PDV Berichte

Projekt Prozeßlenkung mit DV-Anlagen

KFK - PDV 55

Universelles PEARL-Betriebssystem

P. J. Brunner, H. Bösmann, A. Tarabout, W. Werum
Entwicklungsbüro Werum, Lüneburg-Ebensberg

Januar 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 des 2. DV-Programms 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
75 Karlsruhe 1, Postfach 3640
Printed in Western-Germany

KFK-PDV 55

Projekt Prozeßlenkung mit DV-Anlagen
Forschungsbericht KFK-PDV 55

Universelles PEARL-Betriebssystem

P.J. Brunner
H. Bösmann
A. Tarabout
W. Werum

Entwicklungsbüro Werum, Lüneburg-Ebensberg

292 Seiten
3 Abbildungen

Januar 1976

I n h a l t

I n h a l t

I n h a l t

I n h a l t

Inhalt

I n h a l t

I n h a l t

0. EINLEITUNG

0.1 Ziel der Entwicklung, Eigenschaften des Betriebssystems

Die Entwicklung des "Allgemeinen PEARL-Betriebssystems", die im folgenden Bericht dargestellt wird, erfolgte im Rahmen des Projekts PDV der Gesellschaft für Kernforschung unter Förderung der Bundesregierung im Hause Entwicklungsbüro Wulf Werum (Lüneburg-Ebensberg).

Ziel der Entwicklung ist die Bereitstellung eines allgemeinen PEARL-Betriebssystems für Mittelklasserechner mit einer Zentraleinheit (Einprozessoranlage) und Hintergrundspeicher. Der größte Teil des Betriebssystems kann für unterschiedliche Rechner übernommen werden. Der Aufwand zur Anpassung des rechnerunabhängigen Betriebssystems für einen speziellen Prozeßrechner beträgt höchstens zwei Mannjahre, d.h. der Aufwand zum Einsatz weiterer PEARL-Betriebssysteme kann auf etwa ein Drittel gegenüber speziellen rechnerabhängigen Entwicklungen reduziert werden.

Das Betriebssystem ist leicht erweiterbar und modifizierbar, wodurch sichergestellt ist, daß für die Implementierung weiterer Prozeßautomationssprachen (z.B. international genormten) ein Großteil der Entwicklung übernommen werden kann.

Das Betriebssystem umfaßt alle durch PEARL bestimmten Systemdienste:

- die Unterbrechungsauswertung,
- die Einplanung von Steueranweisungen,
- die Prozeß-Verwaltung ("task-control"),
- die Prozeßdatenübertragung,
- die Datei-Verwaltung,
- die Arbeitsspeicher-Verwaltung und
- die Hintergrundspeicher-Verwaltung.

Ausgeschlossen von der Entwicklung sind die von diesen Programmen angesprochenen rechnerabhängigen Geräte-Treiber und die Programme für die formatisierte und für die graphische Ein-/Ausgabe.

Das Betriebssystem ist in der höheren Sprache GBL1 (PL/1-Subset der GfK) programmiert und wird nach der Übersetzung durch dessen Compiler in der maschinenunabhängigen Zwischensprache IL1 vorliegen. Der Anschluß eines Optimizers nach dem Compilier-Vorgang, der ebenfalls im Entwicklungsbüro Werum entwickelt wurde, ist möglich.

Die Schnittstellen des rechnerunabhängigen Betriebssystems zum Anwenderprogramm und zu rechnerspezifischen Teilen sind klar festgelegt.

Das Betriebssystem ist auf einer IBM-Anlage in simulierter Umgebung ausgetestet worden. Damit ist u.a. auch angedeutet, daß das System ohne großen Aufwand zugleich als Hilfsmittel für das Erstellen von Prozeßmodellen mittels Simulation eingesetzt werden kann.

Es ist vorausgesetzt, daß der Leser dieses Berichtes die Sprache PEARL (PDV-Bericht, KFK-PVD1 der Gesellschaft für Kernforschung) beherrscht.

0.2 Einteilung des Berichts

Der folgende Bericht enthält drei Teile:
Im ersten Teil wird der Kern des Betriebssystems dargestellt. Damit werden die Programme zur Unterbrechungsauswertung, Einplanung von Steueranweisungen, Prozeß-Verwaltung beschrieben.

Im zweiten Teil werden die Programme zur Prozeßdatenübertragung und Datei-Verwaltung beschrieben.

Im dritten Teil werden die Programme zur Speicherplatzverwaltung (Arbeitsspeicher und Hintergrundspeicher) beschrieben.

Zum Abschluß des Berichts folgen einige Anhänge:
Anhang A enthält eine Liste der aufgeführten Systemdienste.
Anhang B gibt einige Erläuterungen über die Programmstruktur des PEARL-Betriebssystems.

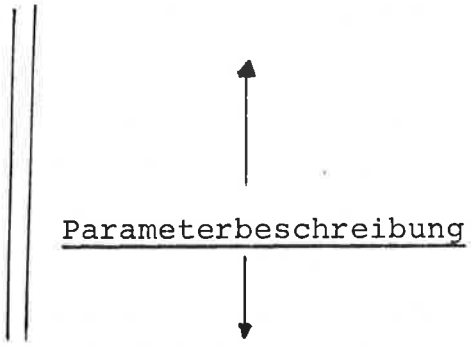
Anhang C beschreibt das Testsystem, das das Betriebssystem und seine Testumgebung bilden.
Anhang D enthält einen Index der wichtigsten Marken und Datenstrukturen des PEARL-Betriebssystems.

0.3 Darstellungsform

Im folgenden Bericht wird beschrieben, wie die Systemfunktionen vom übersetzten Anwenderprogramm aufgerufen werden und wie sie ihren Auftrag abwickeln. Die Art des Aufrufs der Betriebssystemfunktionen ist derart dargestellt, daß für die Anpassung des Betriebssystems an spezielle Rechner eindeutig erkennbar ist, wie die Befehlsfolge, die den Aufruf ausführt, organisiert werden soll.

Der Aufruf einer Funktion mit Parametern wird so dargestellt:

CALL Funktionseingangsname



Wo die Parameterbeschreibung abgelegt ist, hängt von der Implementation ab. Die Parameteradressen (bzw. Werte) können sowohl nach dem Funktionsaufruf im Objektcode liegen, als auch vor dem Aufruf in einem systembekannten Gebiet abgelegt werden oder auch durch einen systembekannten Zeiger (evtl. im Accumulator) angezeigt sein, usw. Es wird nur beschrieben, welche Parameter anzugeben sind.

Adressen können direkt oder (mehrfach) indirekt angegeben werden.

Sie werden in der Parameterbeschreibung mit

ADDR Name oder Beschreibung der adressierten Größe

gekennzeichnet. In speziellen Fällen ist anstatt der Adresse einer Größe eine "leere Adresse" anzugeben. Sie wird in der Parameterbeschreibung mit NULL gekennzeichnet.

Wenn eine Zahl, die dem Compiler bekannt ist, als Parameter anzugeben ist, wird sie in der Parameterbeschreibung mit

VALUE Wert oder Beschreibung der Zahl

gekennzeichnet. Wenn aber die direkte Übergabe einer Zahl nicht möglich ist, darf die Adresse ihrer Ablagestelle als Parameter übergeben werden.

Einige Betriebssystemfunktionen werden mit einer Liste von Parametern gleicher Art aufgerufen, deren Anzahl unbestimmt ist. Wie eine solche Liste organisiert wird, hängt von der Implementation ab. Man kann davon ausgehen, daß die Anzahl der Parameter in der Liste angegeben wird oder daß die Liste durch ein Kennzeichen (z.B. die Zahl 0) abgeschlossen ist. Unabhängig davon wird hier eine Parameterliste durch

Liste von:
Beschreibung der Parameter in der Liste

gekennzeichnet.

In diesem Bericht wird eine bedingte Maßnahme bzw. Tätigkeit (d.h. die Maßnahme bzw. Tätigkeit ist nur unter bestimmten Bedingungen anzugeben bzw. auszuführen) zwischen eckigen Klammern [ ] angegeben.
Alternativen, von denen unbedingt eine zu wählen ist, werden zwischen {und } angegeben.
Drei Punkte ... nach [ ] oder { } deuten eine Folge von eingeklammerten Klausen (Maßnahmen, Tätigkeiten,..) an.

Beispiele:

[A]... steht für eine (evtl.leere) Folge von A mit unbestimmter Länge.

$\left\{\begin{matrix}A\\B\end{matrix}\right\}$... steht für eine beliebige (nicht leere) Folge von A oder B.

Ein Sprung auf eine bestimmte Marke oder über eine Markenvariable wird mit

GOTO [*] Marke

dargestellt. Der Stern * kennzeichnet den Fall, in dem indirekt über die Marke gesprungen wird.

Die Form a.b steht für

"das Element b aus der Datenstruktur a".

Die Form p -> x steht für

"die Größe x, deren Adresse in p steht (durch p angezeigt)".

Aus diesen zwei Formen ist die Form p -> a.b hergeleitet; sie steht für

"das Element b aus der Datenstruktur a, deren Adresse in p steht".

Die Form ADDR(x) steht für die Adresse von x.

1. DER KERN DES BETRIEBSSYSTEMS

Die Module des Kerns des Betriebssystems lösen folgende Aufgaben:

- das Bereitstellen von Verwaltungsstrukturen (PKS) beim Eintritt in den Block für alle darin vereinbarten Prozesse (TASK, TASKNAME-Vereinbarung),

- das Einplanen von Prozeßsteueranweisungen bei Überlaufen von Prozeßsteueranweisungen mit Startbedingung (*schedule*),

- die Ausführung von Prozeßsteueranweisungen zum Starten von Prozessen (ACTIVATE),
  Verzögern von Prozessen (SUSPEND, RESUME),
  Fortsetzen von Prozessen (CONTINUE, RESUME),
  Beenden von Prozessen (TERMINATE oder Erreichen des normalen Endes eines Prozesses) und
  Ausplanen von Prozeßsteueranweisungen (PREVENT)

- die Kopplung der ON-Anweisung an Signale.

- das Registrieren von Meldungen und die Einleitung aller zugeordneten Anweisung (ON-Anweisungen und eingeplante Prozeßsteueranweisungen),

- die Synchronisierung der Prozeßabläufe über Synchronisiervariable (SEMA,BOLT) mit Hilfe von Synchronisieranweisungen zum Sperren (REQUEST, RESERVE, ENTER) und
  Freigeben (RELEASE, FREE, LEAVE).

Bevor auf die einzelnen Funktionen genauer eingegangen wird, werden die von den Betriebssystem-Funktionen angesprochenen Datenstrukturen beschrieben.

1.1 DATENSTRUKTUREN

Die Datenstruktur zur Steuerung des Ablaufs von PEARL-Programmen ist in ihren wesentlichen Teilen in Bild 1.1 insoweit dargestellt, wie dies zum Verständnis der später beschriebenen Betriebssystem-Funktionen benötigt wird.

Mit dem Symbol $\rightarrow\!\!\!\!\rightarrow$ (Vielfach-Pfeil) wird angedeutet, daß an der angezeigten Stelle eine Kette beginnt; so soll mit dem Vielfach-Pfeil, der von einer SEMA-Variablen auf die Prozeßkontrollsätze (PKS) zeigt, angedeutet werden, daß die Kontrollsätze von Prozessen, die durch die gleiche SEMA-Variable gesperrt sind, miteinander verzeigert sind. Mit dem Bild wird nicht angegeben, wie die Verzeigerung erfolgt (Verzeigerung in einer Richtung oder zwei Richtungen).

Die Datenstrukturen werden untergliedert in

- Meldungsbeschreibungen (Ereignisvariable ER, Zeitkontrollsätze ZK),
- Einplankontrollsätze EPS,
- Prozeßkontrollsätze PKS,
- Synchronisiervariable (SEMA-, BOLT-Variable).

Durch die *Meldungsbeschreibungen* werden die Reaktionen auf alle (extern oder intern erzeugten) Meldungen spezifiziert. Meldungen werden untergliedert in

1. Meldungen für die Zeitsteuerung (Dauer-, Zeitangabe). Sie werden über die *Zeitkontrollsätze* ZK verwaltet.

2. Meldungen aus einem Prozeß (technischer Prozeß = externer Prozeß, Rechenprozeß = interner Prozeß). Sie werden über *Ereignisvariable* ER verwaltet.

   Die Ereignisvariablen werden durch Auswertung der INTERRUPT-Anschlüsse im Systemteil des PEARL-Programms erzeugt.

Wenn eine Meldung im PROBLEM-Teil als SIGNAL spezifiziert ist, wird an die Ereignisvariable ER ein SIGNAL-Beschreibungselement SG angehängt.

Die *Einplankontrollsätze* (EPS) werden durch Auswertung der Startbedingungen (SCHEDULES) aufgebaut; sie werden an Ereignisvariable bzw. Zeitkontrollsätze gekettet.

Einplankontrollsätze, die sich auf dieselbe Meldung beziehen, sind untereinander verkettet. Diese Ketten bilden die Warteschlangen für die Zeit- und Dauerkontrolle und für die Reaktion auf Prozeßmeldungen.

Einplankontrollsätze eines Prozesses sind untereinander verkettet; auf den Beginn dieser Kette weist ein Zeiger im Prozeßkontrollsatz (PKS).
Zur Ansprache des zu steuernden Prozesses wird vom EPS auf den zugeordneten PKS hingewiesen.

Prozesse können durch explizites Sperren mit *Synchronisiervariablen* (SEMA- oder BOLT-Variable) ihre Abläufe miteinander synchronisieren. Falls mehrere Prozesse auf die Freigabe derselben SEMA- oder BOLT-Variable warten, sind ihre Prozeßkontrollsätze untereinander und mit der Synchronisiervariablen verkettet.

Die *Prozeßkontrollsätze* (PKS) werden eingerichtet, sobald beim Programmablauf eine TASK-Vereinbarung überlaufen wird. Zur Steuerung der Rechenzeitvergabe werden alle PKS in eine Prioritätskette eingeordnet.

Anhand des Prozeßkontrollsatzes können alle Angaben gewonnen werden über:

- die dem Prozeß zugeordneten Betriebsmittel (Arbeitsspeicher, Hintergrundspeicher, Geräte),
- den Stand der Prozeßbearbeitung (Befehlszähler, Registerinhalte, Prozeßstatus abhängig von Prozeßsteueranweisungen und Sperr- und Freigabeanweisungen),

. die Einbettung in die Prozeßhierarchie (Zeiger auf die "Mastertask", Kette der unmittelbaren "Subtasks", Prioritätsangabe) und

. die Abhängigkeit des Prozesses von Meldungen (Verkettung mit EPS').

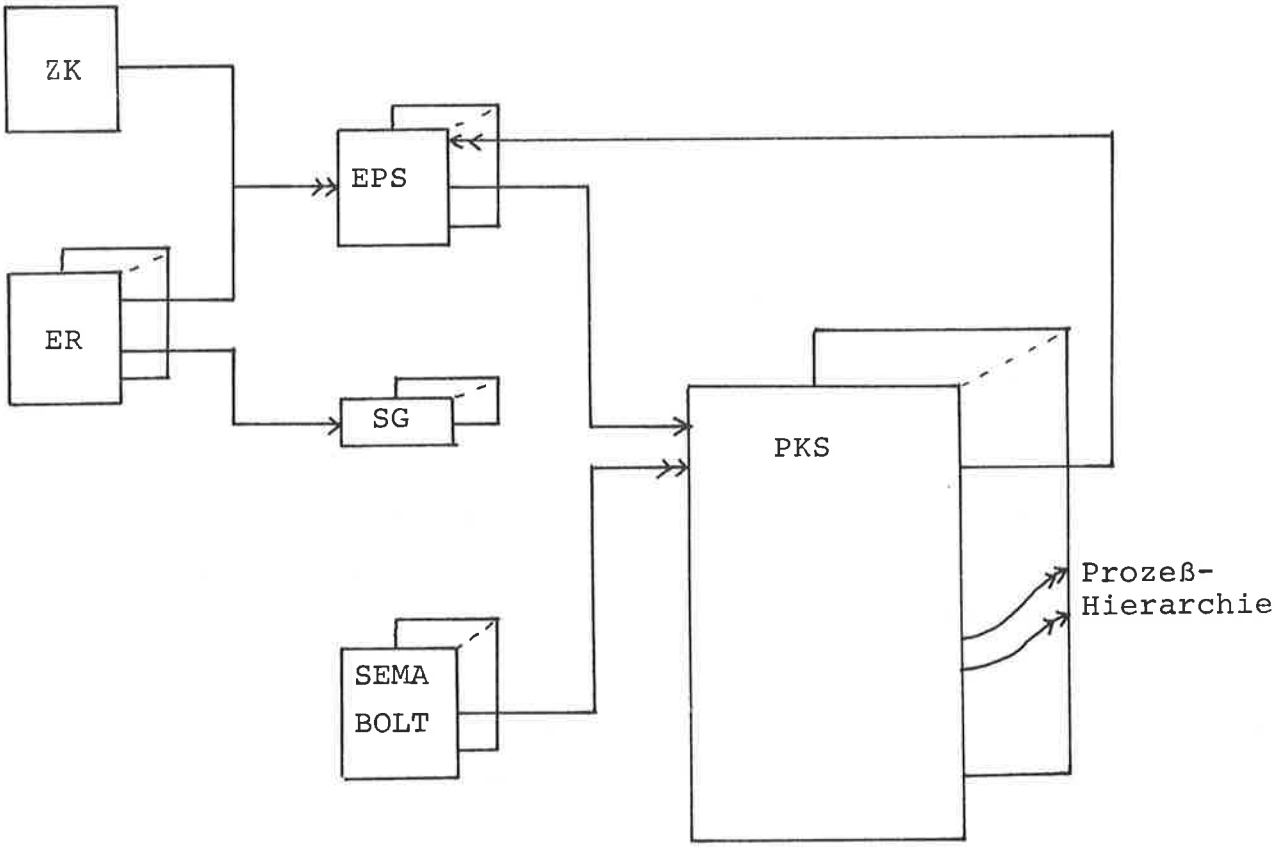


Bild 1.1: Datenstrukturen

1.2 PROZESS-STATUS, STATUSÜBERGÄNGE

Zur simultanen (quasi-parallelen) Abwicklung mehrerer Teilaufgaben organisiert der PEARL-Programmierer die Gesamtaufgabe in mehrere Prozesse (Tasks).

Bei Blockeintritt können Prozesse vereinbart werden (sie werden *bekannt*); bei Blockaustritt können Prozesse *unbekannt* werden.
Zur Steuerung des Prozeßablaufs dienen die Prozeßsteueranweisungen und Synchronisieranweisungen; die genaue Beschreibung der Systemfunktionen für diese Anweisungen erfolgt später. Die möglichen Stati und Statusübergänge eines Prozesses sind (in Abhängigkeit von PEARL-Steueranweisungen) in Bild 1.2 dargestellt.

Ein Prozeß kann vielfach *blockiert* werden (z.B. durch Ausführung von SUSPEND und durch Ausführung einer Synchronisieranweisung). Um aus diesem blockierten Zustand herauszukommen, müssen alle entsprechenden Anweisungen zur Fortsetzung des Prozesses (z.B. durch Ausführung von CONTINUE und Freigabe der Synchronisiervariable) ausgeführt werden. Sonst bleibt der Prozeß blockiert.

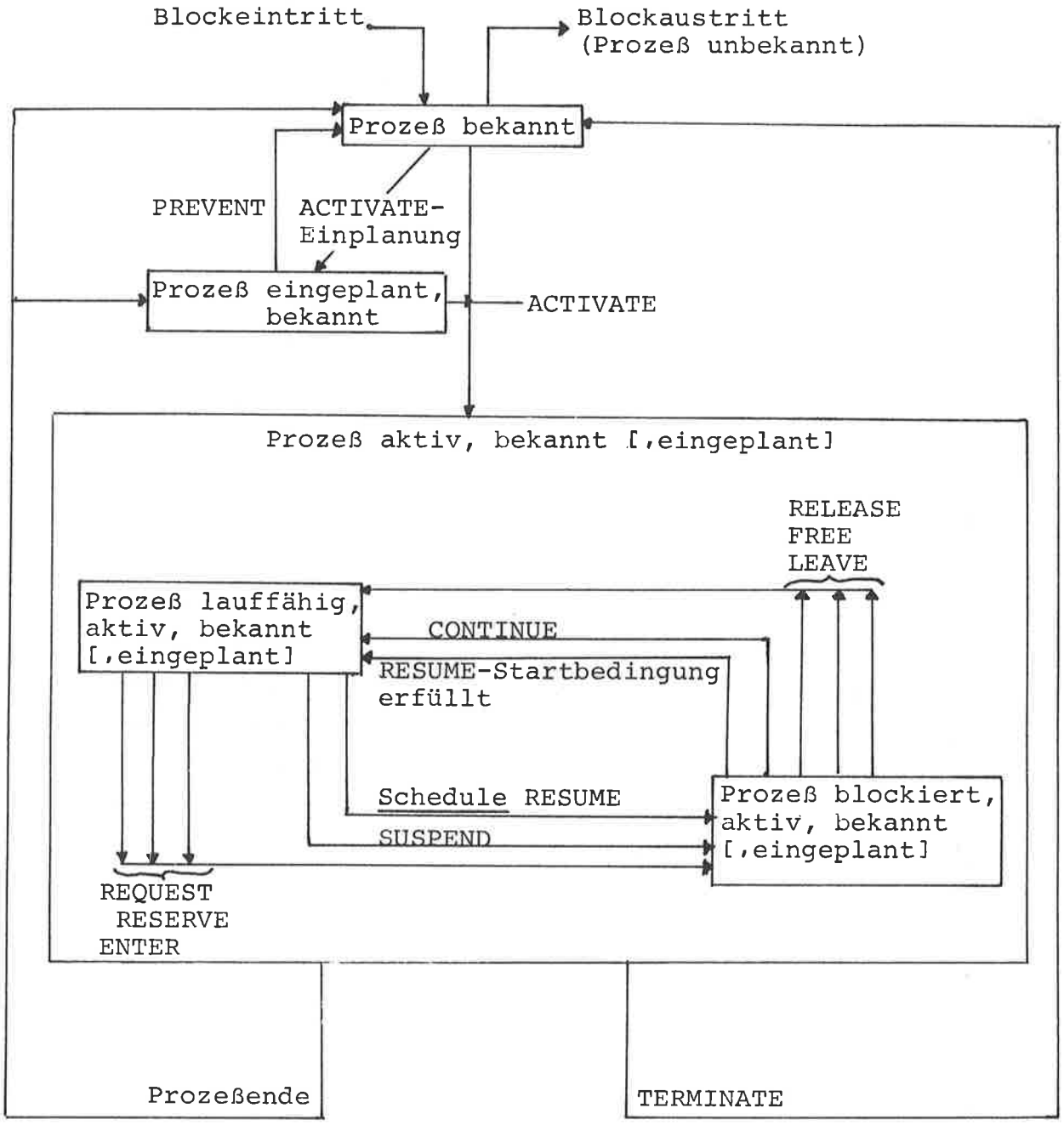


Bild 1.2: Prozeßstatus

Das Betriebssystem bearbeitet jeweils den wichtigsten lauffähigen Prozeß (≡laufender Prozeß); die Wichtigkeit bestimmt der Programmierer durch Prioritätsangabe.

1.3 HIERARCHIE DER ARBEIT IM RECHNER

Das Betriebssystem läßt die Rechenoperationen auf einer von drei Ebenen ablaufen, indem es seine Verwaltungstätigkeiten hierarchisch zwischen die Reaktion auf Meldungen und die Ausführung des Anwenderprogramms einordnet. Diese *Schichtenstruktur* des Betriebssystems gewährleistet einerseits eine schnelle Reaktion auf Meldungen (Interrupts) und andererseits die ungestörte Abwicklung der Verwaltungsarbeiten.

Bild 1.3 zeigt die Untergliederung der Betriebssystemaufgaben und die Hierarchie der Rechenvorgänge.

Nach Eintritt einer Meldung wird in der *Unterbrechungsebene* die laufende Arbeit (der laufende Prozeß) angehalten und der Punkt, an dem sie fortzusetzen ist, definiert. In einer kurzen Zeitphase (kleiner als eine Millisekunde bei heute üblichen Prozeßrechnern) nach Identifizierung der Meldung wird primär reagiert und,sofern mit der Primärreaktion die Meldung noch nicht vollständig ausgewertet ist, eine sekundäre Reaktion eingeplant, die als wichtigste nächste Arbeit auf der Verwaltungsebene durchzuführen ist. Nur während der primären Reaktion (einschließlich Einplanung einer sekundären Reaktion) ist der Rechner nicht unterbrechbar.
Die Reaktion auf Meldungen ist die wichtigste Aufgabe in der Bearbeitungshierarchie, da dadurch die momentane Arbeit in anderen Ebenen beeinflußt werden kann und wichtigere als zuvor zu bearbeitende Prozesse lauffähig werden können.

Auf der *Verwaltungsebene* erfolgt die Prozeß- und Betriebsmittelverwaltung. Es wird sichergestellt, daß kein Wechsel von Rechenprozessen (Aufnehmen eines anderen wichtigeren Prozesses) stattfindet, solange sich die Bearbeitung in der Verwaltungsebene befindet, damit Betriebssystemfunktionen nicht zu Zeitpunkten, in denen Datenstrukturen in einem nicht definierten Zustand sind, unterbrochen werden und evtl. unter Kontrolle eines anderen Prozesses auf eben diesen Datenstrukturen zu arbeiten versuchen. Jede dieser Systemfunktionen ist logisch unteilbar, d.h. ihre

Befehlsfolge kann zwar durch Arbeit auf der Unterbrechungsebene unterbrochen werden, danach wird sie aber auf jeden Fall vor weiteren Systemfunktionen zu Ende geführt. Diese Systemfunktionen werden unter Regie von Prozessen (Anwenderprozesse oder vom Betriebssystem kreierte Prozesse) ausgeführt.

Auf der *Anwenderebene* wird das Anwenderprogramm ausgeführt. Der Rechner arbeitet solange auf der Anwenderebene, bis entweder eine Meldung ihn auf die Unterbrechungsebene anhebt oder nach Aufruf einer Systemfunktion durch den Prozeß auf der Verwaltungsebene fortgefahren wird. Um jeweils den dringlichsten Prozeß aufzunehmen, wird jedesmal beim Ende einer Systemfunktion (falls seit dem Verlassen der Anwenderebene ein weiterer Prozeß lauffähig wird) der wichtigste lauffähige Prozeß bestimmt und aufgenommen.

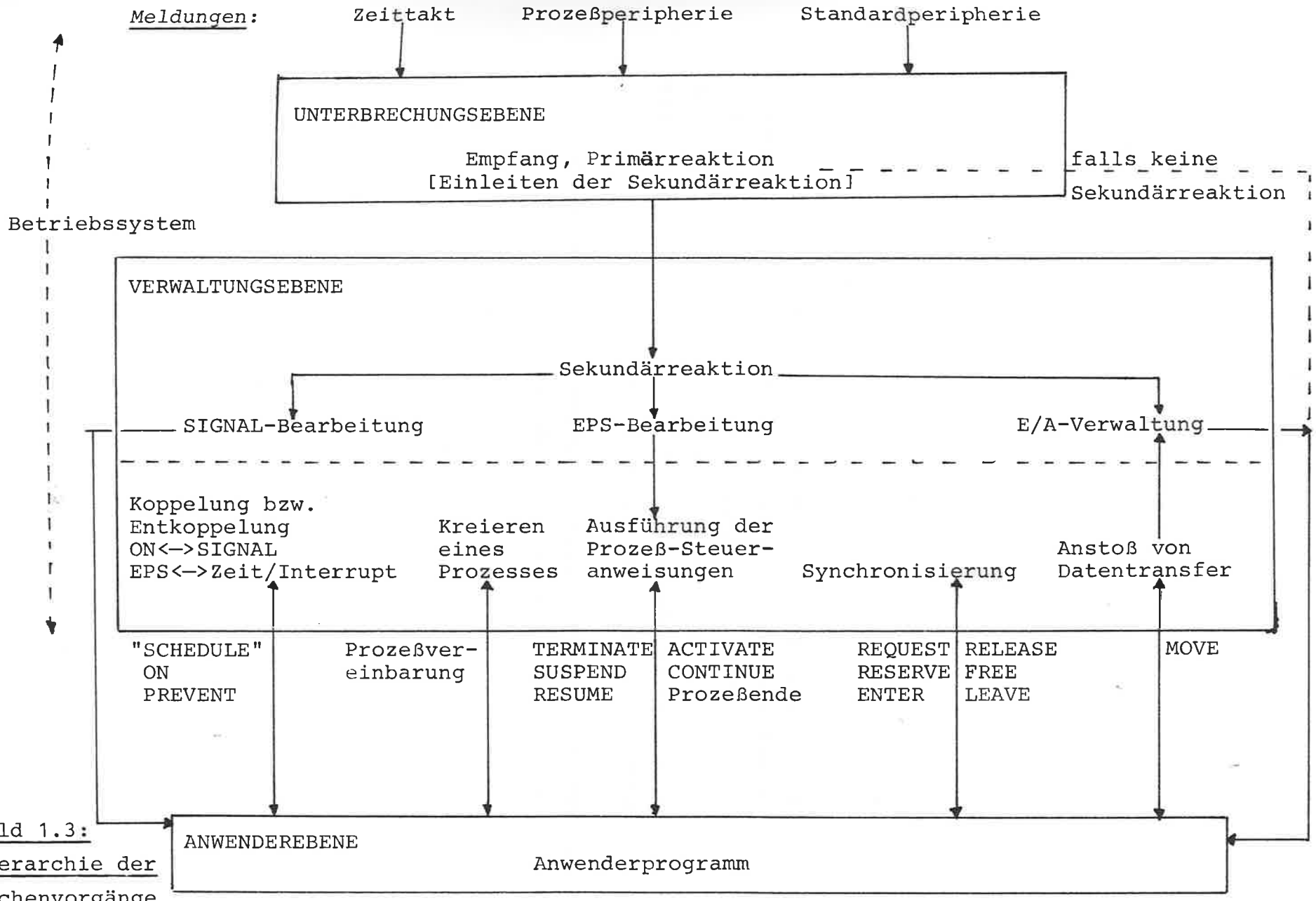


Bild 1.3: Hierarchie der Rechenvorgänge

1.4 NOTIEREN VON PROZESSTEUERANWEISUNGEN MIT STARTBEDINGUNGEN

Beim Überlaufen einer Prozeßsteueranweisung pst für einen Prozeß p werden folgende Operationen ausgeführt:

1. Es wird getestet, ob für den Prozeß p die Steueranweisung pst eingeplant ist (wie die Einplanung erfolgt, wird im folgenden beschrieben); ist dies der Fall, so wird die Einplanung wieder gestrichen (Ausplanung von Prozeßsteueranweisungen).

2a. Falls für die Prozeßsteueranweisung eine Startbedingung angegeben ist, erfolgt die Einplanung der Prozeßsteueranweisung, indem alle Angaben (Startbedingung, Steueranweisung, Task-Angabe und Optionen) in einem *Einplankontrollsatz* (EPS) notiert werden und indem der EPS in die Warteschlangen (Uhrzeit-, Dauer- oder Ereignisketten) eingeordnet wird; im Fall der Prozeßsteueranweisung RESUME wird außerdem der angesprochene Prozeß blockiert (bis die eingeplante Steueranweisung aufgrund von Meldungen oder beim Eintreffen der angegebenen Zeitpunkte diese Blockierung auflöst).

2b. Ist keine Startbedingung angegeben, so wird die Prozeßsteueranweisung sofort ausgewertet.

1.4.1 Übersetzung von Prozeßsteueranweisungen mit Startbedingungen

Eine Anweisung der Form

startbedingung···prozeßsteuerung [task-name][optionen]

wird intern durch folgende Befehlssequenz dargestellt:

(1) CALL SCHANF || ADDR Anfang des Einplankontrollsatzes EPS (für statische Speicherplatzverwaltung)
oder
VALUE Länge des Einplankontrollsatzes EPS in vielfachen von 8 Wörtern (für dynamische Speicherplatzverwaltung)

(2) In der Reihenfolge ihrer Niederschrift werden die Komponenten der Startbedingungen durch folgende Funktionsaufrufe ersetzt.

CALL AZ	|| ADDR Uhrzeit	für AT	Uhrzeit
CALL AI	|| ADDR Interrupt(ER)	für WHEN	Interrupt
CALL WAL	|| ADDR Dauer	für ALL	Dauer
CALL WEV	|| ADDR Dauer	für EVERY	Dauer
CALL VERZ	|| ADDR Dauer	für AFTER	Dauer
CALL EUN	|| ADDR Uhrzeit	für UNTIL	Uhrzeit
CALL EDU	|| ADDR Dauer	für DURING	Dauer

Die Folge der Aufrufe für eine Startbedingung, deren letzte Komponente nicht UNTIL Uhrzeit oder DURING Dauer ist, wird durch CALL LIM abgeschlossen.

(3) Falls als Prozeßsteuerung CONTINUE und in den Optionen eine Priorität angegeben ist:

CALL CNTMP || ADDR Priorität

(4) Falls in den Optionen EXCEPT task-name,·· angegeben ist:

CALL EXCPT || ADDR (Anzahl der Prozesse, Adreßliste der Prozeßkontrollsätze PKS aller spezifizierten Prozesse)

(5a) Falls als Prozeßsteuerung SUSPEND, CONTINUE, TERMINATE oder PREVENT und kein Task-Name angegeben ist:

```
CALL SCHOT  || VALUE   Steuerungsart
```

Falls bei diesen Prozeßsteuerungen ein Task-Name angegeben ist:

```
CALL SCHMT  || VALUE   Steuerungsart
            || ADDR    Prozeßkontrollsatz PKS des spezi-
                       fizierten Prozesses
```

Die Steuerungsart ist folgendermaßen verschlüsselt:

1 für SUSPEND
2 für CONTINUE mit Prioritätsangabe
3 für CONTINUE ohne Prioritätsangabe
4 für TERMINATE
6 für PREVENT

(5b) Falls als Prozeßsteuerung RESUME und ein Task-Name angegeben ist:

```
CALL RSUMTA || ADDR    Prozeßkontrollsatz PKS des spezi-
                       fizierten Prozesses
```

Falls als Prozeßsteuerung RESUME angegeben ist und der Task-Name fehlt:

```
CALL  RSUMLT
```

(5c) Falls als Prozeßsteuerung ACTIVATE angegeben ist:

```
CALL  ACTNP || ADDR    Prozeßkontrollsatz PKS des spezi-
            ||         fizierten Prozesses
            ||
            || ADDR    Priorität
            || ADDR    SEMA-Variable, falls die USING-
            ||         Option benutzt ist, anderenfalls
            ||         ist der leere Hinweis NULL zu über-
            ||         geben
            ||
            || ADDR    Segmentanfang des zu aktivierenden
                       Prozesses
```

Falls in der ACTIVATE-Anweisung das Segment des zu aktivierenden Prozesses angegeben ist, folgt ein Sprungbefehl auf den ersten Befehl des aktivierenden Prozesses hinter dem angegebenen Segment.
GOTO Adresse hinter "CALL PREND"

Das Segment des zu aktivierenden Prozesses wird hinter diesem Sprungbefehl abgelegt und durch

CALL PREND

abgeschlossen, die Funktion PREND verwaltet den normalen Abschluß eines Prozesses.

Es gibt eine Variante der Aktivierung, die berücksichtigt, daß Prozesse unter gewissen Bedingungen nur einen Teil eines Prozeßkontrollsatzes PKS benötigen und statt eines eigenen den Laufzeitkeller des übergeordneten Prozesses benutzen können. Um diese Möglichkeit des Betriebssystems zu nutzen, muß zur Übersetzungszeit festgestellt werden, ob ein Prozeß folgende Bedingungen erfüllt:

. er hat keine untergeordneten Prozesse
. er kann den Laufzeitkeller des übergeordneten Prozesses benutzen
. seine Aktivierung braucht nie gepuffert zu werden
. bei seiner Aktivierung wird nie die Sema-Option benutzt.

Für solche Prozesse wird statt des Aufrufs von ACTNP

CALL	ACTKP	ADDR	Prozeßkontrollsatz PKS des spezifizierten Prozesses
		ADDR	Priorität
		ADDR	Segmentanfang des zu aktivierenden Prozesses

abgesetzt und danach wie nach dem Aufruf von ACTNP fortgefahren.

1.4.2 Aufbau eines Einplankontrollsatzes

Durch die Ausführung der internen Befehlsfolge für eine Prozeßsteueranweisung mit Startbedingungen wird ein Einplankontrollsatz EPS aufgebaut.
Er hat die Form:

Kopf Einplan-Gruppe···[Prioritäts-Element][Except-Liste]

Dabei ist die Information einer Startbedingung in einer Einplan-Gruppe abgelegt.

1.4.2.1 Kopf eines Einplankontrollsatzes

L	Länge des EPS in ganzen Vielfachen von 8 Wörtern
EPSK	Zeiger zur Verkettung aller für einen Prozeß eingerichteten EPS
OP	Identifikation der Prozeßsteuerung
ZTV	Zeiger auf den Prozeßkontrollsatz PKS des zu steuernden Prozesses
PEO	Zeiger auf die Except-Liste, falls die EXCEPT-Option benutzt wurde oder auf die Konstante O
P	Zeiger auf das Prioritätselement, falls als Prozeßsteuerung CONTINUE und die PRIORITY-Option angegeben wurde

Funktion: SCHANF

Aufruf der Speicherplatzverwaltung zur Reservierung eines Bereichs der angegebenen Länge

Initialisierung von L

Ablage des Hinweises auf die Konstante O in PEO und des leeren Hinweises in P

1.4.2.2 Prioritätselement, Except-Liste

Die Funktion CNTMP erzeugt ein *Prioritätselement* indem sie den übergebenen Wert kopiert und seine Adresse im Kopf des Einplankontrollsatzes ablegt.

Aufgrund einer Anweisung mit der EXCEPT-Option wird in EXCPT eine *Except-Liste* erzeugt und ihre Adresse im Kopf des Einplankontrollsatzes abgelegt.

Anzahl (n≥1) der in der EXCEPT-Option genannten Prozesse

Zeiger auf den Prozeßkontrollsatz PKS des ersten Prozesses

⋮

Zeiger auf den Prozeßkontrollsatz des PKS des n-ten Prozesses

1.4.2.3 Interne Darstellung einer Startbedingung

Eine Startbedingung besteht aus maximal vier Komponenten

- Anfang (AT Uhrzeit, WHEN Interrupt)
- Verzögerung (AFTER Dauer)
- Wiederholung (ALL Dauer, EVERY Dauer)
- Ende (UNTIL Uhrzeit, DURING Dauer)

Die interne Darstellung einer Startbedingung heißt Einplan-Gruppe und hat die Form

Einplan-Element [Einplanelement[Einplanelement]] Endekennung

Einplan-Element

CA	Identifizierung (Marke) des Programmstücks, das auszuführen ist, wenn die Uhrzeit erreicht, die Dauer verstrichen oder die Meldung eingetroffen ist
N,V	Zeiger zur Einordnung in eine Zeit- oder Ereigniskette
KP	Zeiger auf den Kopf des EPS
TYP	Identifizierung (Marke) des Programmstücks, das beim Abbau des EPS die Ausordnung des Einplanelementes aus einer Zeit- oder Ereigniskette einleitet
U,D	Zeitangaben

Aus Optimierungsgründen werden auch verkürzte Einplan-Elemente (ohne U,D) benutzt.
Im folgenden wird beschrieben, welche Einplanelemente für die Komponenten einer Startbedingung die entsprechende Einplan-Gruppe bilden und wie sie initialisiert sind. Dabei bedeutet der Eintrag '-' in U,D,N u.V , daß keine Initialisierung vorliegt und die Einträge ZKU, ZKD, ERK in N,V deuten an, daß das Einplanelement in eine Zeit- oder Ereigniskette eingeordnet ist. Die Adresse von OP im Kopf des EPS wird mit POP bezeichnet.
In den Funktionen AZ, VERZ, WAL, WEV, EUN und EDU wird vor Aufbau des entsprechenden Einplanelementes die Zeitangabe geprüft und bei Werten, die außerhalb von 0 bis 24 Stunden liegen eine Fehlermeldung gegeben. Falls eine Dauer unterhalb einer implementationsabhängigen Schranke MIND liegt, wird sie durch MIND ersetzt.

AT t

Funktion: AZ

CA	ATST
N,V	ZKU
TYP	LUR
KP	POP
U	t

WHEN i

Funktion: AI

CA	IST
N,V	ERK
TYP	LON
KP	POP

WHEN i AFTER d1 [ALLd2]

Funktionsfolge: AI, VERZ[,WAL]

CA	IAST
N,V	ERK
TYP	LOF
KP	POP
U	d1

CA	WAST/W
N,V	---
TYP	LDA
KP	POP
U	---
D	---/d2

Wenn die Komponente ALLd2 existiert, wird in CA die Marke W und in D die Zeitangabe d2 eingetragen

AFTER d1 [ALLd2] als Anfang einer Startbedingung

Funktionsfolge: VERZ[,WAL]

CA	WAST/W
N,V	ZKD
TYP	LDA
KP	POP
U	ZP+d1
D	---/d2

Wenn die Komponente ALLd2 existiert, wird in CA die Marke W und in D die Zeitangabe d2 eingetragen.
ZP bezeichnet den Zeitpunkt, zu dem die Anweisung überlaufen wurde.

ALL d als Anfang einer Startbedingung

Funktion: WAL

CA	W
N,V	ZKD
TYP	LDA
KP	POP
U	ZP+d
D	d

ZP bezeichnet den Zeitpunkt, zu dem die Anweisung überlaufen wurde

ALL d hinter WHEN i oder AT t

Funktion: WAL

CA	W
N,V	---
TYP	LDA
KP	POP
U	---
D	d

Dieses Einplanelement wird hinter dem WHEN i bzw. AT t entsprechenden Einplanelement abgelegt.

EVERY d

Funktion: WEV

Es gibt eine implementationsabhängige Schranke für die Dauerangabe d hinter EVERY; liegt d unterhalb dieser Schranke, wird ein Einplanelement wie für ALL d hinter AT t erzeugt.

CA	W
N,V	---
TYP	LEV
KP	POP
U	---
D	d

Dieses Einplanelement wird hinter dem AT t entsprechenden Einplanelement abgelegt.

DURING d

Funktion: EDU

CA	E
N,V	---/ZKD
TYP	LDA
KP	---
U	---/ZP+d
D	d

UNTIL t

Funktion: EUN

CA	E
N,V	---/ZKU
TYP	LUR
KP	---
U	t

Diese Einplanelemente werden hinter dem Einplanelement abgelegt, das der Komponente EVERY d bzw. ALL d der Startbedingung entspricht. Wenn die erste Komponente der Startbedingung AFTER d1 oder ALL d1 ist, gilt ZKD bzw. ZKU.

1.4.2.4 Beispiel eines Einplankontrollsatzes

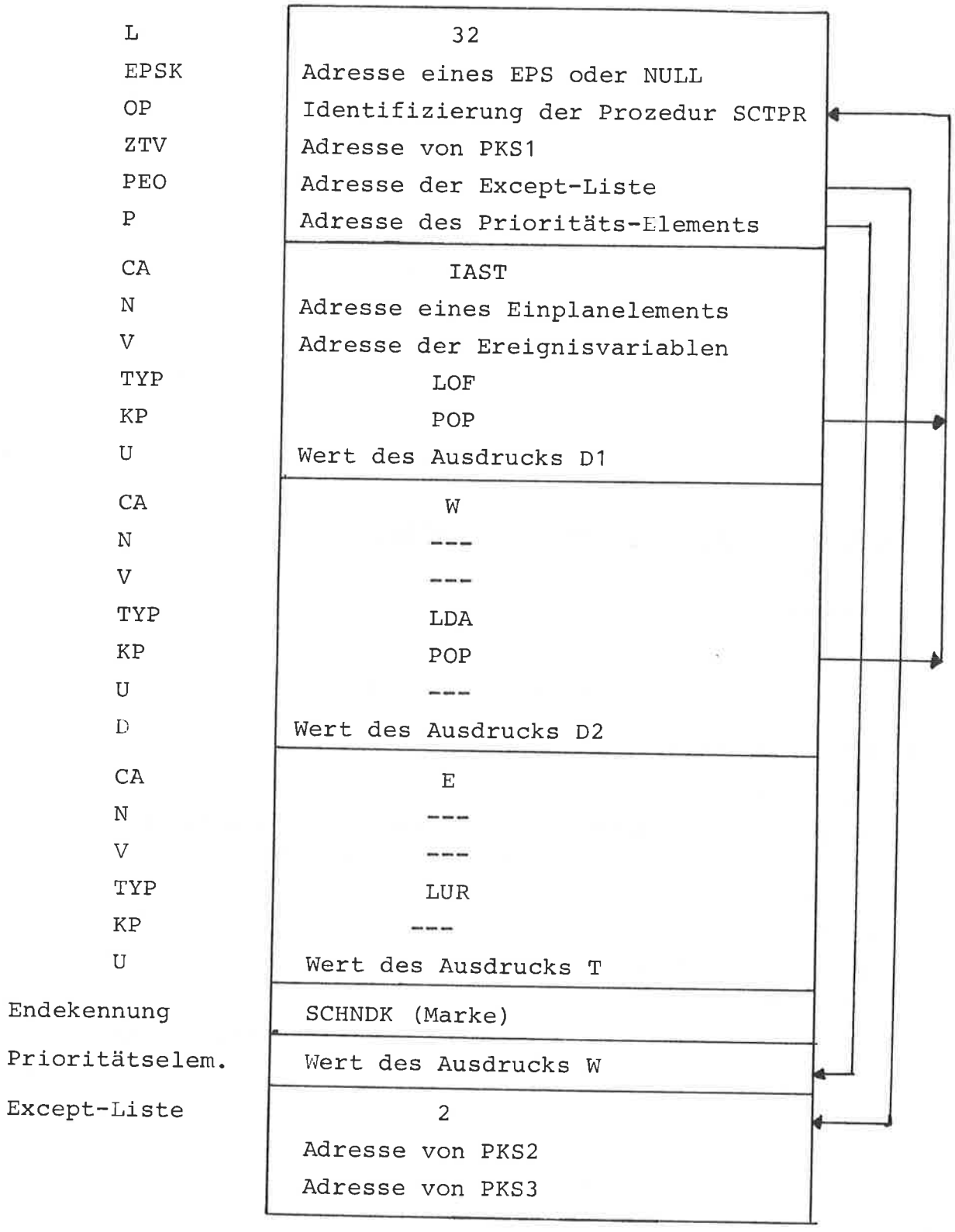
Die Steueranweisung

```
WHEN I AFTER D1 ALL D2 UNTIL T
CONTINUE P1 PRIORITY W EXCEPT P2, P3;
```

wird wie folgt übersetzt:

```
CALL SCHANF   || VALUE  32
CALL AI       || ADDR   der dem Namen I zugeordneten
                        Ereignisvariable ER
CALL VERZ     || ADDR   D1
CALL WAL      || ADDR   D2
CALL EUN      || ADDR   T
CALL CNTMP    || ADDR   W
CALL EXCEPT   || ADDR   Liste (Zeiger auf PKS2,Zeiger auf PKS3
CALL SCHMT    ||| VALUE 2
              ||| ADDR  PKS1
```

Bei Ausführung dieser Befehlsfolge wird der folgende Einplankontrollsatz EPS aufgebaut und initialisiert:



1.4.3 Ein- und Ausordnen von Einplanelementen in Zeit- und Ereignisketten

Aus Einplanelementen, die aufgrund einer der Komponenten

AT Uhrzeit, EVERY Dauer, UNTIL Uhrzeit

erzeugt wurden, wird die *Uhrzeitkette* gebildet.

