;
        ITRP(2) <- EREIG2: ;
        ITRP(3) <- EREIG3: ;


BEISPIELE SIGNAL:

        SIGN(1) <- ERROR: ;
        SIGN(2) <- ENDFILE: ;
        SIGN(3) <- OVERFLOW: ;
```

7-8.3.3 Prozeß-Endstellen-Geräte

BEISPIEL 1:

```
DED*1 <- DIGWIN: ;
DAD*5 -> DIGOUT: ;
```

BEISPIEL 2:

```
DAD*1*3,1 -> BITAU1: ;
DAD*2*5,1 -> BITAU2: ;
DAD*4*1,1 -> BITAU3: ;
```

BEISPIEL 3:

```
DES*(6:9)      <- DIGAR(1:4): ;
DES*(3:5)*5,1 <- BITAR(1:3): ;
```

BEISPIEL 4:

```
DAS*(6:6)*13,1    +
DAS*(9:9)*13,1    +
DAS*(12:12)*13,1  +
DAS*(15:15)*13,1  +
DAS*(18:18)*13,1  +
DAS*(21:21)*13,1  +
DAS*(24:24)*13,1 -> BITAR(1:7): ;
```

ODER

```
DAS*6*14,1  +  DAS*9*14,1 + DAS*12*14,1 + DAS*15*14,1
DAS*18*14,1 + DAS*21*14,1 + DAS*24*14,1 -> BITAR(1:7);
```

BEISPIEL 5:

```
DAS*(6:6)    +
DAS*(9:9)    +
DAS*(12:12)  +
DAS*(15:15)  +
DAS*(18:18)  +
DAS*(21:21)  +
DAS*(24:24) -> DIGAR(1:7): ;
```

ODER

```
DAS*5   +  DAS* 8  +  DAS*11  +  DAS*14  +
DAS*17  +  DAS* 20 +  DAS*23  ->  DIGAR(1:7): ;
```

BEISPIEL 6:

```
DAD*(6:6)*13,1    +
DAD*(9:9)*13,1    +
DAD*(12:12)*13,1 -> BITAR1(1:3): ;

DAD*(7:7)*11,1    +
DAD*(8:8)*11,1    +
DAD*(11:11)*11,1 -> BITAR2(1:3): ;
```

ODER

```
DAD*6*14,1 +  DAD*9*14,1 + DAD*12*14,1 -> BITAR1(1:3): ;

DAD*7*12,1 + DAD*8*12,1  + DAD*12*11,1 -> BITAR2(1:3): ;
```

BEISPIEL 7:

```
DAD*(6:6) + DAD*(9:9) + DAD*(12:12) -> DIGAR1(1:3): ;
DAD*(7:7) + DAD*(8:8) + DAD*(11:11) -> DIGAR2(1:3): ;
```

ODER

```
DAD*3  + DAD*16 + DAD*10 -> DIGAR1(1:3): ;
DAD*14 + DAD*15 + DAD*13 -> DIGAR2(1:3): ;
```

BEISPIEL 8:

```
ABE*3  <-  ANIN1: ;
ABE*12 <-  ANIN2: ;
ABE*23 <-  ANIN3: ;
```

BEISPIEL 9:

```
ABE*(4:8)  <-  ANAR1(1:5): :
ABE*(10:17/3)  <-  ANAR2(1:3): :
ABE*3 + ABE*(6:6) + ABE*(10:14) + ABE*18  <-  ANAR3(1:8): ;
```

7-8.3.4 CAMAC-Gerät

```
BEISPIEL CAMAC-GERAET-ANSCHLUSS:

      CAMAC -> KPV: ;


BEISPIEL 1:

      CAMAC*2 <-> CRT(2) ;
         CRT(2)*(1:3) <-> ST(2:4) ;
            ST(2)*1   +
            ST(3)*1   +
            ST(4)*1 <-> DEVAR(1:3): ;

  ODER

      CAMAC*2 <-> CRT(2) ;
         CRT(2)*(1:3) <-> ST(2:4) ;
            ST(2)*(1:1)   +
            ST(3)*(1:1)   +
            ST(4)*(1:1) <-> DEVAR(1:3): ;

  ODER

      CAMAC*2 <-> CRT(2) ;
         CRT(2)*(1:1) +
         CRT(2)*(2:2) +
         CRT(2)*3     <->  ST(2:4) ;
            ST(2)*1   +
            ST(3)*1   +
            ST(4)*1 <-> DEVAR(1:3): ;


BEISPIEL 2:

      CAMAC*(2:4) <-> CRT(3:5) ;
         CRT(3)*1       +
         CRT(4)*(2:2)   +
         CRT(5)*(3:4) <-> ST(5:8) ;
            ST(5)*4       +
            ST(6)*4       +
            ST(6)*5       +
            ST(7)*6       +
            ST(8)*(6:7) <-> DEVAR(1:6): ;


BEISPIEL 3:

      CAMAC*6 <-> CRT(6) ;
         CRT(6)*(2:18/7) <-> ST(14:16) ;
            ST(14)*(3:3) +
            ST(15)*(2:4) +
            ST(16)*1     +
            ST(16)*4     +
            ST(16)*(6:6) <-> DEVAR(1:7): ;
```

BEISPIEL 4:

```
CA C*3 <-> CRT(7) ;
   CRT(7)*(4:7) <-> ST(21:24) ;
      ST(21)*1  +
      ST(22)*6  +
      ST(23)*9 <-> ARRAY1(1:3): ;
      ST(21)*2  +
      ST(22)*5  +
      ST(23)*(4:5) +
      ST(24)*(2:4) <-> ARRAY2(1:7): ;
```

BEISPIEL 5:

```
CA C*2 <-> CRT(7) ;
   CRT(7)*(11:15) <-> ST(25:29) ;
      ST(25)*(1:1) +
      ST(27)*(3:4) +
      ST(29)*(6:8) <-> ARRAY1(1:6): ;
      ST(26)*(2:3) +
      ST(27)*(5:7) +
      ST(28)*(2:5) <-> ARRAY2(1:9): ;
```

BEISPIEL 6:

```
CA C*5 <-> CRT(5) ;
   CRT(5)*(8:10) <-> ST(11:13) ;
      ST(11)*3     +
      ST(12)*(2:4) +
      ST(13)*(1:5/2) <-> DEVAR(1:7): ;
```

BEISPIEL 7:

```
CA C*(2:4) <-> CRT(3:5) ;
   CRT(3)*1     <-> ST(5) ;
   CRT(4)*(2:2) <-> ST(6:6) ;
   CRT(5)*(3:4) <-> ST(7:8) ;
      ST(5)*(5:6) +
      ST(6)*6     +
      ST(7)*7     +
      ST(8)*8    <->  DEVAR(1:5): ;
```

BEISPIEL 8:

```
CA C*3 <-> CRT(7) ;
   CRT(7)*(1:6/2) <-> ST(34:36) ;
      ST(34)*1  +
      ST(35)*3  +
      ST(36)*10 <-> ARRAY1(1:3): ;
   CRT(7)*(2:6/2) <-> ST(37:39) ;
      ST(37)*2  +
      ST(38)*3  +
      ST(39)*2 <-> ARRAY2(1:3): ;
```

BEISPIEL 9:

```
CA C*3 <-> CRT(3) ;
   CRT(3)*(6:8) <-> ST(5:7) ;
      ST(5)*(7:22/5) <-> DEVAR1(1:4): ;
      ST(6)*(7:21/5) <-> DEVAR2(1:3): ;
      ST(7)*(2:22/5) <-> DEVAR3(1:5): ;
```

BEISPIEL 10:

```
CA C*3 <-> CRT(3) ;
   CRT(3)*23 <-> ST(6) ;
      ST(6)*(1:4) <-> FELDA(1:4): ;
      ST(6)*(5:5) <-> VIRAR(1:1): ;
      ST(6)*(6:8) <-> FELDB(1:3): ;
      ST(6)*(11:15) <-> FELDC(1:5): ;
```

BEISPIEL 11:

```
CA C*2 <-> CRT(2) ;
   CRT(2)*(21:23) <-> ST(2:4) ;
   CRT(2)*(17:18) +
   CRT(2)*15     <->  ST(5:7) ;
      ST(2)*2    <-> ENDEL: ;
      ST(3)*(3:6) <-> ENDAR1(1:4): ;
      ST(4)*1       +
      ST(5)*(2:5)   +
      ST(6)*(6:9)   +
      ST(7)*3      <-> ENDAR2(1:10): ;
```

BEISPIEL 12:

```
CA C*1 <-> CRT(1) ;
   CRT(1)*GNZ <-> SONRE1: ;
   CRT(1)*TSI <-> SONRE2: ;
   CRT(1)*RGL <-> SONRE3: ;
   CRT(1)*1 <-> ST(1) ;
   CRT(1)*(2:4) <-> ST(2:4) ;
   CRT(1)*7 <-> ST(7) ;
      ST(1)*2 <-> ENDST1: ;
      ST(2)*1 <-> ENDST2: ;
      ST(3)*1 <-> ENDST3: ;
      ST(4)*1 <-> ENDST4: ;
      ST(7)*1 <-> ENDST5: ;
```

BEISPIEL 13:

```
CA C*1 <-> CRT(3) ;
   CRT(3)*2 <-> ST(2) ;
   CRT(3)*3 <-> ST(3) ;
   CRT(3)*(5:7) <-> ST(5:7) ;
   CRT(3)*9 <-> ST(9) ;
      ST(2)*4 <-> ENDST6: ;
      ST(3)*2  +
      ST(5)*2  +
      ST(7)*2  +
      ST(9)*2 <-> FELDD(1:4): ;
      ST(6)*6 <-> ENDST7: ;
```

BEISPIEL 14:

```
CA C*4 <-> CRT(4) ;
   CRT(4)*2 <-> ST(8) ;
      ST(8)*3 <-> DEVEL: ;
   CRT(4)*9 <-> ST(9) ;
      ST(9)*(2:4) <-> DEVAR1(1:3): ;
   CRT(4)*16 <-> ST(10) ;
      ST(10)*3 + ST(10)*(6:6) + ST(10)*8 <-> DEVAR2(1:3): ;
```

BEISPIEL 15:

```
CA C*3 <-> CRT(3) ;
   CRT(3)*10  -  ST(10) ;
      ST(10)*4 <-> ENDST1: ;
   CRT(3)*11 <-> ST(11) ;
      ST(11)*2 <-> ENDST2: ;
   CRT(3)*(12:14) <-> ST(12:14) ;
      ST(12)*2  +
      ST(13)*2  +
      ST(14)*2 <-> ARRAY(1:3): ;
   CRT(3)*15 <-> ST(15) ;
      ST(15)*6 <-> ENDST3: ;
```

BEISPIEL 16:

```
CA C*1 <-> CRT(1) ;
   CRT(1)*(1:4/3)  <->  ST(3:4) ;
      ST(3)*(0:1) + ST(4)*(0:0)  <->  DEVAR(1:3): ;
```

7-8.4 BEISPIEL GESAMTER SYSTEMTEIL (Erlangen)

```
MODULE BAERLG ;
SYSTEM ;
        BSEA  <->  KONSOLE: ;
        LSEI  <-   LESER: ;
        SDAU   ->  DRUCKER: ;

        SALP <-> BILDS: ;

       CA C*2 <-> CRT(2) ;
          CRT(2)*(1:3) <-> ST(2:4) ;
             ST(2)*1   +
             ST(3)*1   +
             ST(4)*1 <-> DEVAR(1:3): ;

       CA C*1 <-> CRT(1) ;
          CRT(1)*GNZ <-> SONRE1: ;
          CRT(1)*TSI <-> SONRE2: ;
          CRT(1)*RGL <-> SONRE3: ;

          DED*1 <- DIGWIN: ;
          DAD*5 -> DIGOUT: ;

          DES*(6:9)     <- DIGAR(1:4): ;
          DES*(3:5)*5,1 <- BITAR(1:3): ;

       ITRP(1) <- EREIG1: ;
       ITRP(2) <- EREIG2: ;
       ITRP(3) <- EREIG3: ;

       SIGN(1) <- ERROR: ;
       SIGN(2) <- ENDFILE: ;
       SIGN(3) <- OVERFLOW: ;

PROBLEM ;
DECLARE (KONSOLE,LESER,DRUCKER)  VAL DEVICE GLOBAL ;
DECLARE  BILDS  VAL DEVICE GLOBAL ;

DECLARE (SONRE1,SONRE2,SONRE3)  VAL DEVICE GLOBAL ;

DECLARE  DIGWIN      VAL DEVICE GLOBAL ,
         DIGOUT      VAL DEVICE GLOBAL ,
         DIGAR(4)    VAL DEVICE GLOBAL ;

DECLARE  BITAR3  (3)  VAL DEVICE GLOBAL ;

DECLARE (EREIG1,EREIG2,EREIG3)  VAL INTERRUPT GLOBAL ;
DECLARE (ERROR,ENDFILE,OVERFLOW)  VAL SIGNAL GLOBAL ;
```

```
T0:TASK
   DECLARE  EF(10) LABEL INITIAL (L1) , I FIXED ;

   ON OVERFLOW GOTO L1 ;
   ON ERROR GOTO EF(I) ;
   ON ENDFILE SYSTEM ;
   DISABLE EREIG1 ;
   ENABLE  EREIG2 ;
   TRIGGER EREIG3 ;
   WHEN EREIG2 ACTIVATE T1 ;
L1: ;
   END ;

T1:TASK
   DECLARE
   P VAL FILE ,
   U(10) FLOAT ,
   V FIXED ,
   W (12) FIXED ,
   I FIXED ,
   F (100) FIXED ,
   B BIT(10) ,
   BF (100) BIT(24) ,
   KF (10) VAL FIXED IDENT(0) ;

   MOVE TO DEVAR     OPT( F(8),Q(0) ) ;
   MOVE TO DEVAR(2)  OPT( F(9),Q(1) ) ;
   MOVE TO SONRE2 OPT (GAUGE('101011'B1)) ;

   MOVE DIGWIN TO (V) OPT (F(1)) ;
   MOVE BITAR(3) TO (BF(F(I+3)+2)) ;
   MOVE P TO (V,F(I),BF) ;
   MOVE P TO (V,W,U) ;
   MOVE (BF(I)) TO DIGOUT ;

   MOVE DIGAR(3) TO (W(I*3+F(10))) OPT ( F(1) , Q(9) ) ;
   MOVE (V+1*3+F(10)) TO P ADV (-2) ;
   MOVE (V,KF,BF) TO P ADV (I) ;
   MOVE (I,V+3,KF(I)+4,KF,BF(I) OR BF(1),BF,444,F(I)) TO P POS(000);

   DRAW BILDS EDIT (V,F(I),BF) (R(F1)) ;
   DRAW BILDS EDIT (555) ((1000)XA,(001)XR,(0)XS) ;
   DRAW BILDS EDIT (V+3,F(I)+10,KF,BF(1) EXOR BF(2),KF(01)) (R(F1));
F1:FORMAT (XA,MAP,YA,MAP) ;
   SEE  BILDS EDIT (V) ((4)XA,(0)MAP) ;
   SEE  BILDS EDIT (V,F(I),BF) (R(F2)) ;
F2:FORMAT (XS,XO,YI,BR,(3)MAP,FRAME,(5)OM) ;
   END ;
MODEND ;
```

KAPITEL 8

FEHLERMELDUNGEN

8-1. ALLGEMEINES

Der geringe Kernspeicherausbau an den Zielmaschinen bedingt einen in mehrere "Läufe" unterteilten Übersetzungsvorgang. Die während des Übersetzungsvorganges generierten Fehlermeldungen beziehen sich auf logisch in sich abgeschlossene Teilüberprüfungen des zu übersetzenden Quellcodes, wobei eine solche logische Einheit meist mehrere "Läufe" umfaßt. In Anlehnung an die Sprachwissenschaft, spez. die der mathematischen Linguistik, in der eine Sprachdefinition sich im wesentlichen auf die Unterteilung in

Syntax und
Semantik

abstützt, werden die Fehlermeldungen aus den einzelnen Läufen zu den folgenden logischen Einheiten zusammengefaßt:

formale Syntax
Verwendung von Namen
Semantik.

Bei der Übersetzung und Prüfung des PEARL-Programmes wird jeweils eine fehlerfreie "formale Syntax" vor der Prüfung auf die "Verwendung von Namen" bzw. eine fehlerfreie "Verwendung von Namen" für die Prüfung der Semantik" vorausgesetzt, andernfalls erfolgt ein Abbruch der Übersetzung.

8-2. FEHLERMELDUNGEN AUS DER ÜBERPRÜFUNG DER FORMALEN SYNTAX

Dazu gehören Fehler aus der Überprüfung der Lexikal-Analyse (Konstanten, Bezeichner, Trennsymbole, Schlüsselwörter, etc.) sowie Fehler der Grammatik (sog. "parsing") und auch die Überprüfung der Blockstruktur.

Ausgabeform:

Die Fehlermeldungen aus der formalen Syntaxprüfung bestehen aus Fehlertexten, denen die Quellzeile mit Angabe der Zeilen-Nummer und der markierten Fehlerstelle vorgestellt wird, wobei die Quellzeile von syntaktisch nicht bedeutsamen Blanks bereinigt wurde.

Fehlermeldungen, die direkt an die Auflistung des Quellprogramms anschließen, weisen auf eine irreguläre Beendigung des Analysevorgangs hin, hervorgerufen durch den Überlauf compilerinterner Listen oder durch eine zum Programmende nicht "aufgehende" Blockstruktur.

Allgemeine Ausgabeform bei "formalen Syntaxfehlern":

```
Zeile   quellprogrammzeile
        fehlermarkierung
        fehlertext
        fehlertext
            .
            .
            .
Zeile   quellprogrammzeile
        fehlermarkierung
        fehlertext
        fehlertext
            .
            .
            .
```

Beispiel 1:

```
  6  DCL  NAME1  FIXED, (NAME2, NAME3) FLOAT, (NAME4,) CHAR(1);
                                                  *
            NACH DEM KOMMA EINEN NAMEN ERWARTET
  7  DECLARE  NAME5  NAME6  FIXED;
                     *
            KEIN ZULAESSIGES ATTRIBUT AUF MODULEBENE ODER
            FEHLERHAFTE NAMENSLISTE
            FEHLERHAFTE DATEN-VEREINBARUNG AUF MODULEBENE
```

Interpretation zu Beispiel 1:

Die Fehlermeldung bezüglich Zeile 6 weist mit der Markierung auf eine fehlerhafte Namensliste hin.
In Zeile 7 führt eine fehlerhafte Namensliste die Analyse auf eine nicht mehr eindeutige Fehlererkennung, da entweder der zweite Name "NAME6" ein falsch geschriebenes oder ein nichtzulässiges Attribut oder aber es sich um eine fehlerhafte Namensliste handeln kann. In dem Beispiel für Zeile 7 ist ein weiterer Fehlertext ausgegeben worden, der eine allgemeinere Beschreibung dieses Fehlers angibt.

Beispiel 2:

```
 11  DCL  NAME  BIT(1);
          *
           ABSCHLUSS-SEMIKOLON AN DER MARKIERTEN STELLE ERWARTET
 88  AFTER  5MIN  RESUME
                       *
           ABSCHLUSS-SEMIKOLON AN DER MARKIERTEN STELLE ERWARTET
```

Interpretation zu Beispiel 2:

Beide Fehlermeldungen deuten auf einen Fehler durch ein fehlendes Abschluß-Semikolon hin, wobei in Zeile 88 die Fehlermarkierung an der Stelle steht, an der das fehlende Semikolon stehen müßte während in Zeile 11 erst ein Blick in die Zeile 10 das Fehlen eines Abschluß-Semikolons bestätigt.

```
10 DCL NAME FIXED
```

Diese Besonderheit kommt dadurch zustande, daß der Analyser in Zeile 11 nach dem Attribut FIXED aus Zeile 10 eine weitere mit einem Komma getrennte Deklaration erwartet, wobei in diesem Falle erst in Zeile 11 das Fehlen eines Semikolons eindeutig festliegt.

Beispiel 3:

```
84  END;
    *
    BLOCKSTRUKTUR GEHT NICHT AUF
    PEARL-ZEILE NICHT INTERPRETIERBAR, EVENTUELL VERURSACHT DURCH:
    FALSCHES SCHLUESSELWORT BZW. FALSCHE ABKUERZUNG ODER
    NICHT ZUGELASSENE PEARL-ANWEISUNG AUF MODULEBENE ODER
    FEHLERHAFTE BLOCKBILDUNG ODER
    MODUL-ABSCHLUSS FEHLT::= MODEND;