Aus Einplanelementen, die aufgrund einer der Komponenten

AFTER Dauer, ALL Dauer, DURING Dauer

erzeugt wurden, wird die *Dauerkette* gebildet.

Einplanelemente, die aufgrund der Komponente

WHEN Interrupt

erzeugt wurden, sind in einer interruptspezifischen *Ereigniskette* zusammengefaßt.

1.4.3.1 Kettenköpfe

Zur Verwaltung der Uhrzeitkette und der Dauerkette führt das Betriebssystem die Datenstrukturen ZKU bzw. ZKD.

Für jeden im Systemteil eingeführten Interrupt wird eine Datenstruktur vom Typ ER (Ereignisvariable) erzeugt.

ZKU/ZKD

EW	Marke PRIMU/PRIMD der Primärreaktion auf den Auflösungstakt eines Zeitgebers	*)
CA	Marke CORR	
N	Adresse eines Einplanelementes	
V	Adresse eines Einplanelementes	
DW	Marke, Wert von ZKU.ABF oder ZKU.NABF/ Wert von ZKD.ABF oder ZKD.NABF	
F	Adresse eines Einplanelementes	
U	Konstante TAG, Tageslänge in Millisekunden	
D	Zähler, Dauer in Millisekunden, nach der das Einplanelement bearbeitet werden muß, dessen Adresse in ZKU.F/ZKD.F steht	
NXU	Zeitpunkt, zu dem das durch ZKU.F/ZKD.F identifizierte Einplanelement eingeplant ist	
ABF	Marke ABFU/ABFD in der Primärreaktion	*)
NABF	Marke NABFU/NABFD in der Primärreaktion	*)
ZMV	Marke, Wert von ZKU.NOP oder ZKU.SEKR/ Wert von ZKD.NOP oder ZKD.SEKR	
NOP	Marke NOPU/NOPD: keine Sekundärreaktion	*)
SEKR	Marke SEKRU/SEKRD der Sekundärreaktion	*)

ER

EW	Marke NOTIER oder ZAEHLE der Primärreaktion	*)
ERK	Adresse eines Einplanelementes	
ZEIT	Zeitpunkt des Eintritts der Meldung	
ZAHL	Zähler	

*) Primär- und Sekundärreaktion sind im Kapitel "Bearbeitung von Meldungen" beschrieben.

1.4. 3.2 Einordnen in eine Zeitkette

Die Position an der ein Einplanelement in die Uhrzeit- oder Dauerkette einzuordnen ist, ergibt sich aus dem in U abgelegten Zahlwert.
In Einplanelementen, die aufgrund einer Komponente AT Uhrzeit oder UNTIL Uhrzeit existieren, wird beim Aufbau des EPS U mit dem Wert des Ausdrucks Uhrzeit initialisiert.

Bei den Einplanelementen, die aufgrund einer Komponente AFTER Dauer, ALL Dauer, EVERY Dauer oder DURING Dauer existieren, wird beim Aufbau des EPS D mit dem Wert des Ausdrucks Dauer initialisiert. Bei ihrer Einordnung in eine Zeitkette wird die Summe

(aktueller Zeitpunkt+D) modulo (TAG)

in U abgelegt. Der *aktuelle Zeitpunkt* ist durch

ZKU.NXU + ZKU.D für die Uhrzeitkette
ZKD.NXU + ZKD.D für die Dauerkette

definiert.

Einordnungsvorschrift für die Zeitketten:

Ein Einplanelement, dessen U den Wert U1 hat, ist Vorgänger aller Einplanelemente, deren U einen Wert U2 mit U1≤U2 hat.

Der Kopf der Zeitkette wird als Einplanelement interpretiert, dessen U den Wert TAG = 24 Stunden in Millisekunden hat.
(Der Teil von CA bis D in ZKU und ZKD hat die Struktur eines Einplanelementes.)

Verzeigerung der Zeitketten

In N und in V eines in einer Zeitkette eingeordneten Einplanelementes ist die Adresse seines unmittelbaren Nachfolgers bzw. Vorgängers abgelegt.
In N und in V von ZKU/ZKD ist die Adresse des Einplanelementes mit dem kleinsten bzw. größten Wert von U abgelegt; im Initial-

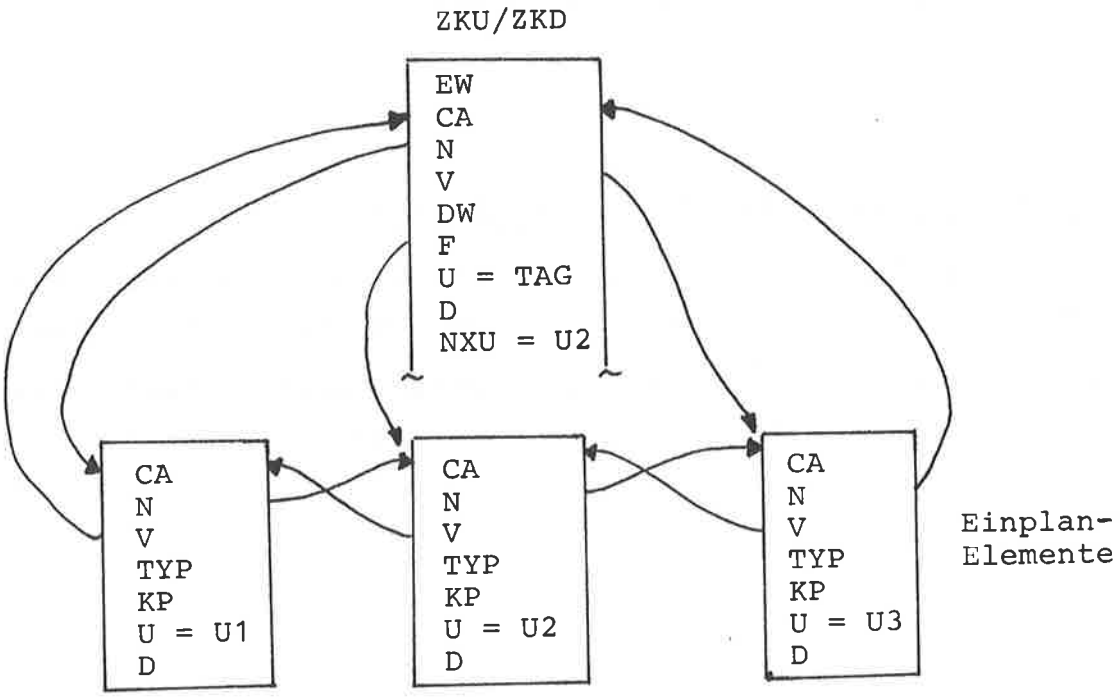
zustand (leere Zeitkette) enthalten diese Variablen die Adresse von CA in ZKU/ZKD.

Das ausgezeichnete Element einer Zeitkette

In jeder Zeitkette existiert ein Einplanelement (bei leerer Kette der Kopf), für das im Vergleich zu den anderen Kettenelementen folgende Differenz (Zeitabstand) minimal wird

U - aktueller Zeitpunkt

und für dessen Vorgänger diese Differenz negativ ist. Dieses Einplanelement wird *ausgezeichnetes Element* genannt; seine Adresse ist in F und sein U in NXU von ZKU/ZKD abgelegt.



$0 \leq U1 < U2 \leq U3 \leq TAG$

Einordnungsalgorithmen

Mit X werde das einzuordnende Einplanelement bezeichnet.

Dauerkette

Funktion: EIND

(1) Berechnung der Position von X durch
(aktueller Zeitpunkt+D) modulo (TAG)
und Zuweisung an U. F ortsetzung bei (2).

Uhrzeitkette

Funktion: EINU

(2) Liegt U (die Position von X) zwischen dem aktuellen Zeitpunkt und der Position des ausgezeichneten Elements der Uhrzeit- bzw. Dauerkette, wird bei ERST, anderenfalls bei EINKET fortgefahren.

ERST: X wird ausgezeichnetes Element der Uhrzeit- bzw. Dauerkette indem in ZKU bzw. ZKD die Werte von F, NXU und D entsprechend geändert werden.
X wird als Vorgänger des bisherigen ausgezeichneten Elements eingekettet.

EINKET: Die Uhrzeit- bzw. Dauerkette wird nach steigenden Werten von U' untersucht und X als Vorgänger des ersten Einplanelementes mit $U \leq U'$ eingekettet.

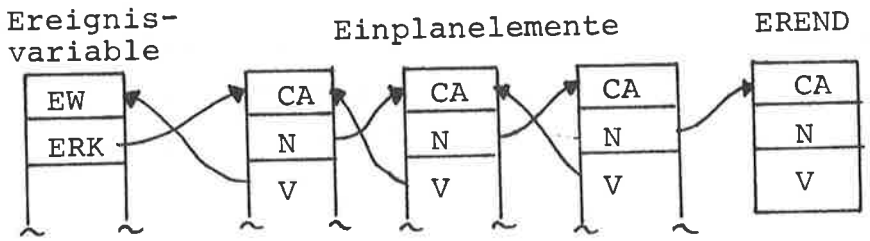
1.4.3.3 Ausordnen aus einer Zeitkette

Funktion: AUSORD

Falls X nicht das ausgezeichnete Element der Zeitkette ist, wird es aus der Verzeigerung gelöst und als ausgekettet gekennzeichnet.
Anderenfalls wird zunächst sein Nachfolger zum ausgezeichneten Element der Uhrzeit- bzw. Dauerkette gemacht, indem in ZKU bzw. ZKD die Werte von F, NXU und D entsprechend geändert werden.

1.4.3.4 Ein- und Ausordnen in Ereignisketten

Bei jeder Ereignisvariablen kann eine Kette von Einplanelementen geführt werden, ERK weist auf das erste Einplanelement und N des letzten Einplanelements enthält die Adresse der Struktur EREND, die das Ende aller Ereignisketten darstellt. Falls kein Einplanelement eingeordnet ist, weist ERK auf EREND (Initialzustand).



Einordnen

Funktion: EINER Parameter: Adresse der Ereignisvariablen

Das einzuordnende Einplanelement wird als erstes in die Ereigniskette eingekettet.

Ausordnen

Funktion: AUSKET

Das Einplanelement wird aus der Verzeigerung gelöst und als ausgekettet gekennzeichnet.

1.4.4 Notieren und Vorbereiten der Prozeßsteuerung

Eine der Funktionen SCHOT, SCHMT, RSUMLT, RSUMTA, ACTNP und ACTKP beendet die Ausführung einer Prozeßsteueranweisung mit Startbedingungen.

Funktion: SCHOT Steuerungsart, keine Prozeßangabe
Funktion: SCHMT Steuerungsart, mit Prozeßangabe

(1) Weitere Initialisierung im Kopf des EPS:

Ablage der Adresse des Prozeßkontrollsatzes PKS in ZTV und der Identifizierung der Prozeßsteuerung in OP.

(2) In TESTA wird geprüft, ob für den betroffenen Prozeß schon ein EPS für dieselbe Steuerungsart existiert. In diesem Fall wird der Kopf des alten EPS aus der prozeßspezifischen Kette der EPS entfernt und in die Liste der für die spätere Ausplanung (Ausordnen der Einplanelemente und Speicherplatzfreigabe) vorgesehenen EPS aufgenommen. In OP wird notiert, daß keine Prozeßsteuerung auszuführen ist.

(3) Der neue EPS wird in die prozeßspezifische Kette der EPS aufgenommen.
Im PKS wird notiert, daß eine Einplanung für die angegebene Steuerungsart existiert.

(4) Wenn eine der Startbedingungen die Form

ALL d [UNTIL t|DURING d1]

hat, wird die Prozeßsteuerung sofort ausgeführt.

Funktion: RSUMTA Prozeßangabe
Funktion: RSUMLT keine Prozeßangabe

Der betroffene Prozeß wird zurückgestellt und in seinem PKS notiert, daß seine Wiederaufnahme aufgrund der Prozeßsteuerung RESUME eingeplant ist.
Fortsetzung bei (1).

Funktion: ACTNP
Funktion: ACTKP

Diese Funktionen sind im Abschnitt 1.7.7 über Prozeßsteuerung beschrieben.

1.5 AUSWERTUNG VON ON-ANWEISUNGEN

1.5.1 Vereinbarung von Signalen und Übersetzung von On-Anweisungen

Die nach Auswertung des Systemteils eines PEARL-Programms vorliegenden Ereignisvariablen können im Problemteil als SIGNAL spezifiziert werden. In diesen Fällen wird ein *Signal-Beschreibungselement* SG

CA	Marke PSIG des Programmstücks in der Sekundärreaktion auf Meldungen, das die Ausführung der dem SIGNAL zugeordneten Befehlsfolge einleitet
N V	Zeiger zur Einordnung in eine Ereigniskette
STDR	Marke, Standardreaktion

erzeugt und in Kette ERK der genannten Ereignisvariablen ER eingeordnet.

Eine On-Anweisung schreibt eine prozeßspezifische Reaktion auf ein Signal vor.
Die interne Darstellung von

```
ON Signal : Anweisung...
```

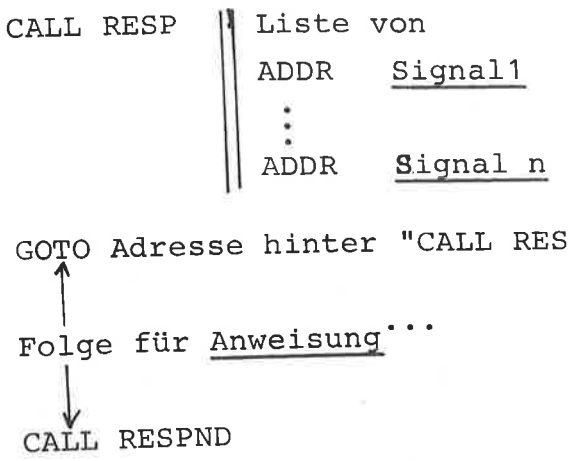
ist

```
CALL RESP1 || ADDR Signal
GOTO Adresse hinter "CALL RESPND"
  ↑
Folge für Anweisung...
  ↓
CALL RESPND
```

Die interne Darstellung von

ON Signal1,...,Signal n : Anweisung...

ist



Die als Parameter übergebenen Adressen sind die Adressen der zugehörigen Ereignisvariablen ER.
Eine GOTO-Anweisung der Folge Anweisung..., deren Sprungziel außerhalb dieser Folge liegt, ist durch

CALL RSPAUS || ADDR Markenbeschreibung (siehe 3.4.2.1.3)

zu übersetzen.

Die Funktionen RESP1 und RESP werden bei Überlaufen der On-Anweisung aufgerufen.
Im Rahmen der Sekundärreaktion auf Meldungen der Prozeßperipherie wird die spätere Ausführung der Anweisungsfolge und des Aufrufs von RESPND durch den Prozeß eingeplant, dessen Code die On-Anweisung enthält.

1.5.2 Blockspezifische Verwaltung von On-Anweisungen

Jeder Aktivierungssatz AS enthält einen Zeiger SIGK, der mit dem leeren Hinweis NULL initialisiert wird und folgendermaßen zur blockspezifischen Verwaltung von On-Anweisungen eines Prozesses benutzt wird.
Bei Überlaufen einer On-Anweisung wird für jedes angegebene Signal im aktuellen Aktivierungssatz AS des laufenden Prozesses ein *On-Beschreibungsblock* RSPB

KR	Zeiger zur Verkettung aller RSPB eines AS
ID	Hinweis auf die zugeordnete Ereignisvariable ER
AD	Adresse der internen Darstellung der Anweisungsfolge
AS	Hinweis auf den AS, in dem dieser RSPB liegt

abgelegt, initialisiert und in die Kette der On-Beschreibungsblöcke des Aktivierungssatzes aufgenommen.

Funktion: RESP1
Erzeugung und Initialisierung eines RSPB in der oben beschriebenen Weise. Einordnen dieses RSPB als erstes Element der durch SIGK identifizierten Kette.

Funktion:RESP
Für jedes Element der Parameterliste wird wie in RESP1 verfahren.

Anhand dieser Datenstrukturen wird nach Eintritt eines als SIGNAL spezifizierten Ereignisses, in der Sekundärreaktion geprüft, ob der die Meldung verursachende Prozeß in einer On-Anweisung eine Reaktion definiert hat (siehe:1.6.2.3).

1.6 BEARBEITUNG VON MELDUNGEN

Das Betriebssystem leitet die Ausführung einer unter Startbedingungen stehenden Prozeßsteueranweisung oder die Ausführung der in einer On-Anweisung spezifizierten Anweisungsfolge ein, wenn ein in den Startbedingungen angegebener Interrupt eintritt oder ein Zeitpunkt erreicht ist, bzw. wenn das spezifizierte Signal gegeben wird. Parallel zur Zentraleinheit arbeitende E/A-Gerät signalisieren das Ende eines Datentransports, um das Betriebssystem zur weiteren Steuerung von Übertragungen zu veranlassen.

Der Eintritt eines Interrupts, Signals, Zeittaktes oder eines Kontrollsignals von einem E/A-Gerät wird im folgenden als Meldung bezeichnet.
Der Anwender erwartet, daß jederzeit Meldungen empfangen werden können und die Reaktionszeit, d.h. die Zeit zwischen der Meldung in der eingeplanten Reaktion, möglichst kurz ist. Daher bearbeitet das Betriebssystem eine Meldung in zwei Phasen.

Zunächst wird in der *Primärreaktion* die Herkunft der Meldung festgestellt und eine kurze, meldungsspezifische Befehlsfolge ausgeführt, die ggf. notiert, daß diese Meldung in der Sekundärreaktion zu bearbeiten ist. Die Primärreaktion kann nicht unterbrochen werden.

In der *Sekundärreaktion* werden die für die Meldung eingeplanten Anweisungen ausgeführt. Diese Arbeit kann durch die Bearbeitung einer Meldung in der Primärreaktion unterbrochen werden. Erst nach vollständiger Bearbeitung aller Meldungen wird der wichtigste lauffähige Prozeß fortgesetzt.

1.6.1 Primärreaktion auf Meldungen

Unmittelbar nach einer Meldung wird das folgende Programmstück - die Primärreaktion - ausgeführt.

(1) In INH wird verhindert, daß aufgrund weiterer Meldungen die Primärreaktion eingeleitet wird.

Falls die Befehlsfolge eines Prozesses unterbrochen wurde, lagert STRETT den Befehlszähler und die Registerinhalte in dessen Prozeßkontrollsatz PKS aus.
Es gibt zwei Betriebssystemprozesse (der eine zur Ausführung der Sekundärreaktion und der andere für Arbeiten der Speicherplatzverwaltung), die in diesem Zusammenhang wie die im Anwenderprogramm definierten Prozesse behandelt werden.

(2) Identifizierung der Meldung in IDINT, indem die Adresse der zugeordneten Datenstruktur ermittelt wird

- Zeitkettenkopf ZKU oder ZKD bei Zeittakt
- Ereignisvariable ER bei Meldungen der Prozeßperipherie
- Übertragungskontrollsatz UKS bei Rückmeldungen eines E/A-Geräts.

(3) Ausführung der in dieser Datenstruktur durch
- Marke EW in ZKU bzw. ZKD
- Marke EW in ER
- Marke REA in UKS

spezifizierten Befehlsfolge, in der ggf. eine Sekundärreaktion auf diese Meldung eingeplant wird.

(4) In ENB wird die Primärreaktion auf weitere Meldungen erlaubt.
Falls seit der Ausführung von INH Meldungen eingegangen sind, wird bei (1) fortgefahren.
Anderenfalls Fortsetzung bei (5).

(5) Falls keine Sekundärreaktion auszuführen ist, wird der unterbrochene Prozeß an dem Programmpunkt fortgesetzt, den er bei Eintritt in die Primärreaktion (siehe (1)) erreicht hatte.
Anderenfalls Fortsetzung bei (6).

(6) Falls der unterbrochene Prozeß eine Systemfunktion bearbeitete, führt er sie zunächst zu Ende, bevor bei (7) fortgefahren wird.

(7) Einleitung der Sekundärreaktion als wichtigster laufähiger Prozeß.

1.6.1.1 Primärreaktion auf Zeittakte

Es wird vorausgesetzt, daß zwei unabhängige Zeitgeber existieren. Die 'externe Uhr' meldet sich alle DELTAU Millisekunden und die 'interne Uhr' alle DELTAD Millisekunden. Die Impulsfrequenz ist durch diese Geräte bestimmt, es wird jedoch angenommen, daß DELTAU≥DELTAD gilt. Wenn nur die 'interne Uhr' existiert, simuliert das Betriebssystem den Zeittakt der 'externen Uhr'.
Die Markenvariablen EW in ZKU und ZKD identifizieren die meldungsspezifische Primärreaktion auf den Uhrzeittakt PRIMU bzw. den Dauertakt PRIMD.
PRIMU (PRIMD) wird nach jeder Meldung der 'externen (internen) Uhr' ausgeführt.

Funktion: PRIMU(PRIMD)

Erhöhung des Zählers D in ZKU(ZKD) um DELTAU (DELTAD).

Falls keine Sekundärreaktion auf die Meldung der 'externen (internen) Uhr' eingeplant ist und der Zähler D in ZKU(ZKD) nicht negativ ist, wird die Sekundärreaktion zur Bearbeitung der Uhrzeitkette (Dauerkette) eingeplant.

Falls die aufgrund der letzten Meldung eingeplante Sekundärreaktion noch nicht beendet ist, wird bei ihrer Fortsetzung der neue Zählerstand berücksichtigt.
Ein negativer Zählerstand bedeutet, daß der Zeitpunkt zur Bearbeitung des ausgezeichneten Elements der Uhrzeitkette (Dauerkette) noch nicht erreicht ist.
In allen Fällen wird bei (4) fortgefahren.

1.6.1.2 Primärreaktion auf eine Meldung der Prozeßperipherie

Aufgrund einer Meldung der Prozeßperipherie wird über die Marke EW in der zugeordneten Ereignisvariablen ER in eine der folgenden Befehlsfolgen gesprungen.

(i) Aufgrund einer Disable-Anweisung soll nicht auf diese Meldung reagiert werden (EW = DSBL).

(ii) Die Sekundärreaktion auf die letzte Meldung für diese Ereignisvariable ER ist noch nicht beendet (EW = ZAEHLE).
Erhöhung von ZAHL in ER um eins; diese Zahl wird nur bei der 'gepufferten' Aktivierung von Prozessen benutzt.
Notieren des aktuellen Zeitpunktes in ZEIT von ER.
Der aktuelle Zeitpunkt ist ZKD.NXU+ZKD.D.

(iii) Die Sekundärreaktion auf die letzte Meldung für diese Ereignisvariable ER ist abgeschlossen (EW = NOTIERE).
Notieren des Hinweises auf ER im Auftragspuffer (PUF) für die Sekundärreaktion auf Meldungen der Prozeßperipherie.
Notieren des aktuellen Zeitpunktes in ZEIT von ER.
Notieren in EW, daß auf weitere Meldungen mit (ii) zu reagieren ist.
Einplanen der Sekundärreaktion auf Meldungen der Prozeßperipherie.

In allen Fällen wird bei (4) fortgefahren.

1.6.2 Sekundärreaktion auf Meldungen

Die Sekundärreaktion wird als wichtigster Rechenprozeß ausgeführt.
Zur Einleitung der Sekundärreaktion wird dieser Prozeß laufähig gemacht und im Unterprogramm PRAUSF der wichtigste Prozeß aufgenommen.
Falls vor Einleitung der Sekundärreaktion die Primärreaktion mehrmals ausgeführt wurde, ist die Reihenfolge, in der die Meldungen bearbeitet werden, durch folgenden Algorithmus festgelegt.

Funktion: SEKRE

(1) Sprung über die Marke GMV nach SEKG oder NOPG

SEKG: Falls Rückmeldungen von E/A-Geräten zu bearbeiten sind, enthält der Auftragspuffer GPUF alle Adressen der zugeordneten Übertragungskontrollsätze UKS. Durch RES jedes UKS ist die gerätespezifische Routine zur Steuerung der Übertragung bekannt. Sie sind in 2.1.2 beschrieben. Nach der Bearbeitung aller Geräterückmeldungen wird GMV = NOPG gesetzt und bei (1) fortgefahren.

NOPG: Es liegen keine Rückmeldungen von E/A-Geräten vor. Fortsetzung bei (2).

(2) Sprung über die Marke ZMV in ZKD nach SEKRD oder NOPD

SEKRD: In der Dauerkette existiert mindestens ein Einplanelement, das zu diesem Zeitpunkt zu bearbeiten ist (siehe: 1.6.2.1). Nach der Ausführung aller für diesen Zeitpunkt eingeplanten Prozeßsteueranweisungen wird ZMV in ZKD gleich NOPD gesetzt und bei (1) fortgefahren.

NOPD: Keine Bearbeitung der Dauerkette eingeplant. Fortsetzung bei (3).

(3) Sprung über die Marke ZMV in ZKU nach SEKRU oder NOPU

SEKRU: In der Uhrzeitkette existiert mindestens ein Einplanelement, daß zu diesem Zeitpunkt zu bearbeiten ist (siehe:1.6.2.1). Nach der Ausführung aller für diesen Zeitpunkt eingeplanten Prozeßsteueranweisungen wird ZMV in ZKU gleich NOPU gesetzt und bei (1) fortgefahren.

NOPU: Keine Bearbeitung der Uhrzeitkette.
Fortsetzung bei (4).

(4) Sprung über die Marke IMV nach SEKE oder NOPE

SEKE: Falls Meldungen der Prozeßperipherie zu bearbeiten sind, enthält der Auftragspuffer PUF alle Adressen der zugeordneten Ereignisvariablen ER. Über den Zeiger ERK in ER werden alle angeketteten Einplanelemente und, falls diese Ereignisvariable als SIGNAL spezifiziert wurde, ein Signal-Beschreibungselement erreicht, um die in CA dieser Datenstrukturen angegebenen Befehlsfolgen auszuführen (siehe:1.6.2.2 und1.6.2.3). Nach der Bearbeitung aller Meldungen der Prozeßperipherie wird IMV = NOPE gesetzt und bei (1) fortgefahren.

NOPE: Es liegen keine Meldungen der Prozeßperipherie vor.
Fortsetzung bei (5).

(5) Der Sekundärreaktionsprozeß wird beendet und der wichtigste lauffähige Prozeß aufgenommen.

1.6.2.1 Sekundärreaktion auf Zeittakte

Funktion: SEKRD Bearbeitung der Dauerkette
Funktion: SEKRU Bearbeitung der Uhrzeitkette

Der Zeiger F in ZKD bzw. ZKU weist auf das ausgezeichnete Element der Dauer- bzw. Uhrzeitkette. Dessen Markenvariable CA wurde bei Aufbau des Einplankontrollsatzes einer der folgenden Werte zugewiesen:

Komponente der Startbedingung	Wert von CA
AT	ATST
AFTER	WAST
EVERY/ALL	WI
UNTIL/DURING	G

(1) Zunächst wird der Nachfolger des ausgezeichneten Elements durch entsprechende Korrekturen des Zeigers F, des Zählers D und des Zeitpunktes NXU in ZKD bzw. ZKU ausgezeichnet.
Danach wird durch Sprung über CA des bisher ausgezeichneten Elements die Ausführung einer der folgenden Befehlsfolgen eingeleitet.

ATST: In OPAUSF wird die durch OP im Kopf des EPS spezifizierte Prozeßsteuerung ausgeführt.
Falls die Startbedingung die Komponenten EVERY/ALL und ggf. UNTIL/DURING enthielt, werden die entsprechenden Einplanelemente bzgl. des aktuellen Zeitpunktes in die Uhrzeit- oder Dauerkette eingeordnet, falls sie schon eingekettet waren - d.h. seit dem Aufbau des EPS sind 24,48,... Stunden vergangen -, werden sie vorher ausgeordnet.
Fortsetzung bei (2).

WAST: Ausführung der Prozeßsteuerung in OPAUSF.
Ausketten des Elements aus der Dauerkette.
Fortsetzung bei (2).

WI: Ausführung der Prozeßsteuerung in OPAUSF.
Ausketten des Elements aus der Dauer- bzw. Uhrzeitkette.
Einordnen des Elements in die Dauer- bzw. Uhrzeitkette an der Stelle, die durch den aktuellen Zeitpunkt und die in D des Elements angegebene Dauer bestimmt ist.
Fortsetzung bei (2).

G: Ausketten des Elements aus der Dauer- bzw. Uhrzeitkette.
Ausordnen des Einplanelements, das aufgrund der Komponente EVERY/ALL existiert.
Fortsetzung bei (2).

(2) Falls auch das neu ausgezeichnete Elemente zu bearbeiten ist (D≥0 in ZKD bzw. ZKU), wird bei (1) fortgefahren. Anderenfalls wird in ZKD bzw. ZKU notiert, daß keine Sekundärreaktion zur Bearbeitung der Dauer- bzw. Uhrzeitkette auszuführen ist.
Fortsetzung bei SEKRE (siehe:1.6.2).

1.6.2.2 Sekundärreaktion auf Meldungen der Prozeßperipherie

Funktion: SEKE

Für alle von der Primärreaktion in den Auftragspuffer PUF eingetragenen Ereignisvariablen ER wird folgender Algorithmus ausgeführt.

(1) Die durch ERK in ER identifizierte Ereigniskette besteht aus Einplanelementen, die aufgrund einer Komponente WHEN existieren, deren Markenvariable den Wert IST bzw. IAST (für WHEN AFTER) hat, und/oder eines Signal-Beschreibungs-Elements, in dessen CA die Marke PSIG notiert ist. Das letzte Element jeder Ereigniskette ist die Datenstruktur EREND mit der Marke SEKEND in CA.
Für alle Elemente der Ereigniskette wird die in CA spezifizierte Befehlsfolge ausgeführt.

IST: Ausführung der Prozeßsteuerung in OPAUSF (siehe ATST). Falls die Startbedingung von der Form WHEN ALL [DURING/UNTIL] war, werden die entsprechenden Einplanelemente bzgl. des aktuellen Zeitpunktes (in ZEIT von ER notiert) in die Dauer- bzw. Uhrzeitkette eingeordnet, falls sie schon eingekettet waren - d.h. seit dem Aufbau des EPS ist die Meldung schon mindestens einmal gegeben worden -, werden sie vorher ausgeordnet.
Fortsetzung mit der Bearbeitung des nächsten Elements der Ereigniskette.

IAST: Die aufgrund der Komponenten AFTER [ALL[UNTIL/DURING]] existierenden Einplanelemente werden in die Zeitketten bzgl. des aktuellen Zeitpunktes (in ZEIT von ER notiert) eingeordnet; dabei wird auf die in U des Elements der Ereigniskette aufgrund der Komponenten WHEN interrupt AFTER dauer abgelegte Dauergröße zugegriffen. Falls diese Elemente schon in den Zeitketten eingekettet waren, werden sie vorher ausgeordnet.
Fortsetzung bei der Bearbeitung des nächsten Elements der Ereigniskette.

PSIG : Die Reaktion auf den Eintritt eines SIGNAL wird im nächsten Abschnitt beschrieben.

SEKEND: Abschluß der Bearbeitung einer Ereigniskette. In der Ereignisvariablen wird notiert, daß die Sekundärreaktion auf die zugeordnete Meldung beendet ist (EW = NOTIERE); der Zähler ZAHL wird auf 0 gesetzt.
Falls der Auftragspuffer PUF leer ist, wird notiert, daß keine Sekundärreaktion auf Meldungen der Prozeßperipherie auszuführen ist (IMV = NOPE).

Fortsetzung bei SEKRE (siehe:1.6.2).

1.6.2.3 Reaktion auf ein Signal

Falls eine Ereignisvariable als SIGNAL spezifiziert wurde, befindet sich in seiner Ereigniskette ERK ein Signal-Beschreibungselement SG, über dessen CA nach PSIG gesprungen wird.

PSIG: (1) Beginnend mit dem aktuellen Aktivierungssatz AS des unterbrochenen Prozesses wird anhand der über SIGK verketteten On-Beschreibungsblöcke RSPB in den Aktivierungssätzen AS untersucht, ob der Prozeß in einer On-Anweisung eine Reaktion vorgeschrieben hat. Sobald ein RSPB mit ID = Adresse von ER gefunden ist, wird bei (2) fortgefahren. Falls keine prozeßspezifische Reaktion definiert ist, wird die in STDR des SG spezifizierte Standardreaktion eingeleitet und danach mit der Bearbeitung des nächsten Elements in der Ereigniskette von ER fortgefahren (siehe: 1.6.2.2).

(2) Folgendermaßen wird notiert, daß der Prozeß, sobald er wieder aufgenommen wird, die in seiner On-Anweisung spezifizierte Befehlsfolge ausführt:

(i) Aufruf der Speicherplatzverwaltung zur Reservierung eines *On-Elements* RSPEL (siehe: 3.3.2).
RSPEL:

KEL	Verkettungszeiger
RAS	Aktueller Befehlszähler
AS	Hinweis auf den aktuellen AS
LZKE	Aktuelles Ende des Laufzeitkellers

(ii) Retten des aktuellen Prozeßstatus aus dem PKS in RSPEL
Befehlszähler: LF aus PKS nach RAS in RSPEL
Aktivierungssatz: AS aus PKS nach AS in RSPEL
Ende des Laufzeitkellers nach LZKE in RSPEL

(iii) Einketten des RSPEL in die durch SGK des PKS identifizierte Kette von On-Elementen.

(IV) Anhand des in (1) gefundenen On-Beschreibungselements RSPB wird ein neuer Prozeßstatus definiert.
Befehlszähler: AD aus RSPB nach LF in PKS
Aktivierungssatz: AS aus RSPB nach AS in PKS

(V) Fortsetzung mit der Bearbeitung des nächsten Elements der Ereigniskette von ER (siehe: 1.6.2.2).

1.6.3 Prozeßspezifische Reaktion auf ein Signal

Im folgenden wird ein Rechenprozeß betrachtet, für den in der Sekundärreaktion (1.6.2.3) notiert wurde, daß ein SIGNAL gegeben wurde, für das er in einer On-Anweisung eine Reaktion spezifiziert hatte (1.5).
Nach seiner Aufnahme als wichtigster lauffähiger Prozeß führt er die in der entsprechenden On-Anweisung spezifizierte Anweisungsfolge aus. Aktivierungssätze der hier aufgerufenen Prozeduren werden seinem Laufzeitkeller an der Stelle zugefügt, die in LZKE des On-Elements RSPEL notiert ist.

Falls die Anweisungsfolge nicht mit einem Sprungbefehl verlassen wird, ruft er danach RESPND auf. Das Verlassen durch einen Sprung wird durch RSPAUS verwaltet.

Funktion: RESPND

Die Zeigervariable SGK im PKS des Prozesses weist auf das On-Element RSPEL, in das der bei der Meldung bestehende Prozeßstatus übertragen wurde.
Diese Daten werden in den PKS übertragen
Befehlszähler: RAS aus RSPEL in LF des PKS
Aktivierungssatz: AS aus RSPEL in AS des PKS
Danach wird RSPEL aus der Kette der On-Elemente entfernt und der von ihm belegte Speicherplatz freigegeben.
Der Prozeß wird, sobald er aufgenommen ist, an der Stelle seiner Befehlsfolge fortgesetzt, die er bei Eintritt des Signals erreicht hatte.

Funktion: RSPAUS Parameter: Markenbeschreibung

Eine Markenbeschreibung MB (siehe: 3.4.2.1.3) enthält die Adresse der Marke, einen Hinweis auf den Aktivierungssatz der Prozedur und die Schachtelungsstufe des Blocks, in dem die Marke lokal ist.
Der Laufzeitkeller LZK des aufrufenden Prozesses wird bis zu dem Block in dem Aktivierungssatz, der in der als Parameter übergebenen Markenbeschreibung spezifiziert ist, freigegeben.

Die zu den abgebauten Aktivierungssätzen gehörenden On-Elemente RSPEL werden ausgekettet und freigegeben.

Der Prozeß wird, sobald er aufgenommen ist, an der Stelle seiner Befehlsfolge fortgesetzt, die in der Markenbeschreibung spezifiziert ist.

1.6.4 Die Trigger-, Induce-, Disable- und Enable-Anweisungen

Die Anweisung

TRIGGER Interrupt;

wird intern durch

CALL TRIGG || ADDR Ereignisvariable ER, die Interrupt zugeordnet ist

dargestellt.

Funktion: TRIGG, Parameter: Adresse eines ER

(1) In INH wird verhindert, daß aufgrund einer Meldung die Primärreaktion eingeleitet wird.
In RRETT werden der Befehlszähler und die Registerinhalte in den PKS des aufrufenden Prozesses übertragen.
Fortsetzung bei (2).

(2) Die Funktion wird durch den Sprung auf die in EW der angegebenen Ereignisvariablen ER notierte Marke verlassen.
Danach wird wie nach einer Meldung der Prozeßperipherie fortgefahren (siehe: 1.6.1.2).

Die Anweisung

INDUCE Signal;

wird intern durch

CALL INDUCE || ADDR Ereignisvariable ER, die Signal zugeordnet ist

dargestellt.

Funktion: INDUCE, Parameter: Adresse eines ER

(1) Beginnend mit dem aktuellen Aktivierungssatz AS des aufrufenden Prozesses wird anhand der über SIGK verketteten On-Beschreibungsblöcke RSPB in den AS untersucht, ob der Prozeß in einer On-Anweisung eine Reaktion vorgeschrieben hat.

Sobald ein RSPB gefunden ist, dessen ID mit dem aktuellen Parameter übereinstimmt, wird bei (2) fortgefahren.
Falls keine prozeßspezifische Reaktion definiert ist, wird die in STDR des Signal-Beschreibungselements SG der Ereigniskette von ER angegebene Standardreaktion ausgeführt.

(2) Ausführung des in 1.6.2.3 (2) beschriebenen Algorithmus.

Die Anweisungen

{DISABLE / ENABLE} Interrupt;

werden intern durch

CALL {DISB / ENAB} || ADDR Ereignisvariable ER, die Interrupt zugeordnet ist

dargestellt.

Funktion: DISB|ENAB , Parameter: Adresse eines ER
Zuweisung von DSBL|NOTIER an EW in ER.

1.7 PROZESS-STEUERUNG

Die Steuerung von Prozessen geschieht aufgrund von *Prozeßsteueranweisungen* im Anwenderprogramm.
Prozeßsteueranweisungen können mit *Startbedingungen* (schedule) versehen werden.
Die Bearbeitung von Prozeßsteueranweisungen wird in zwei Phasen ausgeführt:

- Beim *Überlaufen* der Prozeßsteueranweisung in der sequentiellen Abarbeitung der Anweisungen werden, falls vorhanden, die Startbedingungen eingeplant und evtl. einige vorbereitende Arbeiten ausgeführt.

- Die eigentliche *Ausführung* der Prozeßsteueranweisung wird erst zu einem Zeitpunkt durchgeführt, wenn eine Startbedingung erfüllt ist.
  (Falls keine Startbedingung angegeben ist, fällt der Zeitpunkt der Ausführung mit dem Zeitpunkt des Überlaufens zusammen.)

Solange ein Prozeß *bekannt* ist, existiert für ihn ein *Prozeßkontrollsatz* PKS, und im Aktivierungssatz des Blocks, in dem der Prozeß mit dem Attribut TASK oder TASKNAME vereinbart ist, ist ein Hinweis auf den PKS abgelegt.

Den Programmen zur Ausführung von Prozeßsteueranweisungen wird ein Hinweis auf den PKS des betroffenen Prozesses als Parameter übergeben. Sie interpretieren und ändern die im PKS abgelegten Informationen.
Solange ein Prozeß *aktiv* ist, d.h. zwischen der Ausführung einer Activate-Anweisung und des letzten Befehls seines Codes oder einer Terminate-Anweisung, steht sein PKS in der *Prioritätskette* hinter den PKS aller wichtigeren Prozesse. Ein aktiver Prozeß ist entweder lauffähig oder zurückgestellt (blockiert, im Wartezustand). Dem wichtigsten lauffähigen Prozeß wird, falls keine Reaktion auf Meldungen durchzuführen ist, die Zentraleinheit zugeteilt, er wird als *laufend* bezeichnet.

Durch Ausführung einer ACTIVATE-Anweisung wird der betroffene Prozeß *lauffähig* und unter folgenden Bedingungen wird er *zurückgestellt:*

. Ausführung einer Suspend-Anweisung für ihn,

. Notieren einer Resume-Anweisung für ihn,

. Erreichen des Endes eines Blocks, in dem das Segment eines Prozesses eingeschachtelt ist, der noch aktiv ist oder für den die Ausführung einer Activate-Anweisung eingeplant ist,

. Zugriff auf eine gesperrte Synchronisier-Variable,

. unerfüllbare Speicherplatzanforderungen,

. Ausführung einer E/A-Anweisung.

Im PKS wird jeweils notiert, aufgrund welcher Bedingung der Prozeß zurückgestellt wurde, darüberhinaus wird er ggf. in die Warteschlange eines Sema-, Bolt-, File- oder Übertragungskontrollsatzes eingekettet.
Der Prozeß ist erst wieder lauffähig, wenn alle Sperrbedingungen aufgehoben sind, durch

. Ausführung einer Continue-Anweisung für ihn,

. Meldung, daß eine der in der Resume-Anweisung angegebenen Startbedingungen erfüllt ist,

. Beenden aller Unterprozesse, die dem Block zugeordnet sind, den er verlassen will,

. Freigabe der sperrenden Synchronisier-Variablen,

. Freigabe von Speicherplatz,

. Beenden einer Übertragung.

1.7.1 Der Prozeßkontrollsatz PKS

CA	Identifizierung eines Befehls: entweder Aufnehmen des Prozesses oder Ausführen des CA im nachfolgenden PKS
PN	Hinweis auf den nachfolgenden PKS der Prioritätskette
SPCA	Für Speicherplatzverwaltung reserviert
PV	Hinweis auf den vorhergehenden PKS der Prioritätskette
VN	Hinweis auf einen PKS eines Prozesses mit vergleichbarer Priorität
P	Priorität
VV	wie VN
LF	Laufender Befehl
AC	Accumulator
AS	Hinweis auf den aktuellen Aktivierungssatz AS
SN	Hinweis auf einen PKS, Warteschlange
ABSP	Absolutpriorität
SV	wie SN
SK	Hinweis auf die Kette der Einplankontrollsätze EPS
SZ	Prozeßstatus
B	Hinweis auf einen PKS in der Kette der demselben Prozeß untergeordneten Prozesse
V	Hinweis auf den PKS des übergeordneten Prozesses
BL	Hinweis auf einen Blockzähler
KA	Hinweis auf den PKS eines untergeordneten Prozesses mit Relativpriorität
K	Hinweis auf die Kette der untergeordneten Prozesse
KE	wie KA
SEG	Segmentadresse, Anfang der Befehlsfolge des Prozesses

SGK	Hinweis auf die Kette der On-Elemente
SEMA	Hinweis auf einen Sema-Kontrollsatz, wenn USING Sema
PZ	Anzahl der gepufferten Aktivierungen
SP	Für Speicherplatzverwaltung reserviert
ANZS	Anzahl fehlender Seiten

Aus Optimierungsgründen wird für einen Prozeß nur ein *kurzer* PKS - von CA bis BL - verwendet, wenn zur Übersetzungszeit festgestellt wurde, daß er folgende Bedingungen erfüllt:

. keine Unterprozesse,

. kein eigener Laufzeitkeller,

. keine On-Anweisung,

. keine Aktivierung mit USING Sema,

. keine Activate-Anweisung mit einer Startbedingung, die zur Pufferung von Aktivierungen führen könnte.

Ungekürzte PKS werden als *normal* bezeichnet.

Die *Status-Zelle* SZ stellt in 16 Bits den Prozeßzustand dar. In der Reihenfolge von höhren zu tieferen Bits (2^{15} bis 2^{0}) haben die in den einzelnen Positionen gesetzten Bits folgende Bedeutung:

ACTBIT: Der Prozeß ist aktiv
SUSK: Der Prozeß ist durch SUSPEND zurückgestellt
RSUMK: Der Prozeß ist durch RESUME zurückgestellt
TERMK: Der Prozeß soll aufgrund von TERMINATE beendet werden
WSPBIT: Der Prozeß wartet auf Speicherplatzvergabe
KPKSK: Kurzer PKS
WUPK: Der Prozeß wartet auf das Ende eines Unterprozesses

WAC:	Der Prozeß wartet auf das Ende eines neu zu aktivierenden Prozesses
NWBIT:	Für Speicherplatzverwaltung reserviert
TK:	Der Prozeß ist einer Task-Konstanten (TASK) zugeordnet
TERM:	TERMINATE ist eingeplant
ACT:	ACTIVATE ist eingeplant
REPV:	PREVENT ist eingeplant
SUSP:	SUSPEND ist eingeplant
CONT:	CONTINUE ist eingeplant
RSUM:	RESUME ist eingeplant

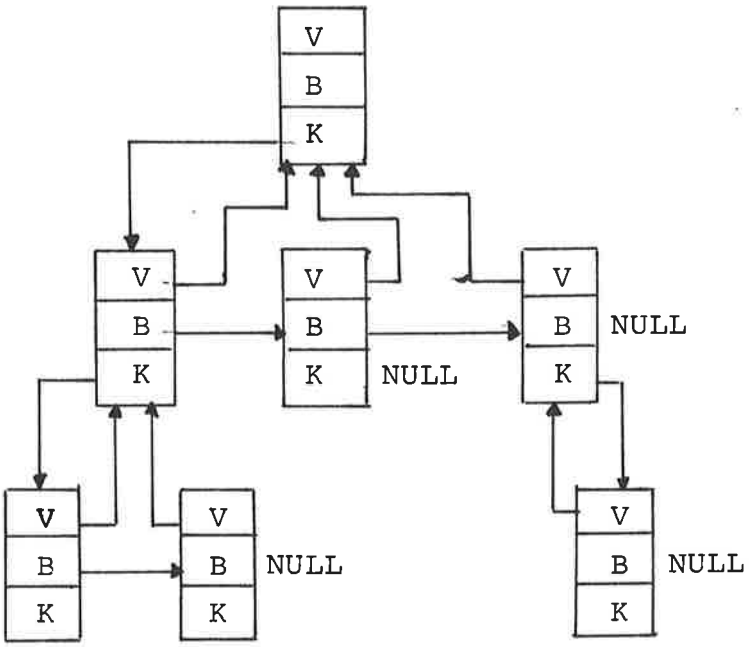
1.7.2 Hierarchie der aktiven Prozesse

Die aufgrund der Blockstruktur des PEARL-Programms bestehende Abhängigkeit (mastertask, subtask) von Prozessen wird intern durch eine entsprechende Verzeigerung der Prozeßkontrollsätze dargestellt. Für den PKS eines aktiven Prozesses gilt (siehe: 1.7.5):

V weist auf den PKS des übergeordneten Prozesses (mastertask).

B weist auf den PKS eines Prozesses, der demselben Prozeß untergeordnet ist oder enthält den leeren Hinweis NULL als Endekennung der Kette der untergeordneten Prozesse.

K weist auf den PKS eines untergeordneten Prozesses (subtask) oder enthält den leeren Hinweis NULL, wenn keine untergeordneten Prozesse existieren bzw. aktiv sind. (K wird nur in normalen PKS geführt.)

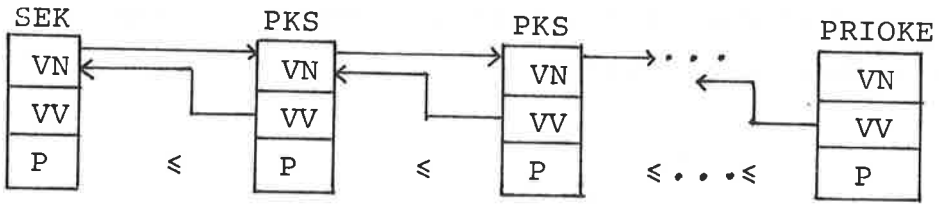


1.7.3 Priorität von Prozessen

Für alle aktiven Prozesse ist entweder durch explizite Angabe einer Priorität in den Activate- oder Continue-Anweisungen oder durch Einsetzen eines implementationsabhängigen Wertes für fehlende Angaben in Activate-Anweisungen eine Wichtiger-Unwichtiger-Relation definiert.

Die Prioritätsangabe kann eine Relativ- oder Systempriorität definieren.
Vergleichbar bezüglich ihrer Priorität sind einerseits Prozesse mit Systempriorität untereinander, andererseits Prozesse, die demselben Prozeß direkt untergeordnet sind, untereinander und mit ihrem direkt übergeordneten Prozeß, für den für diese Betrachtung die Relativpriorität 0 angenommen wird. Prozesse mit vergleichbarer Priorität sind in Vergleichsketten zusammengefaßt.

Vergleichskette der Prozesse mit Systempriorität:

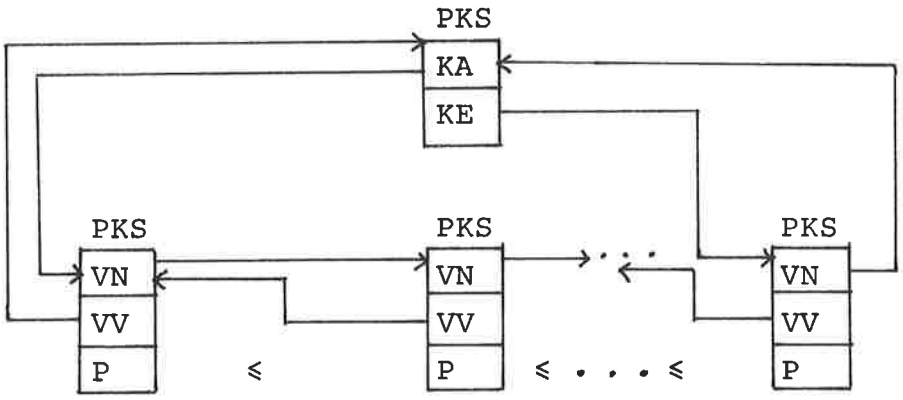


VN und VV bilden eine Vorwärts-rückwärtsverzeigerung über die Prozesse mit Systempriorität; diese Kette ist nach aufsteigenden Werten von P geordnet. Das erste und letzte Element der Kette sind die PKS der Systemprozesse SEK bzw. PRIOKE.

P in SEK und PRIOKE sind mit 2**14 bzw. 2**15-1 initialisiert.

P in PKS enthält den um 2**14 erhöhten Wert der in der Activate- bzw. Continue-Anweisung angegebenen Priorität.

Vergleichskette von Prozessen mit Relativpriorität:



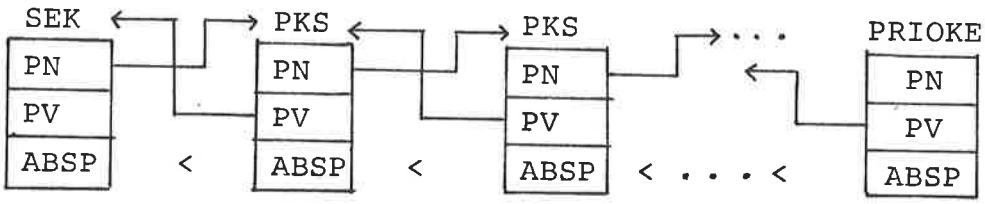
KA und KE zeigen auf den Anfang bzw. das Ende der Kette von direkt untergeordneten Prozessen mit Relativpriorität.

VN und VV bilden eine Vorwärts-rückwärtsverzeigerung über die Prozesse mit vergleichbarer Priorität; diese Kette ist nach aufsteigenden Werten von P geordnet.

P enthält den Wert der in der Activate- bzw. Continue-Anweisung angegebenen Priorität.

Zur Verwaltung im System wird die Prioritätskette über alle PKS aufgebaut.

Prioritätskette



PN, PV bilden eine Vorwärts-rückwärtsverzeigerung über alle Prozesse; diese Kette ist nach aufsteigenden Absolutprioritäten in ABSP geordnet; das erste und letzte Element der Kette sind die PKS der Systemprozesse SEK bzw. PRIOKE.

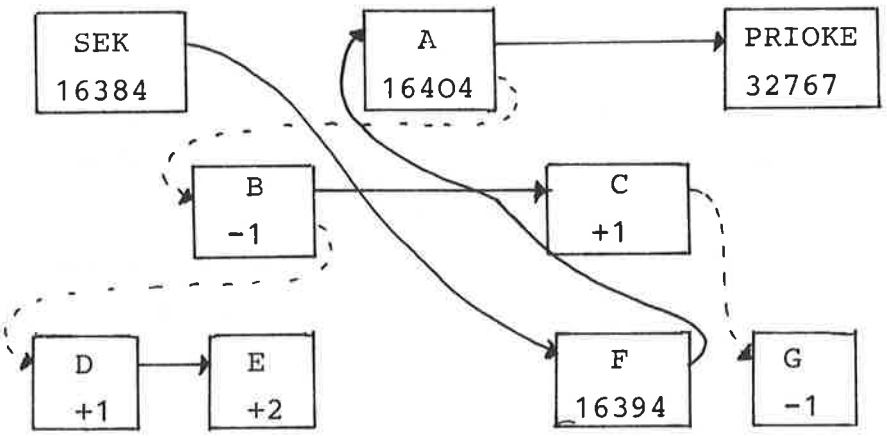
In der Prioritätskette sind die PKS der Prozesse mit Systempriorität in der gleichen Reihenfolge enthalten wie in der Vergleichskette für Systemprioritäten, wobei die PKS der Prozesse mit Relativpriorität dazwischen eingefügt sind und zwar so, daß die PKS einer Vergleichskette mit Relativpriorität entsprechend dieser um den PKS des übergeordneten Prozesses angeordnet sind.
Allen PKS ist gemäß ihrer Reihenfolge in der Prioritätskette eine Absolutpriorität ABSP zugeteilt.
ABSP von SEK ist 2^{14}
ABSP von PRIOKE ist $2^{15}-1$
Die dazwischenliegenden PKS haben in ABSP Werte aus dem Intervall $2^{14}+1$ bis $2^{15}-2$.

Beispiel:

In folgender Reihenfolge werden Prozesse aktiviert:

Prozeß	übergeordneter Prozeß	Priorität
A	(von Bedienung gestartet)	20 SYS
B	A	-1 REL
C	A	+1 REL
D	B	+1 REL
E	B	+2 REL
F	C	10 SYS
G	C	-1 REL

dadurch ergibt sich folgendes Bild:



Die Hierarchie der Prozesse ist durch die Anordnung im Bild wiedergegeben.
Die durchgezogenen Pfeile geben die Ketten vergleichbarer Priorität wieder.
Die in die Kästen eingetragenen Zahlen sind die Prioritäten P.

Die Prioritätskette läuft über die PKS in folgender Reihenfolge:

SEK, F, B, D, E, A, G, C, PRIOKE.

1.7.4 Abschnitte der Prioritätskette

Da PKS von Unterprozessen mit Relativpriorität in der Prioritätskette um den PKS des übergeordneten Prozeß herum angeordnet sind, bildet der PKS eines Prozesses zusammen mit den PKS aller direkt oder indirekt untergeordneten Prozesse einen zusammenhängenden *Abschnitt* der Prioritätskette.

Der erste und letzte PKS eines Abschnitts, der zu einem bestimmten Prozeß gehört, wird durch die Funktionen WIP bzw. UWP bestimmt.

Funktion: WIP/UWP

Parameter: Hinweis auf den PKS des Prozesses, dessen Abschnittsgrenze zu suchen ist.

Durchführung:
(von durch / getrennten Alternativen gilt die erste für die Funktion WIP, die zweite für die Funktion UWP)

(1) der betrachtete PKS ist der durch Parameter übergebene

(2) hat der betrachtete Prozeß keine Unterprozesse (kurzer PKS) oder keine Unterprozesse mit Relativpriorität (KA weist auf sich selbst), ist die Suche beendet und als Ergebnis wird der Hinweis auf den gerade betrachteten PKS geliefert.

(3) Der in KA/KE angezeigte PKS wird nunmehr zum betrachteten PKS.

(4) Ist die Priorität (P) im betrachteten PKS positiv/negativ, ist die Suche beendet und als Ergebnis wird der Hinweis auf den PKS des dem betrachteten Prozeß übergeordneten Prozesses geliefert (V aus PKS).
Ansonsten Fortsetzung bei (2)

1.7.5 Ein- und Ausordnen in Vergleichs- und Prioritätskette

Eingeordnet wird ein PKS beim Überlaufen einer ACTIVATE-Anweisung und bei Ausführung einer CONTINUE-Anweisung mit Prioritätsangabe.

Funktion: EINORD
Vor dem Aufruf von EINORD sind in systembekannten Zeigervariablen die Hinweise auf die PKS des in der Anweisung genannten Prozesses, des unwichtigsten und des wichtigsten Prozesses seines Abschnitts abgelegt. (Beim Aufruf aus der Aktivierung zeigen alle drei Zeiger auf den einzuordnenden PKS, da der Prozeß noch keine Unterprozesse haben kann.)

(1) Bei Relativpriorität wird der PKS in die Vergleichskette beim PKS seines übergeordneten Prozesses, bei Systempriorität wird er in die Vergleichskette der Prozesse mit Systempriorität als Vorgänger aller PKS mit größerer Priorität (P in PKS) eingeordnet.

(2) Der Abschnitt des Prozesses wird zwischen den Abschnitten seines Vorgängers und Nachfolgers in der Vergleichskette in die Prioritätskette eingeordnet, dabei ist zu beachten, daß bei Relativpriorität, falls sie negativ ist und kein Nachfolger in der Vergleichskette mit negativer Priorität existiert, der Abschnitt unmittelbar vor dem PKS des übergeordneten Prozesses in die Prioritätskette eingeordnet wird, und falls sie positiv ist und kein Nachfolger in der Vergleichskette existiert, der Abschnitt hinter dem (bisherigen) Abschnitt des übergeordneten Prozesses in die Prioritätskette eingeordnet wird.

(3) Für alle PKS des Abschnitts wird die Absolutpriorität ermittelt und in ABSP eingetragen: Dabei wird der ganzzahlige Teil des arithmetischen Mittels der Absolutprioritäten des unmittelbaren Vorgängers und Nachfolgers des Abschnitts gebildet, und davon ausgehend werden den PKS des Abschnitts paarweise verschiedene Zahlen zugeordnet.

Falls nicht genügend (noch nicht vergebene) Zahlen zur Verfügung stehen, werden die ABSP in sovielen PKS der angrenzenden Abschnitte modifiziert, bis die ABSP aller PKS der Prioritätskette paarweise verschieden sind.

Bei Ausführung einer Terminate-Anweisung werden die PKS aller betroffenen Prozesse einzeln aus ihrer Vergleichs- und aus der Prioritätskette genommen.
Bei Ausführung einer Continue-Anweisung werden, falls die Priorität geändert wurde, der PKS des genannten Prozesses aus seiner Vergleichskette und sein Abschnitt aus der Prioritätskette genommen.

1.7.6 Anlegen von Prozeßkontrollsätzen

Die interne Befehlsfolge zur Eröffnung eines Blocks enthält den Aufruf

```
CALL PKSINI  || Liste von:
             || VALUE   Art der Task,
```

wenn Taskvereinbarungen auszuwerten sind. In der Parameterliste ist für jede Task eine der Zahlen 1 bis 4 angegeben:

1 für Task-Variable, normaler PKS
2 für Task-Variable, kurzer PKS
3 für Task-Konstante, normaler PKS
4 für Task-Konstante, kurzer PKS

Funktion: PKSINI

Für jedes Element der Parameterliste wird

- Speicherplatz für einen normalen (1,3) bzw. kurzen (2,4) PKS reserviert, und dessen Adresse in der ersten freien Zelle des aktuellen Aktivierungssatzes des laufenden Prozesses eingetragen,
- bei Task-Konstanten (3,4) der Hinweis auf den PKS des laufenden Prozesses in V, der Hinweis auf den aktuellen Aktivierungssatz des laufenden Prozesses in AS und der Hinweis auf die Zelle Z in BD des aktuellen Blocks des laufenden Prozesses in BL des neuen PKS abgelegt,
- der Status-Zelle SZ ein geeigneter Initialwert und SK der leere Hinweis NULL zugewiesen.

Für die auf Modul-Ebene vereinbarten Tasks wird je ein

```
CALL MPINI  || ADDR   Prozeßkontrollsatz
```

abgesetzt, dabei ist die Adresse des zur Bindezeit für den PKS reservierten Speicherplatzes angegeben.
In MPINI werden AS, V und SK mit dem leeren Hinweis NULL und SZ mit einem geeigneten Wert initialisiert.

1.7.7 Starten eines Prozesses, die Activate-Anweisung

Die Anweisung

ACTIVATE Taskname [PRIORITY Priorität{REL/[SYS]}][USING Sema];

wird intern dargestellt durch

CALL ACTT	ADDR	PKS, der Taskname zugeordnet ist
	ADDR	Wert von Priorität
	ADDR	Sema-Kontrollsatz, der Sema zugeordnet ist oder leerer Hinweis NULL
	ADDR	Segment des Prozesses

falls der Prozeß einen normalen PKS besitzt oder

CALL ACTKT	ADDR	PKS, der Taskname zugeordnet ist
	ADDR	Priorität
	ADDR	Segment des Prozesses

falls der Prozeß einen kurzen PKS besitzt.

Es wird vorausgesetzt, daß im Programm bei *Prioritätsangaben* nur Werte zwischen -2**14+1 und 2**14-1 verwendet werden. Bei Relativprioritäten wird in Wert von Priorität die Angabe des Programms angegeben,und bei Systemprioritäten wird in Wert von Priorität die um 2**14 erhöhte Angabe des Programms abgelegt. Wenn die Anweisung keine Priorität spezifiziert, wird zur Übersetzungszeit ein Wert eingesetzt.

Im folgenden werden alle aufgrund einer ACTIVATE-Anweisung auszuführenden Programme beschrieben:

Die Funktionen ACTNP, ACTKP werden bei Überlaufen einer Activate-Anweisung mit Startbedingungen, d.h. unter Regie des aufrufenden Prozesses, ausgeführt.
Sobald eine der Startbedingungen erfüllt ist, wird im Rahmen der Sekundärreaktion die Funktion SCANP oder SCAKP ausgeführt.

Die Funktionen ACTT, ACTKT werden bei Überlaufen einer Activate-Anweisung ohne Startbedingungen, d.h. unter Regie des aufrufenden Prozesses ausgeführt.

Der Laufzeitkeller für den neuen Prozeß wird erst eingerichtet, wenn er als wichtigster lauffähiger Prozeß aufgenommen wird.

Funktion: ACTNP
Funktion: ACTKP
Funktion: ACTT
Funktion: ACTKT

(1) . Falls der betroffene Prozeß aktiv ist: Fortsetzung bei (2).
. Falls für den Prozeß eine Terminate-Anweisung bis auf den Teil, der unter seiner eigenen Regie läuft, ausgeführt wurde, wird bei (2) fortgefahren, wenn er auf das Ende von Unterprozessen wartet, sonst wird zunächst seine Terminierung beendet bevor bei (3) fortgesetzt wird.
. Anderenfalls Fortsetzung bei (3).

(2) Im Rahmen von ACTNP und ACTKP wird der neu aufgebaute Einplankontrollsatz EPS aufgegeben, indem in OP seines Kopfes notiert wird, daß aufgrund von Meldungen keine Prozeßsteuerung auszuführen ist und indem er in die Kette der freizugebenden EPS aufgenommen wird.
Im Rahmen aller Funktionen wird eine Fehlermeldung ausgegeben und in die Fehlerroutine FHLEND gesprungen.

(3) In der Prozedur TESTA wird geprüft, ob für den Prozeß noch ein EPS existiert, in dem seine Aktivierung notiert ist; dieser EPS wird aufgegeben (siehe (2)).

(4) Im Rahmen von ACTNP und ACTT wird der von KA bis ANZS reichende Teil des PKS initialisiert:
In K,KA und KE wird notiert, daß keine Unterprozesse existieren.
In SGK und SP wird der leere Hinweis NULL eingetragen.
PZ erhält den Wert O, d.h. keine Aktivierung ist gepuffert.
In ANZS wird O eingetragen.
In SEMA wird der übergebene Parameter abgelegt, dh. die Adresse des über USING Sema spezifizierten Sema-Kontrollsatzes oder der leere Hinweis NULL.
In SEG wird die als Parameter übergebene Anfangsadresse des Segments abgelegt.
Im Rahmen von ACTKP und ACTKT wird die Anfangsadresse des Segments in LF des PKS übertragen.

(5) Falls in TESTA (siehe (3)) ein EPS für Aktivierung aufgegeben wurde, wird nun in der Prozedur KSPA folgendes ausgeführt:

. Der PKS wird aus der Vergleichs- und Prioritätskette und aus der Kette der demselben Prozeß untergeordneten Prozesse entfernt.

. Der in BL des PKS angezeigte Blockzähler wird um 1 vermindert; falls der Zähler dabei = 0 wird, wird der übergeordnete Prozeß (V in PKS) aus dem Wartezustand auf Unterprozesse befreit und evtl. lauffähig gemacht.

(6) Wenn der Prozeß als Taskvariable (TASKNAME) vereinbart ist, werden in AS,V und BL des PKS die Adressen des aktuellen Aktivierungssatzes des PKS bzw. der Zelle Z des aktuellen Blockes des laufenden Prozesses eingetragen.

(7) Der in BL des PKS angezeigte Blockzähler wird um eins erhöht.
Der PKS wird in die Kette der Unterprozesse seines übergeordneten Prozesses eingeordnet.
In P wird die als Parameter übergebene Priorität eingetragen.
Der PKS wird in die Prioritätshierarchie (Vergleichskette, Prioritätskette) eingeordnet, dabei wird seine Absolutpriorität berechnet und in ABSP abgelegt (siehe 1.7.5).

(8) Im Rahmen von ACTNP und ACTKP wird der Prozeß in CA seines PKS als nicht lauffähig gekennzeichnet und mit der weiteren Bearbeitung des neu aufgebauten EPS fortgefahren (siehe 1.4.4)

(9) Im Rahmen von ACTT wird der Pufferungszähler für Aktivierungen (PZ im PKS) auf 1 gesetzt.
ACTT wird bei (11) und ACTKT bei (12) fortgesetzt.

Funktion: SCANP

Als Parameter wird ein Hinweis auf den PKS des Prozesses übergeben, der aufgrund einer Meldung zu aktivieren ist.

(10) Wenn die Aktivierung aufgrund einer Meldung der Prozeßperipherie auszuführen ist, wird der Zähler PZ im PKS um den in ZAHL der entsprechenden Ereignisvariablen ER Wert plus 1 erhöht, wenn eine zeitliche Startbedingung erfüllt ist, wird er um 1 erhöht.
Falls der Prozeß aktiv ist, wird mit der weiteren Ausführung der Sekundärreaktion, sonst bei (11) fortgefahren.

(11) In SZ des PKS wird notiert, daß der Prozeß aktiv ist.
Wenn in der Activate-Anweisung USING SEMA benutzt wurde, d.h. in SEMA des PKS steht der Hinweis auf einen Sema-Kontrollsatz (siehe: (4)), wird auf diesen Sema-Kontrollsatz zugegriffen: Falls dessen Zähler S den Wert O hat, wird der Prozeß zurückgestellt, um bei der Freigabe der Sema diesen Zugriff zu wiederholen, anderenfalls wird der Zähler um 1 erniedrigt, in LF des PKS die in SEG abgelegte Anfangsadresse des Segments übertragen und in CA notiert, daß der Prozeß lauffähig ist.
Danach wird im Rahmen SCANP mit der weiteren Ausführung der Sekundärreaktion fortgefahren und im Rahmen ACTT entweder die Sekundärreaktion auf inzwischen eingetroffene Meldungen eingeleitet oder der wichtigste lauffähige Prozeß aufgenommen.

Funktion: SCAKP

Als Parameter wird ein Hinweis auf den PKS des Prozesses übergeben, der aufgrund einer Meldung zu aktivieren ist.

(12) In SZ wird notiert, daß der Prozeß aktiv ist und in CA, daß er lauffähig ist.
Danach wird wie in (11) entweder mit der weiteren Ausführung der Sekundärreaktion oder mit der Einleitung der Sekundärreaktion fortgefahren oder der wichtigste lauffähige Prozeß aufgenommen.

1.7.8 Die Suspend-Anweisung

Die Anweisung

```
SUSPEND [task-name][EXCEPT task-name,..];
```

wird intern dargestellt durch

```
CALL SUSPTA  || ADDR   PKS, der task-name zugeordnet ist
             || ADDR   Except-Liste
```

falls 'task-name' spezifiziert ist oder durch

```
CALL SUSPLT  || ADDR   Except-Liste
```

falls kein 'task-name' spezifiziert ist.

Falls die EXCEPT-Option nicht benutzt ist, wird als Parameter der Hinweis auf die Konstante 0 übergeben.
Falls die EXCEPT-Option benutzt ist, wird als Parameter der Hinweis auf die folgendermaßen aufgebaute Except-Liste übergeben.

Anzahl (n≥1) der angegebenen 'task-namen'
Hinweis auf den PKS, der dem ersten 'task-namen' zugeordnet ist
:
Hinweis auf den PKS, der dem n-ten 'task-namen' zugeordnet ist

Die Funktionen SUSPTA und SUSPLT werden bei Überlaufen der Anweisung aufgerufen.
Sobald eine der Startbedingungen erfüllt ist, unter denen eine Suspend-Anweisung auszuführen ist, wird im Rahmen der Sekundärreaktion SCSUSP aufgerufen.

Funktion: SUSPTA
Funktion: SCSUSP

(1) Falls der Prozeß nicht aktiv ist oder falls er aufgrund einer Terminate-Anweisung mit seiner Beendigung beauftragt ist, wird die Systemfunktion nicht weiter ausgeführt.
Anderenfalls Fortsetzung bei (2).

Funktion: SUSPLT

(2) Notieren, daß im Rahmen des Programms EXCEPT die Prozeßsteuerungsfunktion SUSPEN auszuführen ist.
Ausführung von EXCEPT (1.7.13).

In EXCEPT werden alle betroffenen Prozesse identifiziert, und für jeden wird SUSPEN ausgeführt.

Funktion: SUSPEN

Falls der Prozeß aufgrund einer Suspend-Anweisung zurückgestellt ist, Fortsetzung des EXCEPT-Programms.
Anderenfalls wird in SZ des PKS notiert, daß der Prozeß aufgrund einer Suspend-Anweisung zurückgestellt ist und in CA des PKS, daß der Prozeß nicht lauffähig ist.
Fortsetzung des EXCEPT-Programms.

1.7.9 Die Continue-Anweisung

Die Anweisungen

(a) CONTINUE task-name [EXCEPT task-name,''];

(b) CONTINUE task-name PRIORITY Priorität {REL|[SYS]}
[EXCEPT task-name,''];

(c) CONTINUE PRIORITY priorität {REL|[SYS]}
[EXCEPT task-name];

werden intern dargestellt durch

(a) CALL CONTTA
- ADDR PKS, der task-name zugeordnet ist
- ADDR Except-Liste

(b) CALL CPRIOT
- ADDR PKS
- ADDR Except-Liste
- ADDR Wert von Priorität

(c) CALL CPRIOL
- ADDR Except-Liste
- ADDR Wert von Priorität

Für die Prioritätsangaben gelten die in 1.7.7 angegebenen Regeln. Die Struktur der Except-Liste ist in 1.7.8 beschrieben.
Die Funktionen CONTTA, CPRIOT und CPRIOL werden bei Überlaufen der Anweisung aufgerufen.
Sobald eine der Startbedingungen erfüllt ist, unter denen eine Continue-Anweisung auszuführen ist, wird im Rahmen der Sekundärreaktion SCTPR (mit Prioritätsangabe) oder SCONT aufgerufen.

Ohne Prioritätsangabe
Funktion: CONTTA
Funktion: SCONT

Falls der Prozeß nicht aktiv ist oder falls er aufgrund einer Terminate-Anweisung mit seiner Beendigung beauftragt ist, wird die Systemfunktion beendet.

Notieren, daß im Rahmen des Programms EXCEPT die Prozeßsteuerungsfunktion CONTIN auszuführen ist.
Ausführung von EXCEPT (1.7.13).

Mit Prioritätsangabe

Funktion: CPRIOT

Funktion: SCPTR

Falls der Prozeß nicht aktiv ist oder falls er aufgrund einer Terminate-Anweisung mit seiner Beendigung beauftragt ist, wird die Systemfunktion nicht weiter ausgeführt.
Anderenfalls Ausführung von EXCEPT mit CONTIN.

Funktion: CPRIOL

Falls die neue Priorität nicht mit der alten in P des PKS eingetragenen übereinstimmt, wird sie in P eingetragen, der wichtigste und der unwichtigste Prozeß des Abschnitts in der Prioritätskette (siehe: 1.7.4 WIP und UWP) werden bestimmt, der PKS des Prozesses und sein Abschnitt werden aus der Vergleichs- bzw. Prioritätskette genommen und durch EINORD (siehe: 1.7.4) neu eingeordnet.
In jedem Fall Ausführung von EXCEPT mit CONTIN.

Funktion: CONTIN (aufgerufen vom EXCEPT-Programm)

(1) In SZ des PKS wird notiert, daß der Prozeß nicht aufgrund einer Suspend-Anweisung zurückgestellt ist.
Fortsetzung bei (2).

(2) Falls der Prozeß auf die Freigabe einer Synchronisier-Variablen oder eines Betriebsmittels für Ein-Ausgabe (File-, Datei- oder Geräte-Kontrollsatz) steht, Fortsetzung des EXCEPT-Programms.
Anderenfalls Fortsetzung bei (3).

(3) Falls der Prozeß aufgrund einer Resume-Anweisung noch zurückgestellt ist,
Fortsetzung des EXCEPT-Programms.
Anderenfalls Fortsetzung bei (4).

(4) Falls für die Fortsetzung des Prozesses die Eingabe von Segmenten (siehe: Verwaltung des Hintergrundspeichers) nötig ist, wird in CA des PKS notiert, daß der Prozeß, sobald er als wichtigster aufgenommen wird, zunächst diese Segmente anfordert.
Fortsetzung des EXCEPT-Programms.
Falls alle Segmente des Prozesses im Arbeitsspeicher sind, wird in CA des PKS notiert, daß mit der Bearbeitung seiner Befehlsfolge fortgefahren werden kann.
Fortsetzung des EXCEPT-Programms.

1.7.10 Die Terminate-Anweisung

Die Anweisungen

(a) TERMINATE task-name [EXCEPT taskname,...];

(b) TERMINATE [EXCEPT task-name,..];

werden intern dargestellt durch

(a)	CALL TERMTA	||ADDR	PKS, der task-name zugeordnet ist
		||ADDR	Except-Liste
(b)	CALL TERMLT	||ADDR	Except-Liste.

Die Struktur der Except-Liste ist in 1.7.8 beschrieben.

Die Funktionen TERMTA und TERMLT werden bei Überlaufen der Anweisung aufgerufen.
Sobald eine der Startbedingungen erfüllt ist, unter denen eine Terminate-Anweisung auszuführen ist, wird im Rahmen der Sekundärreaktion SCTERM aufgerufen.

Funktion: TERMTA
Funktion: TERMLT
Funktion: SCTERM

Notieren, daß im Rahmen des Programms EXCEPT die Prozeßsteuerungsfunktion TERMIN auszuführen ist.
Ausführung von EXCEPT (1.7.13).

In EXCEPT werden alle betroffenen Prozesse identifiziert, und für jeden wird TERMIN ausgeführt.

Funktion: TERMIN

(1) Durch Aufruf von OPNOP wird in allen für den Prozeß existierenden Einplankontrollsätzen EPS (in OP) notiert, daß aufgrund von Meldungen keine Steueranweisung auszuführen ist. Diese EPS werden an die Kette der freizugebenden EPS gehängt. In SZ des PKS wird notiert, daß keine EPS für diesen Prozeß existieren, und in SK wird der leere Hinweis NULL eingetragen.
Falls der Prozeß nicht aktiv ist, wird er mit seiner Beendigung durch PRETER beauftragt.
Fortsetzung bei (2).

(2) Der Zähler PZ (gepufferte Aktivierungen) in PKS wird auf 0 gesetzt.
In SZ und CA des PKS wird notiert, daß der Prozeß aufgrund einer Terminate-Anweisung mit seiner Beendigung durch TERMPR beauftragt ist.
In SEMA des PKS wird der leere Hinweis NULL eingetragen.
Fortsetzung des EXCEPT-Programms.

Sobald ein Prozeß, für den TERMIN oder eine Prevent-Anweisung ausgeführt wurde, d.h. der mit seiner Beendigung beauftragt ist, aufgenommen wird, wird (über CA seines PKS) die Funktion TERMPR oder PRETER aufgerufen.

Funktion: TERMPR

(1) Falls der Prozeß in einer Warteschlange steht, wird er ausgekettet (EVAUSK).
Falls der Prozeß einen eigenen Laufzeitkeller besitzt, Fortsetzung bei (2).
Anderenfalls Fortsetzung bei (3).

(2) Durch Ausführung von FREEAS (sekundärer Eingang von PRAUS siehe 3.4.2.6) für alle AS des Laufzeitkellers wird geprüft, ob Unterprozesse auf einen Aktivierungssatz zugreifen und in diesem Fall, wartet der Prozeß auf das Ende dieser Unterprozesse. Falls keine Unterprozesse auf irgendeinen Aktivierungssatz zugreifen, wird der Laufzeitkeller freigegeben.
Fortsetzung bei (3).

Funktion: PRETER

(3) Durch Aufruf von KSPA (siehe 1.7.7 Pkt. (5)) wird der Prozeß aus allen Ketten entfernt,und dem übergeordneten Prozeß wird signalisiert, daß er nicht mehr auf diesen Unterprozeß zu warten hat.

(4) Falls ein Prozeß die Ausführung einer Activate-Anweisung (siehe: 1.7.7 (1)) für diesen Prozeß nicht beenden konnte, weil die Funktion TERMPR noch nicht ausgeführt war, wird unmittelbar mit der unterbrochenen Aktivierung fortgefahren, denn in diesem Fall war die Ausführung von TERMPR nur in das Aktivierungssprogramm eingeschoben.

1.7.11 Das normale Ende eines Prozesses

Der letzte Befehl im Segment eines Prozesses ist

CALL PREND.

Funktion: PREND

(1) Falls der Prozeß einen kurzen PKS besitzt, d.h. er hat weder einen eigenen Laufzeitkeller, noch können für ihm Aktivierungen gepuffert sein und bei seiner Aktivierung wurde kein USING Sema benutzt, Fortsetzung bei (3) des Programms TERMPR (1.7.10).
Anderenfalls Fortsetzung bei (2).

(2) Falls bei seiner Aktivierung USING Sema benutzt wurde, zeigt SEMA seines PKS auf den Sema-Kontrollsatz, der nun freigegeben wird.
Fortsetzung bei (3).

(3) Freigabe des Laufzeitkellers.
Falls Aktivierungen gepuffert sind (PZ in PKS ungleich 0), Fortsetzung bei 1.7.7 (6) zur erneuten Aktivierung.
Falls ein Einplankontrollsatz EPS für eine erneute Aktivierung existiert, wird der Prozeß zurückgestellt und der nächstwichtigste lauffähige Prozeß aufgenommen.
Falls Aktivierungen weder gepuffert noch eingeplant sind, wird Punkt (3) des Programms TERMPR (1.7.10) ausgeführt.

1.7.12 Die Prevent-Anweisung

Die Anweisungen

(a) PREVENT task-name;

(b) PREVENT;

werden intern durch

(a) CALL PREVTA ADDR PKS, der task-name zugeordnet ist

(b) CALL PREVLT

dargestellt.

Diese Funktionen werden bei Überlaufen der Anweisungen aufgerufen; sobald eine der Startbedingungen erfüllt ist unter denen die Ausführung einer Prevent-Anweisung steht, wird die Funktion SCPREV aufgerufen.

Funktion: PREVTA
Funktion: SCPREV
Funktion: PREVLT

(1) Falls für den betroffenen Prozeß Steueranweisungen eingeplant sind, werden ihre Einplankontrollsätze aufgegeben:

- Für jeden seiner EPS wird in OP des Kopfes notiert, daß keine Prozeßsteuerung auszuführen ist; die Kette seiner EPS wird der Kette aller abzubauenden EPS zugefügt.
- Im PKS wird notiert, daß für den Prozeß keine Steueranweisungen eingeplant sind.

(2) Falls der betroffene Prozeß nicht aktiv ist, wird er mit seiner Beendigung durch PRETER beauftragt.

Wenn kein im PEARL-Programm definierter Prozeß mehr lauffähig ist, wird der Systemprozeß aufgenommen, dessen Prozeßkontrollsatz PRIOKE das Ende der Prioritätskette bildet und folgendes Programm ausgeführt:

Funktion: ABBAU

Die Kette der abzubauenden EPS besteht aus EPS, die bei der Ausführung von Prevent- und Terminate-Anweisungen aufgegeben wurden. Ihre Einplanelemente stehen noch in Ereignis- oder Zeitketten.
Für jeden abzubauenden EPS werden alle Einplanelemente aus ihren Ketten genommen (siehe: 1.4.3.3, 1.4.3.4) und danach wird der Speicherplatz freigegeben.

1.7.13 Berücksichtigung von Except-Listen bei der Ausführung von Prozeßsteueranweisungen

Bei der Ausführung einer Suspend-, Continue- oder Terminate-Anweisung werden im allgemeinen mehrere Prozesse gesteuert:

- der Prozeß, der durch Angabe eines Tasknamens in der Anweisung spezifiziert ist oder bei Fehlen dieser Angabe, der Prozeß, in dessen Code die Anweisung steht,
- alle diesem Prozeß untergeordneten Prozesse und deren untergeordnete Prozesse usw., sofern ihre Namen nicht in EXCEPT-Taskname... aufgeführt sind; die in dieser Liste genannten Prozesse und deren untergeordnete Prozesse usw. sind von der Prozeßsteuerung ausgenommen.

Ausgehend von dem PKS des in der Anweisung spezifizierten Prozesses wird die im vorigen Abschnitt beschriebene Hierarchieverzeigerung zur Identifizierung der PKS aller Unterprozesse benutzt.
Falls die Anweisung die EXCEPT-Option enthielt, steht als deren interne Darstellung eine Except-Liste (siehe: 1.7.8) mit den Hinweisen auf die PKS der ausgenommenen Prozesse. In diesem Fall wird für jeden über die Hierarchieverzeigerung erreichten PKS eines Unterprozesses geprüft, ob seine Adresse in der Except-Liste auftaucht.

Funktion: EXCEPT
Mit "Haupt-Prozeß" wird der in der Anweisung spezifizierte Prozeß und mit "Prozeß" der jeweils bei Durchlaufen der Hierarchieverzeigerung erreichte Prozeß bezeichnet.

(1) Ausführung der Prozeßsteuerung für den Hauptprozeß.
Fortsetzung bei (2).

(2) Einstellen einer Weiche: Except-Liste vorhanden oder nicht.
Fortsetzung bei (3).

(3) Falls der Prozeß Unterprozesse hat, wird über K seines PKS der erste ihm untergeordnete Prozeß aufgenommen.
Fortsetzung bei (5).
Falls der Prozeß keine Unterprozesse hat, wird geprüft, ob er der Hauptprozeß ist,und in diesem Fall sind alle betroffenen Prozesse gesteuert: Ende.
Falls der Prozeß keine Unterprozesse hat und nicht der Hauptprozeß ist, Fortsetzung bei (4).

(4) Falls B auf einen PKS weist, wird dieser als nächster aufgenommen.
Fortsetzung bei (5).
Falls B den leeren Hinweis enthält und V auf den PKS des Hauptprozesses weist, sind alle betroffenen Prozesse gesteuert: Ende.
Falls B den leeren Hinweis enthält und V nicht auf den PKS des Hauptprozesses weist, wird über V der nächste Prozeß aufgenommen.
Fortsetzung bei (4).

(5) Sprung über die in (2) eingestellte Weiche nach (a) oder (b).

(a) *Except-Liste vorhanden*
In der Except-Liste steht ein Hinweis auf den PKS des Prozesses:
Fortsetzung bei (4).
Die Except-Liste enthält keinen Hinweis auf den PKS des Prozesses:
Fortsetzung bei (b).

(b) *Keine Except-Liste vorhanden*
Ausführung der Prozeßsteuerung für den Prozeß.
Fortsetzung bei (3).

1.7.14 Ablauf der Programme zur Prozeßsteuerung

Prozeßsteueranweisungen werden auf der Verwaltungsebene ausgeführt, d.h. diese Arbeit kann durch die Primärreaktion auf Meldungen unterbrochen werden, und vor ihrem Ende kann weder die Sekundärreaktion eingeleitet noch ein wichtigerer als der steuernde Prozeß laufen.

Um zu vermeiden, daß die Ausführung einer Activate-, Terminate- oder Prevent-Anweisung die Reaktion eines wichtigeren Prozesses auf eine Meldung verzögert, wird in diesen Fällen die Prozeßsteuerung in mehreren Phasen durchgeführt, zwischen denen alle lauffähigen Prozesse, die wichtiger als der gesteuerte Prozeß sind, aufgenommen werden.

Die Continue-, Suspend- und Resume-Anweisungen werden ungeteilt ausgeführt.

1.8 EXPLIZITES SYNCHRONISIEREN VON PROZESSABLÄUFEN MIT HILFE VON SEMA- UND BOLT-VARIABLEN

1.8.1 Interne Darstellung von Sema- und Bolt-Variablen

Alle *Vereinbarungen* von Sema-Variablen und Bolt-Variablen in einem Block werden intern in je einem Befehl zusammengefaßt dargestellt.

CALL SEMINI	VALUE Anzahl der Sema-Variablen
CALL BOLINI	Liste von: VALUE Maximumwert der Bolt-Variablen

Beim Aufruf von BOLINI ist für Bolt-Variable, für die kein Maximumwert vereinbart wurde, an der entsprechenden Stelle der Parameterliste die größte darstellbare Zahl anzugeben.

Die Funktionen SEMINI und BOLINI sind im einzelnen in 3.4.2.4 beschrieben. Hier soll nur dargestellt werden, welche *Datenstrukturen* von diesen Funktionen beim Eintritt in den Block aufgebaut werden.

Für jede Sema-Variable wird ein Sema-Kontrollsatz SEMA eingerichtet und dessen Adresse im Aktivierungssatz abgelegt.

Sema-Kontrollsatz (SEMA):

SK	Zeiger auf die Warteschlange von PKS
S	Zähler, initial 0

S = 0 bedeutet: Semavariable gesperrt

S > 0 bedeutet: Semavariable frei
Die Zahl in S gibt an, wieviele Requestanweisungen auf diese Semavariable ausgeführt werden können, ohne daß der ausführende Prozeß blockiert wird.

Für jede Bolt-Variable wird ein Bolt-Kontrollsatz BOLT eingerichtet und dessen Adresse im Aktivierungssatz abgelegt.

Bolt-Kontrollsatz (BOLT):

BK	Zeiger auf die Warteschlange von PKS
SZ	Zähler, initial 0
SUP	Zahl, übergebener Maximumwert

SZ = 0 bedeutet: Boltvariable frei

SZ > 0 bedeutet: Boltvariable ist durch Enteranweisungen gesperrt; die Zahl in SZ gibt an, wieviele Enteranweisungen ausgeführt sind

SZ = -1bedeutet: Boltvariable ist durch Reserveanweisung gesperrt.

Die Zeiger SK in SEMA und BK in BOLT werden mit der Adresse der Struktur SEMEND

SN	Zeiger, unbenutzt
ABSP	Zahl, 2**16
SV	Zeiger, bedeutungslos

initialisiert, die als Endeelement aller Warteschlangen bei Sema- und Bolt-Kontrollsätzen benutzt wird.

1.8.2 Warteschlangen

Die Warteschlange eines Sema- oder Bolt-Kontrollsatzes besteht aus allen Prozeßkontrollsätzen PKS von Prozessen, die aufgrund einer Request-Anweisung bzw. Reserve- oder Enter-Anweisung zurückgestellt wurden. Zur Verkettung der PKS werden ihre Zeiger SN und SV benutzt. Der Zeiger SK in SEMA weist auf den zuletzt eingeketteten PKS und der Zeiger BK in BOLT auf den PKS mit der kleinsten Absolutpriorität ABSP (wichtigster Prozeß der Warteschlange). Eine leere Warteschlange ist durch den Hinweis auf das Endeelement SEMEND in SK bzw. BK gekennzeichnet.

1.8.3 Die Request- und Release-Anweisungen

Die Anweisungen

$$\left\{\begin{matrix}\text{REQUEST}\\ \text{RELEASE}\end{matrix}\right\}\ \underline{\text{Sema}};$$

werden intern durch

$$\text{CALL}\ \left\{\begin{matrix}\text{REQ1}\\ \text{REL1}\end{matrix}\right\}\ \ ||\ \text{ADDR}\quad \text{Sema-Kontrollsatz SEMA}$$

und die Anweisungen

$$\left\{\begin{matrix}\text{REQUEST}\\ \text{RELEASE}\end{matrix}\right\}\ \underline{\text{Sema1}},\ldots,\underline{\text{Sema n}};$$

durch

$$\text{CALL}\ \left\{\begin{matrix}\text{REQSL}\\ \text{RELSL}\end{matrix}\right\}\ \ \Big\|\ \begin{matrix}\text{Liste von:}\\ \text{ADDR}\quad \text{Sema-Kontrollsatz 1 SEMA}\\ \vdots\\ \text{ADDR}\quad \text{Sema-Kontrollsatz n SEMA}\end{matrix}$$

dargestellt.

Funktion: REQ1, Parameter: Hinweis auf einen Sema-Kontrollsatz

(a) Falls die Sema-Variable frei ist (S > 0), wird S um eins vermindert und der aufrufende Prozeß bleibt lauffähig.

(b) Anderenfalls (S = 0) wird der PKS des aufrufenden Prozesses als erstes Element in die Warteschlange, auf die SK im Semakontrollsatz zeigt, eingekettet (Funktion EINK), und der Prozeß wird zur Wiederholung der Requestanweisung zurückgestellt.

Funktion: REQSL Parameter: Liste von Hinweisen auf Sema-Kontrollsätze

Bei allen Semavariablen der Liste wird geprüft, ob sie frei sind (S > 0).

(a) Wird eine Semavariable gefunden, die nicht frei ist, wird wie bei REQ1 unter (b) fortgefahren.

(b) Anderenfalls werden die Zähler S aller Semavariablen um eins vermindert und der aufrufende Prozeß bleibt lauffähig.

Funktion: REL1 Parameter: Hinweis auf Sema-Kontrollsatz

Der Zähler S wird um eins erhöht.
Falls die von SK angezeigte Kette nicht leer ist, werden alle darin notierten PKS aus dem "Warten auf Sema" befreit und evtl. wieder lauffähig gemacht zur Wiederholung ihrer Requestanweisung.
Der aufrufende Prozeß bleibt lauffähig.

Funktion: RELSL Parameter: Liste von Hinweisen auf Sema-Kontrollsätze

Für jedes Element der Liste wird entsprechend der Funktion REL1 verfahren.

1.8.4 Die Reserve-, Enter-, Free- und Leave-Anweisungen

Die Anweisungen

```
{RESERVE}
{ENTER  }
{FREE   }  Bolt;
{LEAVE  }
```

werden intern durch

```
          {RES1 }
          {ENT1 }
CALL      {FREE1}  || ADDR   Bolt-Kontrollsatz BOLT
          {LEAV1}
```

und die Anweisungen

```
{RESERVE}
{ENTER  }  Bolt1,...,Bolt n;
{FREE   }
{LEAVE  }
```

durch

```
          {RESBL }
          {ENTBL }  || Liste von:
CALL      {FREEBL}  || ADDR   Bolt-Kontrollsatz 1 BOLT
          {LEAVBL}  ||   :
                    || ADDR   Bolt-Kontrollsatz n BOLT
```

dargestellt.

Funktion: RES1, Parameter: Hinweis auf einen Bolt-Kontrollsatz

(a) Falls SZ = 0 ist, wird SZ = -1 gesetzt und der aufrufende Prozeß fortgesetzt.

(b) Anderenfalls wird der aufrufende Prozeß zur Wiederholung seiner Reserveanweisung zurückgestellt und sein PKS in die Kette, die bei BK beginnt, eingeordnet und zwar entweder als erstes Element, falls er wichtiger (ABSP) als das bislang erste Element ist oder als zweites Element, falls er unwichtiger ist als das erste Element.
Der wichtigste lauffähige Prozeß wird aufgenommen.

Funktion: RESBL, Parameter: Liste von Hinweisen auf Bolt-Kontrollsätze

Bei allen Boltvariablen der Liste wird geprüft, ob sie frei sind (SZ = 0).
Wird eine Boltvariable gefunden, bei der SZ ≠ 0 ist, wird wie bei RES1 unter (b) fortgefahren.
Anderenfalls wird in allen Bolt-Kontrollsätzen der Liste SZ = -1 gesetzt und der aufrufende Prozeß fortgesetzt.

Funktion: ENT1, Parameter: Hinweis auf einen Bolt-Kontrollsatz

(a) Folgende Kriterien sind zu prüfen, ob die Enteranweisung ausgeführt werden kann oder der aufrufende Prozeß zurückgestellt werden muß:

1. SZ = 0: ausführen
2. SZ = -1: zurückstellen
3. SZ = SUP (Maximalzahl für Enteranweisungen erreicht): zurückstellen
4. sonst (0 ≤ SZ< SUP):
4.1 BK-Kette leer: ausführen
4.2 aufrufender Prozeß wichtiger als 1. Element der BK-Kette: ausführen
4.3 sonst: zurückstellen

(b) Zur Ausführung der Enteranweisung wird SZ um eins erhöht und der aufrufende Prozeß fortgesetzt.

(c) Zum Zurückstellen des aufrufenden Prozesses wird wie bei RES1 unter (b) fortgefahren.

Funktion: ENTBL, Parameter: Liste von Hinweisen auf Bolt-Kontrollsätze

Für alle Boltvariablen der Liste wird die bei ENT1 unter (a) angegebene Prüfung durchgeführt.
Wird eine Boltvariable gefunden, bei der die Prüfung "Zurückstellen" ergibt, wird sinngemäß wie bei RES1 unter (b) fortgefahren.
Anderenfalls wird in allen Boltkontrollsätzen der Liste SZ um eins erhöht und der aufrufende Prozeß fortgesetzt.

Funktion: FREE1, Parameter: Hinweis auf Bolt-Kontrollsatz

(a) Ist SZ ≠ -1 (nicht durch Reserveanweisung gesperrt), wird die Anweisung übergangen.

(b) Anderenfalls wird SZ = 0 gesetzt,und

(c) die von BK angezeigte Kette von wartenden Prozessen wird aufgelöst und alle darin verketteten PKS aus dem Zustand "Warten auf Bolt" befreit und evtl. zur Wiederholung ihrer Reserve- oder Enter-Anweisung lauffähig gemacht.
Der aufrufende Prozeß bleibt lauffähig.

Funktion: FREEBL, Parameter: Liste von Hinweisen auf Bolt-Kontrollsätze

Für jedes Element der Liste wird entsprechend der Funktion FREE1 verfahren.

Funktion: LEAV1, Parameter: Hinweis auf Bolt-Kontrollsatz

Ist SZ ≤ 0 (nicht durch Enter-Anweisung gesperrt), wird die Anweisung übergangen.
Anderenfalls wird SZ um eins vermindert und wie bei FREE1 unter (b) fortgefahren.

Funktion: LEAVBL, Parameter: Liste von Hinweisen auf Bolt-Kontrollsätze

Für jedes Element der Liste wird entsprechend der Funktion LEAV1 verfahren.

2. EIN- UND AUSGABE

In diesem Kapitel sind Datenstrukturen und Steuerprogramme für den Datenaustausch zwischen dem Arbeitsspeicher und externen Datenträgern - soweit sie geräteunabhängig sind - dargestellt.
Wie gerätespezifische Routinen (Treiber, Fehlerbehandlung) angeschlossen werden, ist in dieser Beschreibung definiert.

Datenstrukturen zur Beschreibung der technischen Umgebung

Für jedes angeschlossene E/A-Gerät wird ein Übertragungskontrollsatz geführt; er definiert wie mit dem Gerät Übertragungen auszuführen sind.
Wie auf einem Gerät der Standardperipherie Dateien verwaltet werden, ist in einem zugeordneten Dateiverwaltungssatz definiert.
Über diese Kontrollsätze werden gerätespezifische Routinen angesprochen: E/A-Treiber, Fehlerreaktionen, Speicherplatzverwaltung.

Datenstrukturen zur Beschreibung der programmdefinierten Umgebung

Die interne Beschreibung aller im Systemteil eines PEARL-Programms eingeführten Einheiten (device) der *Prozeßperipherie* steht in den drei Bereichen BVK, BVW und BGB, die für jede Einheit die Information über das zugeordnete E/A-Gerät und die Anschlußstellen (connector-description) enthalten.

Die im Systemteil definierten Einheiten der *Standardperipherie* werden durch je einen Gerätekontrollsatz repräsentiert, der Informationen über das zugeordnete E/A-Gerät und den Speicherplatz enthält.

Dateiverwaltung

Für jede vom Programm (in einer Create-Anweisung) auf einer Einheit der Standardperipherie angelegte Datei wird ein Dateikontrollsatz geführt mit einem Hinweis auf den Gerätekontrollsatz der Einheit und der Beschreibung des zugeteilten Spei-

cherplatzes.
Eine Variable der Art FILE ist als Bezug auf einen Filekontrollsatz realisiert. Nach Ausführung einer Open-Anweisung ist er einem Dateikontrollsatz zugeordnet und legt eine Übertragungsart für die Datei fest.
Aufgrund einer E/A-Anweisung wird er mit den notwendigen Steuergrößen für das E/A-Gerät initialisiert und dient danach als Kontrollstruktur für die Durchführung der Übertragung.

Steuerung von Übertragungen

E/A-Anweisungen werden auf Zulässigkeit geprüft, in Übertragungsaufträge für das E/A-Gerät übersetzt und dem Geräte-Steuerprogramm (zur Übertragung von jeweils einer gerätespezifischen Übertragungseinheit) weitergeleitet; während der Übertragung sind die beauftragenden Rechenprozesse zurückgestellt.

2.1 ANSPRACHE UND KONTROLLE VON E/A-GERÄTEN

Nach Auswertung des Systemteils eines PEARL-Programms existiert für jedes angeschlossene E/A-Gerät (device-type) ein *Übertragungskontrollsatz* UKS; über die in den entsprechenden Anweisungen (device-connection) eingeführten Einheiten (device-identification) ist er ansprechbar. Das Betriebssystem benutzt ihn zur Ablage der für Übertragungen mit dem E/A-Gerät notwendigen Steuerinformationen wie Gerätestatus (arbeitsbereit, arbeitend, gesperrt), Gerätetyp (Ein- und/oder Ausgabe, Prozeß- oder Standardperipherie), Identifikation gerätespezifischer Routinen, Beschreibung des laufenden Übertragungsauftrags und ein Hinweis auf die Warteschlange zurückgestellter Aufträge für das E/A-Gerät.

Nach dem Transport einer (gerätespezifischen) technischen Übertragungseinheit gibt das E/A-Gerät eine *Rückmeldung*. Nach der Programmunterbrechung wird der zugeordnete UKS identifiziert und die in ihm spezifizierte *Reaktion auf die Rückmeldung* ausgeführt. Eine nichterwartete Rückmeldung führt zu einer Fehlerbehandlung. Das Betriebssystem erkennt den Ausfall eines E/A-Geräts, indem es die Rückmeldung innerhalb einer gewissen Zeit erwartet.

2.1.1 Der Übertragungskontrollsatz UKS

REA	Marke der Primärreaktion auf die Rückmeldung des Geräts
AKT	Bei Geräten der Prozeßperipherie Hinweis auf den PKS des Prozesses, der den UKS für eine Übertragung belegt hat. Bei Geräten der Standardperipherie Hinweis auf den Filekontrollsatz FKS, der die Parameter für die laufende Übertragung spezifiziert
W	Hinweis auf die Kette der auf die Freigabe des UKS wartenden PKS (Prozeßsperipherie) oder FKS (Standardperipherie)
RES	Marke der Sekundärreaktion auf die Rückmeldung des Geräts
AN1 AN2	Marke der Geräteansprache für Eingabe/Ausgabe Fehlerreaktion bei gesperrtem Gerät Zurückstellen bei belegtem Gerät Ausführung der in AN3/AN4 spezifizierten Befehlsfolge bei freiem Gerät
AN3 AN4	Marke einer Fehlerreaktion, falls keine Eingabe/Ausgabe Marke der gerätespezifischen Befehlsfolge zur Durchführung einer Eingabe/Ausgabe
FD	Marke der Fehlerreaktion bei gesperrtem Gerät
ETR ATR	Identifizierung der Treiberprozedur für die Eingabe/Ausgabe
UE	Technische Übertragungseinheit Bei Geräten der Prozeßperipherie 0 oder 8 für 16-Bit bzw. 8-Bit-Kanäle. Bei Geräten der Standardperipherie Anzahl der bei einer Übertragung transportierten Bytes oder Wörter
FZ	Marke der Reaktion auf die Zeitüberschreitung des Gerätes (eine technische Übertragungseinheit muß innerhalb einer bestimmten Zeit transportiert werden)

ZEIT	Bitmuster zur zeitlichen Kontrolle des Geräts
ART	Bitmuster zur Beschreibung des Gerätetyps
BP	Identifizierung einer Positionierungsprozedur z.B. Rückspulen eines Bandes
SKONV	Identifizierung der Prozedur für die Standardkonvertierung
Die folgenden Elemente werden nur in UKS von Geräten der Prozeßperipherie geführt	
LAG	Identifizierung des dem Gerät zugeordneten Teilbereichs im Bereich der Ausgabewerte BAG
LKN	Nummer des (technischen)Kanals,über den ein- oder ausgegeben wird
LKS	Nummer der ersten Anschlußstelle des in LKN spezifizierten Kanals, die bei der Übertragung berücksichtigt wird
LKL	Anzahl der Anschlußstellen, die bei der Übertragung zu berücksichtigen sind
PINT	Hinweis auf die Variable, der der Eingabewert zugewiesen wird
UEW	Zwischenspeicher für den über einen Kanal eingelesenen Wert bzw. für den Ausgabewert

2.1.2 Reaktion auf Geräterückmeldungen

Nachdem eine Unterbrechungsursache als Rückmeldung eines E/A-Gerätes erkannt ist, wird die in REA seines UKS spezifizierte *Primärreaktion* ausgeführt.

(1) *Unerwartete Rückmeldung*

Funktion: GRUECF

Verhindern weiterer Rückmeldungen dieses Gerätes und in RES des UKS notieren, daß eine Fehlerreaktion auszuführen ist.
Fortsetzung bei (2).

(2) *Erwartete Rückmeldung*

Funktion: GRUECK

Eintragen der Adresse des UKS in den Auftragspuffer
Löschen der Zeitkontrolle
Einplanen der Sekundärreaktion auf Geräterückmeldungen

Im Rahmen der *Sekundärreaktion* auf Unterbrechungen wird der Auftragspuffer für Geräterückmeldungen in der Reihenfolge abgearbeitet, in der ihn die Primärreaktion gefüllt hat.

(1) *Einleiten der gerätespezifischen Sekundärreaktion*

Funktion: SEKG

Übernahme der nächsten Adresse aus dem Auftragspuffer und Fortsetzung bei der in RES des UKS spezifizierten Befehlsfolge (DAT, EPD, APD, FMELD)

(2) *Erwartete Rückmeldung*

Funktion: DAT, Gerät der Standardperipherie
Funktionen: EPD, APD, Gerät der Prozeßperipherie

Diese Funktionen werden im einzelnen in den Abschnitten 2.3.4 und 2.2.3 beschrieben.

Falls die Übertragung einer weiteren technischen Einheit nötig ist, wird das Gerät damit beauftragt, anderenfalls wird im UKS notiert, daß keine Rückmeldung erwartet wird und das Gerät arbeitsbereit ist, der beauftragende Prozeß wird aus dem Wartezustand befreit.
Fortsetzung bei (4).

(3) *Unerwartete Rückmeldung*

Funktion: FMELD

Im UKS wird notiert, daß das Gerät nicht arbeitsbereit ist.
Ausgabe einer Fehlermeldung.
Fortsetzung bei (4).

(4) *Ende einer gerätespezifischen Sekundärreaktion*

Funktion: RMG

Falls der Auftragspuffer leer ist, wird notiert, daß keine Sekundärreaktion auf Geräterückmeldungen auszuführen ist.
Anderenfalls Fortsetzung bei (1).

2.1.3 Zeitkontrolle der Geräte

Um feststellen zu können, ob die erwartete Rückmeldung eines E/A-Gerätes ausbleibt, wird eine Systemkonstante T nach der Arbeitsgeschwindigkeit des langsamsten Geräts gewählt und von allen Geräten verlangt, daß sie eine Übertragung innerhalb von T Millisekunden ausführen.
Das System führt zwei Kontrollvariable LZK1 und LZK2 und in den UKS die paarweise verschiedenen Bitmuster ZEIT mit jeweils genau einem gesetzten Bit. Jedes gesetzte Bit in LZK1 bedeutet, daß die Rückmeldung des entsprechenden Geräts erwartet wird,und jedes länger als T Millisekunden in LZK2 gesetzte Bit bedeutet eine Zeitüberschreitung des Gerätes.

Setzen der Zeitkontrolle

Beim Anstoß eines Gerätes für eine Übertragung wird ZEIT seines UKS in LZK1 übernommen.

Löschen der Zeitkontrolle

Im Rahmen der Primärreaktion auf die Rückmeldung eines Gerätes wird ZEIT seines UKS aus LZK1 und LZK2 entfernt.

Feststellen von Zeitüberschreitungen

Alle T Millisekunden wird zunächst geprüft, ob in LZK2 Bits gesetzt sind und ggf. für die entsprechenden Geräte die Fehlerreaktion auf Zeitüberschreitung eingeleitet.
Danach wird LZK2 = LZK1 gesetzt.
D.h. jedem Gerät stehen für eine Übertragung mindestens T und höchstens 2T Millisekunden zur Verfügung.

2.2 ÜBERTRAGUNG VON PROZESSDATEN

Das Betriebssystem steuert den Datenaustausch zwischen dem Arbeitsspeicher und den Anschlußstellen der Datenendgeräte. Es wird vorausgesetzt, daß nach Auswertung des Systemteils die in 2.2.1 beschriebenen Datenstrukturen initialisiert sind. Die als Einheiten definierten Folgen von Anschlußstellen können eine Länge bis zu einem Wort*) haben und müssen zusammenhängend sein. Sie können sich über mehrere benachbarte Kanäle erstrecken. Hier bezeichnet Kanal eine Gruppe von Anschlußstellen, die bei der Übertragungsarbeit des Geräts immer gleichzeitig angesprochen werden.
Das Steuerprogramm für die Datenübertragung mit einer Einheit identifiziert aufgrund der Einheitennummer das Gerät und die belegten Kanäle, es zerlegt den Gesamtauftrag in soviele Schritte wie Kanäle betroffen sind. In jedem Schritt stößt es das Gerät an, Daten über einen Kanal zu übermitteln. Bei langsamen Geräten stellt es nach Auftragseingang den laufenden Rechenprozeß zurück und erwartet für jeden Schritt eine Rückmeldung des Geräts; in der Wartezeit werden andere Arbeiten ausgeführt. Nach Abschluß der Gesamtübertragung werden der beauftragende Prozeß und die auf Freigabe des Geräts wartenden Prozesse aus dem Wartezustand befreit.

*) implementiert für eine Wortlänge von 16 Bits.

2.2.1 Datenstrukturen zur Beschreibung der Prozeßperipherie

Jedem Namen des Systemteils, der eine Folge von Anschlußstellen bezeichnet, wird eine Nummer VK zugeordnet, unter der im *Bereich der virtuellen Kanalnummern* BVK ein Wort (16 Bit) reserviert wird.

Die ersten 8 Bits von BVK(VK) enthalten eine Nummer VW, unter der im *Bereich der virtuellen Werke* BVW eine Struktur vom Typ (Zeiger, Zahl) abgelegt ist. Der Zeiger BVW(VW).PPW enthält die Adresse des Übertragungskontrollsatzes UKS, der zur Ansprache des entsprechenden Datenanschlußgerätes benutzt wird.

Die zweiten 8 Bits von BVK(VK) enthalten die relative Kanalnummer RK, die zusammen mit der Zahl BVW(VW).NOK zur Identifizierung der Anschlußstellen dient. Bei Analogwerken ist

$$KNR = BVW(VW).NOK+RK$$

die Nummer des Kanals.

Diese Berechnung wird besonders kurz, wenn die Kanallänge eine Potenz von 2 ist. Im folgenden werden 8 und 16 als Kanallängen zugelassen; somit haben die Elemente von BGB folgendes Format:

Kanallänge	Bit	Interpretation
8	1-13	Kanalnummer
16	1-12	
8	14-16	Anschlußstellennummer
16	13-16	

Anschlußstellen und Kanäle werden von 0 an numeriert.

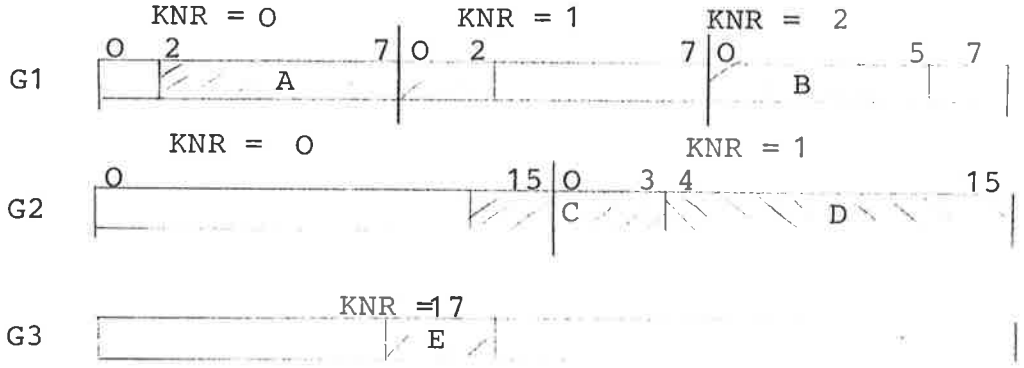
Für Anschlüsse zur Auswertung digitaler Prozeßdaten, die über Abschnitte von *Hauptkanälen* (subdivided connection) übertragen werden, erfolgt die Beschreibung dieser Abschnitte durch Einträge im Bereich BGB.
Für jede Prozeßdate wird in einem (16-Bit) Wort von BGB definiert:

- die Nummer des Hauptkanals und
- die Nummer der Anschlußstelle innerhalb des Hauptkanals, über die das erste Bit (oberstes Bit) der Prozeßdate übertragen wird.

Im Bereich BGB liegen die Beschreibungen für Übertragung von Prozeßdaten über benachbarte Anschlußstellen hintereinander; für nichtbenutzte Abschnitte von Digitalwerken wird ebenfalls eine Beschreibung abgelegt. Die unteren Bits der Beschreibung in BGB bestimmen die Nummer der Anschlußstelle, die oberen Bits die Hauptkanalnummer. Mit Hilfe zweier aufeinander folgender Beschreibungen in BGB wird die Länge eines Abschnittes berechnet.

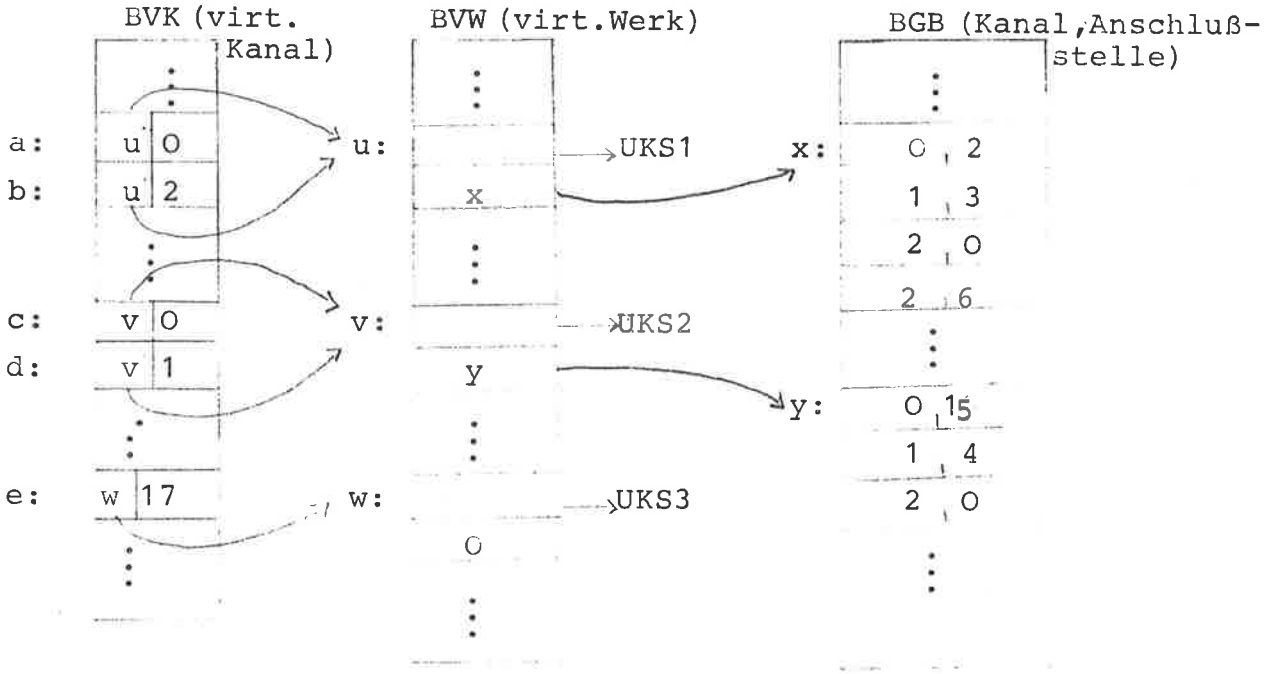
Für jedes Gerät zur digitalen Übertragung existiert im *Bereich der Ausgabewerte BAG* ein Teilbereich, so daß für jeden Kanal ein Wort zur Speicherung des letzten über ihn ausgegebenen Wertes zur Verfügung steht. Im UKS dieses Gerätes ist das Element LAG so initialisiert, daß für einen Kanal mit der Nummer KNR der Index des zugeordneten Elementes von BAG durch LAG + KNR gegeben ist.

Beispiel:
Die beiden Digitalgeräte G1, G2 und das Analoggerät G3 gehören zur Rechnerkonfiguration; die Kanallänge von G1 ist 8 und die von G2 ist 16. Die in der Skizze schraffierten Anschlußstellen



sind unter den Bezeichnern A,B,C,D,E im Systemteil definiert.

Zur Laufzeit stehen hierfür folgende Informationen zur Verfügung, um A,..,E über die Nummern a,..,e zu identifizieren.



2.2.2 Steuerprogramme der Übertragung

Zur Übertragung zwischen der Prozeßperipherie und einem Wort des Arbeitsspeichers werden die Funktionen PDIN (Eingabe) und PDOUT (Ausgabe) aufgerufen.

CALL PDIN	VALUE	virtuelle Kanalnummer (Index in BVK)
	ADDR	Wert im Arbeitsspeicher
CALL PDOUT	VALUE	virtuelle Kanalnummer (Index in BVK)
	ADDR	Wert im Arbeitsspeicher

(1) Anhand der virtuellen Kanalnummer wird in WERKBE der Übertragungskontrollsatz UKS die Beschreibung der betroffenen Anschlußstellen identifiziert.

(2a) Bei Eingabe wird mit dem in AN1 des UKS spezifizierten Programmstück (DSP, UEFHL, BEL, EINUK, ESY, ESY1) fortgefahren.

(2b) Bei Ausgabe wird mit dem in AN2 des UKS spezifizierten Programmstück (DSP, UEFHL, BEL, AUSUK, ASY, ASY1) fortgefahren.

DSP : Das Gerät ist nicht arbeitsbereit. Ausführung der in FD des UKS spezifizierten Reaktion: z.B. Versuch, das Gerät einzuschalten oder Ausgabe einer Fehlermeldung und Abbruch des laufenden Prozesses.

UEFHL: Die gewünschte Übertragungsart ist auf dem Gerät nicht möglich. Ausgabe einer Fehlermeldung und Abbruch des laufenden Prozesses.

BEL : Der Übertragungskontrollsatz UKS ist von einem anderen Prozeß für die Dauer einer Übertragung belegt. Der laufende Prozeß wird zur Wiederholung seines Aufrufs von PDIN oder PDOUT zurückgestellt.

Gerät mit interner Arbeitsgeschwindigkeit

Eingabe

EINUK: In UKK werden der erste (physikalische) Kanal, die erste Anschlußstelle und die Zahl der Anschlußstellen ermittelt. In einer durch NUK gesteuerten Folge wird über die Kanäle eingegeben, so daß der Wert rechtsbündig im angegebenen Wort des Arbeitsspeichers abgelegt wird.

Ausgabe

AUSUK: In UKK werden der erste (physikalische) Kanal, die erste Anschlußstelle und die Zahl der Anschlußstellen ermittelt. In einer durch NUK gesteuerten Folge wird jeweils ein Teil des Ausgabewertes einem Element im Bereich der Ausgabewerte (BAG) überlagert und dieses Bereichselement zum Kanal übertragen.

Gerät mit langsamer Arbeitsweise

Eingabe

ESY : Auch Ausgabe möglich.
Im UKS wird notiert, daß die Rückmeldung des Geräts die Ausführung einer Eingabe signalisiert.
Fortsetzung bei ESY1.

ESY1 : Nur Eingabe möglich.
In UKK werden der erste (physikalische) Kanal, die erste Anschlußstelle und die Zahl der Anschlußstellen ermittelt. Ablage dieser Steuergrößen und der Zieladresse (zweiter Parameter von PDIN) im UKS.
Anstoß des Geräts, den ersten Kanal einzulesen.
Fortsetzung bei EALANG.

Ausgabe

ASY : Auch Eingabe möglich.
Im UKS wird notiert, daß die Rückmeldung des Geräts die Ausführung einer Ausgabe signalisiert.
Fortsetzung bei ASY1.

ASY1: Nur Ausgabe möglich.
In UKK werden der erste (physikalische) Kanal, die erste Anschlußstelle und die Zahl der Anschlußstellen ermittelt. Ablage dieser Steuergrößen und des Ausgabewertes im UKS. Modifikation des dem ersten Kanal zugeordneten Elementes von BAG.
Anstoß des Geräts, dieses Element von BAG zum ersten Kanal auszugeben.
Fortsetzung bei EALANG.

EALANG: Der UKS wird als belegt gekennzeichnet, so daß bei weiteren Ansprachen Prozesse zurückgestellt werden (siehe BEL).
Eine Rückmeldung des Geräts wird erwartet.
Die Adresse des aktuellen PKS wird im UKS notiert und der Prozeß bis zum Abschluß des letzten Übertragungsschritts zurückgestellt.

2.2.3 Sekundärreaktion auf eine Geräterückmeldung

Übertragungsgeräte, die nicht mit interner Geschwindigkeit arbeiten, geben nach der Eingabe von oder Ausgabe zu einem Kanal eine Meldung.
Die Sekundärreaktion auf die erwartete Rückmeldung eines Geräts der Prozeßperipherie ist die Ausführung der in RES seines UKS spezifizierten Befehlsfolge EPD oder APD.

Eingabe von einem Kanal beendet

EPD: Der eingelesene Wert ist im UKS abgelegt. Anhand der im UKS notierten Steuerparameter wird er ganz oder teilweise in das beim Aufruf von PDIN angegebene Wort übernommen.
In NUK wird geprüft, ob ein weiterer Übertragungsschritt einzuleiten ist (Fortsetzung bei WLE) oder die Gesamtübertragung beendet ist (Fortsetzung bei FPD).

WLE: Ablage der geänderten Steuerparameter im UKS.
Anstoß des Geräts,über den nächsten Kanal einzulesen.
Fortsetzung bei RMG.

Ausgabe zu einem Kanal beendet

APD: In NUK wird geprüft, ob ein weiterer Übertragungsschritt einzuleiten ist (Fortsetzung bei WLA) oder die Gesamtübertragung beendet ist (Fortsetzung bei FPD).

WLA: Anhand der im UKS notierten Steuerparameter wird der nächste Teil des im UKS abgelegten Ausgabewertes in das entsprechende Element von BAG (Bereich der Ausgabewerte) übernommen. Ablage der geänderten Steuerparameter im UKS. Anstoß des Geräts, das Element von BAG zum Kanal auszugeben.
Fortsetzung bei RMG.

Abschluß einer Übertragung

FPD: Alle auf Freigabe des UKS wartenden Prozesse und der beauftragende Prozeß werden aus dem Wartezustand befreit.
Der UKS wird für weitere Übertragungen freigegeben und es wird keine Rückmeldung erwartet.
Fortsetzung bei RMG.

Ende der Sekundärreaktion auf die Rückmeldung eines Geräts

RMG: Falls im Auftragspuffer für Geräterückmeldungen weitere Einträge existieren, wird die nächste Sekundärreaktion auf die Rückmeldung eines Geräts eingeleitet.

Bei leerem Auftragspuffer wird notiert, daß keine Sekundärreaktion auf Geräterückmeldungen einzuleiten ist.

2.3 DATEIVERWALTUNG UND ÜBERTRAGUNG

Im folgenden sind die Datenstrukturen und Verwaltungsfunktionen beschrieben, die das Betriebssystem für das Arbeiten mit Dateien zur Verfügung stellt.
Zur Verwaltung von Teilen des Hintergrundspeichers als Einheiten, auf denen Dateien geführt werden können, liegen nach Auswertung des Systemteils Gerätekontrollsätze vor, die die Datenträger und die Übertragungsgeräte spezifizieren. Zur Laufzeit erzeugt das Betriebssystem aufgrund von File-Vereinbarungen und Create-Anweisungen File- und Dateikontrollsätze und verknüpft sie mit den spezifizierten Gerätekontrollsätzen. Jeder Dateikontrollsatz kann jedem Gerätekontrollsatz und jeder Filekontrollsatz jeden Dateikontrollsatz zugeordnet werden, sofern die in der Create- oder Open-Anweisung angegebene Arbeitsweise auf dem Gerät zugelassen ist. Einem Gerätekontrollsatz können mehrere Dateikontrollsätze und diesen mehrere Filekontrollsätze zugeordnet werden.

Die Steuerfunktionen für die Ein-Ausgabe realisieren die Übertragungen zwischen zusammenhängenden Bereichen des Arbeits- und Hintergrundspeichers. Sie werden ggf. mehrmals bei der Ausführung von Ein-Ausgabeanweisungen aufgerufen.
Wenn Rechenprozesse gleichzeitig den Zugriff auf Betriebsmittel (Speicherplatz, Kontrollsätze) oder Dateien verlangen, läuft der wichtigste weiter und die anderen werden zur Wiederholung ihrer Anforderung zurückgestellt.

2.3.1 Datenstrukturen der Dateiverwaltung

2.3.1.1 Dateiverwaltungssatz

Nach Auswertung des Systemteils eines PEARL-Programms existiert für jedes angeschlossene Massenspeichergerät, auf dem Dateien angelegt werden können, ein *Dateiverwaltungssatz* DVS. Er spezifiziert, wie auf diesem Gerät Dateien angelegt und entfernt werden.

CST	Marke des Programmstücks zum Anlegen einer Datei mit der gerätespezifischen Standardgliederung	
CNST	Marke des Programmstücks zum Anlegen einer Datei mit einer programmdefinierten Gliederung	
SPBB	Marke des Programmstücks zur Umrechnung der im Programm spezifizierten Dateigröße in die gerätespezifische Darstellung des Speicherplatzbedarfs	
SBEL	Marke des Programmstücks zur Reservierung des für eine Datei benötigten Speicherplatzes	
RVOLL	Marke der Reaktion auf Speicherplatzmangel	
MTLG(1) MTLG(2) MTLG(3) MTLG(4)	Dateilänge Seitenlänge Zeilenlänge Einheitenlänge	Obere Grenzen für die programmdefinierten Gliederungen
STLG(1) STLG(2) STLG(3) STLG(4)	Dateilänge Seitenlänge Zeilenlänge Einheitenlänge	Standardgliederung
SPBD	Speicherplatzbedarf einer Datei mit Standardgliederung	

OPN	Marke des Programmstücks zum Eröffnen eines FILE, falls für den in der TITLE-Option genannten Namen kein Dateikontrollsatz existiert; vor dem Programmlauf existierende Dateien können greifbar gemacht werden
CL	Marke des gerätespezifischen Programmstücks zum Abschließen eines FILE, z.B. Ausgabe einer Schreibendekennung
DEL	Marke des gerätespezifischen Programmstücks zum Löschen einer Datei: Freigabe des Speicherplatzes, Entfernen eines Eintrags aus einem Hintergrundkatalog
DKSKOP	Marke des Programmstücks, um nach dem Anlegen einer Datei ggf. eine gerätespezifische Beschreibung in den Hintergrundkatalog einzutragen
DKSU	Identifizierung der Prozedur zum Suchen einer Dateibeschreibung im Hintergrundkatalog des Geräts

2.3.1.2 Gerätekontrollsatz

Eine im Systemteil eines PEARL-Programms auf einem Massenspeichergerät eingeführte Einheit (device) wird durch einen *Gerätekontrollsatz* GKS repräsentiert. Die mit dem Attribut DEVICE vereinbarten Variablen, die in Create- oder Open-Anweisungen angesprochen werden, haben als Wert den Hinweis auf einen GKS.

PDT	Hinweis auf den DVS des zugeordneten Speichergeräts
PU	Hinweis auf den UKS des zugeordneten E/A-Geräts
ANF/ END	Anfangs- und Endadresse des über diesen GKS erreichbaren Teils des Hintergrundspeichers
VOLAD	Anfangsadresse des freien Speicherplatzes
FVOL	Länge des freien Speicherplatzes
DKSK	Hinweis auf die Kette der Dateikontrollsätze, aller auf dieser Einheit angelegten Dateien
WCR	Hinweis auf die Kette der PKS, aller auf Speicherplatzfreigabe für diese Einheit wartenden Prozesse
NDA	Hinweis auf die Kette von Parametersätzen, wird nur beim Anlegen einer Liste von Dateien benutzt

2.3.1.3 Dateikontrollsatz

Ein Stück des Hintergrundspeichers, das über Variable der Art FILE ansprechbar ist, heißt *Datei*. Bei Ausführung einer Create-Anweisung (oder der ersten Open-Anweisung für eine existierende Datei) wird ein *Dateikontrollsatz* DKS eingerichtet. Er wird aufgrund einer Delete-Anweisung entfernt.

G	Hinweis auf den GKS, der über die in UPON device genannte Variable identifiziert wurde
NDKS	Hinweis auf einen DKS oder leerer Hinweis, Verkettung der DKS bei einem GKS
FILK	Hinweis auf die Kette der zugeordneten FKS
USE	Anzahl der zugeordneten FKS (File-Kontrollsatz)
WP	Hinweis auf die Kette von PKS der Prozesse, die bis zur Aufhebung der Schreibsperre warten
LOK	Zähler, > 0 Schreibsperre gesetzt

BK	Hinweis auf die Kette der FKS, die für eine Übertragung belegt sind, um simultane Übertragungen zu koordinieren	
DEL	Hinweis auf den PKS des Prozesses, der auf die Ausführbarkeit seiner Delete-Anweisung wartet	
PUF	Hinweis auf den zugeordneten Pufferbereich	
ANF	Anfangsadresse des zugeteilten Speicherplatzes auf dem externen Datenträger	
TLG(1) TLG(2) TLG(3) TLG(4)	Dateilänge Seitenlänge Zeilenlänge Einheitenlänge	Die in der Create-Anweisung spezifizierte Dateigliederung bzw. die Standardgliederung des entsprechenden DVS
SCHE	Länge des beschriebenen Teils, Schreibende	
NAM	Der in TITLE zeichenkette angegebene Dateiname	

2.3.1.4 Filekontrollsatz

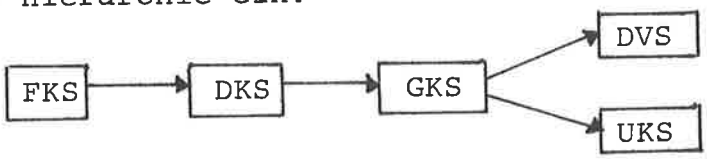
Variable der Art FILE haben als Wert den Hinweis auf einen *Filekontrollsatz* FKS. Er wird aufgrund einer Open-Anweisung oder Create-Anweisung mit einem DKS verknüpft, diese Verbindung wird durch eine Close- oder Delete-Anweisung gelöst. Das Betriebssystem benutzt sie zur Ablage von Steuergrößen für Übertragungen.

FN	Zeiger zur Verkettung aller mit einem DKS verknüpften FKS
DSZ	Zugriffsart (DIRECT, SEQUENTIAL, STRM)
FVA	Zeiger wie FN
DS	Hinweis auf den DKS der Datei, mit dem das FILE eröffnet ist
OP	Hinweis auf den PKS des eröffnenden Prozesses

TLG(1) TLG(2) TLG(3) TLG(4)	Dateilänge Seitenlänge Zeilenlänge Einheitenlänge	Gliederung (Adreßstruktur) nach der auf die Datei zuzugreifen ist
BINEK BINAK ALPEK ALPAK	Marken der Kontrollprogramme für die binäre Eingabe, binäre Ausgabe, zeichenorientierte Eingabe, zeichenorientierte Ausgabe	
EIN AUS	Marken der Steuerprogramme zur Einleitung einer Eingabe oder Ausgabe	
DOG	Identifizierung einer Prozedur zur Koordinierung simultaner Übertragungen für eine Datei und zur Ansprache des E/A-Geräts	
AP	Hinweis auf den PKS des Prozesses, der den FKS für eine Übertragung belegt hat	
WP	Hinweis auf die Kette der PKS von Prozessen, die auf Freigabe des belegten FKS warten	
REND	Marke des Programmstücks zum Abschluß einer Übertragung	
PUK	Hinweis auf den UKS des E/A-Geräts, das für den Datentransport zuständig ist	
N V	Zeiger zur dateispezifischen Verkettung aller für Übertragungen belegten FKS	
ZA	Hinweis auf die Kette der FKS, die für Übertragungen belegt sind, die mit der über diesen FKS abgewickelten Übertragung unverträglich sind	
SN	Zeiger zur Verkettung in einer Warteschlange des UKS	
PRIO	Absolutpriorität des belegenden Prozesses	
SV	Zeiger wie SN	
ANF	Anfangsadresse der Datei	
AST END	Stelle auf der Datei (relativ zu ANF),von der bzw. bis zu der übertragen werden soll	
STARB	Adresse eines Bereichs im Arbeitsspeicher, von dem bzw. auf den übertragen werden soll	

2.3.2 Dateiverwaltungsfunktionen

Das Betriebssystem bindet die in 3.1 dargestellten Datenstrukturen nach den Angaben des Programms in folgende Hierarchie ein.



Hier bedeutet A—>B, daß über A genau ein B erreicht wird und B mehrere A zugeordnet sein können.

Der Systemteil eines PEARL-Programms beschreibt, welche Peripheriegeräte (device-type) angeschlossen sind und führt auf ihnen Einheiten ein, die über ihre Bezeichnung (device-identification) ansprechbar sind. Bei seiner Auswertung wird für jedes in einer Anweisung (device-connection) genannte Gerät der Standardperipherie ein UKS und ein DVS erzeugt und initialisiert; jede Einheit wird durch einen GKS repräsentiert, der mit den Hinweisen auf einen UKS und einen DVS initialisiert ist.

Im Problemteil sind die Einheiten über ihre Bezeichnung (device-identification) bekannt und Variable der Art DEVICE, die eine Einheit der Standardperipherie identifizieren, haben als Wert einen Hinweis auf den entsprechenden GKS.

2.3.2.1 Vereinbarung einer Variablen der Art FILE

Für jeden mit dem Attribut FILE vereinbarten Bezeichner wird für die Lebensdauer dieser Variablen ein Filekontrollsatz FKS geführt.
Eine Anweisung der Form

DECLARE bezeichner FILE

wird intern durch

CALL ALFKS

dargestellt. Dieser Aufruf ist Bestandteil der Befehlsfolge zum Eröffnen eines Blockes.

(1) Aufruf von RESFKS (Speicherplatzverwaltung):
Im Systemraum wird Platz für einen FKS reserviert.

(2) Ablage der Adresse des FKS im Aktivierungssatz.

(3) Aufruf von INF1: Initialisierung des FKS als abgeschlossen d.h. der FKS kann durch Ausführung einer Create- oder Open-Anweisung einem DKS zugeordnet werden.

Beim Austritt aus einem Bereich wird für jede in ihm vereinbarte Variable der Art FILE folgendes Programm ausgeführt:

(1) Falls der FKS noch einem DKS zugeordnet ist, Aufruf von FILAB zur Auflösung dieser Zuordnung (implizites Abschließen).

(2) Aufruf von GFKSF (Speicherplatzverwaltung)
Freigabe des vom FKS belegten Platzes.

2.3.2.2 Die Create-Anweisung und die Open-Anweisung

Interne Darstellung einer Dateibeschreibung (file-specification)

Für jede in einer Create- oder Open-Anweisung angegebene Dateibeschreibung (file-spezification)

FILE(file-name) TITLE char-expression UPON (device-name)

{INPUT / OUTPUT / UPDATE} [(length,length,length)] {STREAM / SEQUENTIAL / DIRECT}

{[(length)] {ALPHA|BASIC} / {ALPHA|BASIC} data-attribut}

ist ein *Parametersatz* PARCO abzulegen.

FIL	Hinweis auf den über file-name erreichten FKS	
TIT	Zeichenkette der TITLE-Option	
DEV	Hinweis auf den über device-name erreichten GKS oder leerer Hinweis, falls Prozeßperipherie	
ART	Bitmuster zur Beschreibung des Dateityps	
TLG(1)	Seitenzahl der Datei	
TLG(2)	Zeilen pro Seite	
TLG(3)	Informationseinheiten pro Zeile	
TLG(4)	Länge einer Informationseinheit (z.B.in Bytes)	
GTLG(1) . . . GTLG(4)	Zahl . . . Zahl	Diese Größen sind nicht zu initialisieren, sie werden zur Zwischenspeicherung in den Funktionen CRLIST und OPLIST benutzt
SPBD	Zahl	
NPS	Zeiger	

Folgende Tabelle stellt den Wertebereich der Variablen ART dar:

Bit	
1 - 9	0
10	1, falls Standardgliederung, 0 sonst
11, 12	00 für ALPHA; 01 für BASIC
13, 14	01 für INPUT, 10 für OUTPUT, 11 für UPDATE
15, 16	00 für STREAM, 01 für SEQUENTIAL, 11 für DIRECT

Anlegen einer Datei

Eine Anweisung

CREATE file-specification;

wird intern dargestellt durch

(a) bei Angabe einer Gliederung (length,length,length)
CALL CRDE || ADDR Parametersatz

(b) bei Fehlen einer Gliederung
CALL CRSE || ADDR Parametersatz

und einen entsprechend initialisierten Parametersatz, der im Fall (a) nur bis TLG(4) und im Fall (b) bis ART abzulegen ist.

(a) Funktion: CRDE

(1) Testprogramm CRMD

(1.1) In PARMIN werden die Parameter übernommen und geprüft.
Eine Fehlerreaktion (Ausgabe einer Meldung und Prozeßabbruch) wird eingeleitet, wenn der FKS eröffnet ist, keine Einheit der Standardperipherie spezifiziert ist (leerer Hinweis in DEV) oder der in ART spezifizierte Dateityp auf dem E/A-Gerät nicht zulässig ist.
Anderenfalls Fortsetzung bei (1.2)

(1.2) In TEIL wird aus der angegebenen Gliederung eine normierte Darstellung z.B. Bytes pro Zeile, Bytes pro Seite und Bytes pro Datei ermittelt.
Fortsetzung bei (1.3)

(1.3) Ausführung des in CNST des Dateiverwaltungssatzes DVS spezifizierten Programmstücks (C1D, C1DK, CMD, CMDK):

Bei Geräten, auf denen nur eine Datei zugelassen ist, wird in C1D bzw. C1DK eine Fehlerreaktion

(Ausgabe einer Meldung und Prozeßabbruch) eingeleitet, falls schon eine Datei existiert. Anderenfalls Fortsetzung bei (1.3.1)

Bei Geräten, auf denen mehrere Dateien zugelassen sind, wird CMD bzw. CMDK ausgeführt. In NMVGL wird geprüft, ob der in TIT des Parametersatzes angegebene Dateiname schon für eine existierende Datei verwendet wurde und ggf. eine Fehlerreaktion (Ausgabe einer Meldung und Prozeßabbruch) eingeleitet.
Anderenfalls Fortsetzung bei (1.3.1)

(1.3.1) Falls einer der in TEIL ermittelten Werte für Datei-, Seiten- und Zeilenlänge größer als der entsprechende Wert von MTLG im DVS ist (Überschreiben der zugelassenen Maximalgliederung), wird eine Fehlerreaktion (Ausgabe einer Meldung und Prozeßabbruch) eingeleitet.
Anderenfalls wird die in SPBB des DVS angegebene gerätespezifische Routine zur Berechnung des Platzbedarfs ausgeführt. Bei Geräten mit beschränkter Speicherkapazität (C1DK, CMDK) wird bei (1.3.2) und bei Geräten mit unbeschränkter Speicherkapazität bei (2) fortgesetzt.

(1.3.2) Falls auf der Einheit zu wenig freier Speicherplatz für die anzulegende Datei existiert, wird die in RVOLL der DVS spezifizierte Reaktion auf Platzmangel eingeleitet: Ausgabe einer Fehlermeldung und Prozeßabbruch oder Zurückstellen des PKS in der Warteschlange WCR des GKS zur Wiederaufnahme des Prozesses bei Freigabe von Speicherplatz.
Wenn genügend Platz zur Verfügung steht, wird bei (2) fortgesetzt.

(2) Ausführungsprogramm CREO

(2.1) Anlegen eines DKS durch die Prozedur ALDKS:
Die Speicherplatzverwaltungsfunktion RESDKS reserviert Platz für einen DKS.

Einfügen des DKS in die bei DKSK des GKS geführte Liste. Übernahme der Adresse des GKS und des Dateinamens in den DKS.
Weitere Initialisierungen.
Fortsetzung bei (2.2)

(2.2) Ausführung der in SBEL des DVS spezifizierten Befehlsfolge: Belegen des benötigten Speicherplatzes und Notieren der Anfangsadresse im DKS.
Fortsetzung bei (2.3)

(2.3) Ausführung der in DKSKOP des DVS spezifizierten Befehlsfolge: Bei Geräten mit einem Hintergrundkatalog für Dateien wird dieser erweitert.

Fortsetzung ~~bei (02.1)~~ zur Eröffnung des in FIL des Parametersatzes angegebenen FKS mit dem neuen DKS bei der Funktion OPDO.

(b) Funktion: CRSE

(1) Testprogramm CROD

(1.1) Aufruf der Prozedur PARMIN wie in (a1.1)
Fortsetzung bei (1.2)

(1.2) Ausführung des in CST des DVS spezifizierten Programmstücks (CS1D, CSMDK, CSMD):

Bei Geräten, auf denen nur eine Datei zugelassen ist, wird in CS1D eine Fehlerreaktion (Ausgabe einer Meldung und Prozeßabbruch) eingeleitet, falls schon eine Datei existiert. Anderenfalls Fortsetzung bei (1.3).
Bei Geräten, auf denen mehrere Dateien zugelassen sind, wird CSMDK bzw. CSMD ausgeführt. In NMVGL wird geprüft, ob der in TIT des Parametersatzes angegebene Dateiname schon für eine existierende Datei verwendet wurde und ggf. eine Fehlerreaktion (Ausgabe einer Meldung und Prozeßabbruch) eingeleitet.
Anderenfalls Fortsetzung bei (1.3)

(1.3) Die in STLG und SPBD des DVS abgelegten Werte werden im folgenden als Dateigliederung und Speicherplatzbedarf übernommen. Fortsetzung bei (a1,3.2)

Anlegen einer Liste von n≥2 Dateien

Eine Anweisung

CREATE file-spezification,'';

wird intern dargestellt durch

```
CALL CRLIST  ||  ADDR   1. Parametersatz
             ||   :
             ||  ADDR   n. Parametersatz
```

und die n entsprechend initialisierten, vollständigen Parametersätze.

Funktion: CRLIST

(1) Zunächst wird für jeden Parametersatz eines der Testprogramme CRMD (siehe (a1)) oder CROD (siehe (b1)) ausgeführt. Falls sie nicht wegen eines Fehlers mit dem Abbruch des Prozesses oder mit dem Zurückstellen des Prozesses bis zur Freigabe von Speicherplatz enden, werden jeweils die ermittelten Werte der Dateigliederung und des Platzbedarfs in GTLG bzw. SPBB des Parametersatzes zwischengespeichert. Der Parametersatz wird in die bei NDA des GKS geführte Liste aufgenommen. Nach Bearbeitung des letzten Parametersatzes Fortsetzung bei (2).

(2) Für jeden angegebenen GKS, wird die Liste der angehängten Parametersätze daraufhin untersucht, ob die Summe der Speicherplatzanforderung aller auf dieser Einheit neu anzulegenden Dateien erfüllt werden kann. Bei Speicherplatzmangel wird das in RVOLL des entsprechenden DVS spezifizierte Programm ausgeführt, das entweder mit der Zurückstellung des Prozesses oder einer Fehlermeldung und dem Abbruch des Prozesses endet.
Eine Fehlerreaktion wird eingeleitet, wenn zwei auf der Einheit anzulegende Dateien denselben Namen haben.
Fortsetzung bei (3).

(3) Für jeden der n Parametersätze wird das unter (a2) beschriebene Ausführungsprogramm CREO abgearbeitet: Reservierung des Speicherplatzes (SPBD des Parametersatzes bestimmt die Größe), Aufbau und Initialisierung eines DKS (GTLG des Parametersatzes bestimmt die Gliederung), Eröffnen des in FIL bezeichneten FKS mit dem DKS.

Eröffnen einer Datei

Eine Anweisung

OPEN file-spezification;

wird intern dargestellt durch

CALL OPEN || ADDR Parametersatz

und einen entsprechend initialisierten Parametersatz, der bis TLG(4) abgelegt ist.

Funktion: OPEN

(01) Testprogramm OPTEST

(1.1) In PARMIN werden die Parameter übernommen und geprüft.
Eine Fehlerreaktion (Ausgabe einer Meldung und Prozeßabbruch) wird eingeleitet, wenn der FKS eröffnet ist, keine Einheit der Standardperipherie spezifiziert ist (Leerer Hinweis in DEV) oder der in ART spezifizierte Dateityp auf dem E/A-Gerät nicht zulässig ist.
Anderenfalls Fortsetzung bei (1.2)

(1.2) In NAMVGL wird die beim GKS geführte Liste der DKS nach dem in TIT des Parametersatzes angegebenen Namen durchsucht.

(i) DKS gefunden: Ist im DKS notiert, daß ein Prozeß auf die Ausführung seiner Delete-Anweisung wartet, wird der laufende Prozeß abgebrochen und eine Fehlermeldung ausgegeben.
Anderenfalls Fortsetzung bei (1.3)

(ii) DKS nicht gefunden: Die in OPN des DVS spezifizierte Routine wird ausgeführt.
Bei Geräten, die einen Hintergrundkatalog führen, wird dort nach einem Eintrag des vorgegebenen Namens gesucht. Bei Erfolg wird ein DKS eingerichtet, in die Liste des GKS aufgenommen und mit den im Hintergrundkatalog abgelegten Werten für die Gliederung initialisiert;

danach wird bei (1.3) fortgefahren.
wird kein Eintrag gefunden, erfolgt - wie bei Geräten ohne Hintergrundkatalog - eine Fehlerreaktion (Ausgabe einer Meldung und Prozeßabbruch).

(1.3) Ausführung von TEIL zur Berechnung der normierten Darstellung der Dateigliederung oder Übernahme der in TLG des DKS spezifizierten Größen, je nachdem, ob in der Anweisung eine Dateigliederung angegeben wurde oder nicht.
Fortsetzung bei (O2).

(O2) Ausführungsprogramm OPDO

(2.1) Zuordnung des ersten FKS zum DKS:
In DOG des FKS wird notiert, daß bei Übertragungen das E/A-Gerät direkt angesprochen wird.
Fortsetzung bei (2.3).

(2.2) Dem DKS sind schon andere FKS zugeordnet.
In DOG des (zu eröffnenden) FKS wird notiert, daß bei Übertragungen vor Ansprache des E/A-Geräts geprüft wird, ob auf das Ende einer Übertragung über einen der anderen FKS gewartet werden muß. Falls bisher nur ein FKS mit dem DKS eröffnet ist, wird sein DOG ebenso geändert.

(2.3) Initialisierung des FKS:
Einordnung in die Liste der mit dem DKS eröffneten FKS.
Kennzeichnung des FKS als unbenutzt und bereit für die in der Anweisung spezifzierte Art der Übertragung.

Eröffnen einer Liste von n≥2 Dateien

Eine Anweisung

```
OPEN file-spezification,'';
```

wird intern dargestellt durch

```
CALL OPLIST  ‖ ADDR   1. Parametersatz
             ‖  :
             ‖ ADDR   n. Parametersatz
```

und die n entsprechend initialisierten, vollständigen Parametersätze.

Funktion: OPLIST

(1) Zunächst wird für jeden Parametersatz das Testprogramm OPTEST (siehe (O1)) ausgeführt.
Falls es nicht wegen eines Fehlers mit dem Abbruch des Prozesses endet, werden die ermittelten Werte der Dateigliederung in GTLG des Parametersatzes und der Hinweis auf den DKS im FKS zwischengespeichert.
Nach Bearbeitung des letzten Parametersatzes, Fortsetzung bei (2).

(2) Für jeden der n Parametersätze wird das unter (O2) beschriebene Ausführungsprogramm OPDO abgearbeitet.

2.3.2.3 Die Close-Anweisung

Eine Anweisung

```
CLOSE FILE (file-name);
```

wird intern durch

```
CALL CLOSE  || ADDR   Filekontrollsatz
```

dargestellt.

Funktion: CLOSE

(1) Bei nichteröffnetem FKS wird eine Fehlermeldung ausgegeben und der Prozeß abgebrochen.

Ist der FKS für eine Übertragung belegt, wird der Prozeß zur Wiederholung des Aufrufs nach dem Ende der Übertragung zurückgestellt.
Anderenfalls Fortsetzung bei (2).

(2) In FILAB wird der FKS aus der Kette der dem DKS zugeordneten FKS gelöst. Falls nur ein anderer FKS zugeordnet bleibt und ein Prozeß zur Ausführung einer Delete-Anweisung zurückgestellt ist, wird er aus dem Wartezustand befreit. In der in CL des DVS spezifizierten Befehlsfolge wird z.B. eine Schreibendekennung auf die Datei ausgegeben.
Fortsetzung bei (3).

(3) In INF1 wird der FKS als abgeschlossen gekennzeichnet.

2.3.2.4 Die Delete-Anweisung

Eine Anweisung

```
DELETE FILE (file-name);
```

wird intern durch

```
CALL DELETE  || ADDR   Filekontrollsatz
```

dargestellt.

Funktion: DELETE

(1) Bei nichteröffnetem FKS, bei gesetzter Schreibsperre oder falls ein Prozeß zur Ausführung einer DELETE-Anweisung für diese Datei zurückgestellt ist, wird eine Fehlermeldung gegeben und der Prozeß abgebrochen.

Ist der FKS für eine Übertragung belegt, wird der Prozeß zur Wiederholung des Aufrufs nach dem Ende der Übertragung zurückgestellt.
Anderenfalls Fortsetzung bei (2).

(2) Sind mehr als ein FKS mit dem DKS eröffnet, wird der FKS in INF2 als abgeschlossen gekennzeichnet und der Prozeß zur Wiederholung seines Aufrufs zurückgestellt bis alle anderen FKS abgeschlossen sind (siehe Beschreibung von FILAB in CLOSE).
Ist der genannte FKS allein dem DKS zugeordnet, wird der DKS aus der Liste der Dateikontrollsätze des GKS entfernt und die Speicherplatzverwaltungsfunktion gibt den vom DKS belegten Platz frei. In der durch DEL des DVS spezifizierten Befehlsfolge wird - falls auf dem Gerät möglich - der Eintrag im Hintergrundkatalog gelöscht und der von der Datei belegte Speicherplatz freigegeben; dabei werden die beim GKS wegen Speicherplatzmangel zur Wiederholung ihrer Create-Anweisung zurückgestellten Prozesse aus dem Wartezustand befreit.
In INF1 wird der FKS als abgeschlossen gekennzeichnet.

2.3.2.5 Die Lock- und die Unlock-Anweisung

Die Anweisungen

$$\left\{\begin{matrix}\text{LOCK}\\ \text{UNLOCK}\end{matrix}\right\}\ \text{FILE(file-name)}\ [,\text{FILE(file-name)}\cdots];$$

werden intern durch

CALL LOCK	ADDR	1. Filekontrollsatz
CALL UNLOCK	⋮	
	ADDR	n. Filekontrollsatz

dargestellt.

Funktion: LOCK

(1) Falls einer der angegebenen FKS abgeschlossen ist, wird eine Fehlermeldung gegeben und der Prozeß abgebrochen.
Anderenfalls Fortsetzung bei (2).

(2) Für jeden der FKS wird im zugeordneten DKS der Sperrzähler LOK um eins erhöht. (Für die Datei ist die Schreibsperre gesetzt, wenn LOK > 0)

Funktion: UNLOCK

Für jeden angegebenen FKS wird geprüft, ob er eröffnet ist; falls er abgeschlossen ist oder der Sperrzähler LOK des zugeordneten DKS den Wert 0 hat, wird eine Fehlermeldung ausgegeben und der Prozeß abgebrochen.
Anderenfalls wird LOK um eins erniedrigt und beim Stand LOK = 0 werden in RELFKS alle bei dem DKS wegen gesetzter Schreibsperre zurückgestellten Prozesse zur Wiederholung ihrer Ausgabeanweisung aus dem Wartezustand befreit.

2.3.3 STEUERPROGRAMME FÜR DIE ÜBERTRAGUNG ZWISCHEN DATEIEN UND DEM ARBEITSSPEICHER

Die interne Darstellung einer E/A-Anweisung für eine Datei besteht aus einer Folge von Funktionsaufrufen für die Formatierung mit einer geeigneten Darstellung der notwendigen Parameter und einem Funktionsaufruf zur Steuerung des Datenaustauschs zwischen einem Bereich des Arbeitsspeichers und dem externen Datenträger.
Eine Get-, Put-, Take- oder Send-Anweisung spezifiziere durch

FILE(file-name [{ADVANCE / POSITION} {(seite[,zeile,einheit]) / ([seite],[zeile],[einheit])}]

die externe Endstelle; dann enthält ihre interne Darstellung einen der folgenden Funktionsaufrufe

Get-Anweisung :	CALL GETR	ADDR Parametersatz
Put-Anweisung :	CALL PUTR	
Take-Anweisung:	CALL TAKR	
Send-Anweisung:	CALL SENR	

und einen Parametersatz PARS

FIL	Hinweis auf den FKS
BA	Anfangsadresse des Bereichs im Arbeitsspeicher
BL	Länge des Bereichs (z.B. in Bytes)
T	Positionierungstyp
P(2)	Seitenangabe
P(3)	Zeilenangabe
P(4)	Einheitenangabe

Die folgende Tabelle zeigt, wie die Positionierungs-parameter zu initialisieren sind

Sprachform	Parameterwerte
Keine Positionsangabe	T = 0
ADVANCE (seite) POSITION (seite)	T = 1 für ADVANCE T = 4 für POSITION P(2) = seite
ADVANCE(seite,zeile,einheit) POSITION(seite,zeile,ein-heit)	T = 2 für ADVANCE T = 5 für POSITION P(2) = seite P(3) = zeile P(4) = einheit
POSITION([seite],[zeile], [einheit])	T = 6 P(2) = seite/-1 P(3) = zeile/-1 P(4) = einheit/-1

Im letzten Fall (T = 6) ist für eine im Quellcode fehlende Angabe der Wert -1 zu übergeben.
Falls keine Position oder nur seite angegeben ist, braucht der Parametersatz nur bis T bzw. P(2) abgelegt zu werden.

Funktion: GETR/PUTR/TAKR/SENR

(1) Ausführung der in ALPEK/ALPAK/BINEK/BINAK des FKS spezifizierten Befehlsfolge.

(i) Der FKS ist entweder nicht eröffnet oder bei seiner Eröffnung wurde eine andere Übertragungsart angegeben.
UEFHL: Ausgabe einer Fehlermeldung und Abbruch des Prozesses.

(ii) Der FKS ist für eine Übertragung belegt.
FBEL: Einketten des PKS in die Warteschlange WP des FKS und Zurückstellen des Prozesses zur Wiederholung seines Aufrufs von GETR/PUTR/TAKR/SENR bei Freigabe des FKS.

(iii) In DIR des FKS wird notiert, daß er für zeichenorientierte Eingabe/zeichenorientierte Ausgabe/binäre Eingabe/binäre Ausgabe benutzt werden soll.
Fortsetzung bei (3)/(2)/(3)/(2).

(2) Wenn die Datei unter Schreibschutz steht (Lock-Anweisung), wird der Prozeß zur Wiederholung seines Aufrufs von PUTR/SENR bei Aufhebung des Schreibschutzes zurückgestellt und der PKS in die Warteschlange WP des DKS gekettet.
Anderenfalls Fortsetzung bei (3).

(3) Auswertung der Positionierungsparameter

Bei der Eröffnung des FKS wurde definiert, wie bei seiner Ansprache für Übertragungen eine Position innerhalb der Datei bestimmt werden kann.
In TLG(1) = Gesamtlänge, TLG(2) = Seitenlänge, TLG(3) = Zeilenlänge, TLG(4) = Länge einer Informationseinheit ist die *Dateigliederung* beschrieben.
Die Zahl DSZ bestimmt den *Freiheitsgrad der Positionierung* bzgl. der laufenden Position: DSZ = 1 für STREAM (nur die laufende Position), DSZ = 2 für SEQUENTIAL (alle Positionen von der laufenden aufwärts), DSZ = 3 für DIRECT (keine Einschränkung).
Die *laufende Position* ist in END notiert und bezeichnet den Abstand der Informationseinheit von Dateianfang bis zu der beim letzten Zugriff auf den FKS übertragen wurde.

In POSBE wird eine durch DSZ des FKS und T des Parametersatzes PARS bestimmte Befehlsfolge ausgeführt.

(i) Die Angabe einer Position ($T \neq 0$) für einen mit STREAM (DSZ = 3) eröffneten FKS oder die Benutzung von POSITION ($T \geq 4$) bei einem nicht mit DIRECT ($DSZ \neq 3$) eröffneten FKS führt zu einer Fehlerreaktion (Ausgabe einer Meldung und Prozeßabbruch).

(ii) In allen anderen Fällen wird ggf. unter Benutzung der laufenden Position aus den Zahlen TLG(2), TLG(3), TLG(4) des FKS und den Zahlen P(2), P(3), P(4) des Parametersatzes PARS die Anfangsposition als Abstand zum Dateianfang ermittelt.
Ist diese Zahl größer als TLG(1) des FKS, d.h. ist das für diesen FKS definierte Dateiende überschritten, wird eine Fehlermeldung ausgegeben.
Anderenfalls Fortsetzung bei (4).

(4) Belegen des FKS für die Übertragung

Die in POSBE ermittelte Anfangsposition wird in AST des FKS abgelegt.
Die Endeposition für die Übertragung ergibt sich durch Addition von BL (Länge des Bereichs im Arbeitsspeicher) des Parametersatzes; ist diese Summe größer als TLG(1),

wird TLG(1) als Endeposition gewählt. Sie wird in END des FKS abgelegt.
In STARB des FKS wird die in BA des Parametersatzes angegebene Adresse des Bereichs im Arbeitsspeicher notiert.

In AP des FKS wird der Hinweis auf den PKS des laufenden Prozesses notiert.
Fortsetzung bei (5).

(5) Koordinierung simultaner Übertragungen für eine Datei

Vor der Ansprache des E/A-Geräts zur Ausführung des im FKS beschriebenen Übertragungsauftrags, wird die in DOG des FKS bezeichnete Prozedur aufgerufen.

Wenn der FKS seit seiner Eröffnung als einziger dem DKS zugeordnet ist, wird GERE aufgerufen.
Fortsetzung bei (6).
Anderenfalls wird zur Prüfung der Verträglichkeit des aktuellen Auftrags mit allen bereits an die Datei übergebenen Aufträgen (BK-Kette des DKS) das Programm DATE ausgeführt.
Unterträglichkeit besteht zwischen zwei Aufträgen, wenn die durch AST und END in den FKS beschriebenen Intervalle auf der Datei sich überschneiden und wenigstens einer der Aufträge ein Schreibauftrag ist.
Ist der aktuelle Auftrag (in FKS) unverträglich mit einem bereits an die Datei übergebenen Auftrag (FKS'), wird FKS in die ZA-Kette des FKS' eingekettet und bei (7) fortgefahren.
Anderenfalls wird der Auftrag an die Datei übergeben, d.h. der FKS wird in die BK-Kette des DKS aufgenommen und bei (6) fortgefahren.

(6) Ansprache des E/A-Geräts

Ausführung der in AN1 (Eingabekontrolle) bzw. AN2 (Ausgabekontrolle) des dem E/A-Gerät zugeordneten UKS spezifizierten Befehlsfolge (DFE, DFA, DBE, DSP)

DSP: Das Gerät ist nicht arbeitsbereit.
Über FD des UKS kann eine gerätespezifische Reaktion eingeleitet werden.
Ausgabe einer Fehlermeldung und Abbruch des Prozesses.

DBE: Das Gerät ist für eine Übertragung belegt.
Zurückstellen des FKS in der Warteschlange W des UKS.
Fortsetzung bei (7).

DFE: Einleiten der Eingabe durch Aufruf der durch ETR des UKS identifizierten Treiberroutine.
Belegen des Geräts für den FKS, Erwarten der Rückmeldung.
Fortsetzung bei (7).

DFA: Einleiten der Ausgabe durch Aufruf der durch ATR des UKS identifizierten Treiberroutine.
Belegen des Geräts für den FKS, Erwarten der Rückmeldung.
Fortsetzung bei (7).

(7) Der laufende Prozeß wird bis zum Abschluß der Übertragung zurückgestellt.

2.3.4 Sekundärreaktion auf eine Geräterückmeldung

Nach der Rückmeldung eines E/A-Geräts der Standardperipherie wird im Rahmen der Sekundärreaktion (siehe 1.6.2) der Transport einer weiteren Übertragungseinheit eingeleitet oder die laufende Übertragung abgeschlossen.

Funktion: DAT
Über den UKS des E/A-Geräts sind die notwendigen Steuerparameter zugänglich: AKT weist auf den FKS, der den laufenden Auftrag beschreibt und W auf eine (evtl. leere) Kette von FKS, die zurückgestellte Aufträge beschreiben.

(1) Falls alle Daten für den laufenden Auftrag übertragen sind, wird bei (2) fortgefahren. Anderenfalls wird das Gerät zum Transport einer weiteren technischen Übertragungseinheit veranlaßt und der Auftragspuffer für Geräterückmeldungen (1.6.2) weiter bearbeitet.

(2) Falls die Warteschlange W des UKS nicht leer ist, wird der erste Datentransport für den ersten zurückgestellten FKS veranlaßt. Anderenfalls wird im UKS notiert, daß das Gerät nicht belegt ist.
Fortsetzung bei (3)

(3) Abschluß einer Übertragung durch Ausführung der in REND des FKS spezifizierten Befehlsfolge.

(i) LANG: Mehr als ein FKS sind dem DKS zugeordnet.
Für alle FKS' aus der Kette ZA des aktuellen FKS wird die Verträglichkeitsprüfung fortgesetzt (siehe : 2.3.3(5)). Der FKS wird aus der Kette BK des DKS entfernt.
Fortsetzung bei (ii).

(ii) KURZ: Nur ein FKS ist dem DKS zugeordnet.
Im FKS wird notiert, daß er frei für eine Übertragung ist, alle in der Warteschlange W des FKS zurückgestellten Prozesse werden ebenso wie der beauftragende Prozeß aus dem Wartezustand befreit.

Weitere Bearbeitung des Auftragspuffers für Geräterückmeldungen (1.6.2).

3. SPEICHERPLATZVERWALTUNG

3.1 Einführung

In diesem Teil des Berichts wird die Speicherplatzverwaltung des PEARL-Betriebssystems beschrieben. Sie umfaßt die Verwaltung im Arbeitsspeicher sowie das Verfahren zur Verdrängung von momentan unbenutzten Datensätzen vom Arbeitsspeicher auf den Hintergrundspeicher.
Der Arbeitsspeicher wird in *Seiten* geteilt. Die Länge einer Seite ist eine Potenz von 2 und wird bei der Systemgenerierung festgesetzt (siehe: 3.4). Daten, die zusammenhängend in einem Satz bleiben müssen, werden in einem *Segment* abgelegt, das aus mehreren aufeinanderfolgenden Seiten bestehen kann, wenn der Datensatz länger als eine Seite ist. Segmente, die verdrängbar sind (siehe 3.5.3), dürfen auf den Hintergrundspeicher ausgelagert werden.

Da das allgemeine PEARL-Betriebssystem auf Rechnern mit unterschiedlichen Hardware-Konfigurationen implementiert werden kann, müssen die Systemdienste zur Speicherplatzverwaltung modular aufgebaut sein, so daß anhand der bei einer Implementierung gewählten Strategie die richtige Konstellation von Systemdiensten aus den allgemeinen Speicherplatzverwaltungsprogrammen gebildet werden kann. Dieses impliziert aber, daß im Konzept der Speicherplatzverwaltung viele der möglichen Strategien berücksichtigt werden.
Die Art der bei einer Implementierung gewählten Speicherplatzorganisation hängt aber nicht nur von der Hardware-Konfiguration sondern auch von den Möglichkeiten des PEARL-Compilers ab.
Eine Organisation, die z. B. erlaubt, daß alle Segmente (außer denen, die RESIDENT sind) auf den Hintergrund ausgelagert werden, verlangt vom Compiler eine sehr feine Analyse des Anwenderprogramms (z.B. um alle möglichen Programmgrößen herauszufinden, auf die ein gewisser Zeiger hinweisen kann). Es ist klar, daß eine umfangreiche Programmanalyse durch den Compiler, wenn überhaupt möglich, nicht immer wünschenswert ist, da ein solcher Compiler wahrscheinlich nicht arbeiten könnte, ohne bestimmte

Zwischendateien auf den Hintergrund auslagern zu können, was wiederum andere Speicherplatzverwaltungsprogramme notwendig machen würde.

Diese Betrachtungen waren bei der Realisierung der Systemdienste zur Speicherplatzverwaltung Leitfaden unserer Arbeit. Das Speicherplatzverwaltungsprogramm ist einheitlich in seinem Konzept, modular in seiner Realisierung, einerseits in Hinsicht auf die mögliche Hardware-Konfiguration und andererseits in Hinsicht auf die Schnittstellen zum Compiler.
Es wurde eine Speicherverwaltung implementiert, die

. nur Zumutbares vom PEARL-Compiler verlangt,

. nicht zuviel Verwaltungszeit beansprucht,

. eine günstige, aber nicht die optimale Speicherbesetzung ermöglicht.

Sie ist leicht reduzierbar in Richtung auf statische Verwaltung und leicht erweiterbar auf volldynamische Verwaltung.

In Kapitel 3.2 werden erst die möglichen Strategien zur Speicherplatzverwaltung beschrieben, die sich aus dem aufgeführten Programm herleiten lassen; danach wird anhand der verschiedenen Dateneigenschaften die Einteilung des Arbeitsspeichers dargestellt. In Kapitel 3.3 wird die Verwaltung der Prozeßsteuerlisten beschrieben. In Kapitel 3.4 wird das Konzept des Laufzeitkellers eines Prozesses erläutert und die interne Organisation der Größen im Laufzeitkeller beschrieben. Schließlich wird in Kapitel 3.5 die Strategie zur Auslagerung von Segmenten auf den Hintergrundspeicher dargestellt.

3.2 Organisation im Arbeitsspeicher

3.2.1 Daten und Datensätze

Die Methoden zur Verwaltung des Arbeitsspeichers werden stark von den Eigenschaften der benutzten Daten und Datensätze beeinflußt. Diese Daten und Datensätze zerfallen in drei Gruppen:

1. Systemdaten:
   Unterbrechungsbeschreibungen (Zeitkontrollsätze, Ereignisvariable, SIGNAL-Beschreibungselemente),
   Gerätekontrollsätze,
   Puffer und Kontrollisten des Systems,
   Systemfunktionen,
   Systemdaten.

Den Speicherplatzbedarf für die Unterbrechungsbeschreibungen und die Gerätekontrollsätze berechnet der Compiler anhand des Systemteils im Anwenderprogramm. Puffern und Kontrollisten werden beim Systemgenerieren feste Längen zugewiesen. Systemfunktionen und Systemdaten benutzen ebenfalls eine bekannte Anzahl von Speicherzellen.

2. Prozeßsteuerlisten:
   Prozeßkontrollsätze,
   SEMA- und BOLT-Kontrollsätze,
   ON-Elemente,
   Einplankontrollsätze,
   Adreßkontrollsätze (siehe 3.5.2),
   Datei- und Filekontrollsätze.

Prozeßkontrollsätze, SEMA- und BOLT-Kontrollsätze können in Bereichen mit evtl. variablen Grenzen - wenn nicht auf Modulebene vereinbart - zusammengefaßt werden. Einplankontrollsätze haben, je nach Komplexität der Startbedingungen variable Länge. Prozeßsteuerlisten (ausgenommen die Einplankontrollsätze) sind gewissermaßen durch die Blockstruktur des Programms verwaltet.

3. Programmdaten :
   arithmetische Daten,
   Zeitgrößen,
   Marken und Programmsteuerdaten (Rückkehradressen, usw.),
   Anwenderprozeduren und Moduln.

Arithmetische Daten, Zeitgrößen und Marken können indiziert mit evtl. variablen Grenzen (wenn nicht auf Modulebene vereinbart) auftreten; wenn nicht als GLOBAL vereinbart, sind sie durch die Blockstruktur des Programms verwaltet. Sie können RESIDENT sein.

3.2.2 Strategien zur Speicherplatzverwaltung

Hier werden mögliche Strategien zur Verwaltung des Arbeitsspeichers kurz beschrieben, zu deren Implementierung die aufgeführten Speicherplatzverwaltungsprogramme eingesetzt werden können.

3.2.2.1 Statische Speicherplatzverwaltung

Der Compiler berechnet die maximale Speicherbelastung durch ein Programm; er weist den Programmgrößen je nach der Blockstruktur des Programms Adressen zu; allen Daten und Dateien werden beim Binden feste, unveränderliche Adressen zugeordnet. Zur Laufzeit hat jede Programmgröße eine eindeutige Speicheradresse.
Statische Speicherplatzorganisation fordert also:

1. Programmdatenbereiche, SEMA-, BOLT- und TASK-Variablen-Bereiche mit fester, zur Compile-Zeit bekannter Länge.

2. Prozeduren, für die zur Compile-Zeit die maximale Anzahl der gleichzeitigen Benutzer berechnet bzw. festgesetzt werden kann. Hierfür gibt es zwei Möglichkeiten:

   entweder
   - Reentrantfähige Prozeduren (Attribut REENTRANT) sind verboten. Jede Prozedur darf nur einmal gleichzeitig ausgeführt werden, so daß die Größen, die innerhalb der Prozedur (lokale Größen) vereinbart sind, einen festen Speicherplatz benutzen.

     Dies kann auch virtuell realisiert werden, wenn das Attribut REENTRANT in der Sprachsyntax zugelassen ist, indem der Compiler einen REQUEST-Befehl über eine prozedurspezifische SEMA-Variable mit dem Initialwert 1 im Prozedurkopf und einem RELEASE-Befehl über dieselbe SEMA-Variable beim Prozedurrücksprung ablegt.

oder

- wenn der Rechner mindestens ein Basisregister besitzt, dürfen reentrantfähige Prozeduren n-mal gleichzeitig laufen, wobei die Anzahl n vom Compiler berechnet ist. In diesem Fall werden die prozedurlokalen Größen in einem sogenannten *Aktivierungssatz* (AS) zusammengefaßt und ihnen der Abstand zum Aktivierungssatzanfang als relative Adresse zugeordnet. Das Basisregister enthält die Anfangsadresse des Aktivierungssatzes. Prozedurlokale Größen werden indirekt über das Basisregister plus Abstand angesprochen.

  Der während der Laufzeit notwendige Test, ob die zulässige Anzahl von gleichzeitigen Ausführungen der Prozedur erreicht ist, wird durch die Erzeugung (vom Compiler) eines REQUEST-Befehls über eine prozedurspezifische SEMA-Variable mit Initialwert n (n kann entweder durch die Programmniederschrift vom Compiler einfach berechnet sein oder einen maximalen implementationsdefinierten Wert zugeordnet bekommen) im Prozedurkopf und eines RELEASE-Befehls beim Prozedurrücksprung gemacht.

Solche Verfahren werden nur für reentrantfähige Prozeduren benötigt. Programmgrößen, die in normalen Prozeduren definiert sind, werden feste Adressen zugewiesen.

Bei statischer Speicherplatzorganisation ist die Aufgabe des Betriebssystems zur Laufzeit gering: Testen, ob der aktuelle Block verlassen werden darf (d.h. Testen, ob ein im Block lokaler Prozeß aktiv oder eingeplant ist), die REQUEST- und RELEASE-Befehle bei reentrantfähigen Prozeduren ausführen.

Prozeßkontrollsätze, SEMA- und BOLT-Variable können im Aktivierungssatz abgelegt werden.
Einplankontrollsätzen wird der zur Compile-Zeit berechnete maximale Speicherraum zugeordnet. Der Compiler berechnet:

$$m^* \left(\sum \text{(maximale Speicherbelastung des Einplankontrollsatzes der betrachteten Steueranweisung)} \right)$$

wobei m die Dimension der TASK-Variablen ist (m = 1, wenn einfache TASK-Variable).

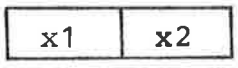
3.2.2.2 Pseudostatische Speicherplatzverwaltung

Der Unterschied zur statischen Speicherplatzverwaltung besteht im wesentlichen in der Benutzung von Bereichen variabler Länge. Die Ablage und die Verwaltung aller Größen, die nicht indiziert sind oder deren Index in festen, zur Compile-Zeit bekannten Grenzen liegt, wird statisch, wie oben erklärt, organisiert.

Die Anforderung und Freigabe von Speicherplatz für Bereiche mit variabler Länge (d.h. mit zur Laufzeit bestimmter Grenze) werden dagegen zur Laufzeit gesondert verwaltet. Bei Blockeröffnung wird die Länge des Bereiches berechnet und angemessener Speicherplatz im freien Raum angefordert. Wenn nicht genügend Speicherplatz zur Verfügung steht, muß der anfordernde Prozeß in Wartezustand auf Speicherplatzfreigabe gesetzt werden.

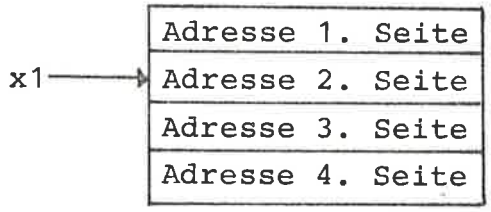
Bei Blockaussprung wird der besetzte Speicherbereich freigegeben. Da die Prozesse asynchron zueinander laufen können, entstehen Lücken im Hauptspeicher, die wieder für die Ablage anderer Bereiche benutzt werden könnten. Um die Benutzung dieser Lücken zu organisieren, stehen im Prinzip zwei Methoden zur Verfügung:

- die Seitenabbildungsmethode: Das freie Gebiet des Hauptspeichers wird in Seiten, deren Länge eine Potenz von 2 ist (z.B. 512 Wörter), geteilt. Bei Blockeröffnung wird die Länge des Bereiches berechnet und die notwendige Anzahl von Seiten angefordert. Wenn nicht genügend freie Seiten zur Verfügung stehen, wird der anfordernde Prozeß zurückgestellt und in Wartezustand auf Speicherplatzfreigabe gesetzt. Anderenfalls werden die notwendigen Seiten genommen. Da diese Seiten in völlig getrennten physikalischen Gebieten liegen können, ist die Benutzung einer Abbildungsfunktion für Adressen von Bereichselementen notwendig: Der Abstand eines Bereichselements zum Bereichsanfang sei x; x wird in zwei unterteilt:

  wobei x2 den Abstand zum Seitenanfang und x1 die Seitennummer ist.

x1 gibt die Komponente eines Beschreibungsvektors, in dem die Seitenanfangsadressen abgelegt sind, an



Die Seitenanfangsadresse plus x2 ergibt die physikalische Adresse des Bereichselements.

Die Benutzung einer Abbildungsfunktion kostet aber viel Speicherplatz, wenn die Abbildung im Anwenderprogramm eingebettet ist (vier bis sechs zusätzliche Befehle pro Adressberechnung) und Zeit (i.a. das Dreifache der normalen Referenzzeit in einem zusammenhängenden Bereich). Ein Vorteil der Methode ist aber, daß sie den Systemdienst nicht belastet und daß keine Zeit beim Verschieben verloren wird (siehe unten).
Für Rechner mit Hardware-Seitenabbildung wird natürlich die Seitenabbildungsmethode verwendet.

- die Verschiebungsmethode: Bereiche, die von aktivierten oder eingeplanten Prozessen benutzt werden, werden geschoben und zusammengefaßt, so daß nur eine Lücke bleibt.
Die Lücken sind auf diese Art in einem Speichergebiet gesammelt. Die Verschiebung eines Bereiches dauert aber relativ lange; während dieser Zeit müssen die Prozesse, die auf diesen Bereich Zugriff haben, gesperrt werden. Diese Methode birgt dadurch die Gefahr, daß das Betriebssystem bei Unterbrechungen i.a. nicht die geforderte, kurze Reaktionszeit im Anwenderprogramm sicherstellen kann.
Diese Gefahr kann aber vermindert werden, indem die Segmente prinzipiell zusammengeschoben werden, wenn kein Anwenderprozeß im Rechner arbeitet. Die Verschiebung wird in diesem Fall unter Regie von PRIOKE (siehe 3.5.1.1) ausgeführt. Der Programmierer hat auch die Möglichkeit, wichtige Datenbereiche

als RESIDENT zu spezifizieren. Solche Bereiche werden dann in einem statischen Raum des Arbeitsspeichers abgelegt, in dem sie nicht berührt werden.
Ein Vorteil der Verschiebungsmethode ist, daß große Bereiche nicht in Stücke geschnitten werden, so daß keine Abbildungsfunktion für Adressen von Bereichselementen notwendig ist.

Bei pseudostatischer Speicherplatzverwaltung sind neben den Unterprogrammen zur statischen Speicherplatzorganisation folgende Funktionen des Betriebssystems notwendig:

- Speicherplatzanforderungen der Prozesse bearbeiten. Wenn kein Speicherplatz mehr frei ist, den anfordernden Prozeß zurückstellen.
- Speicherplatzfreigabe der Prozesse notieren. Evtl. auf Speicherplatzfreigabe wartende Prozesse fortsetzen.

3.2.2.3 Dynamische Speicherplatzverwaltung

Bei dieser Verwaltung muß mindestens ein Basisregister zur Verfügung stehen. Bereiche mit variabler Länge und reentrantfähige Prozeduren sind erlaubt.
Systemdaten werden in ihrem reservierten *Systemraum* des Arbeitsspeichers und Prozeßsteuerlisten im sogenannten *statischen Raum* abgelegt (siehe 3.2.3).
Einfache Programmgrößen sind in aufrufspezifischen Aktivierungssätzen zusammengefaßt. Aktivierungssätze und lokale Bereiche werden dynamisch verwaltet. Adressen von einfachen Programmgrößen ergeben sich aus ihrem relativen Abstand zum AS-Anfang plus AS-Anfangsadresse.
Die Folge aller Aktivierungssätze eines Prozesses, die die dynamische Ansprache von Prozeduren bis zu einem bestimmten Zeitpunkt darstellen, werden im sogenannten *Laufzeitkeller* zusammengefaßt. Ein Laufzeitkeller bildet ein Segment. Andere Segmente enthalten den Befehlscode der Anwendermoduln und Prozeduren. Große Bereiche werden getrennt vom Laufzeitkeller in eigenen Segmenten abgelegt.

Alle Angaben zur Organisation der Ansprache der lokalen, einfachen Daten und alle Verwaltungsgrößen des Laufzeitkellers (Rücksprungparameter, Umgebungsvektor zur Ablage von AS-Anfangsadresse der übergeordneten Prozedur, usw.), die vom System benötigt werden, werden im Prozedurkopf sowie beim Prozeduraufruf behandelt. Alle Angaben zur Organisation der Ansprache von lokalen Bereichen werden beim Blockeintritt oder Blockaussprung behandelt. Beim Blockeintritt wird Speicherplatz für die blocklokalen Größen angefordert. Steht nicht genügend Platz zur Verfügung, wird der anfordernde Prozeß zurückgestellt.

Bei Blockaussprung wird der besetzte Speicherplatz freigegeben. Der aufgerufene Systemdienst setzt dann evtl. zurückgestellte Prozesse wieder fort.

3.2.2.4 Zusammenfassung der Strategien und gewählte Organisation

Man kann die Strategien zur Speicherplatzverwaltung so klassifizieren:

Statische Speicherplatzverwaltung:
Kein Bereich mit variabler Länge zugelassen.
Reentrantfähigkeit der Prozeduren:
- Verboten, wenn kein Basisregister zur Verfügung steht,
- begrenzt (Compiler-organisiert), wenn ein Basisregister existiert.

Pseudostatische Speicherplatzverwaltung:
Bereiche mit variabler Länge zugelassen.
Reentrantfähigkeit der Prozeduren: wie bei der statischen Speicherplatzverwaltung.
Auswählbare Strategien zur Benutzung der Lücken im dynamischen Raum:
- Seitenabbildungsmethode,
- Verschiebungsmethode,
- keine Seitenabbildung und keine Verschiebung.

Dynamische Speicherplatzverwaltung:

Bereiche mit variabler Länge und reentrantfähige Prozeduren zugelassen.
Auswählbare Strategien zur Benutzung der Lücken im dynamischen Raum: wie bei der pseudostatischen Speicherplatzverwaltung.

Gewählte Organisation:

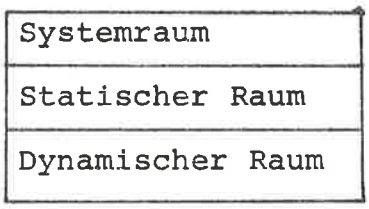
Es wurde die dynamische Speicherplatzverwaltung implementiert, ohne Seitenabbildung und ohne Verschiebung im Arbeitsspeicher und zwar aus folgenden Gründen:

- Die dynamische Verwaltung ist die allgemeinste und läßt sich leicht zur statischen reduzieren.
- Die Verschiebungsmethode verlangt eine umfangreiche Analyse vom Compiler, und das Verschieben ist zeitaufwendig.
- Die Seitenabbildung ist bei Rechnern ohne Basisregister auch zeitaufwendig.

Sowohl die Verschiebungs- wie die Seitenabbildungsmethode sind aber leicht in das implementierte System einzubauen.

3.2.3 Einteilung des Arbeitsspeichers

Der Arbeitsspeicher wird entsprechend den drei Gruppen von Daten und Datensätzen (siehe 2.1) in drei Gebiete geteilt:



Systemdaten werden im *Systemraum* des Arbeitsspeichers bei Systemgenerierung abgelegt. Für Prozeßkontrollsätze von Prozessen, die als GLOBAL TASK vereinbart sind, SEMA- und BOLT-Variable und Programmdaten, die das Attribut GLOBAL besitzen, werden ebenfalls beim Binden feste Speicherplätze im Systemraum vergeben.

Im *statischen Raum* werden die Prozeßsteuerlisten abgelegt. Der Compiler berechnet die Länge des statischen Raums, indem er die größte Summe der Prozeßsteuerlisten bildet, die gleichzeitig existieren können. Innerhalb des statischen Raums wird Speicherplatz für die Prozeßsteuerlisten nach Bedarf zur Laufzeit reserviert oder freigegeben. Datensätze im statischen Raum werden nie auf den Hintergrund ausgelagert.

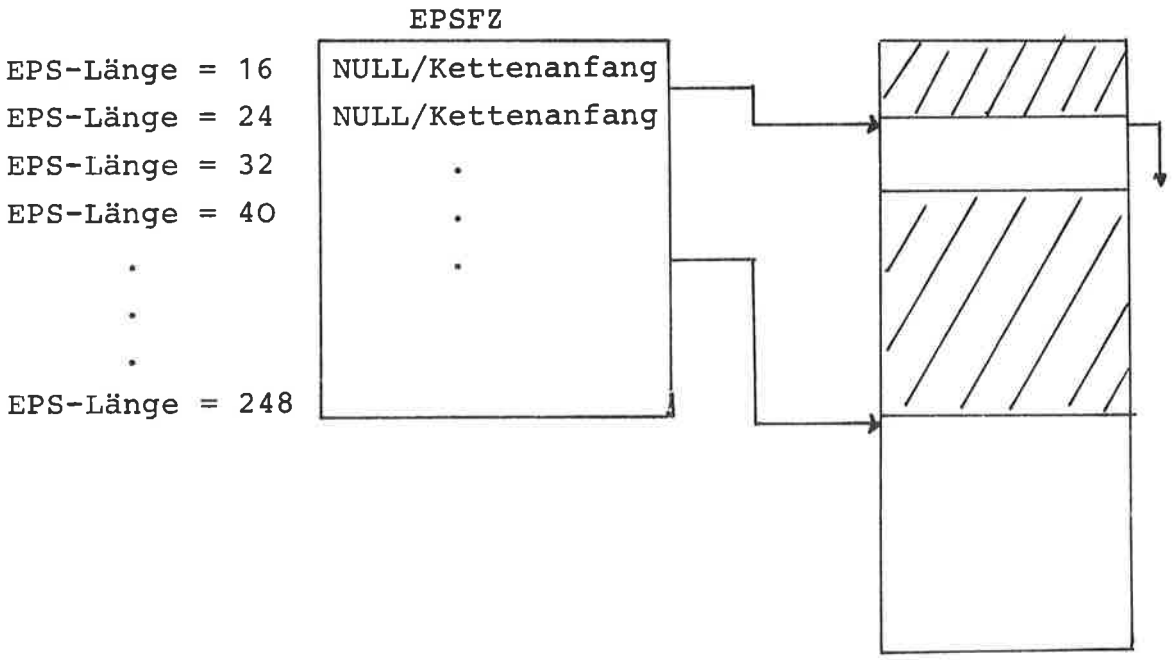
Der *dynamische Raum* enthält die Segmente, in denen die Programmdaten abgelegt sind, die nicht als GLOBAL vereinbart sind. Segmente von Programmdaten, die nicht als RESIDENT spezifiziert sind, dürfen auf den Hintergrund (falls vorhanden!) ausgelagert werden.

3.3 Verwaltung der Prozeßsteuerlisten

Prozeßsteuerlisten werden im statischen Raum des Arbeitsspeichers abgelegt. Der statische Raum wird in Seiten geteilt, die entweder Einplankontrollsätze oder die anderen Steuerlisten enthalten. Wir haben angenommen, daß eine Seite 256 Zellen besitzt. Steuerlisten bleiben im Arbeitsspeicher unberührt solange sie leben.

3.3.1 Verwaltung von Einplankontrollsätzen (EPS)

EPS sind innerhalb des statischen Raumes in Seiten, die nur EPS enthalten, gespeichert.
Freigewordene EPS von gleicher Länge (auf die nächste durch 8 teilbare Zahl aufgerundet), werden miteinander verkettet und können später für EPS der gleichen Länge wieder benutzt werden.
Im Bereich EPSFZ ist unter Index i der Zeiger auf die Kette der freien EPS der Länge 8*i eingetragen.



Speicherplatzanforderung für einen EPS führt das Unterprogramm STPEPS aus:
Wenn Platz für ein EPS von der Länge L angefordert wird, wird erst gesucht, ob die durch EPS-Länge = L definierte Kette leer ist. Ist sie nicht leer, wird der EPS reserviert und die Kette um das entsprechende Glied gekürzt. Ist sie leer, so wird gesucht, ob in einer Kette von freigewordenen EPS mit größerer Länge L' ein Element zur Verfügung steht. Wenn ja, wird es für den neu aufzubauenden EPS genommen und der restliche Teil in die Kette von freigewordenen EPS mit Länge L'-L eingetragen. Wenn kein Element gefunden ist, wird eine bisher freigebliebene Seite im statischen Raum für die Ablage von EPS genommen (die eigentlich durch den Zeiger von EPS-Länge = 248 angezeigt sein würde). Der neu aufzubauende EPS wird am Anfang dieser Seite abgelegt und der restliche Teil der Seite in die Kette der freigewordenen EPS mit Länge 248-L eingetragen.

Die Freigabe eines EPS wird im Unterprogramm GEPSF ausgeführt: Wenn ein EPS freigegeben ist, wird erst gesucht, ob die EPS unmittelbar danach und/oder davor ebenfalls frei sind; falls ja, werden die angrenzenden freigewordenen EPS zusammengefaßt. Dieser freie Platz wird in die Kette der passenden Länge eingekettet.

3.3.2 Verwaltung der übrigen Steuerlisten

Alle übrigen Steuerlisten haben je nach Art eine feste Länge. Für jede Art wird eine Kette von freien Listen geführt.

Die folgende Tabelle zeigt zugehörig zu jeder Steuerlistenart den Anforderungs- und Freigabeaufruf und den Index, unter dem im Feld FGEL der Anfang der Freikette zu finden ist:

Steuerlistenart	Index	Anforderung	Freigabe
DKS	1	RESDKS	GDKSF
normaler PKS	2	RENPKS	GPKSF
FKS	3	RESFKS	GFKSF
kurzer PKS	4	REKPKS	GPKSF
ADKS	5	RADKS	GADKSF
RSPEL	6	RESRSP	GRSPF
BOLT	7	REBOLT	GBOLTF
SEMA	8	RESEMA	GSEMAF
HSKS	9	RHSKS	GHSKSF

In STEFZ ist der Anfang des freien Teils der letzten unbenutzten Seite im statischen Speicherraum eingetragen und in STREST die noch verbleibende freie Länge auf dieser Seite.

Bei Anforderung einer Steuerliste eines bestimmten Typs wird geprüft, ob der entsprechende Zeiger im Feld FGEL leer ist (NULL) oder eine Kette von freien Steuerlisten anzeigt.
Ist eine Kette vorhanden, wird ihr erstes Element bereitgestellt und der Anfangszeiger in FGEL berichtigt.
Ist der Zeiger leer, wird der benötigte Speicherplatz vom Ende des letzten belegten Segments bzw. wenn das nicht reicht, von einem neuen Segment, bereitgestellt und STEFZ und STREST berichtigt.

Bei Freigabe einer Steuerliste eines bestimmten Typs wird die nunmehr freie Liste in die entsprechende Kette eingehängt.

3.4 Verwaltung der Programmdaten, Laufzeitkeller

Programmdaten werden im dynamischen Raum des Arbeitsspeichers in Segmenten verwaltet. Ein Segment besteht aus einer oder mehreren aufeinanderfolgenden Seiten. Die Länge einer Seite ist eine Potenz von 2 und wird bei der Systemgenerierung festgesetzt, so daß auch der längste Aktivierungssatz der Anwenderprozeduren in einer Seite Platz hat.

In diesem Betriebssystem haben wir eine Seitenlänge von 256 Zellen angenommen.

Es gibt Segmente von Prozedurcode, Laufzeitkellern von Prozessen und großen Bereichen, die getrennt von Laufzeitkellern gespeichert werden.

3.4.1 Allgemeine Vorgehensweise

Beim Start des Anwenderprogramms befinden sich schon alle Code-Segmente - oder zumindest die, die Modul-Prozessen angehören - im Arbeitsspeicher. Segmente, die Laufzeitkeller oder Datenbereiche enthalten, werden dynamisch reserviert und freigegeben. Im Kopf jedes Segmentes ist ein *Segmentkontrollsatz* SKS abgelegt.

Eine ausführliche Beschreibung des SKS ist im Abschnitt 3.5 zu finden; ein SKS enthält unter anderem folgende Informationen:

- eine Kennung: Befehlscode oder Laufzeitkeller oder Datenbereich oder Lücke (leeres Segment),
- zwei Zeiger zur Verkettung,
- Anzahl der Seiten.

Freie Segmente werden mit Hilfe ihrer SKS in Ketten, die jeweils Segmente gleicher Länge enthalten, verwaltet.

Das Feld FS enthält unter dem Index i den Zeiger auf die Kette der freien Segmente, die i Seiten groß sind oder eine Leerinformation, falls keine freien Segmente dieser Größe vorhanden sind, wobei Segmente, die größer als 32 Seiten sind, alle in einer Kette, die bei FS(33) beginnt, geführt werden.

Wird ein Segment von n Seiten angefordert, wird in FS ab Index n gesucht, ob freie Segmente von n oder mehr Seiten vorhanden sind. Wird ein freies Segment von n+m Seiten gefunden, werden seine ersten n Seiten bereitgestellt und die restlichen m Seiten mit einem eigenen SKS versehen und unter FS(m) eingekettet.

Wird ein Segment freigegeben, wird geprüft, ob unmittelbar davor und/oder dahinter freie Segmente liegen und gegebenenfalls all diese Segmente zusammengefaßt und als ein einziges größeres in die entsprechende Kette eingehängt.

3.4.2 Laufzeitkeller

Für jeden Prozeß mit normalem PKS wird ein *Laufzeitkeller* LZK eingerichtet. In ihm werden (dynamisch) für alle vom Prozeß aufgerufenen Prozeduren *Aktivierungssätze* AS auf- und abgebaut.

LZK

Segmentkontrollsatz SKS
AS des Segments
AS der ersten aufgerufenen Prozedur
AS der zweiten aufgerufenen Prozedur
. . .

Ein Prozeß mit kurzem PKS hat keinen eigenen LZK. Für ihn gelten alle folgenden Bedingungen:

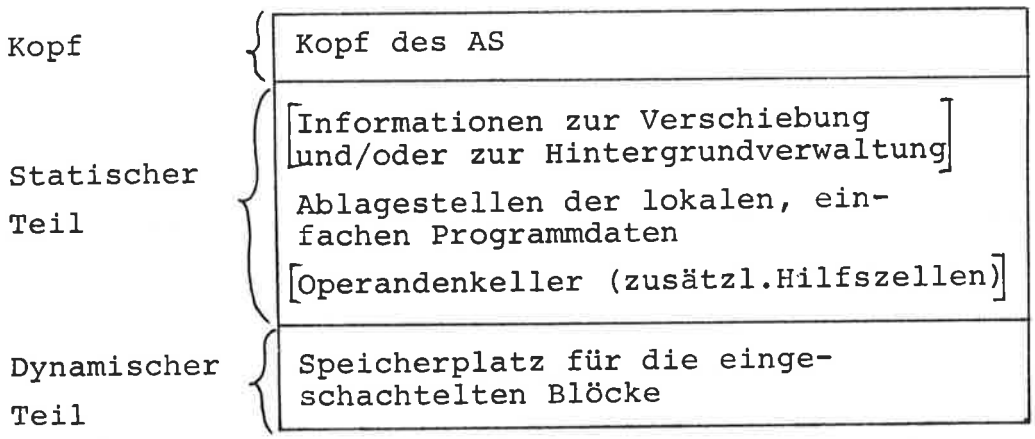
- Der Code enthält keine Vereinbarungen, Prozeduraufrufe oder On-Anweisungen.
- Bei seiner Aktivierung wird nicht die Sema-Option benutzt und die Aktivierung braucht nicht "gepuffert" zu werden.
- Er hat keine Unterprozesse.

Er benutzt den Laufzeitkeller des übergeordneten Prozesses.

3.4.2.1 Aktivierungssatz

Der Aktivierungssatz einer Prozedur oder Task enthält Verwaltungsdaten, die während der gesamten Ausführungszeit der Prozedur benötigt werden und alle in der Prozedur lokalen Programmdaten, mit Ausnahme der Bereiche oder Strukturbereiche, die getrennt abgelegt werden. Ein Aktivierungssatz ist für einen speziellen Aufruf abgelegt und wird beim Verlassen der Prozedur ungültig.

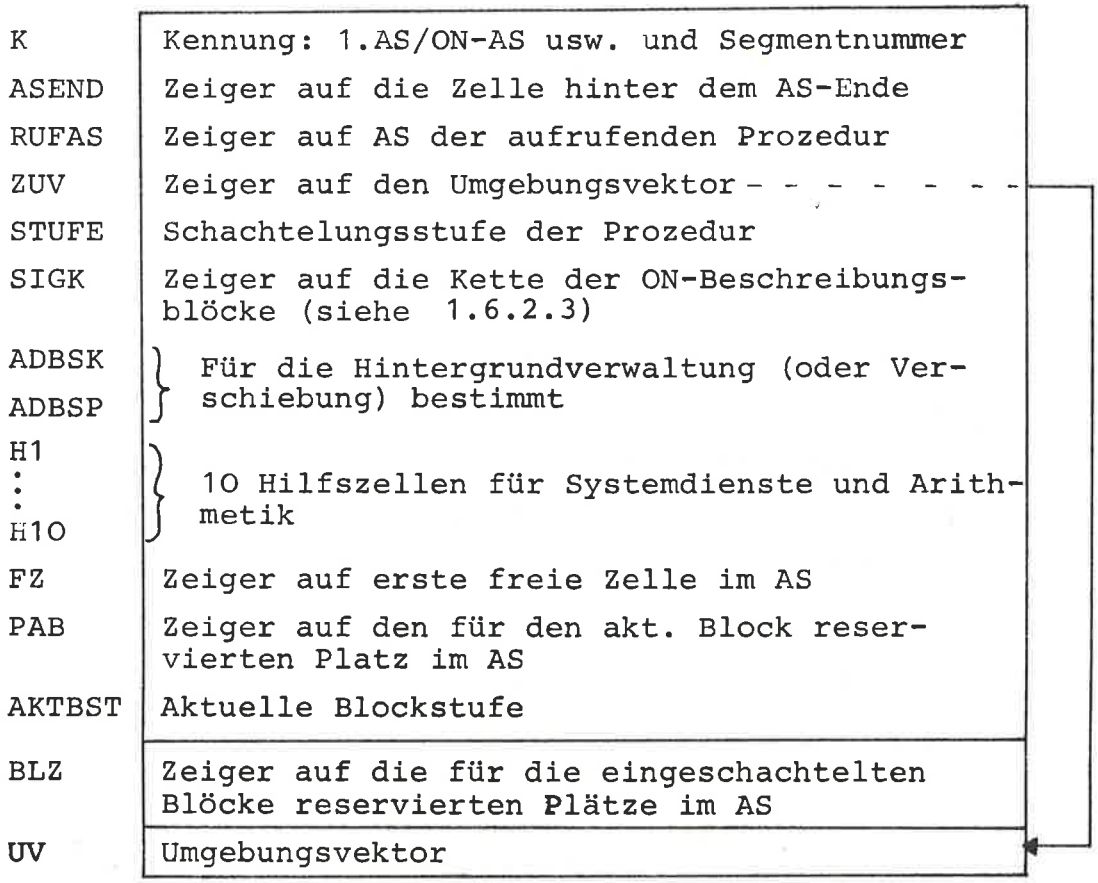
Ein Aktivierungssatz besteht also aus einem Kopf, aus einem statischen Teil und aus einem dynamischen Teil. Er sieht im allgemeinen so aus:



Der Operandenkeller dient zur Ablage von Zwischenergebnissen, Adressen zur Ansprache von globalen Größen und Parametern etc.

Falls in der Prozedur, für die der Aktivierungssatz aufgebaut wurde, Segmente (im Sinne von PEARL) von Prozessen mit kurzen PKS eingebettet sind, wird für jeden dieser Prozesse im statischen Teil ein Operandenkeller reserviert.

3.4.2.1.1 Der Kopf des AS enthält folgende Informationen:



Aktivierungssätze werden aus Gründen der Verwaltungsvereinfachung nur für Tasks und Prozeduren angelegt. Eingeschachtelte BEGIN-Blöcke werden anders verwaltet.

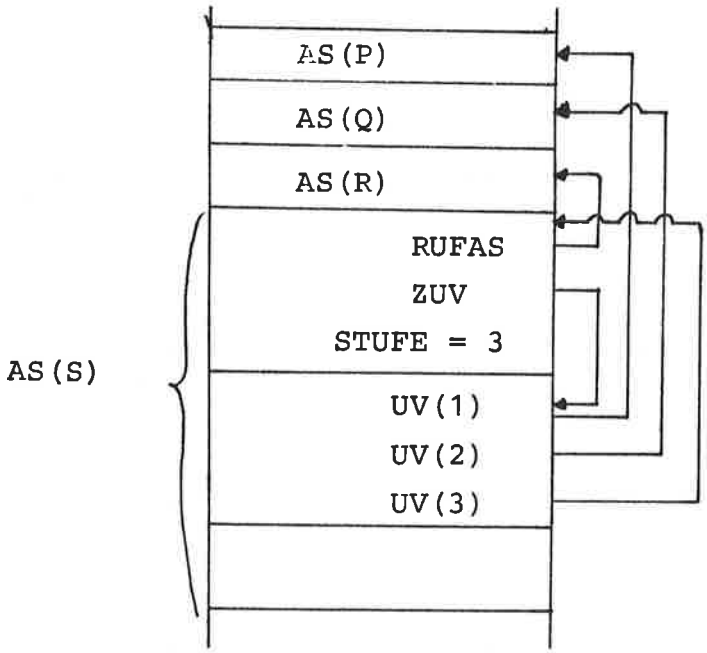
STUFE ist die von der Programmniederschrift her statisch bestimmte Schachtelungsstufe, wobei reine Beginblöcke nicht mitgezählt werden.

Der Umgebungsvektor (UV) enthält Zeiger auf die Aktivierungssätze aller Task- und Prozedurblöcke, die von der Programmniederschrift her die Prozedur umgeben; die Anzahl der Umgebungszeiger ist STUFE; der letzte Umgebungszeiger zeigt auf den Anfang des aktuellen AS selbst. Der Umgebungsvektor wird benötigt, um globale Größen in allen umgebenden Blöcken anzusprechen.

Beispiel: Sei die Prozedurschachtelung so organisiert:

```
P: ⌈  CALL Q
   Q: ⌈   CALL R
      R: ⌈  CALL S
         ⌊
      S: ⌈
         ⌊
   ⌊
```

Wenn **P** die Schachtelungsstufe 1 hat, hat S die Stufe 3. Der Umgebungsvektor im Aktivierungssatz der Prozedur S enthält Hinweise auf die Aktivierungssätze von P und Q (aber nicht R) und sich selbst..



3.4.2.1.2 Verwaltung von BEGIN-Blöcken

Die vereinfachte Verwaltung von reinen BEGIN-Blöcken wird im AS der umgebenden Task oder Prozedur durchgeführt. Einfache lokale Variable werden dem Speicherplatz für lokale Variable im statischen Teil des AS der umgebenden Task oder Prozedur zugeschlagen.
Nur BEGIN-Block-lokale Bereiche, TASK-, SEMA-, BOLT- und FILE-Variable, ON-Anweisungen und Segmente (im Sinne von PEARL) werden blockspezifisch im dynamischen Teil des AS der umgebenden Task oder Prozedur verwaltet. BEGIN-Blöcke, die nicht solche lokalen Größen enthalten, kann der Compiler vernachlässigen.

BLZ ist ein Bereich der Länge mb, wobei mb die maximale Blockstufe in der Prozedur bzw. Task ist, zu der der AS gehört. BLZ(i) zeigt auf den Anfang des Speicherplatzes, der in Blockstufe i reserviert wurde.

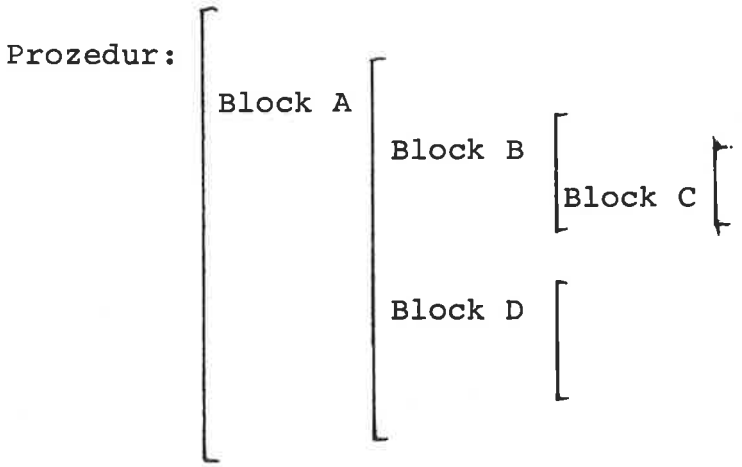
FZ zeigt auf die erste freie Zelle des AS, d.h. hinter den letzten reservierten Speicherplatz für die aktuelle Blockstufe.

AKTBST ist die aktuelle Blockstufe.
PAB ist gleich BLZ(AKTBST).

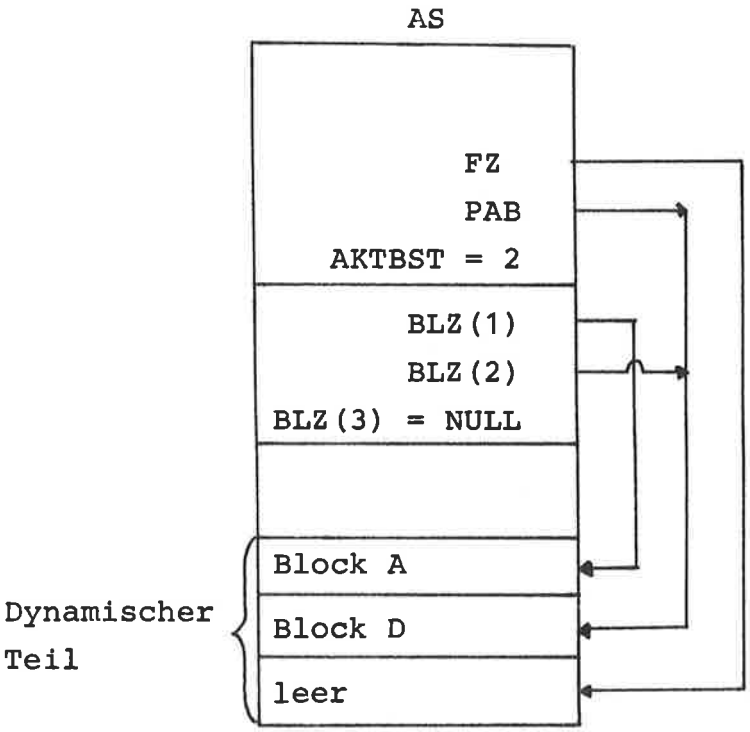
Der für einen Block reservierte Speicherplatz enthält:

- einen Kopf BD, der folgende Komponenten hat:
  - K Kennung
  - Z Anzahl der zugreifenden Prozesse
  - PA Anzahl der lokalen TASK-Variablen
  - SA Anzahl der lokalen SEMA-Variablen
  - BA Anzahl der lokalen BOLT-Variablen
  - FA Anzahl der lokalen FILE-Variablen
  - BLA Anzahl der lokalen Bereiche
  - ADBSP für die Hintergrundverwaltung,
- für jede lokale Variable der oben genannten Arten einen Zeiger auf die entsprechende Struktur zur Verwaltung der Variablen.
  Die Zeiger auf diese Variablen sind in der im Kopf aufgeführten Reihenfolge hintereinander abgelegt.

Beispiel: Die Blockschachtelung in der Prozedur sei so organisiert:



Wenn man sich im Block D befindet, ist die Blockstufe gleich 2.
BLZ hat drei Komponenten, da die maximale Blockstufe 3 ist (Block C).
Der Aktivierungssatz der Prozedur enthält folgendes:



3.4.2.1.3 Darstellung von Marken im Aktivierungssatz einer Prozedur

Hier wird die Darstellung von Markenkonstanten (in PEARL *value-label*) und ähnlich verwalteten Marken (Prozedureingangsstellen, Rückkehradressen) beschrieben. Markenvariable (in PEARL *reference-one-label*) werden wie Zeiger (*reference-two-identifier*) verwaltet.

Markenkonstanten, Adressen von Prozedureinsprungstellen und Rückkehradressen von Prozeduren sind in Codesegmenten definiert und werden in Laufzeitkellern der Prozesse beschrieben.
Bei Prozedureintritt werden Beschreibungen von lokalen Markenkonstanten im Aktivierungssatz der Prozedur eingetragen.

Ein *Markenbeschreibungsblock* (MB) im AS hat drei Einträge:

- Zeiger auf den aktuellen AS
- BEGIN-Blockstufe in der die Marke definiert ist
- Physikalische Markenadresse.

Ein solcher Markenbeschreibungsblock kann durch Aufruf einer Bibliotheksfunktion LABINI, die nicht Bestandteil des gelieferten Betriebssystems ist, erzeugt werden; sie würde folgendermaßen parametriert:

CALL LABINI	relative Adresse der Marke im Code-Segment
	BEGIN-Blockstufe
	relative Ablage des MB im AS

Für Markenkonstante, die weder von weiter eingeschachtelten Blöcken oder Prozeduren aus angesprungen,noch an Markenvariable zugewiesen, noch als Parameter übergeben werden, ist es nicht nötig, einen Markenbeschreibungsblock anzulegen. Sprünge zu solchen Marken können ohne Aufruf des Systemdienstes BLAUS (siehe 3.4.2.6) übersetzt werden.

Markenvariable (*reference-one-label*) werden wie Zeiger auf Markenkonstanten verwaltet. Dabei ist zu berücksichtigen, daß sie in der internen Darstellung nicht Markenkonstanten von höheren Block- oder Prozedurstufen zugewiesen bekommen, die nach dem Verlassen der Stufe fälschlicherweise noch benutzt werden könnten.
Parametermarken werden wie normale Parameter verwaltet.
Wichtig ist auch, zu überprüfen, ob ein "GOTO"-Befehl nicht zu einem Sprung aus einem Prozeß führen kann.
Die Marke, auf die gesprungen wird, muß immer innerhalb des Codes des Prozesses sein.

3.4.2.2 Eröffnung des Laufzeitkellers

Ein Prozeß, in dessen Segment (im Sinn von PEARL) Variable vereinbart sind oder Prozeduren aufgerufen werden, besitzt einen Laufzeitkeller LZK und seine Befehlsfolge beginnt mit Aufrufen von Systemfunktionen, in denen Speicherplatz für den LZK reserviert und der erste Aktivierungssatz AS initialisiert wird.

Speicherplatzreservierung:

```
CALL  LZKINI  || VALUE   Seitenzahl
```

Als Parameter wird die Anzahl der für den einzurichtenden LZK benötigten Seiten im Arbeitsspeicher angegeben. Das Programm LZKINI ist im Abschnitt 3.5.4 beschrieben. Es stellt, falls genügend Speicherplatz frei ist, ein Segment der angegebenen Größe zur Verfügung.

Initialisierung

Hinter dem Segmentkontrollsatz des LZK wird der erste Aktivierungssatz AS eingerichtet; dabei werden aus Optimierungsgründen zwei Fälle unterschieden.

a) Vollständiger Aktivierungssatz
   Falls das Segment (im Sinne von PEARL) des Prozesses ein Block mit Vereinbarungen von Variablen ist, folgt dem Aufruf von LZKINI der Befehl

```
CALL  BASINI  || VALUE   Länge des AS
              || VALUE   Länge von Kopf + statischer Teil
              || VALUE   Anzahl der eingeschachtelten
                         Blöcke
```

   Durch diese Funktion wird ein AS der Form

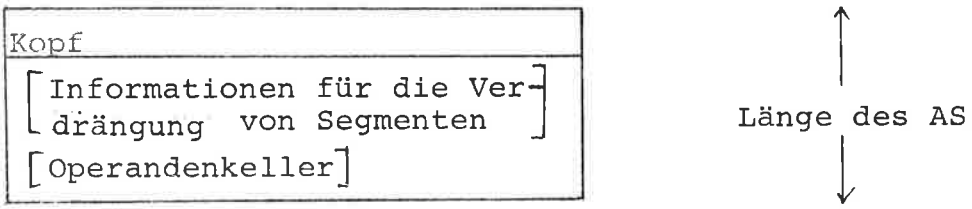
   eingerichtet und entsprechend initialisiert.

b) Verkürzter Aktivierungssatz

Falls das Segment (im Sinne von PEARL) des Prozesses keine Vereinbarungen von Variablen enthält und, wenn es ein Block ist, keine eingeschachtelten Blöcke besitzt, folgt dem Aufruf von LZKINI der Befehl

CALL CASINI || VALUE Länge des AS

Durch diese Funktion wird ein AS der Form



eingerichtet und entsprechend initialisiert. Dieser Aktivierungssatz enthält keinen Platz für lokale Programmdaten oder für die Verwaltung eingeschachtelter Blöcke.

Im Rahmen von BASINI und CASINI wird der Kopf des Aktivierungssatzes folgendermaßen initialisiert:

K = Segmentnummer und Kennung als erster AS des LZK

RUFAS = Adresse des AS der statisch umgebenden Prozedur (evtl. NULL)

SIGK, ADBSK, PAB = NULL

ADBSP = NOPP

AKTBST = 0

ASEND = Adresse der ersten Zelle hinter dem AS

FZ = Adresse der ersten freien Zelle des AS

Aufbau des Umgebungsvektors: Kopie des Umgebungsvektors aus dem AS der umgebenden Prozedur und Eintrag der Adresse des aktuellen AS als letzte Komponente.

$$\text{STUFE} = \begin{cases} X+1 & \text{im Rahmen von BASINI} \\ X & \text{im Rahmen von CASINI} \end{cases}$$

Dabei ist X die in STUFE des AS der umgebenden Prozedur eingetragene Schachtelungsstufe.

Im PKS des Prozesses wird in AS der Hinweis auf den aktuellen Aktivierungssatz eingetragen.

Falls eine Hintergrundverwaltung implementiert ist, folgen dem Aufruf von BASINI bzw. CASINI die Befehle

```
{CALL  DCLSN}
{CALL  DCLSP}
[CALL  ADKBZA]
```

In den Funktionen DCLSN und DCLSP (siehe 3.5.5) wird dem Prozeß das Zugriffsrecht auf weitere Segmente erteilt,und in ADKBZA (siehe 3.6.6.) werden Vorbereitungen durchgeführt, um bei Verlagerung von Segmenten Adreßvariable (Zeiger, Marken) des LZK entsprechend korrigieren zu können.

3.4.2.3 Einrichten weiterer Aktivierungssätze

Bei jedem Prozeduraufruf wird im Laufzeitkeller des aufrufenden Prozesses ein Aktivierungssatz aufgebaut.

3.4.2.3.1 Vorbereitung der Argumente vor Aufruf einer Prozedur

Beim Aufruf werden der Prozedur die Adressen der Argumente übergeben. Diese Adressen müssen vor dem Einsprung in die Prozedur vorbereitet werden. Dabei ist die Vorgehensweise weitgehend vom Rechner und Compiler abhängig. Wir wollen hier eine Methode vorschlagen, die günstig für die Anpassung an die anderen Systemdienste zur Speicherplatzverwaltung ist, die aber unverbindlich ist.
Wichtig ist jedenfalls, wenn die Segmentverschiebung oder unsere Nachschubspeicherverwaltung eingesetzt werden soll, daß die *physikalischen* Adressen der Argumente übergeben werden.

Die Argumente werden in drei Gruppen unterteilt und entsprechend behandelt:

1. Programmdaten, die als GLOBAL vereinbart sind:
   Sie sind im statischen Teil des Arbeitsspeichers mit festen, unveränderlichen Adressen abgelegt.
   Ihre Adressen sind zur Laufzeit bekannt und werden ohne Schwierigkeit im Aktivierungssatz der aufrufenden Prozedur direkt abgelegt.

2. Programmdaten,die weder als GLOBAL noch als Parameter in der aufrufenden Prozedur bekannt sind, d.h. die daselbst oder in einer übergeordneten Prozedur oder Task lokal vereinbart sind: Diese Daten liegen in den verschiedenen Aktivierungssätzen des Laufzeitkellers. Ihre physikalischen Adressen können von einer Bibliotheksfunktion (ARGINI) berechnet werden, die aber nicht Bestandteil des gelieferten Betriebssystems ist.

   Die Parametrierung von ARGINI wäre

CALL ARGINI	Prozedurschachtelungsstufe,in der die Variable vereinbart ist
	relative Ablage im entsprechenden AS
	relative Ablage der physikalischen Adresse im aktuellen AS

   Ist das Argument eine Marke (oder Prozedureingangsstelle), ist dafür im AS der Prozedur, in der die Marke vereinbart ist, eine Markenbeschreibung (MB, siehe 3.4.2.1.3) abgelegt; diese Markenbeschreibung wird als Argument verwendet.
   Steht ein Ausdruck an Argumentposition, ist sein Wert zu berechnen, in einer Hilfszelle (im Operandenkeller) abzulegen und diese Hilfszelle als Argument zu verwenden.

3. Programmdaten, die in der aufrufenden Prozedur als Parameter bekannt sind: Im AS der Prozedur, in der eine Variable als Parameter übernommen wurde, besteht bereits eine Zelle, die die physikalische Adresse des jeweiligen

Argumentes enthält; der Inhalt dieser Zelle ist nun an die neue Argumentablagestelle zu übertragen.
Dies kann mit Hilfe einer Bibliotheksfunktion (PARGINI) geschehen, die ebenfalls nicht Bestandteil des gelieferten Betriebssystems ist.
Die Parametrierung von PARGINI wäre wie die von ARGINI.

Ist das Argument ein VALUE-Parameter, wird die Ablagestelle des übernommenen Argumentwertes wie eine lokale Variable, die nicht Parameter ist, unter Punkt 2 behandelt.

3.4.2.3.2 Prozeduraufruf

Prozeduraufrufe werden auf 3 Arten übersetzt:

1. Normalfall, die aufgerufene Prozedur ist keine Parameterprozedur und der Aufruf steht nicht in einer ON-Anweisung:

   CALL Prozedureingang || Argumentliste

   Der Aufbau des AS der aufgerufenen Prozedur wird allein durch die aufgerufene Prozedur organisiert (siehe 3.4.2.3.3).

2. Aufruf einer Parameterprozedur:

   CALL PARPRO || ADDR Parameterprozedur
   || Argumentliste

   Unter "ADDR Parameterprozedur" ist der relative Abstand zum Anfang des AS der aufrufenden Prozedur zu verstehen, wo die physikalische Adresse der Markenbeschreibung der Argumentprozedur eingetragen wird.

   Aus dem ersten Wort der Markenbeschreibung der Argumentprozedur wird der Hinweis auf den AS entnommen, aus dem der Umgebungsvektor zu kopieren ist,und über das zweite Wort wird in die Prozedur eingetreten; der Aufbau des neuen AS wird durch die aufgerufene Prozedur organisiert, wobei bei dieser Aufrufart in AS.K die Kennung PARPRO eingetragen wird.

3. Aufruf in einer ON-Anweisung:

   CALL RESPRO || { ADDR Prozedureingang
   { - (ADDR Parameterprozedur) }
   || Argumentliste

   Die erste Form der Prozedurangabe wird verwendet, wenn es sich nicht um eine Parameterprozedur handelt; "ADDR Prozedureingang" ist die relative Adresse des Prozedureingangs im aktuellen Code-Segment (Segment im Sinne der Hintergrundverwaltung).
   Die zweite Form der Prozedurangabe wird für Parameterprozeduraufrufe verwendet.

Je nach Art der Prozedurangabe wird entweder der aktuelle AS oder der in der Markenbeschreibung der Argumentprozedur angegebene AS zur Kopie des Umgebungsvektors bereitgestellt und in die Prozedur eingetragen. Der Aufbau des neuen AS wird durch die aufgerufene Prozedur organisiert, wobei bei dieser Aufrufart in AS.K die Kennung RSPMSK eingetragen wird.

Die"Argumentliste" enthält

1. ADDR der Rückkehrmarke
2. die Liste der ADDR der eigentlichen (im Quellprogramm angegebenen) Argumente.

Für die Rückkehr aus der aufgerufenen Prozedur hinter den Aufruf ist vom Compiler hinter jedem Prozeduraufruf eine Marke zu generieren, welche, durch ARGINI (siehe 3.4.2.3.1) vorbehandelt, als erstes Argument an die aufgerufene Prozedur übergeben wird.

Jede Argument-ADDR (inkl. Rückkehrmarke) ist der relative Abstand zum AS der aufrufenden Prozedur, wo die physikalische Adresse des Argumentes eingetragen ist.

Wir nehmen hier an, daß die Argumente verstreut im Aktivierungssatz liegen; wenn sie aber in einem zusammenhängenden Bereich abgelegt sind, soll beim Aufruf als einziges Argument der Hinweis auf diesen Speicherbereich übergeben werden.

3.4.2.3.3 Eintritt in eine Prozedur

Beim Eintritt in eine Prozedur wird erst Speicherplatz im Laufzeitkeller des Prozesses für den neu aufzubauenden Aktivierungssatz belegt. Der Kopf des Aktivierungssatzes wird initialisiert. Danach erfolgt die Parameterübertragung und evtl. die Initialisierung der Programmdaten, die in dem statischen Teil des Aktivierungssatzes abgelegt werden müssen.

Übersetzung von "DCL P PROC = (P1,...,Pn) BEGIN...":

```
    GOTO  Adresse hinter dem Prozedurende
P:  CALL  ASINI   ||VALUE   Gesamtlänge des AS
                  ||VALUE   Schachtelungsstufe
                  ||VALUE   Länge von (Kopf + statischer Teil)
                  ||VALUE   Anzahl der eingeschachtelten Blöcke

    CALL  PARUEB  ||VALUE   n+1
                  ||ADDR    Ablagestelle

    [CALL ADKBZA]
          ↑
    Befehlsfolge zur Initialisierung der im
    statischen Teil abgelegten Programmdaten
          |
          ↕
    Normale Übersetzung des Prozedurkörpers
          |
          ↓
          :
```

Falls die Prozedur innerhalb einer ON-Anweisung aufgerufen wurde, ist in AS.K die Kennung RSPMSK bereits eingetragen (siehe 3.4.2.3.2 Punkt 3.).

Durch ASINI wird der Kopf des AS nun wie folgt initialisiert:

Anfang des neuen AS = ASEND aus dem AS der aufrufenden Prozedur

AS.ASEND = Anfang des neuen AS +"Gesamtlänge des AS" (erster Parameter von ASINI)

AS.ADBSK, AS.PAB = NULL

AS.ADBSP = NOPP

AS.AKTBST = 0

Der Bereich AS.BLZ wird in der Länge "Anzahl der eingeschachtelten Blöcke" reserviert (4. Parameter von ASINI).

AS.STUFE = "Schachtelungsstufe"(2. Parameter von ASINI)

Der Bereich AS.UV wird in der Länge AS.STUFE reserviert und initialisiert. Die ersten AS.STUFE-1 Zellen werden aus dem UV der aufrufenden bzw. umgebenden Prozedur (siehe 3.4.2.3.2) kopiert, in die letzte Zelle wird der Zeiger auf den neuen AS selbst eingetragen.

AS.ZUV = Zeiger auf den Anfang des AS.UV

AS.FZ = Anfang des AS + "Länge Kopf + statischer Teil" (3. Parameter von ASINI)

Durch die Bibliotheksfunktion PARUEB, die nicht Bestandteil des gelieferten BS ist, werden die Adressen der Argumente aus dem AS der aufrufenden Prozedur in den neuen AS - in die Parameter - übertragen.

Danach kann für die Hintergrundverwaltung das Unterprogramm ADKBZ.A (siehe 3.6.6) aufgerufen werden.

3.4.2.4 Blockeröffnung

Zur Eröffnung eines BEGIN-Blocks ist vom Compiler folgendes abzulegen:

```
CALL BLKINI
[CALL PKSINI   || Liste von: VALUE  Art der Task]
[CALL SEMINI   || VALUE   Anzahl der SEMA-Variablen]
[CALL BQLINI   || Liste von: VALUE  Maximalwert der
               ||                   Boltvariablen   ]
[CALL ALFKS      ...
[CALL ASN      || VALUE   Anzahl Seiten
               || VALUE   Typ des Segments]   ...
Übersetzte Anweisungsfolge des Blocks
```

Durch BLKINI wird folgendes ausgeführt:

AS.AKTBST wird um 1 erhöht
AS.PAB, AS.BLZ(AS.AKTBST) = AS.FZ (bisheriges Ende)
AS.FZ = AS.FZ + Länge der Struktur BD
Ab der von AS.PAB angezeigten Stelle wird die Struktur BD angelegt, BD.Z wird mit 1 initialisiert und alle anderen Komponenten werden 0 gesetzt.

PKSINI hat als Parameter eine Liste von Taskarten.
Für jede lokale Taskvariable ist die Taskart wie in 1.7.2 beschrieben anzugeben.
Für jedes Element der Parameterliste wird folgendes ausgeführt:

- es wird Speicherplatz für einen PKS (TASK-Art = 1 oder 3) oder KPKS (TASK-Art = 2 oder 4) im statischen Raum des Arbeitsspeichers reserviert (siehe 3.2)
- wenn TASK-Art = 3 oder 4, d.h. es handelt sich um eine TASK-Konstante (Attribut TASK), wird in die AS-Zelle des PKS (bzw. KPKS) der Zeiger auf den Aktivierungssatz des laufenden Prozesses und in die V-Zelle des PKS (bzw. KPKS) der Zeiger auf den PKS des laufenden Prozesses eingetragen;

. der SZ-Zelle im PKS wird ein Initialwert zugewiesen. Die SK-Zelle wird mit dem Zeiger NULL initialisiert;

. die erste freie Zelle im Aktivierungssatz des laufenden Prozesses wird mit dem Zeiger auf den gerade reservierten PKS (bzw. KPKS) initialisiert. Der Zeiger FZ im Kopf des AS wird um eins weitergeschaltet. Die Anzahl der im Block vereinbarten Prozesse (Zelle BD.PA) wird um eins erhöht.

Falls keine TASK-Variablen im Block lokal sind, entfällt der Aufruf von PKSINI.

Durch SEMINI wird folgendes ausgeführt:

. die Anzahl der im Block vereinbarten SEMA-Variablen (Zelle BD,SA) wird um die hier angegebene Anzahl erhöht;

. die angegebene Anzahl von SEMA-Kontrollsätzen wird im statischen Raum des Arbeitsspeichers reserviert (siehe 4.3.2); die SK-Zelle jedes SEMA-Kontrollsatzes wird mit dem Zeiger auf die SEMEND-Struktur initialisiert; der Zelle S jedes SEMA-Kontrollsatzes wird der Wert 0 zugewiesen. Zeiger auf diese reservierten SEMA-Kontrollsätze werden im blockspezifischen Teil des Aktivierungssatzes abgelegt.

. Der Zeiger FZ im Kopf des AS wird um die Anzahl der SEMA-Variablen weitergeschaltet.

Falls keine SEMA-Variablen im Block lokal sind, entfällt der Aufruf von SEMINI.

BOLINI hat als Parameter eine Liste von Maximalwerten für BOLT-Variable; diese Maximalwerte sind die bei den BOLT-Vereinbarungen angegebenen Zahlen, bzw.,wenn nichts angegeben ist, ist vom Compiler eine möglichst große Zahl einzusetzen.

Für jedes Element der Parameterliste wird folgendes ausgeführt:

. Es wird Speicherplatz für einen BOLT-Kontrollsatz im statischen Raum des Arbeitsspeichers reserviert (siehe 3.2).

. In diesem BOLT-Kontrollsatz wird

  . die BK-Zelle mit dem Zeiger auf das SEMEND-Element initialisiert,
  . die SZ-Zelle mit O initialisiert,
  . die SUP-Zelle mit dem angegebenen Maximalwert initialisiert.

. Die erste freie Zelle im Aktivierungssatz des laufenden Prozesses wird mit dem Zeiger auf den gerade reservierten BOLT-Kontrollsatz initialisiert.

. Der Zeiger FS im Kopf des AS wird um eins weitergeschaltet. Die Anzahl der im Block vereinbarten BOLT-Variablen (Zelle BD.BA) wird um eins erhöht.

Falls keine BOLT-Variablen im Block lokal sind, entfällt der Aufruf von BOLINI.

Ein Aufruf von ALFKS ist für jede im Block lokale FILE-Variable abzusetzen.

Durch ALFKS wird folgendes ausgeführt:

. Ein FKS wird im statischen Raum des Arbeitsspeichers reserviert (siehe 4.3.2).

. In diesem FKS wird
  . die DS-Zelle mit dem Zeiger NULL initialisiert,
  . die WP-Zelle mit dem Zeiger auf das SEMEND-Element initialisiert,
  . die AP-Zelle mit dem Zeiger auf das EREND-Element initialisiert,
  . die Weichen BINEK,BINAK,ALPEK,ALPAK werden auf Fehler gesetzt.

. Die erste freie Zelle im Aktivierungssatz des laufenden Prozesses wird mit dem Zeiger auf den gerade reservierten FKS initialisiert. Der Zeiger FZ im Kopf

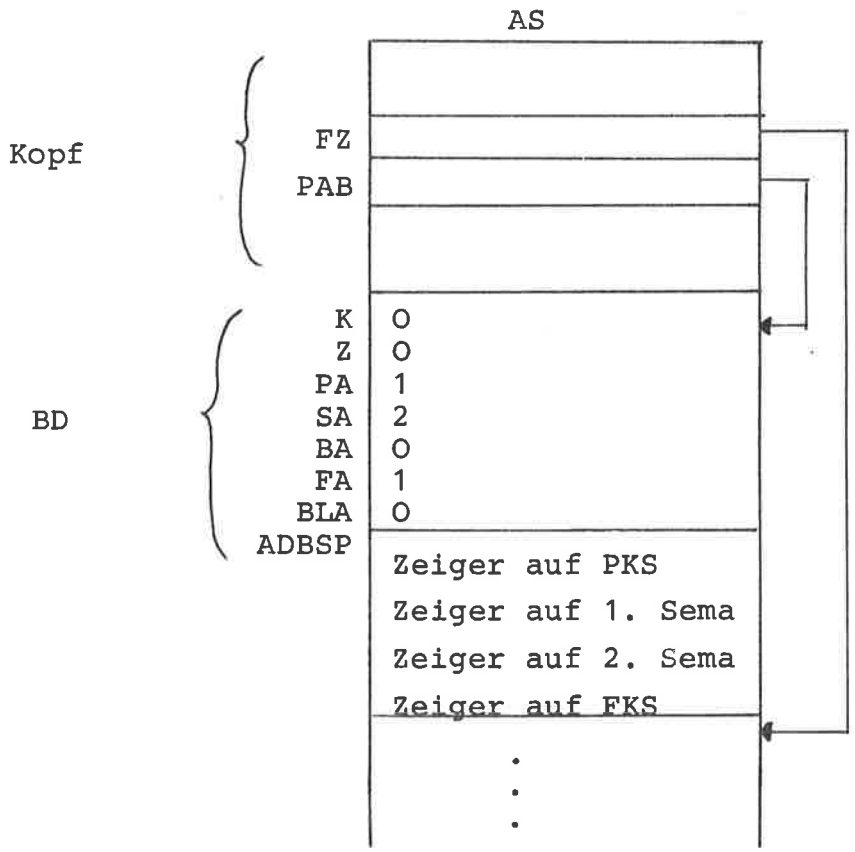
des AS wird um eins weitergeschaltet. Die Anzahl der im Block vereinbarten FILE-Variablen (Zelle BD.FA) wird um eins erhöht.

Ein Aufruf von ASN ist für jeden im Block lokalen Bereich abzusetzen. Wenn lokale Bereiche im Block vereinbart sind, wird evtl. vom Compiler ein Aufruf von ADKBZB (siehe 3.6.6) abgelegt.

Beispiel: Nehmen wir an, daß im Block eine TASK-Variable der Art 1, zwei SEMA-Variable, keine BOLT-Variable und eine FILE-Variable vereinbart sind. Beim Eintritt in den Block wird folgende Befehlsfolge abgelegt:

```
CALL BLKINI
CALL PKSINI  || VALUE   1  (in Form einer Liste)
CALL SEMINI  || VALUE   2
CALL ALFKS
```

Nach dem Überlaufen dieser Befehlsfolge wird der blockspezifische Raum im Aktivierungssatz des laufenden Prozesses so aussehen:



Die relativen Adressen im Aktivierungssatz der Zeiger auf die PKS, SEMA, BOLT und FKS sind also dem Compiler wohlbekannt. Sie werden bei der Übersetzung der Steuer- und Synchronisieranweisungen benutzt.

Hinter dieser Aufruf-Folge wird evtl. "CALL ADKBZB" abgelegt. ADKBZB dient der Verwaltung von Adressen in Adreßvariablen-Bereichen, falls die Segmentverschiebung und/oder die Verdrängung von Segmenten auf den Hintergrund implementiert ist. Es wird im Abschnitt 4.3.6 beschrieben.

Dann werden die im Block vereinbarten Bereiche betrachtet. Ihnen wird getrennt vom Aktivierungssatz Speicherplatz zugewiesen. Bereichen mit dem Attribut RESIDENT und Bereichen ohne RESIDENT-Attribut werden verschiedene Segmente zugewiesen. Der Compiler sorgt für die Einstellung der evtl. notwendigen Adreß-Abbildungsfunktionen dieser Bereiche. Der aufgerufene Systemdienst vergibt nur Speicherplatz für die Bereichssegmente.
Dieses geschieht durch den Aufruf:

```
CALL ASN  ||VALUE  Anzahl der Seiten
          ||VALUE  Typ des Segments
```

ASN wird im Abschnitt 5 beschrieben.
Die Zeiger auf die verschiedenen Bereiche werden an der durch AS.FZ angezeigten Stelle abgelegt. AS.FZ muß jedesmal entsprechend weitergeschaltet werden.

Nachdem alle Bereiche eingerichtet worden sind, wird die normale Folge der Anweisungen im Block ausgeführt. Wenn eine ON-Anweisung überlaufen wird, wird ein (oder mehrere) ON-Beschreibungsblock RSPB an der durch AS.FZ angezeigten Stelle abgelegt. Dieses ist in 1.5.2 erklärt.
Der für einen Block reservierte Abschnitt im Aktivierungssatz einer Prozedur sieht also im allgemeinen so aus:

BD
Zeiger auf Prozeßkontrollsätze PKS
Zeiger auf Semakontrollsätze SEMA
Zeiger auf Boltkontrollsätze BOLT
Zeiger auf Filekontrollsätze FKS
Informationen für die Verwaltung von Bereichen
ON-Beschreibungsblöcke RSPB

3.4.2.5 Verlassen eines Blockes

Der Objektcode eines Blockes, in dem Task-, Sema-, Bolt- oder File-Variable vereinbart sind, enthält

CALL SBLEND

als letzten Befehl. Der Objektcode anderer Blöcke wird durch

CALL BLEND

abgeschlossen.

Eine Goto-Anweisung mit einem Sprungziel außerhalb des Blocks und innerhalb der Prozedur wird durch

CALL BLAUS || ADDR Markenbeschreibung

übersetzt. Der Aussprung aus einer Prozedur wird im nächsten Abschnitt beschrieben.

Funktion: SBLEND

Falls ein Unterprozeß, der auf diesen Block zugreift, aktiv ist oder seine Aktivierung eingeplant ist, wird der laufende Prozeß zurückgestellt.
Anderenfalls werden alle vom Block angezeigten Prozeß-, Sema-, Bolt- und File-Kontrollsätze freigegeben, und danach wird wie in BLEND fortgefahren.

Funktion: BLEND

Der außerhalb des Laufzeitkellers für Bereiche, die in diesem Block vereinbart sind, belegte Speicherplatz wird freigegeben.
Die Verwaltungsgrößen im Kopf des aktuellen Aktivierungssatzes werden auf den Stand gebracht, den sie vor der Eröffnung in dem Block hatten:
Der Zeiger FZ weist auf den Anfang der Blockbeschreibung BD des aktuellen Blocks.
Die aktuelle Blockstufe AKTBST wird um eins erniedrigt.
Der Zeiger PAB weist auf die Blockbeschreibung BD des umschließenden Blocks.

Funktion BLAUS:

Aus der als Parameter übergebenen Markenbeschreibung wird die Schachtelungsstufe des Blocks entnommen, in den gesprungen wird.
Beginnend mit dem aktuellen Block wird der Aktivierungssatz schrittweise bis zu dem Block abgebaut, in dem das Sprungziel liegt. Dabei wird für jeden freizugebenden Block entweder SBLEND oder BLEND ausgeführt.
Nach dem letzten Schritt wird mit dem in der Markenbeschreibung als Sprungziel spezifizierten Befehl fortgefahren.

3.4.2.6 Verlassen einer Prozedür

Die interne Darstellung einer Return-Anweisung im äußersten Block einer Prozedur ist

CALL PEND	||ADDR	Parameterposition der Rückkehradresse

Die Return-Anweisung in eingeschachtelten Blöcken wird durch

CALL PRET	||ADDR	Parameterposition der Rückkehradresse

dargestellt.

Eine Goto-Anweisung mit einem Sprungziel außerhalb der Prozedur wird mit

CALL PRAUS	|| ADDR	Hinweis auf Markenbeschreibung

übersetzt. Dabei liegt die Markenbeschreibung im Aktivierungssatz der Prozedur, in der die Marke vereinbart ist, und im aktuellen Aktivierungssatz ist ein Hinweis auf diese Markenbeschreibung abgelegt.

Funktion: PEND

In RUFAS des aktuellen Aktivierungssatzes ist der Hinweis auf den Aktivierungssatz der aufrufenden Prozedur abgelegt. Er wird in AS des aktuellen PKS übernommen.
Danach wird mit dem Befehl fortgefahren, dessen Marke in der Rückkehradresse spezifiziert ist.

Funktion: PRET

Beginnend mit dem aktuellen Block werden alle Blöcke des Aktivierungssatzes verlassen, indem entweder wie in SBLEND oder wie in BLEND verfahren wird.

Nach dem letzten Schritt wird wie in PEND fortgefahren.

Funktion: PRAUS

Aus der als Parameter übergebenen Markenbeschreibung wird der Hinweis auf den Aktivierungssatz der Prozedur, in der das Sprungziel liegt, entnommen.
Falls dieser Aktivierungssatz im Laufzeitkeller eines anderen Prozesses liegt, wird der laufende Prozeß mit seinen Unterprozessen beendet, der Laufzeitkeller wird freigegeben und eine Fehlermeldung wird ausgegeben.
Wenn das Sprungziel im Code des Prozesses liegt, wird der Laufzeitkeller bis zu dem Ziel-AS schrittweise abgebaut. Dabei wird für jeden eingeschachtelten Block entweder SBLEND oder BLEND ausgeführt.
Im Ziel-AS werden so viele Blöcke verlassen, bis die Blockstufe erreicht ist, die in der als Parameter übergebenen Markenbeschreibung spezifiziert ist.
Danach wird auf das in der Markenbeschreibung angegebene Ziel gesprungen.

3.5 Prozeßverdrängung und Nachschubspeicherverwaltung

3.5.1 Einleitung

Hier wird die implementierte Strategie zur Verwaltung des Speicherplatzes von Rechnern mit Hintergrundspeicher beschrieben.
Es wird keine Annahme über die physikalische Struktur des Hintergrundspeichers gemacht. Er kann sowohl eine Platte oder Trommel als auch ein Magnetbandgerät sein. Die Verwaltung der Schreib- oder Leseaufträge auf den Hintergrund und der Geräte, die den physikalischen Hintergrundspeicher tragen, ist eine Aufgabe der Datei-Organisation, die von der Struktur des Hintergrundspeichers abhängig ist.

Arbeits- und Hintergrundspeicher sind in Seiten eingeteilt. Segmente bestehen aus aufeinanderfolgenden Seiten (siehe 3.4). Es gibt Segmente von Prozedurcode, Laufzeitkellern und Datenmengen (Bereiche oder indizierte Strukturen). Segmente können RESIDENT sein oder nicht.

Segmente können nur ein- oder ausgelagert werden; sie werden nicht im Arbeitsspeicher verschoben. Daher ist es nicht nötig, RESIDENTE und nicht-RESIDENTE Segmente in zwei getrennten Gebieten des Arbeitsspeichers zu halten. Wenn aber Verschiebungsmöglichkeiten von Segmenten zugelassen werden sollten, muß darauf geachtet werden, daß RESIDENT-Segmente unbedingt von den anderen zu trennen sind, was zusätzliche Verwaltung erfordert.
Außer den vom Programmierer explizit als RESIDENT spezifizierten Segmenten sind auch alle Segmente von Laufzeitkellern und Adreßmengen implizit als RESIDENT zu betrachten.

3.5.2 Datensätze zur Verwaltung des Arbeits- und Nachschubspeichers

Im Kopf eines Segments wird ein *Segmentkontrollsatz* SKS abgelegt, in dem folgende Einträge benutzt werden:

SKS

K	Typ: Befehlscode/Datenmenge/RESIDENT
N	Nachfolger in der SKS-Kette
P	Prioritätszahl
V	Vorgänger in der SKS-Kette
Z	Zähler
KBLI	Index im KBL (d.h. Seitennummer)
HINTA	Adresse auf dem Hintergrund
SA	Anzahl der Seiten
SGNR	Segmentnummer
ADKSK	Zeiger auf die ADKS-Kette

An den SKS eines nicht-RESIDENTEN Segments werden *Adreßkontrollsätze* ADKS, die spezifisch für jeden Prozeß sind, der das Segment anspricht (siehe 3.6.1), angefügt.

ADKS

ZSKS	Zeiger auf den SKS
NSADS	Nachfolger in der segmentspezifischen Kette
VSADS	Vorgänger in der segmentspezifischen Kette
ZPKS	Zeiger auf den PKS
NPADS	Nachfolger in der prozeßspezifischen Kette
ALTA	alte Anfangsadresse des Segments
VZADB	Zeiger auf die ADBS-Kette

Wir haben uns auf vier Segmenttypen beschränkt: Befehlscode, Datenbereiche, die nach ihrer ersten Initialisierung unmodifiziert bleiben werden, normale Datenbereiche und RESIDENT-Segmente. Der Segmenttyp und verschiedene Kennungen, die im Lauf des Programms benutzt werden, werden in SKS.K abgelegt.

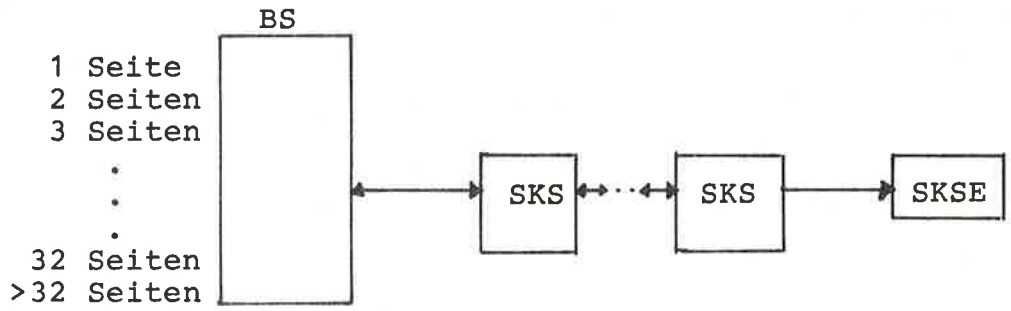
Da der SKS mit dem Segment in den Hintergrund ausgelagert werden kann, wird dem Segment - wenn ausgelagert - ein im statischen Teil des Arbeitsspeichers bleibender Kontrollsatz (HSKS) zugeordnet, in dem folgende Einträge abgelegt werden:

HSKS

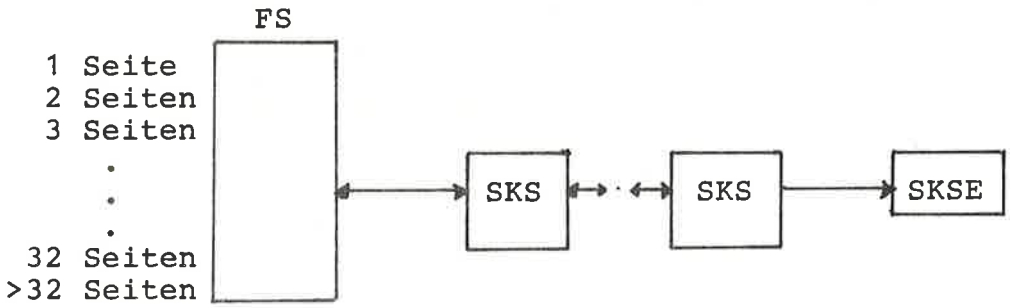
KBLI	Früherer Index im KBL
HINTA	Adresse auf dem Hintergrund
SA	Anzahl der Seiten

SKS werden in verschiedenen Ketten miteinander verkettet, wenn das Segment sich im Kernspeicher befindet. Die Anfangszeiger dieser Ketten sind folgende:

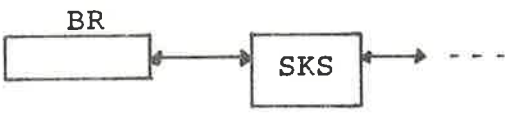
Belegte Segmente (nicht RESIDENT) im Arbeitsspeicher mit Länge =



Freigewordene Segmente im Arbeitsspeicher mit Länge =



RESIDENT belegte Segmente (Länge bedeutungslos):

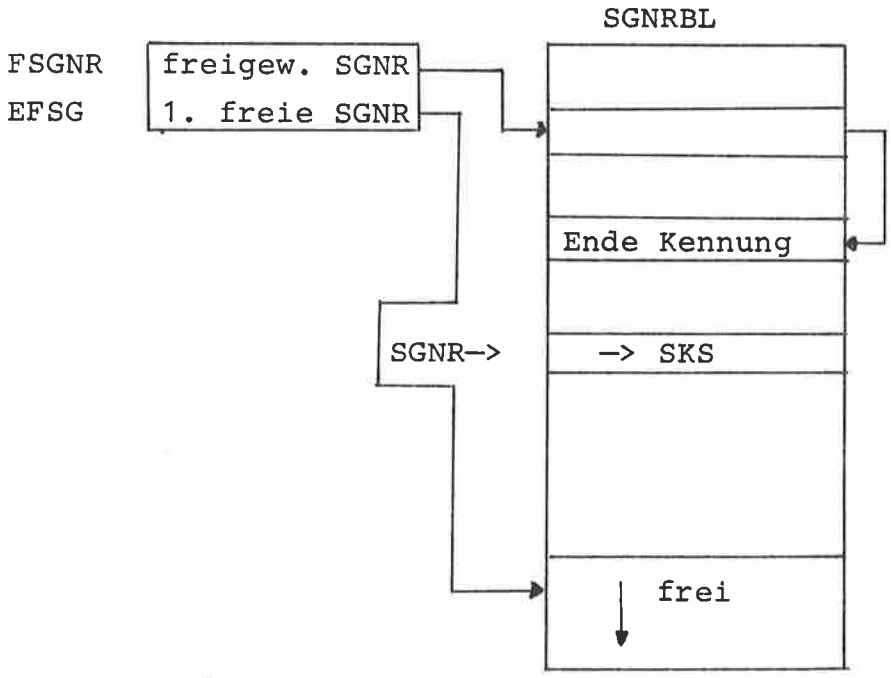


Alle Ketten von SKS sind mit einem besonderen Endeelement SKSE beendet.

SKS von belegten, nicht-RESIDENTEN Segmenten bzw. freigewordenen Segmenten, die dieselbe Anzahl N von Seiten besitzen, sind miteinander an der N. Zelle des BS- bzw. FS-Bereichs verkettet. SKS von RESIDENT belegten Segmenten sind extra an BR gekettet.

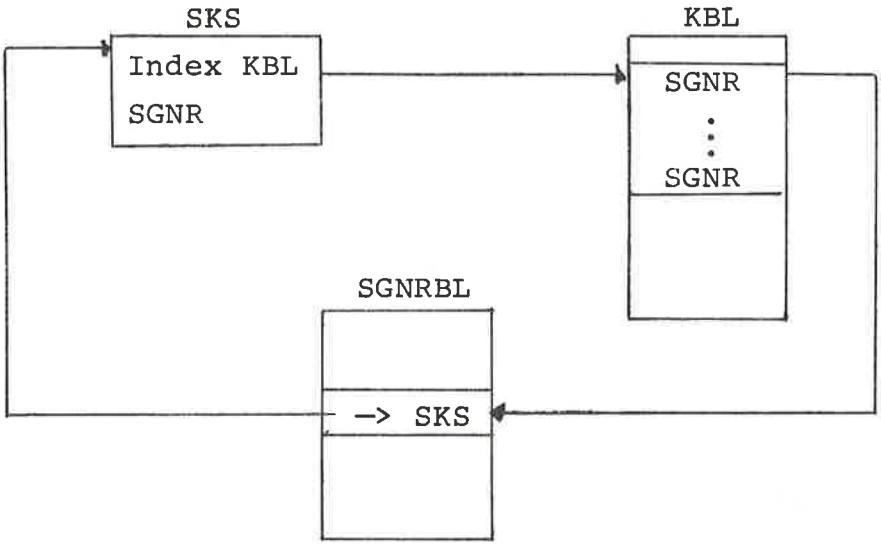
Zur Beschreibung des dynamischen Teils des Kernspeichers wird eine *Kernspeicherbelegungsliste* KBL eingerichtet, in der jedes Wort einer *Seite* im dynamischen Teil des Kernspeichers assoziiert ist. Dieses Wort enthält die Segmentnummer des Segments, das die entsprechende Seite belegt. Der SKS dieses Segments enthält den Index für KBL, der dem Segmentanfang im KBL entspricht (Zelle SKS.KBLI).

Die den Segmenten zugeordneten Segmentnummern SGNR werden dynamisch mit Hilfe einer *Segmentnummerbelegungsliste* SGNRBL vergeben. SGNR dient als Index im SGNRBL. Im SGNRBL-Bereich ist in jeder durch SGNR angegebenen Position ein Zeiger auf den segmentzugehörigen SKS abgelegt. Alle durch freigewordene SGNR angegebenen Positionen im SGNRBL-Bereich sind miteinander verkettet.



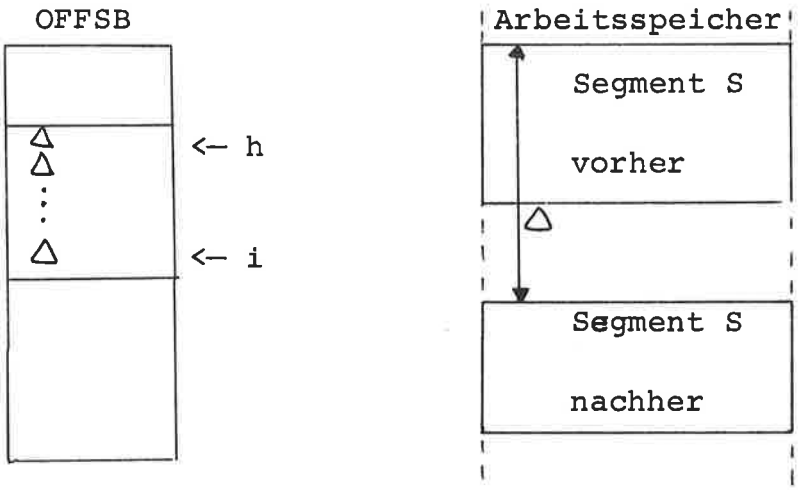
Bei Anforderung einer neuen Segmentnummer wird, falls vorhanden, eine freigewordene SGNR oder, falls die Kette der freigewordenen SGNR leer ist, die 1. freie SGNR des SGNRBL-Bereiches genommen.

Die SKS, KBL-Elemente und SGNRBL-Elemente sind folgendermaßen im Kreis verzeigert:



Wenn ein ausgelagertes Segment wieder in den Arbeitsspeicher gebracht worden ist, werden alle Referenzen auf das Segment korrigiert. Dazu dient ein Offset-Bereich OFFSB, in dem die Differenz zwischen alter Anfangsadresse und neuer Anfangsadresse des Segments eingetragen wird und zwar an den Stellen, die durch die alten Seitennummern des Segments definiert sind (also parallel zum alten KBL-Stand).

Wenn das Segment S z.B. die Seitennummern h bis i im Arbeitsspeicher vorher besessen hat und jetzt die Seitennummern j bis k, wird OFFSB so aussehen:



△ ergibt sich aus j-k.

3.5.3 Kriterien zur Segmentverdrängung

Segmente mit folgenden Eigenschaften können verdrängt werden:

1. Segmente, auf die nur Prozesse zugreifen, die auf explizite Synchronisieranweisungen (RELEASE, FREE, LEAVE, CONTINUE) oder auf Verstreichen einer langen Dauer (RESUME) warten.

2. Segmente, auf die nur Prozesse zugreifen, die unwichtiger als der Prozeß sind, der durch Speicherplatzanforderung die Segmentverdrängung notwendig macht, und die keinen Datei-Übertragungsbefehl ausführen.

Zum Erkennen eines Segments der ersten Gruppe wird der Zähler im SKS des Segments benutzt. Dieser Zähler wird initial gelöscht, bei Zuteilung des Zugriffsrechts zum Segment um eins erhöht und bei Entzug des Zugriffsrechts um eins vermindert. Die Zuteilung des Zugriffsrechts erfolgt aufgrund von Anweisungen im Programm beim Aktivieren von Prozessen (Unterprogrammaufrufe von DCLSN oder DCLSP, siehe 3.5.5). Das Zugriffsrecht wird entsprechend beim Beenden von Prozessen entzogen. Zusätzlich kann Prozessen, die längere Zeit laufunfähig sind, während dieser Zeit der Zugriff entzogen werden.
Der Zählerstand O kennzeichnet den Fall, daß das Segment zur ersten Gruppe der verdrängbaren Segmente gehört.

Zum Erkennen eines Segments der zweiten Gruppe wird die Prioritätszahl im SKS des Segments benutzt; sie wird dynamisch der höchsten absoluten Prioritätszahl der zugreifenden Prozesse gleichgesetzt. Das Betriebssystem sorgt für die Vergabe solcher absoluter Prioritätszahlen, die die Reihenfolge der Prozesse in der Prioritätskette spiegeln (siehe 1.7.4).

Wenn Speicherplatz für ein Segment von n Seiten angefordert ist und im Arbeitsspeicher nicht genügend freie Seiten zur Verfügung stehen, wird gesucht, ob ein verdrängbares Segment von N Seiten, wobei $N \geq n$, vorhanden ist.

Verdrängbare Segmente der ersten Gruppe haben den Vorzug vor verdrängbaren Segmenten der zweiten Gruppe. Innerhalb einer Gruppe haben die Codesegmente den Vorzug vor Datensegmenten und unmodifizierbare Datensegmente vor normalen Datensegmenten.

Prozedursegmente werden nicht auf den Hintergrund geschrieben, weil sie invariant sind und permanent im Hintergrund bleiben. Unmodifizierbare Datensegmente werden nur das erste Mal auf den Hintergrund geschrieben und bleiben dann invariant im Hintergrund. Der Compiler muß solche Datenbereiche erkennen und besonders kennzeichnen, sonst werden sie wie normale Bereiche evtl. mehrmals auf den Hintergrund ausgelagert.
Wenn ein ausgelagertes Segment wieder in den Arbeitsspeicher eingelagert werden soll, wird erst, ohne Rücksicht auf evtl. bestehende Vorzüge von verdrängbaren Segmenten untereinander, geprüft, ob das Segment an der alten Stelle im Arbeitsspeicher eingelagert werden darf. In diesem Fall müssen die Referenzen auf dieses Segment nicht korrigiert zu werden.

3.5.4 Speicherplatzanforderung

Wenn ein Prozeß Speicherplatz anfordert, wird erst geprüft, ob gerade eine andere Speicherplatzanforderung bearbeitet wird. Ist dies der Fall, wird die wichtigste Anforderung aufgenommen und die unwichtigere zurückgestellt. Ein Zeiger weist auf den PKS des wichtigsten anfordernden Prozesses. Wenn keine andere Speicherplatzanforderung vorliegt, werden die SKS-Zähler auf den letzten Stand gebracht, um die verdrängbaren Segmente der ersten Gruppe erkennen zu können.

Wenn ein Prozeß in langen Wartezustand gesetzt worden war, wurde in seinem PKS notiert, daß, falls Speicherplatz angefordert wird, die SKS-Zähler der Segmente, auf die er zugreift, um eins erniedrigt werden müssen. Dafür ist im PKS die SPCA-Zelle reserviert, in die der Befehl "CALL NW" eingetragen wird, falls der Prozeß in langen Wartezustand gesetzt wird oder in die der Befehl "GOTO *PKS.PV" eingetragen wird, falls der Prozeß aus seinem Wartezustand befreit wird oder die SKS-Zähler bereits erniedrigt worden sind. Die Zelle SPCA im PKS kann auch den Befehl "CALL FL" enthalten, wenn gerade eine Datei-Übertragungsanweisung bearbeitet wird.

Um die SKS-Zähler auf den letzten Stand zu bringen, wird die Prioritätskette rückwärts durchsprungen: Wenn "CALL NW" in SPCA des PKS liegt, wird der Systemdienst NW aufgerufen; NW erniedrigt die SKS-Zähler um eins und trägt "GOTO *PKS.PV" in die SPCA-Zelle ein. Wenn "GOTO *PKS.PV" in SPCA liegt, wird einfach der PKS übersprungen. Wenn "CALL FL" in SPCA liegt, wird der Zugriff auf "source" oder "sink" gesichert, indem alle Segmente, auf die der Prozeß zugreift, als unverdrängbar gekennzeichnet werden.

3.5.4.1 Anforderung eines neuen Segments im Arbeitsspeicher

Wenn das Segment für einen Laufzeitkeller bestimmt ist, wird die Anforderung vom Unterprogramm LZKINI bearbeitet:

```
CALL LZKINI  || VALUE   Anzahl der Seiten
```

Wenn das Segment für sonstige Daten bestimmt ist, wird die Anforderung vom Unterprogramm ASN bearbeitet:

```
CALL ASN     || VALUE   Anzahl der Seiten
             || VALUE   Typ des Segments
```

Der zweite Parameter ist als Zahl anzugeben und betrifft:

```
Typ = 1 | Befehlscode
      2 | unmodifizierbarer Datenbereich
      3 | normaler Datenbereich
      4 | RESIDENT-Segment
```

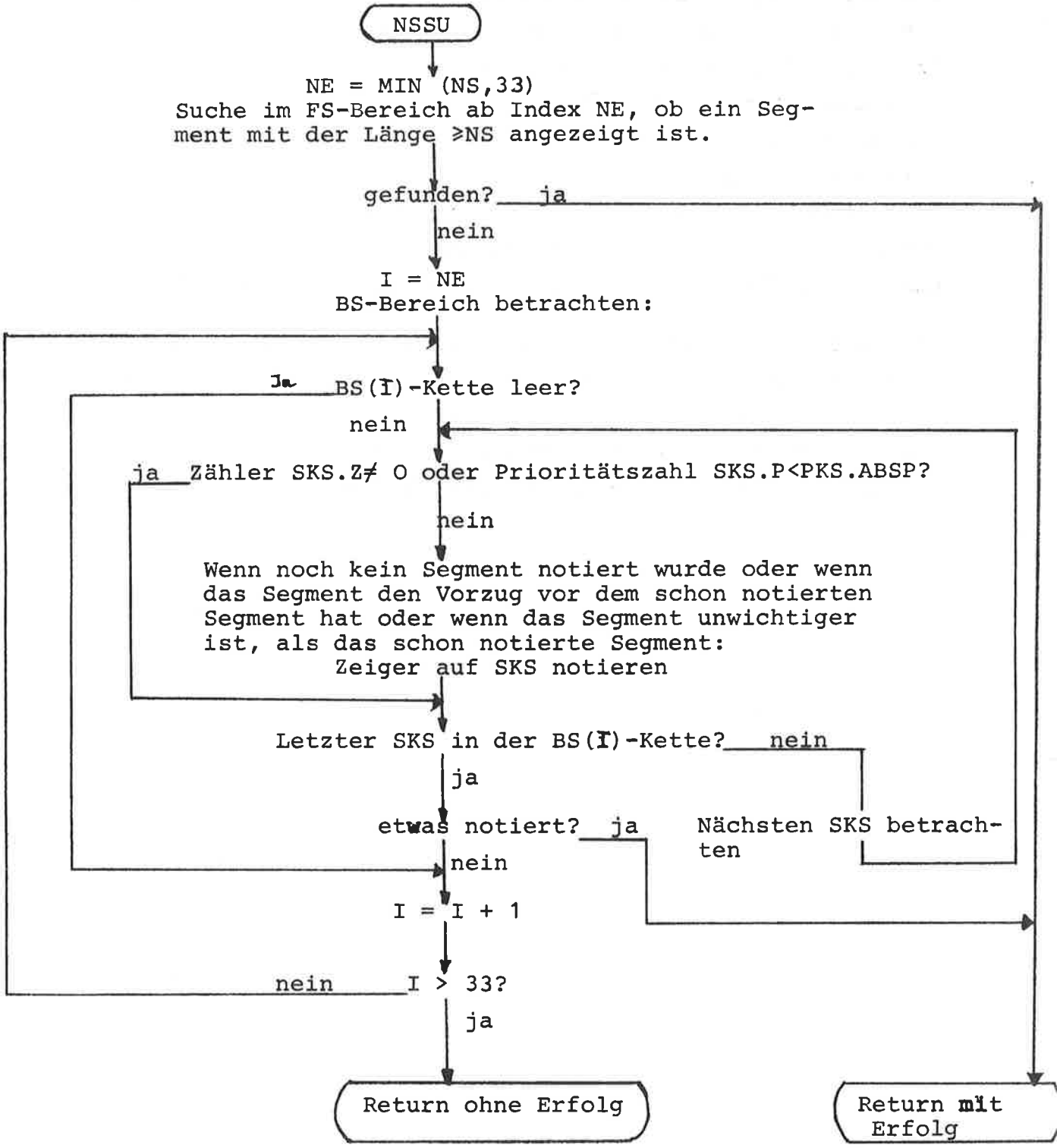
LZKINI und ASN tun folgendes:

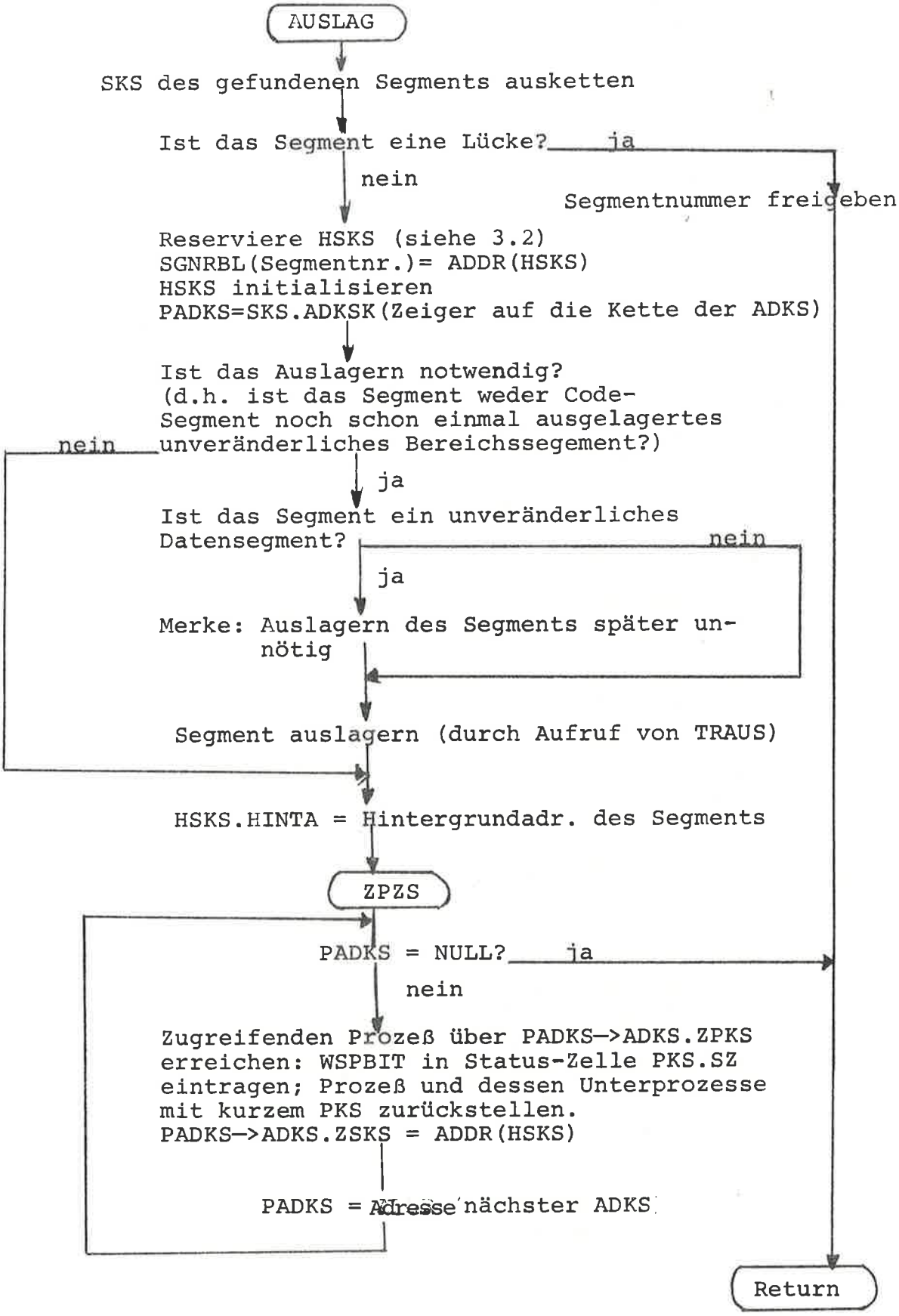
- Beim Eingang von LZKINI wird der Segment-Typ gleich 4 (RESIDENT) gesetzt.
  Beim Eingang ASN wird der Segment-Typ als Parameter geliefert. Die Anzahl der angeforderten Seiten wird NS zugewiesen.

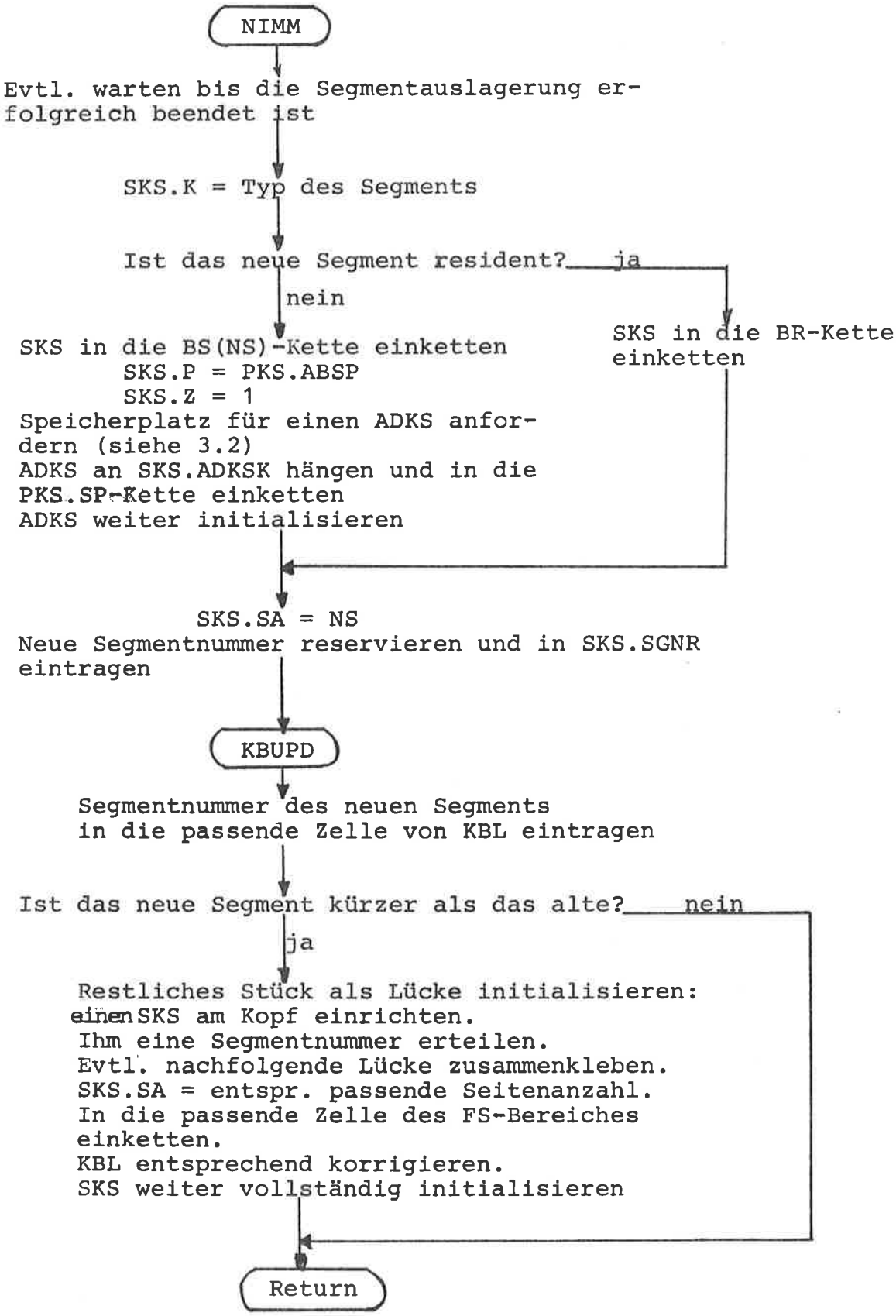
- In SWAF wird getestet, ob eine andere Anforderung unterwegs ist. Falls ja und falls der Prozeß, der die aktuelle Anforderung macht, unwichtiger als der Prozeß ist, der die schon laufende Anforderung gemacht hat, wird der aktuelle Prozeß zurückgestellt. Anderenfalls wird die Prioritätskette rückwärts durchsprungen, um die SKS-Zähler auf den aktuellen Zustand zu bringen (siehe 3.5.4) und der Prozeß fortgesetzt.
  Ein als unwichtiger zurückgestellter Prozeß wird über die SN- und SV-Zellen im PKS in eine durch PANFOR angezeigte Kette eingeordnet. Wenn die laufende Anforderung erfüllt ist, wird er aus dem Wartezustand befreit und zur Wiederholung seiner Anforderung fortgesetzt.

. Ein freies oder verdrängbares Segment mit der Länge ≥ NS wird durch Aufruf von NSSU (siehe Flußdiagramm) gesucht. Falls der Versuch erfolglos beendet ist, wird der Prozeß im Status "Warte auf Speicherplatzfreigabe" zurückgestellt und über die im PKS liegenden SN- und SV-Zeiger an ZURANF gekettet. Falls der Versuch erfolgreich beendet ist, wird das gefundene Segment, wenn nötig, ausgelagert und die darauf zugreifenden Prozesse zurückgestellt. Dies wird durch Aufruf von AUSLAG (siehe Flußdiagramm) ausgeführt.

. NS Seiten werden im Arbeitsspeicher an der freigewordenen Stelle durch Aufruf von NIMM (siehe Flußdiagramm) genommen, und dafür wird ein SKS initialisiert.
Die Adresse des neuen Segments steht im Accumulator nach Rücksprung aus LZKINI oder ASN zur Verfügung.

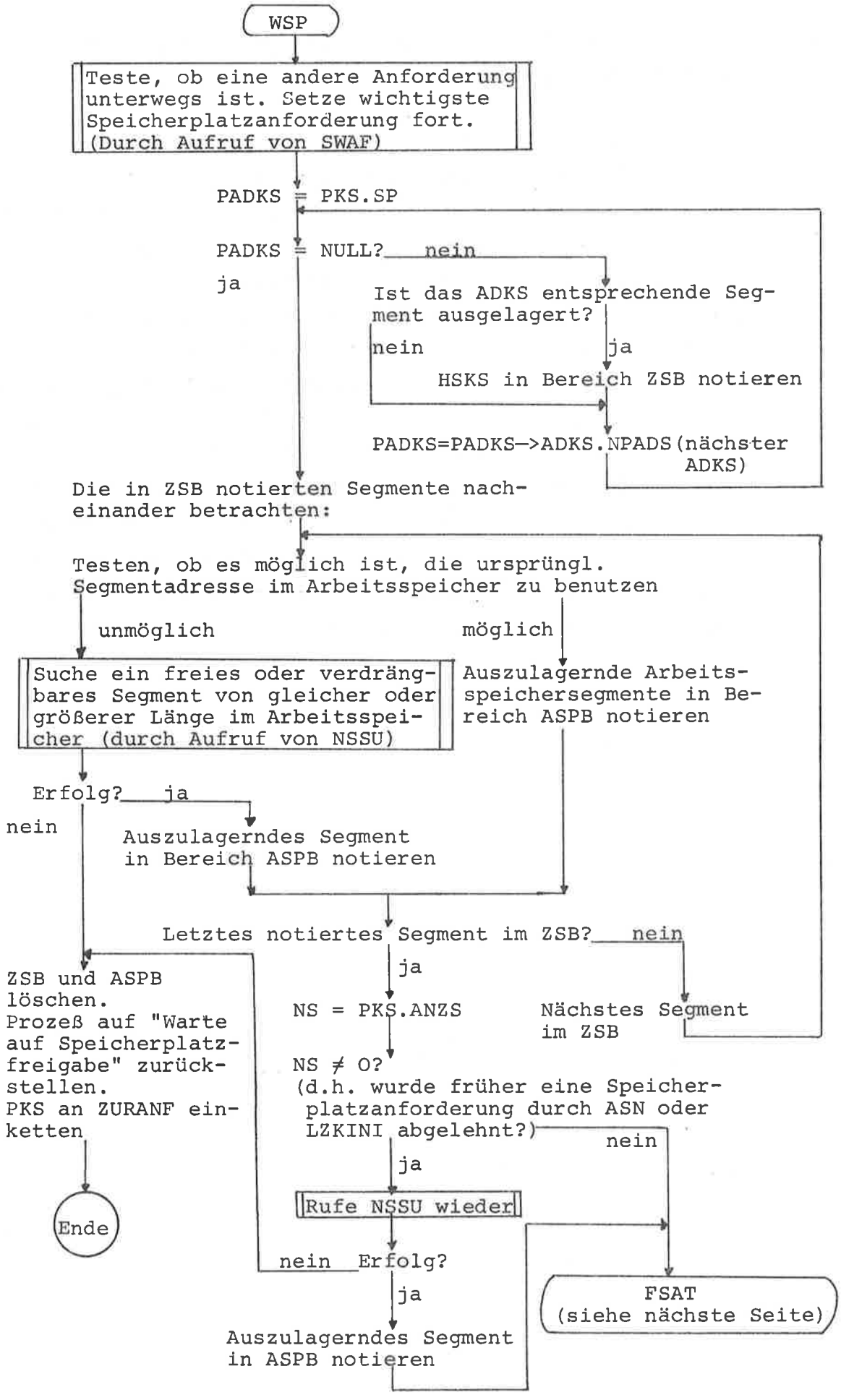

NSSU
NE = MIN (NS,33)
Suche im FS-Bereich ab Index NE, ob ein Segment mit der Länge ≥NS angezeigt ist.
gefunden?
ja
nein
I = NE
BS-Bereich betrachten:
ja
BS(I)-Kette leer?
nein
ja
Zähler SKS.Z≠ 0 oder Prioritätszahl SKS.P<PKS.ABSP?
nein
Wenn noch kein Segment notiert wurde oder wenn das Segment den Vorzug vor dem schon notierten Segment hat oder wenn das Segment unwichtiger ist, als das schon notierte Segment:
Zeiger auf SKS notieren
Letzter SKS in der BS(I)-Kette?
nein
ja
etwas notiert?
ja
Nächsten SKS betrachten
nein
I = I + 1
nein
I > 33?
ja
Return ohne Erfolg
Return mit Erfolg


AUSLAG
SKS des gefundenen Segments ausketten
Ist das Segment eine Lücke?
ja
nein
Segmentnummer freigeben
Reserviere HSKS (siehe 3.2)
SGNRBL(Segmentnr.)= ADDR(HSKS)
HSKS initialisieren
PADKS=SKS.ADKSK(Zeiger auf die Kette der ADKS)
Ist das Auslagern notwendig?
(d.h. ist das Segment weder Code-
Segment noch schon einmal ausgelagertes
nein
unveränderliches Bereichssegement?)
ja
Ist das Segment ein unveränderliches
Datensegment?
nein
ja
Merke: Auslagern des Segments später un-
nötig
Segment auslagern (durch Aufruf von TRAUS)
HSKS.HINTA = Hintergrundadr. des Segments
ZPZS
PADKS = NULL?
ja
nein
Zugreifenden Prozeß über PADKS->ADKS.ZPKS
erreichen: WSPBIT in Status-Zelle PKS.SZ
eintragen; Prozeß und dessen Unterprozesse
mit kurzem PKS zurückstellen.
PADKS->ADKS.ZSKS = ADDR(HSKS)
PADKS = Adresse nächster ADKS
Return


NIMM
Evtl. warten bis die Segmentauslagerung erfolgreich beendet ist
SKS.K = Typ des Segments
Ist das neue Segment resident?
ja
nein
SKS in die BR-Kette einketten
SKS in die BS(NS)-Kette einketten
SKS.P = PKS.ABSP
SKS.Z = 1
Speicherplatz für einen ADKS anfordern (siehe 3.2)
ADKS an SKS.ADKSK hängen und in die PKS.SP-Kette einketten
ADKS weiter initialisieren
SKS.SA = NS
Neue Segmentnummer reservieren und in SKS.SGNR eintragen
KBUPD
Segmentnummer des neuen Segments in die passende Zelle von KBL eintragen
Ist das neue Segment kürzer als das alte?
nein
ja
Restliches Stück als Lücke initialisieren:
einen SKS am Kopf einrichten.
Ihm eine Segmentnummer erteilen.
Evtl. nachfolgende Lücke zusammenkleben.
SKS.SA = entspr. passende Seitenanzahl.
In die passende Zelle des FS-Bereiches einketten.
KBL entsprechend korrigieren.
SKS weiter vollständig initialisieren
Return

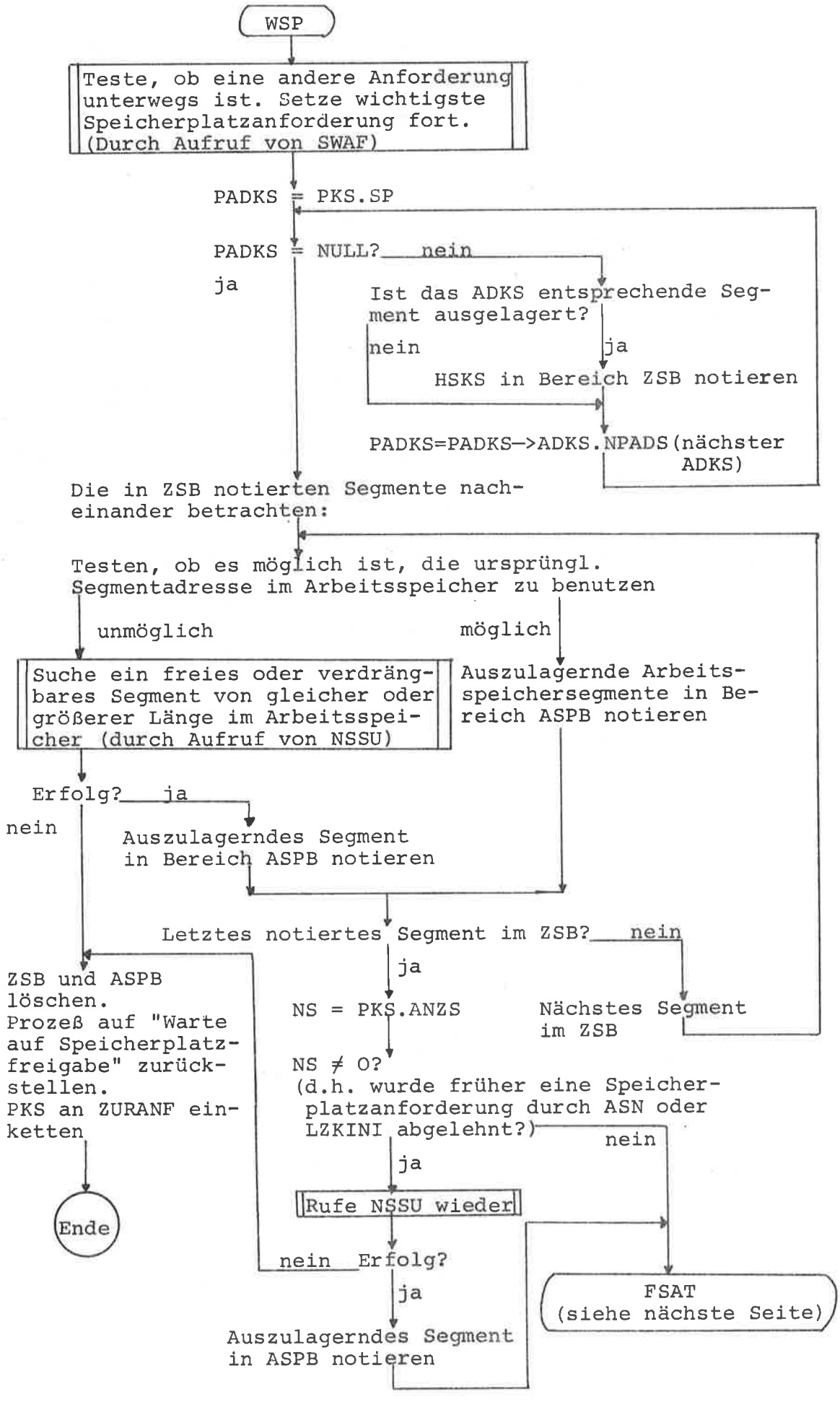
3.5.4.2 Speicherplatzanforderung für Segmente, die bereits auf dem Hintergrundspeicher existieren

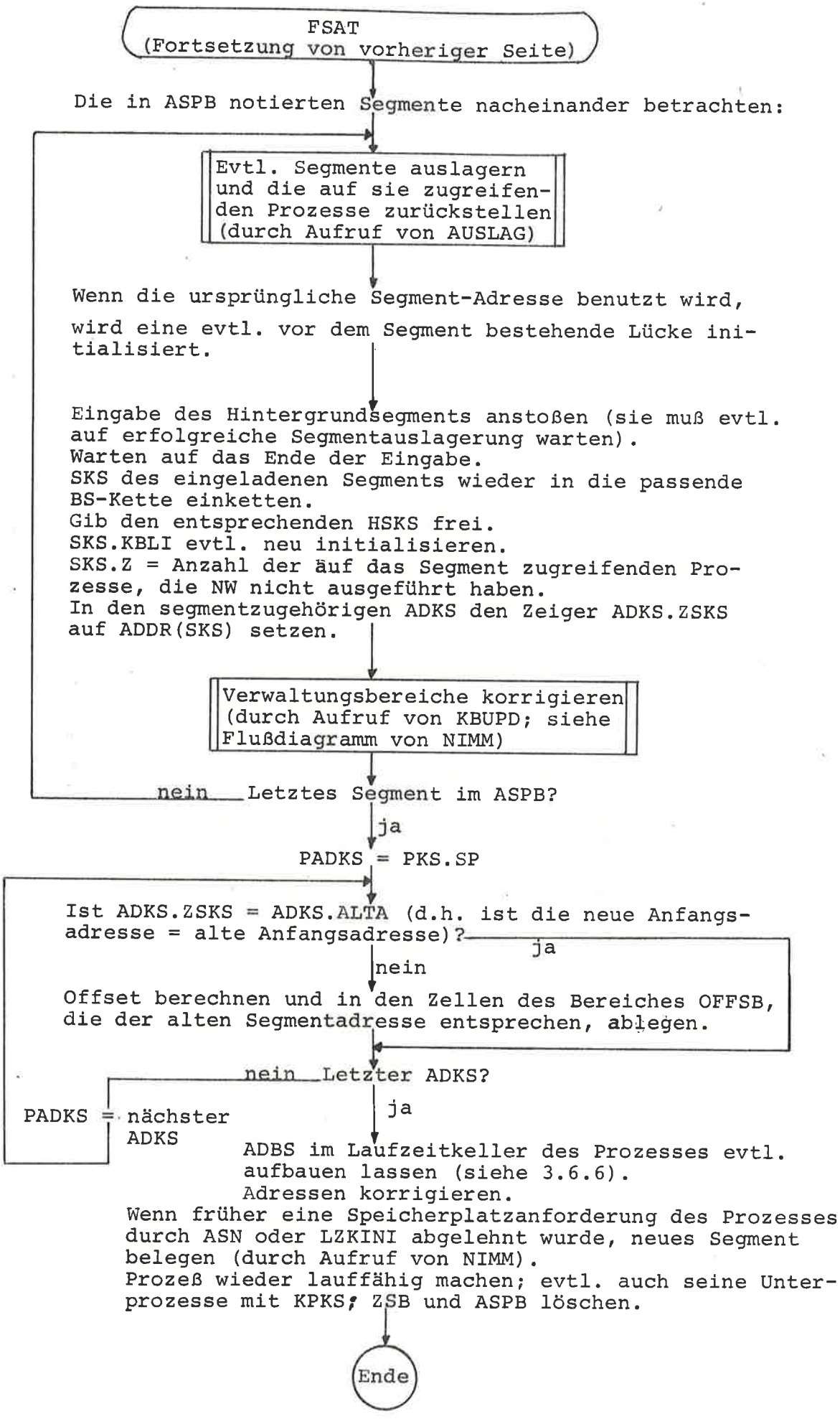
Wenn ein Segment auf dem Hintergrundspeicher existiert und nicht mehr im Arbeitsspeicher vorhanden ist, wurde es im Unterprogramm AUSLAG ausgelagert (siehe 3.5.4.1). Dabei wurden alle auf das Segment zugreifenden Prozesse so zurückgestellt, daß sie Platz im Arbeitsspeicher für die Ablage ihrer ausgelagerten Segmente anfordern, wenn sie wieder laufen: Das WSPBIT wurde in die SZ-Zelle des PKS eines solchen Prozesses notiert und, falls er vorher lauffähig war, wurde in der CA-Zelle seines PKS der Aufruf "CALL EVRUN" abgelegt. Wenn nun der Prozeß zum Laufen kommt, wird das Unterprogramm EVRUN ausgeführt; dabei wird der Aufruf "CALL WSP" in die PKS.CA-Zelle eingetragen, so daß der Prozeß danach das Unterprogramm WSP aufruft, das den notwendigen Speicherplatz für die Segmente, die auf den Hintergrund verdrängt worden sind, im Arbeitsspeicher sucht und ggf. belegt.


WSP
Teste, ob eine andere Anforderung unterwegs ist. Setze wichtigste Speicherplatzanforderung fort. (Durch Aufruf von SWAF)
PADKS = PKS.SP
PADKS = NULL?
nein
ja
Ist das ADKS entsprechende Segment ausgelagert?
nein
ja
HSKS in Bereich ZSB notieren
PADKS=PADKS->ADKS.NPADS (nächster ADKS)
Die in ZSB notierten Segmente nacheinander betrachten:
Testen, ob es möglich ist, die ursprüngl. Segmentadresse im Arbeitsspeicher zu benutzen
unmöglich
möglich
Suche ein freies oder verdrängbares Segment von gleicher oder größerer Länge im Arbeitsspeicher (durch Aufruf von NSSU)
Auszulagernde Arbeitsspeichersegmente in Bereich ASPB notieren
Erfolg?
ja
nein
Auszulagerndes Segment in Bereich ASPB notieren
Letztes notiertes Segment im ZSB?
nein
ja
ZSB und ASPB löschen. Prozeß auf "Warte auf Speicherplatzfreigabe" zurückstellen. PKS an ZURANF einketten
NS = PKS.ANZS
Nächstes Segment im ZSB
NS ≠ 0?
(d.h. wurde früher eine Speicherplatzanforderung durch ASN oder LZKINI abgelehnt?)
nein
ja
Rufe NSSU wieder
Ende
nein
Erfolg?
ja
FSAT
(siehe nächste Seite)
Auszulagerndes Segment in ASPB notieren

3.5.4.2 Speicherplatzanforderung für Segmente, die bereits auf dem Hintergrundspeicher existieren

Wenn ein Segment auf dem Hintergrundspeicher existiert und nicht mehr im Arbeitsspeicher vorhanden ist, wurde es im Unterprogramm AUSLAG ausgelagert (siehe 3.5.4.1). Dabei wurden alle auf das Segment zugreifenden Prozesse so zurückgestellt, daß sie Platz im Arbeitsspeicher für die Ablage ihrer ausgelagerten Segmente anfordern, wenn sie wieder laufen: Das WSPBIT wurde in die SZ-Zelle des PKS eines solchen Prozesses notiert und, falls er vorher lauffähig war, wurde in der CA-Zelle seines PKS der Aufruf "CALL EVRUN" abgelegt. Wenn nun der Prozeß zum Laufen kommt, wird das Unterprogramm EVRUN ausgeführt; dabei wird der Aufruf "CALL WSP" in die PKS.CA-Zelle eingetragen, so daß der Prozeß danach das Unterprogramm WSP aufruft, das den notwendigen Speicherplatz für die Segmente, die auf den Hintergrund verdrängt worden sind, im Arbeitsspeicher sucht und ggf. belegt.


WSP
Teste, ob eine andere Anforderung unterwegs ist. Setze wichtigste Speicherplatzanforderung fort. (Durch Aufruf von SWAF)
PADKS = PKS.SP
PADKS = NULL?
nein
ja
Ist das ADKS entsprechende Segment ausgelagert?
nein
ja
HSKS in Bereich ZSB notieren
PADKS=PADKS->ADKS.NPADS (nächster ADKS)
Die in ZSB notierten Segmente nacheinander betrachten:
Testen, ob es möglich ist, die ursprüngl. Segmentadresse im Arbeitsspeicher zu benutzen
unmöglich
möglich
Suche ein freies oder verdrängbares Segment von gleicher oder größerer Länge im Arbeitsspeicher (durch Aufruf von NSSU)
Auszulagernde Arbeitsspeichersegmente in Bereich ASPB notieren
Erfolg?
ja
nein
Auszulagerndes Segment in Bereich ASPB notieren
Letztes notiertes Segment im ZSB?
nein
ja
ZSB und ASPB löschen. Prozeß auf "Warte auf Speicherplatzfreigabe" zurückstellen. PKS an ZURANF einketten
NS = PKS.ANZS
Nächstes Segment im ZSB
NS ≠ 0? (d.h. wurde früher eine Speicherplatzanforderung durch ASN oder LZKINI abgelehnt?)
nein
ja
Rufe NSSU wieder
Ende
nein
Erfolg?
ja
FSAT (siehe nächste Seite)
Auszulagerndes Segment in ASPB notieren


FSAT
(Fortsetzung von vorheriger Seite)
Die in ASPB notierten Segmente nacheinander betrachten:
Evtl. Segmente auslagern und die auf sie zugreifenden Prozesse zurückstellen (durch Aufruf von AUSLAG)
Wenn die ursprüngliche Segment-Adresse benutzt wird, wird eine evtl. vor dem Segment bestehende Lücke initialisiert.
Eingabe des Hintergrundsegments anstoßen (sie muß evtl. auf erfolgreiche Segmentauslagerung warten).
Warten auf das Ende der Eingabe.
SKS des eingeladenen Segments wieder in die passende BS-Kette einketten.
Gib den entsprechenden HSKS frei.
SKS.KBLI evtl. neu initialisieren.
SKS.Z = Anzahl der auf das Segment zugreifenden Prozesse, die NW nicht ausgeführt haben.
In den segmentzugehörigen ADKS den Zeiger ADKS.ZSKS auf ADDR(SKS) setzen.
Verwaltungsbereiche korrigieren (durch Aufruf von KBUPD; siehe Flußdiagramm von NIMM)
nein
Letztes Segment im ASPB?
ja
PADKS = PKS.SP
Ist ADKS.ZSKS = ADKS.ALTA (d.h. ist die neue Anfangsadresse = alte Anfangsadresse)?
ja
nein
Offset berechnen und in den Zellen des Bereiches OFFSB, die der alten Segmentadresse entsprechen, ablegen.
nein
Letzter ADKS?
PADKS = nächster ADKS
ja
ADBS im Laufzeitkeller des Prozesses evtl. aufbauen lassen (siehe 3.6.6).
Adressen korrigieren.
Wenn früher eine Speicherplatzanforderung des Prozesses durch ASN oder LZKINI abgelehnt wurde, neues Segment belegen (durch Aufruf von NIMM).
Prozeß wieder lauffähig machen; evtl. auch seine Unterprozesse mit KPKS; ZSB und ASPB löschen.
Ende

Die Wirkung von SWAF, NSSU, AUSLAG und NIMM wurde im vorherigen Abschnitt angegeben.
Die Ein- oder Ausgabe von Segmenten zwischen Arbeitsspeicher und Hintergrundspeicher ist hier nicht festgelegt, da sie stark von der Struktur des Hintergrundspeichers abhängig ist. Hier müßte bei der Implementierung ein gewisser Anpassungsaufwand geleistet werden. Wir sind nur von der elementaren Annahme ausgegangen, daß vor der Eingabe eines Segments aus dem Hintergrund die auszulagernden Segmente erfolgreich ausgegeben worden sind.

3.5.5 Erteilung bzw. Entzug des Zugriffsrechts auf ein Segment

Erteilung des Zugriffsrechts

Wenn innerhalb des Codes eines Prozesses Daten aus einem fremden Segment angesprochen werden, soll dem Prozeß das Zugriffsrecht auf das Segment erteilt werden. Dem Prozeß wird dabei evtl. ein ADKS, der spezifisch für das Segment ist, zugeteilt und in die PKS.SP-Kette eingekettet.
Dieses geschieht durch den Aufruf von DCLSN oder DCLSP (siehe 3.4.2.2), der folgende Form hat:

```
CALL DCLSN || Liste von Segmentnummern
```

oder

```
CALL DCLSP || Liste von Zeigern auf SKS.
```

DCLSN und DCLSP führen folgendes aus:

- Parameter in der Liste werden nacheinander geholt. Die entsprechenden Segmente, die auf dem Hintergrund ausgelagert worden sind, werden in den Bereich ZSB notiert.

- Falls im ZSB ausgelagerte Segmente notiert wurden, wird getestet, ob genügend Speicherplatz für ihre Einlagerung im Arbeitsspeicher steht. Wenn ja, werden diese Segmente in den Arbeitsspeicher eingebracht. Wenn nein, wird der Prozeß zurückgestellt. Dies geschieht durch Ausführung eines Teils von WSP.

- Parameter in der Liste werden wieder nacheinander geholt. Wenn das entsprechende Segment nicht RESIDENT ist, wird ein ADKS reserviert, in die segmentspezifische Kette von SKS.ADKSK und in die prozeßspezifische Kette PKS.SP eingekettet und in die VZADB-Zelle des ADKS der Zeiger NULL eingetragen. Wenn die Prioritätszahl ADSP des Prozesses kleiner als die Prioritätszahl P im SKS des Segments ist, wird SKS.P die Zahl PKS.ABSP zugewiesen; damit wird erreicht, daß im SKS des Segments jeweils die Priorität des wichtigsten zugreifenden Prozesses eingetragen ist.

Entzug des Zugriffsrechts

Am Ende eines Prozesses wird mit Hilfe seiner ADKS in den SKS aller Segmente, für die ihm in DCLSN und DCLSP das Zugriffsrecht erteilt wurde, der Zähler Z um eins erniedrigt und die Prioritätszahl P geändert. Die ADKS werden aus den segmentspezifischen Ketten entfernt und freigegeben.

3.5.6 Freigabe von Segmenten

Datensegmente

Beim Verlassen eines Blocks wird für jedes Segment, das ihm bei der Eröffnung zur Ablage eines Bereichs zugeordnet wurde, die Prozedur GSF aufgerufen.
Beim Ende eines Prozesses wird im Rahmen von TERMPR bzw. PREND die Prozedur BFLZK aufgerufen.

Funktion: GSF

Falls das Segment nicht RESIDENT ist, wird der ADKS vom SKS und von PKS abgekettet und freigegeben, weiter wird ggf. der belegte Platz auf dem Hintergrundspeicher freigegeben.
Fortsetzung bei BFLZK.

Funktion: BFLZK

Die Segmentnummer wird freigegeben und der SKS aus der Kette der belegten Segmente entfernt (3.5.2).
Das Segment wird als Lücke betrachtet und, falls möglich, mit seinen Nachbarlücken zusammengefaßt, dabei wird die Liste der freien Segmente (3.5.2) entsprechend geändert.

Auf Speicherplatzfreigabe wartende Prozesse werden aus diesem Wartezustand befreit.

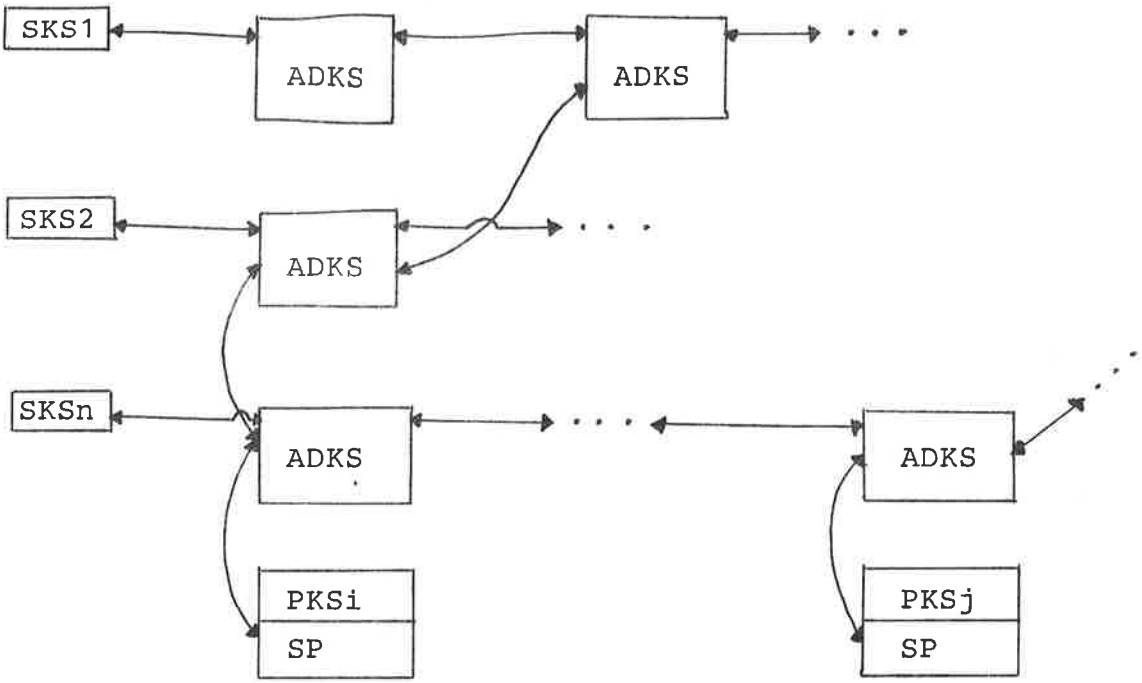
Codesegmente

Codesegmente werden zunächst nicht freigegeben, es wird nur notiert, daß sie nicht benutzt werden, um bei Bedarf (siehe ASN und WSP) ihren Speicherplatz freizugeben. Sie bleiben permanent auf dem Hintergrund und werden bei erneutem Zugriff ggf. wieder in den Arbeitsspeicher gebracht.

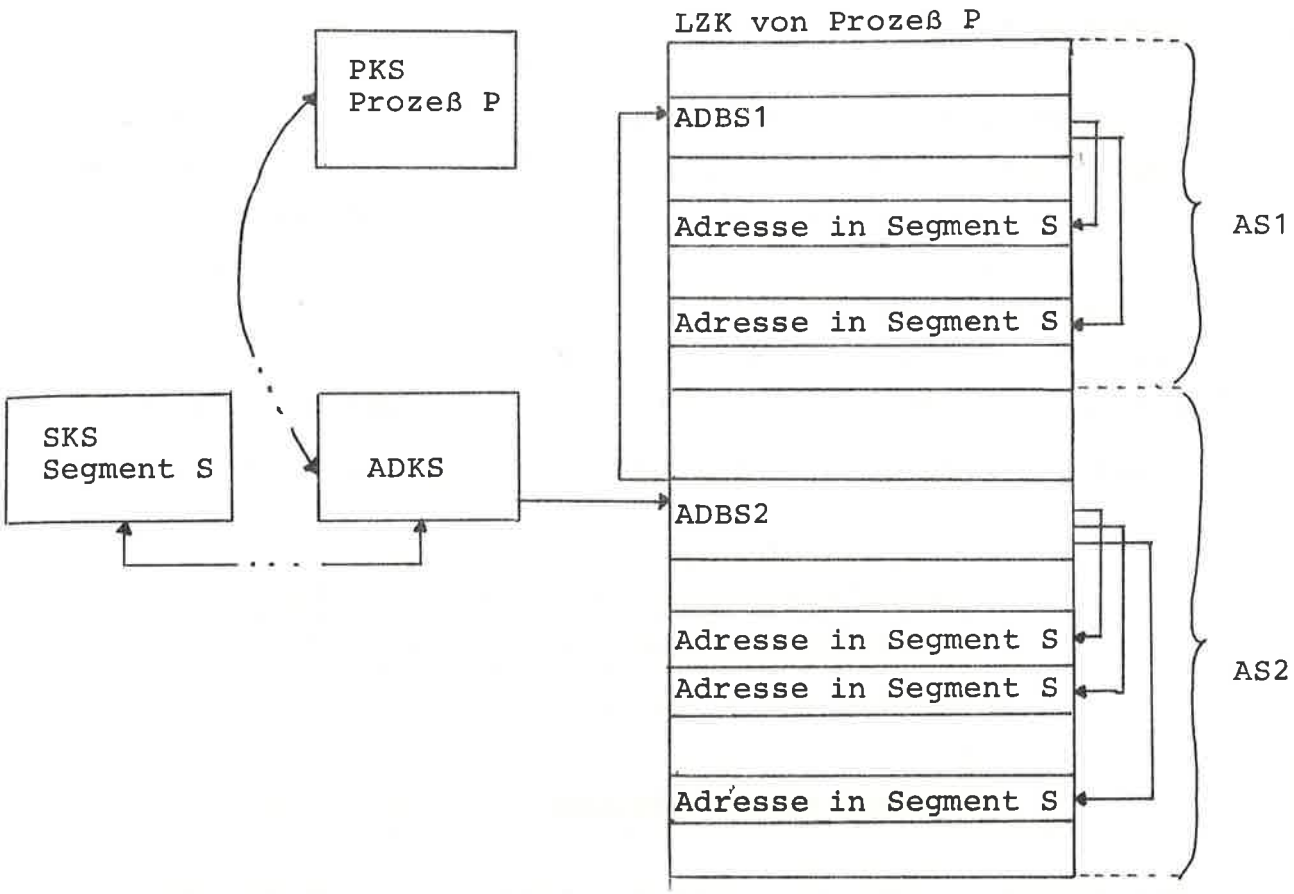
3.6 Korrektur von Referenzen bei Verlagerung von Segmenten

3.6.1 Prinzipien

Unter Regie eines Prozesses werden verschiedene Segmente angesprochen. Wenn ein Segment auf den Hintergrund ausgelagert wird (oder wenn es verschoben wird), müssen alle darauf zugreifenden Prozesse zurückgestellt werden.
Das Betriebssystem braucht also eine Verwaltungsstruktur, die angibt, welche Prozesse auf welche Segmente Zugriff haben.
Wenn ein Segment in den Arbeitsspeicher eingebracht wird oder wenn es verschoben ist, müssen alle Referenzen daraufhin aktualisiert werden. Das System muß also vom Segment her ausfinden, welche Adreßvariablen tatsächlich ins Segment weisen. Diese Adreßvariablen befinden sich in den Laufzeitkellern der Prozesse, die auf das Segment Zugriff haben oder in den Segmenten von Zeigerbereichen oder Markenbereichen, die von diesen Prozessen kreiert wurden.
Dieses wird in unserem System erreicht, indem folgende Datenstrukturen benutzt werden:
An den Prozeßkontrollsatz eines Prozesses werden *Adreßkontrollsätze* ADKS angehängt, die spezifisch für jedes Segment sind, das der Prozeß anspricht. Die ADKS sind an die segmentzugehörigen Segmentkontrollsätze SKS angekettet.



Ein ADKS ist also spezifisch für ein Segment und einen Prozeß. Damit werden einerseits alle Prozesse, die Zugriff auf ein bestimmtes Segment haben, und andererseits alle Segmente, auf die ein bestimmter Prozeß Zugriff hat, bekannt. ADKS sind im Abschnitt 3.5.2 beschrieben. ADKS sind im Systemraum des Arbeitsspeichers abgelegt.
Physikalische Adressen von Daten in Datensegmenten sind in Laufzeitkellern von Prozessen oder in Zeigerbereichen oder Markenbereichen, die von Prozessen kreiert wurden, abgelegt. Die Ablagestelle dieser Adressen muß vom Segment her erreichbar sein, um sie ggf. korrigieren zu können. Dazu dienen die *Adreßblöcke* ADBS, die in den Laufzeitkellern der Prozesse abgelegt werden. In einem ADBS sind also Hinweise auf ABlagestellen von physikalischen Adressen in einem Segment S abgelegt. Ein ADBS gilt für physikalische Adressen, die im Aktivierungssatz einer Prozedur abgelegt sind. Der ADBS wird mit anderen ADBS für dasselbe Segment in anderen Aktivierungssätzen des Laufzeitkellers gekettet. Ein Zeiger im ADKS, der spezifisch für das Segment und den Prozeß ist, weist auf den Anfang dieser Kette.



Eine Adresse ist also vom ADBS her angezeigt.
In diesem Abschnitt wird die Verwaltung der ADBS innerhalb des Laufzeitkellers angegeben.

3.6.2 Adreßblöcke ADBS

Von ADBS werden entweder einzelne im Aktivierungssatz abgelegte Adressen angezeigt, deren relative Position im AS bekannt ist oder Adressen, die in getrennten Segmenten von Zeigerbereichen oder Markenbereichen gespeichert sind.
Zu den letzteren gehören auch die Zeiger oder Markenvariablen, die zu indizierten Datenstrukturen gehören.
Entsprechend werden die ADBS in zwei Gruppen eingeteilt:

- ADBS, von denen einzelne Adressen angezeigt werden:

ADBS

Verkettungszeiger auf nächsten ADBS
-(Anzahl der angezeigten Adressen)
Zeiger auf 1. Adresse ⋮ Zeiger auf n. Adresse

Die einzelnen Adressen sind im statischen Teil des Aktivierungssatzes abgelegt. Die entsprechenden ADBS werden daher auch im statischen Teil des Aktivierungssatzes abgelegt.

- ADBS, von denen Mengen von Adressen (d.h. Adressen in Bereichen oder in indizierten Datenstrukturen) angezeigt werden:

ADBS

Verkettungszeiger auf nächsten ADBS
Anzahl der angezeigten Adreßmengen
Zeiger auf 1. Adreßmenge ⋮ Zeiger auf n. Adreßmenge

Die Bereiche und die indizierten Datenstrukturen sind in Segmenten abgelegt, die getrennt vom Laufzeitkeller sind und die von Zeigern im dynamischen Teil der Aktivierungssätze blockspezifisch angezeigt sind (siehe 3.4.2.1 und 3.4.2.5).

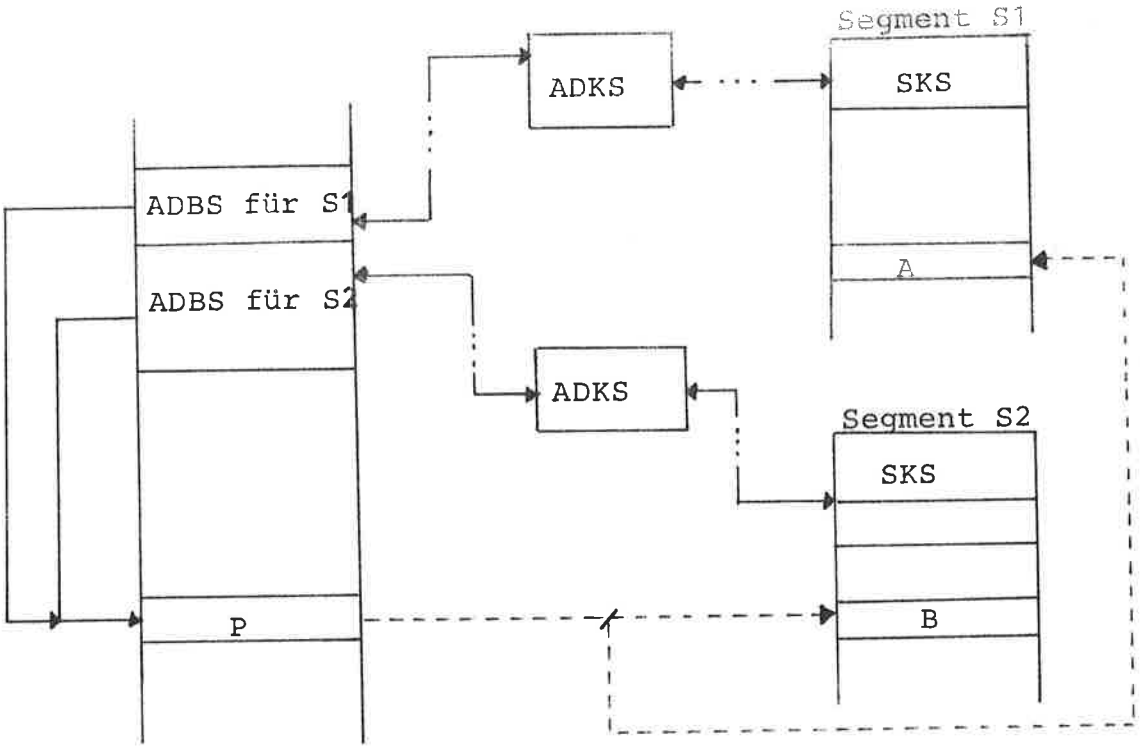
Zu den Bereichen und indizierten Datenstrukturen gehören auch Adreß-Abbildungsfunktionen, die den Zugang zu ihren Elementen ermöglichen. Die im ADBS angegebenen Zeiger auf eine Adreßmenge sind in der Tat Zeiger auf die zugehörigen Adreß-Abbildungsfunktionen. Wie und wo diese Adreß-Abbildungsfunktionen abgelegt werden, ist Sache der Implementierung; da das Unterprogramm zur Korrektur von Adressen in Bereichen oder indizierten Datenstrukturen die Adreß-Abbildungsfunktionen ebenfalls benutzt, ist es nicht in unserem PEARL-Betriebssystem mitgeliefert. ADBS, von denen Mengen von Adressen angezeigt werden, sind im blockspezifischen Teil des dynamischen Teils des Aktivierungssatzes abgelegt.

3.6.3 Schwierigkeiten bei der Analyse von Adressen

Hier werden die Zeiger (in PEARL *reference-two-identifier*) und die Markenvariablen (*reference-one-label*) gemeinsam *Adreßvariable* genannt. Adreßvariablen werden Adressen (Adreß-Konstante oder die in Adreßvariablen bereits enthaltenen Werte) zugewiesen. Im Lauf des Programms kann aber eine Adreßvariable Adressen aus verschiedenen Segmenten zugewiesen bekommen. Solch eine Adreßvariable müßte dann von verschiedenen segmentspezifischen ADBS angezeigt sein.

Beispiel:

Sei P ein Zeiger, der zu einem bestimmten Zeitpunkt auf die Größe A im Segment S1 und zum anderen Zeitpunkt auf die Größe B im Segment S2 hinweist. Die Ablagestelle von P wäre so angezeigt:



Um diese Verzeigerung richtig zu realisieren, brauchte man nur zu wissen, daß P auf A oder B hinweisen kann. Das wäre zur Übersetzungszeit relativ einfach zu erkennen, wenn aus der Programmniederschrift klar und eindeutig hervorgeht, daß P die Adresse der wohlbekannten Größe A bzw. B zugewiesen bekommt. Jetzt gibt es aber auch den Fall, in dem A z.B. in einer basisbezogenen Unterstruktur auftritt, die von einem Zeigerbereich-Element angezeigt ist, das wiederum über einen Parameter-Zeiger bekannt ist; also in der PL/1-Schreibweise etwa:

```
P = ADDR (Q->B(I)->ST.A).
```

Die Frage "Welchem Segment gehört diese Adresse?" ist äußerst schwierig zu beantworten und würde vom Compiler eine sehr sorgfältige Programmanalyse verlangen. Dieses ist natürlich nicht unmöglich, jedoch sind wir von keiner so extremen Annahme ausgegangen.

3.6.4 Implementierungsmöglichkeiten

Die verschiedenen Implementierungsmöglichkeiten stehen unter zwei Gesichtspunkten:

1. Inwieweit soll der Compiler die möglichen Adreßzuweisungen analysieren.
2. Welchen Segmentarten wird es erlaubt, verschoben oder auf den Hintergrundspeicher ausgelagert zu werden. Es ist klar, daß Referenzen auf Segmente, die im Arbeitsspeicher bleiben, nie korrigiert werden; es wird in dem Fall kein ADBS für solche Adressen aufgebaut.

Zu Punkt 1. - der Fall ausgenommen, in dem der Compiler jede Adreßzuweisung ausführlich analysiert - kann man die Adreßvariablen in zwei Klassen einteilen:
Zur ersten Klasse würden die Adreßvariablen gehören, von denen der Compiler eindeutig erkannt hat, auf welche Segmente sie sich beziehen können.
Dieser Klasse würde z.B. die Adreßvariable P angehören, die die einzige Zuweisung P = ADDR(x) bekommt, wo x einem bekannten Segment angehört.
Der zweiten Klasse würden die Adreßvariablen angehören, die Adressen von Größen zugewiesen bekommen, die der Compiler aus Effektivitätsgründen gar nicht analysiert. Dieser Klasse könnte z.B. die Adreßvariable Q gehören, die im Lauf des Programms die Adresse einer basisbezogenen Variablen zugewiesen bekommt. Wenn eine Adreßvariable der zweiten Klasse angehört, wird angenommen, daß sie Adressen aus allen Segmenten, auf die der laufende Prozeß zugreift, zugewiesen bekommen kann. Die Adreßvariable wird also von allen ADBS angezeigt, die spezifisch für die vom Prozeß benutzten Segmente sind.
Der zweiten Klasse können unter Umständen alle Adreßvariablen angehören, wenn der Compiler gar keine Programmanalyse ausführt. Die erste Klasse ist in diesem Fall leer.

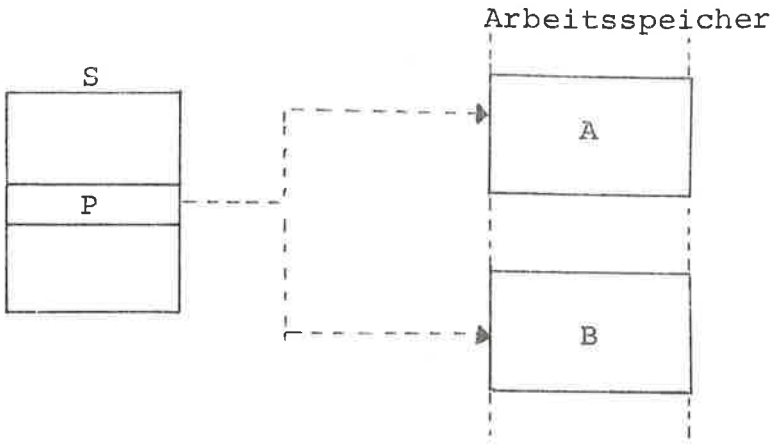
Es ist klar, daß die Grenze zwischen beiden Klassen, je nach Compilermöglichkeit, "beweglich" ist.

Eine triviale Einteilung der Adreßvariablen ist auch einzusehen: Die Zeiger einerseits und die Markenvariablen andererseits. Die ersten werden nur von ADBS angezeigt, die für Datensegmente spezifisch sind, die letzteren werden nur von ADBS angezeigt, die für Codesegmente spezifisch sind.

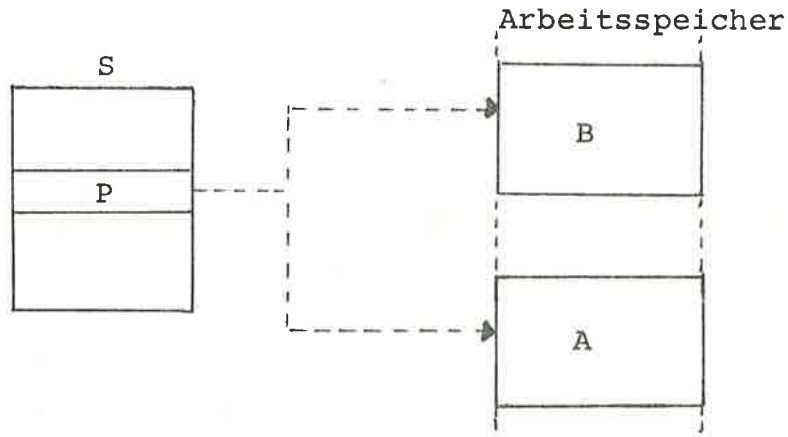
Wenn man diese zwei Einteilungsarten kombiniert, werden Adreßvariable in vier Gruppen eingeteilt. Es ist Implementierung, zu definieren, welche Adresse welcher Gruppe angehört. Dies ändert nichts an den für das Betriebssystem gelieferten Systemdiensten.

Unter dem Gesichtspunkt 2 ist folgendes zu berücksichtigen: Das System wird besonders belastet, wenn die Verschiebung oder Auslagerung von Laufzeitkeller-Segmenten zugelassen ist. Einerseits, weil ein Laufzeitkeller eine große Menge von Verwaltungsgrößen enthält, die sich auf denselben oder auf andere Laufzeitkeller beziehen, und die ebenfalls zu korrigieren wären. Noch belastender wäre andererseits die Notwendigkeit, alle in ADBS abgelegten Hinweise auf Adreßvariable nur virtuell zu realisieren, da sonst wiederum ein ADBS benutzt werden müßte, um Adressen in einem anderen ADBS zu kennzeichnen und so fort.

Entscheidend ist aber folgende Schwierigkeit, die bei der Verdrängung von Laufzeitkeller-Segmenten und Adreßvariable-Segmenten entsteht:
Nehmen wir an, daß ein solches Segment S, das den Zeiger P enthält, auf den HIntergrund ausgelagert worden ist. P kann auf Größen in zwei verschiedenen Segmenten A und B hinweisen.



Nehmen wir an, daß, während S ausgelagert ist, A und B irgendwie ihre Position im Arbeitsspeicher wechseln und daß S danach wieder in den Arbeitsspeicher eingebracht wird. Also folgendes Bild entsteht:



Wie soll man jetzt erkennen, ob P vorher auf A bzw. B hinwies? Eine "ökonomische" Lösung wäre z.B. diese:
Wenn ein Segment, das Adreßvariable enthält, auf den Hintergrund ausgelagert wird, wird zusammen mit ihm ein Bild der Arbeitsspeicherbelegung ausgelagert (d.h. eine Liste, die beschreibt, welches Segment in welchen Arbeitsspeicherseiten steht). Wenn das Segment und das dazugehörige Bild wieder in den Arbeitsspeicher eingebracht werden, wird anhand des Bildes ein Offset-Bereich für die Korrektur von Adreßvariablen eingerichtet,und die im Segment liegende Adreßvariablen werden sofort korrigiert. Man sieht schon aus dieser Beschreibung, daß das Betriebssystem dadurch ziemlich aufgebläht würde.

Hier sind also zwei Implementierungsmöglichkeiten gegeben: In einem komfortablen Betriebssystem kann die Verschiebung oder