99  FI;
    *
    PEARL-ZEILE NICHT INTERPRETIERBAR, EVENTUELL VERURSACHT DURCH:
    FALSCHES SCHLUESSELWORT BZW. FALSCHE ABKUERZUNG ODER
    NICHT ZUGELASSENE PEARL-ANWEISUNG AUF MODULEBENE ODER
    FEHLERHAFTE BLOCKBILDUNG ODER
    MODUL-ABSCHLUSS FEHLT::= MODEND;
```

Interpretation zu Beispiel 3:

Beide Fehlermeldungen bezüglich Zeile 84 und Zeile 99 deuten auf einen Fehler der Blockstruktur hin. Die Ursache(n) für den Blockfehler muß in den vorhergehenden Zeilen gefunden werden.

Fehlermeldungen bei der formalen Syntaxanalyse, die unmittelbar im Anschluß an das Listing stehen können:

. PROGRAMMENDE ERREICHT, KEINEN MODULABSCHLUSS GEFUNDEN
  ODER BLOCKSTRUKTUR NICHT AUFGEGANGEN

. MEHR ALS 299 FEHLERZEILEN, ABBRUCH DER UEBERSETZUNG

. UEBERSETZUNG: ZU TIEF GESCHACHTELT, KELLER VOLL

Allgemeine Regeln für die Interpretation der Fehlermeldungen bei der formalen Syntaxanalyse:

Durch das freie Eingabeformat und die Nicht-Ausschließung von Schlüsselwörtern als Namen sowie die zulässige Einfügung von Kommentar beliebiger Länge an jeder Stelle, ist nicht in jedem Fehlerfall eine exakt genaue Markierung des Fehlers erreichbar (siehe Beispiel 2).

Aus diesem Grunde sollen die nachfolgenden Tips die Interpretation der Fehlermeldungen erleichtern und damit schneller zur Auffindung und Korrektur des Fehlers führen.

Regeln:

- In einer "syntaktischen Einheit", die von einem Semikolon zum jeweils nächstfolgenden reicht, können ein oder mehrere Fehler erkannt werden. Beziehen sich die Fehler auf den selben Ort, dann erfolgt nur eine einzige Markierung, sonst mehrere Markierungen entsprechend der Fehleranzahl.

- Die unter einer Markierung stehenden Fehlertexte müssen nicht unbedingt auf verschiedene Fehler hinweisen. Sie können durch Folgefehler zustandekommen.

- Der Fehler befindet sich meist an der markierten Stelle oder rechts von der Markierung bis zum nächsten Semikolon. Eine Ausnahme dieser Regel besteht, wenn das Abschlußsemikolon fehlt, dann kann die Markierung auf dem ersten Zeichen der nächsten Anweisung stehen (siehe Beispiel 2).

- Blockfehler können nicht genau markiert werden; die Ursache ist in den der Markierung vorangehenden Zeilen zu finden.

  Blockfehler-Meldungen können auch durch fehlerhafte Anweisungen speziell nach THEN und ELSE in der IF-Anweisung oder nach REPEAT in der Schleifen-Anweisung auftreten. Nach Verbesserung der fehlerhaften Anweisung, sind auch diese Blockfehler-Meldungen nicht mehr vorhanden.

- Nachstehend aufgeführte Fehler führen zu vielen Folgefehlern, wobei die Ursache meist schwer markierbar ist.

  . Fehlende Kommentar-Endeklammerung bewirkt, daß alle nachfolgenden Anweisungen (auch Blockbildungsanweisungen) bis zum nächsten */ überlesen werden.

  . Fehlendes Ende-Apostroph bei Charakter-Konstanten bewirkt, daß bis zum nächsten Apostroph (maximal 40 Zeichen) alles als "character-string" interpretiert wird.

  . PEARL-Quelltext in den Spalten 73-80 führt ebenfalls zu unübersichtlichen Folgefehlern.

8-3 FEHLERMELDUNGEN AUS DER ÜBERPRÜFUNG DER VERWENDUNG VON NAMEN UND DER SEMANTIK

Ausgabeform:

Die Fehlermeldungen bestehen aus Fehlertexten, zu denen die Zeilennummer angegeben wird, bei der der Fehler entdeckt wurde:

```
ZEILE zlnr      fehlertext
                fehlertext
                    .
                    .
ZEILE zlnr      fehlertext
          .
          .
          .
```

Bei der Interpretation der Fehlermeldung ist folgendes zu beachten:

- Die angegebene Zeilennummer muß nicht unbedingt auf die fehlerhafte Zeile verweisen. Der Fehler kann sich auch in einer der Zeilen vor der angegebenen befinden.

- Bei einer Zeilennummer können mehrere Fehlertexte erscheinen. Diese Texte müssen nicht unbedingt auf verschiedene Fehler hinweisen. Sie können durch Folgefehler zustande kommen.

Es werden maximal 66 Fehlertexte ausgegeben.

Beispiele:

```
ZEILE 5         NAME 'ANTON' NICHT DEKLARIERT

ZEILE 112       AUSDRUCK NACH 'IF' NICHT VOM TYP 'BIT(1)'
```

ANHANG I

METASYMBOLE ZUR SYNTAXBESCHREIBUNG

Metasymbole sind nicht Bestandteil des Zeichensatzes einer Sprache, sie dienen lediglich zur Sprachdefinition. In Anlehnung an die erweiterte Backus-Naur-Form (BNF) werden folgende Metasymbole zur Beschreibung der Syntax des PEARL-Subsets benutzt:

Meta-symbol	Bedeutung
::=	kann erzeugt werden aus
|	logisches ODER
[]	der geklammerte Teil ist nicht obligatorisch (auch Option genannt)
{}	Zusammenfassung für kunjunktiv oder disjunktiv verknüpfte Elemente
$\cdots$	das vor den hochgestellten Punkten aufgeführte Element kann wiederholt auftreten

Beispiel für die Beschreibung eines allgemeinen Ausdrucks in PEARL:

ausdruck::= [monadischer-operator] operand[{dyadischer-operator operand}$^{\cdots}$]

Bedeutung:

Ein gültiger Ausdruck in PEARL besteht aus einem nicht obligatorischen monadischen Operator gefolgt von einem Operanden und eventuell einer beliebigen Wiederholung der Elementfolge dyadischer-Operator und Operand.

ANHANG II

VERZEICHNIS DER SCHLÜSSELWÖRTER
(ohne Systemteil-Gerätebezeichner, Operatoren und Trennzeichen)

A
ACTIVATE
ADV
AFTER
ALL
ALPHA
AT
B
B1
B3
BASIC
BEGIN
BIT
BR
BY
CALL
CHAR
CHARACTER
CLOCK
CLOSE
CONTINUE
CREATE
D
DCL
DECLARE
DELETE
DEVICE
DIR
DISABLE
DRAW
DURA
DURATION
DURING
E
EDIT
ELSE
ENABLE
END
ENTRY
F
FILE
FIXED
FLOAT
FOR
FRAME
FROM
GET
GLOBAL
GOTO
IDENT
IDENTICAL
IF
INIT
INITIAL
INPUT
INTERRUPT
IP
IRUPT
LABEL
MAP
MODEND
MODULE
MOVE
OM
ON
OPEN
OPT
OUTPUT
PAGE
POS
PREVENT
PRIORITY
PROBLEM
PROC
PROCEDURE
PUT
REENTRANT
RELEASE
REPEAT
R
REQUEST
RESIDENT
RESUME
RETURN
RETURNS
SEE
SEMA
SEND
SEQ
SIGNAL
SKIP
SUSPEND
SYSTEM
T
TASK
TERMINATE
THEN
TITLE
TO
TRIGGER
UNTIL
UPDATE
UPON
VAL
WAIT
WHEN
WHILE
X
XA
XI
XO
XP
XR
XS
YI
YO
YR
YS

ANHANG III

ZUSAMMENSTELLUNG ALLER ZUGELASSENEN VEREINBARUNGEN

1. PROBLEMDATEN

1.1 Deklaration von lokalen Variablen ohne Vorbesetzung auf Modul-, Task-, Prozedur-, Blockebene:

```
DECLARE {name|(namenliste)} {FIXED | FLOAT | BIT(n) | CHARACTER(n)|
        CLOCK| DURATION};
```

1.2 Deklaration von lokalen Variablen mit Vorbesetzung auf Modul-, Task-, Prozedur-, Blockebene:

```
DECLARE {name|(namenliste)} {FIXED | FLOAT | BIT(n) | CHARACTER(n)|
        CLOCK | DURATION} INITIAL  ([+|-] konstante);
```

1.3 Deklaration von globalen Variablen ohne Vorbesetzung auf Modulebene:

```
DECLARE {name|(namenliste)} {FIXED | FLOAT | BIT(n) | CHARACTER(n)|
        CLOCK | DURATION} GLOBAL;
```

1.4 Deklaration von globalen Variablen mit Vorbesetzung auf Modulebene:

```
DECLARE {name|(namenliste)} { FIXED | FLOAT | BIT(n) | CHARACTER(n)|
        CLOCK | DURATION} GLOBAL  INITIAL ([+|-] konstante);
```

1.5 Deklaration von lokalen Konstantennamen auf Modul-, Task-, Prozedur-, Blockebene:

```
DECLARE {name|(namenliste)}  VAL { FIXED | FLOAT | BIT(n)|
        CHARACTER(n) | CLOCK | DURATION}  IDENTICAL ([+|-]
        konstante);
```

1.6 Deklaration von globalen Konstantennamen auf Modulebene:

```
DECLARE {name|(namenliste)} VAL {FIXED | FLOAT | BIT(n) | CHARACTER(n)|
        CLOCK| DURATION } GLOBAL ([+|-] konstante);
```

1.7 Spezifikation von globalen Konstantennamen auf Modulebene:

```
DECLARE {name|(namenliste)} VAL {FIXED | FLOAT | BIT(n) | CHARACTER(n)|
        CLOCK| DURATION } GLOBAL;
```

1.8 Deklaration von lokalen Feldern ohne Vorbesetzung auf Modul-, Task-, Prozedur-, Blockebene:

DECLARE {name|(namenliste)} ([1:]g_1 [,[1:]g_2][,[1:]g_3]) {FIXED|FLOAT BIT(n) | CHARACTER(n)};

1.9 Deklaration von lokalen Feldern mit Vorbesetzung auf Modul-, Task-, Prozedur-, Blockebene:

DECLARE {name|(namenliste)} ([1:]g_1 [,[1:]g_2][,[1:]g_3]) {FIXED|FLOAT| BIT(n) | CHARACTER(n)} INITIAL ([+|-] konstante [, [+]-] konstante]...);

1.10 Deklaration von globalen Feldern ohne Vorbesetzung auf Modulebene:

DECLARE {name|(namneliste)} ([1:]g_1 [,[1:]g_2] [,[1:]g_3]) {FIXED|FLOAT| BIT(n) | CHARACTER(n)} GLOBAL;

1.11 Deklaration von globalen Feldern mit Vorbesetzung auf Modulebene:

DECLARE {name|(namenliste)} ([1:]g_1 [,[1:]g_2][,[1:]g_3]) {FIXED|FLOAT| BIT(n) | CHARACTER(n)} GLOBAL INITIAL ([+|-] konstante [, [+|-] konstante]...);

1.12 Deklaration von lokalen Konstantenfeldern auf Modul-, Task-, Prozedur-, Blockebene:

DECLARE {name|(namenliste)} ([1:]g_1[,[1:]g_2][,[1:]g_3]) VAL {FIXED|FLOAT| BIT(n) | CHARACTER(n)} IDENTICAL ([+|-] konstante [, [+|-] konstante]...);

1.13 Deklaration von globalen Konstantenfeldern auf Modulebene:

DECLARE {name|(namenliste)} ([1:]g_1[,[1:]g_2][,[1:]g_3]) VAL {FIXED FLOAT| BIT(n) | CHARACTER(n)} GLOBAL IDENTICAL ([+|-] konstante [, [+|-] konstante]...);

1.14 Spezifikation von globalen Konstantenfeldern auf Modulebene:

DECLARE {name|(namenliste)} ([1:]g_1 [,[1:]g_2][,[1:]g_3]) VAL {FIXED| FLOAT| BIT(n) | CHARACTER(n)} GLOBAL;

2. PROGRAMM-STEUERUNGSDATEN

2.1 Deklaration von lokalen Anweisungs- und Formatmarken auf Task-, Prozedur-, Blockebene
markenname:

2.2 Deklaration von lokalen Markenfeldern auf Task-, Prozedur-, Blockebene:

DECLARE {name|(namenliste)} ([1:]g_1) LABEL INITIAL (anweisungsmarkenname [, anweisungsmarkenname]...);

2.3 Deklaration von lokalen Semaphor-Variablen ohne Vorbesetzung auf Modulebene:

DECLARE {name|(namenliste)} SEMA;

2.4 Deklaration von lokalen Semaphor-Variablen mit Vorbesetzung auf Modulebene:

DECLARE {name|(namenliste)} SEMA INITIAL (ganze-zahl);

2.5 Deklaration von globalen Semaphor-Variablen ohne Vorbesetzung auf Modulebene:

DECLARE {name|(namenliste)} SEMA GLOBAL;

2.6 Deklaration von globalen Semaphor-Variablen mit Vorbesetzung auf Modulebene:

DECLARE {name|(namenliste)} SEMA GLOBAL INITIAL (ganze-zahl);

2.7 Spezifikation von Interrupts auf Modulebene:

DECLARE {name|(namenliste)} VAL INTERRUPT GLOBAL;

2.8 Spezifikation von Signals auf Modulebene:

DECLARE {name|(namenliste)} VAL SIGNAL GLOBAL;

3. GERÄTE

3.1 Spezifikation von Geräten auf Modulebene:

DECLARE {name|(namenliste)} VAL DEVICE GLOBAL;

3.2 Spezifikationen von Gerätefeldern auf Modulebene:

DECLARE {name|(namenliste)} ([1:]g_1) VAL DEVICE GLOBAL;

4. FILES

4.1 Deklaration von lokalen Files auf Modul-, Task-, Prozedur-, Blockebene:

DECLARE {name|(namenliste)} VAL FILE;

4.2 Deklaration von globalen Files auf Modulebene:

DECLARE {name|(namenliste)} VAL FILE GLOBAL;

5. TASKS

5.1 Deklaration von lokalen Tasks auf Modulebene:

eingangsname: TASK [RESIDENT] task-segment END;

5.2 Deklaration von globalen Tasks auf Modulebene:

eingangsname: TASK GLOBAL [RESIDENT] task-segment END;

5.3 Spezifikation von Tasks auf Modulebene:

eingangsname: TASK GLOBAL [RESIDENT];

6. PROZEDUREN

6.1 Deklaration von lokalen Prozeduren auf Modul-, Task-, Prozedur-, Blockebene:

eingangsname: PROCEDURE [(parameterliste)] [rückkehrattribute]
[REENTRANT] prozedurekörper END;

6.2 Deklaration von globalen Prozeduren auf Modulebene:

eingangsname: PROCEDURE [(parameterliste)] [rückkehrattribute]
GLOBAL [REENTRANT] prozedurkörper END;

6.3 Spezifikation von Prozeduren auf Modulebene:

DECLARE {name|(namenliste)} ENTRY-attribut [rückkehrattribute]
GLOBAL [REENTRANT];

Programmieranleitung für das ASME-PEARL-SUBSET/1

Gesellschaft für Kernforschung mbH, Karlsruhe
PDV-Bericht KFK-PDV 1oo, November 1976
181 Seiten, 2 Abb.

Es wird eine Programmiersprache der mittleren Ebene beschrieben, die es erlaubt, Struktur, Algorithmen, Zeitverhalten und Ein-/Ausgabe von Echtzeitprogrammen zu formulieren. Als wichtige Eigenschaft bietet die Sprache dem Anwender neben den üblichen Unterstützungen wie Blockstruktur, Schleifensteuerung usw. durch das Sprachmittel "Task" die Möglichkeit zur Programmierung zeitlich parallel ablaufender Vorgänge. Mit einer Reihe von Anweisungen kann die zeitliche Koordinierung der Tasks beschrieben werden (sog. "Tasking").

Außerdem wird auf Sprachebene in einem "Systemteil" die Anschlußstruktur von Standard E/A-Geräten und Prozeßsteuerungsgeräten an den Rechner beschrieben, wodurch sich die gesamte Information zur Steuerung von Prozessen auf Programmebene befindet.

Die Programmiersprache ist für den Automatisierungsingenieur oder Experimentator mit Programmiererfahrung bestimmt.

Der Bericht beschreibt Syntax und Semantik der einzelnen PEARL-Anweisungen und gibt Beispiele dazu.

Programmieranleitung für das ASME-PEARL-SUBSET/1

Gesellschaft für Kernforschung mbH, Karlsruhe
PDV-Bericht KFK-PDV 1oo, November 1976
181 Seiten, 2 Abb.

Es wird eine Programmiersprache der mittleren Ebene beschrieben, die es erlaubt, Struktur, Algorithmen, Zeitverhalten und Ein-/Ausgabe von Echtzeitprogrammen zu formulieren. Als wichtige Eigenschaft bietet die Sprache dem Anwender neben den üblichen Unterstützungen wie Blockstruktur, Schleifensteuerung usw. durch das Sprachmittel "Task" die Möglichkeit zur Programmierung zeitlich parallel ablaufender Vorgänge. Mit einer Reihe von Anweisungen kann die zeitliche Koordinierung der Tasks beschrieben werden (sog. "Tasking").

Außerdem wird auf Sprachebene in einem "Systemteil" die Anschlußstruktur von Standard E/A-Geräten und Prozeßsteuerungsgeräten an den Rechner beschrieben, wodurch sich die gesamte Information zur Steuerung von Prozessen auf Programmebene befindet.

Die Programmiersprache ist für den Automatisierungsingenieur oder Experimentator mit Programmiererfahrung bestimmt.

Der Bericht beschreibt Syntax und Semantik der einzelnen PEARL-Anweisungen und gibt Beispiele dazu.

Programmers Handbook for ASME-PEARL-SUBSET/1

Gesellschaft für Kernforschung mbH, Karlsruhe
PDV-report KFK-PDV 1oo, November 1976
181 pages, 2 figs.

A middle-level programming language is described which allows the formulation of the structure, the algorithms, time behaviour and I/O of realtime programs.

Besides the usual programming aids such as blockstructure, loop-control etc. the language offers the user the important feature of the "Task", which gives the ability to program processes that are parallel in time.

There is a series of statements which give the time-correlation of those tasks, the so called "Tasking". Moreover the structure of connections of standard-I/O-devices and process-control-devices is described by the language, so that all information for the process-control is located in the program.

The language is destined for the process control engineer or experimentor with some programming experience. This report describes the syntax and semantics of the various PEARL-statements and gives examples of them.

Programmers Handbook for ASME-PEARL-SUBSET/1

Gesellschaft für Kernforschung mbH, Karlsruhe
PDV-report KFK-PDV 1oo, November 1976
181 pages, 2 figs.

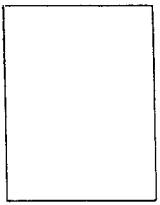
A middle-level programming language is described which allows the formulation of the structure, the algorithms, time behaviour and I/O of realtime programs.

Besides the usual programming aids such as blockstructure, loop-control etc. the language offers the user the important feature of the "Task", which gives the ability to program processes that are parallel in time.

There is a series of statements which give the time-correlation of those tasks, the so called "Tasking". Moreover the structure of connections of standard-I/O-devices and process-control-devices is described by the language, so that all information for the process-control is located in the program.

The language is destined for the process control engineer or experimentor with some programming experience. This report describes the syntax and semantics of the various PEARL-statements and gives examples of them.

Absender:



An die
Gesellschaft für
Kernforschung mbH
Projekt PDV

75 Karlsruhe
Postfach 3640

Lieber Leser !
Da wir daran interessiert sind zu erfahren, wie die KFK-PDV-Berichte in der Fachwelt beurteilt werden, möchten wir Sie bitten, uns Ihr Urteil durch ankreuzen der entsprechenden Antwort mitzuteilen.
Herzlichen Dank !

Betr.: KFK-PDV-[.....]

	ja	ja mit Einschränkung	nein
1) Wir werden FE-Ergebnisse aus dem Bericht verwenden	☐	☐	☐
2) Wir entnehmen dem Bericht neue Erkenntnisse	☐	☐	☐
3) Der Bericht enthält für uns nützliche Informationen	☐	☐	☐
4) Der Bericht ist interessant, aber ohne Nutzen für uns	☐	☐	☐
5) Der Bericht ist verständlich	☐	☐	☐
6) Diese Forschungsrichtung sollte weiter verfolgt werden !	☐	☐	☐

Datum: Unterschrift: Firma: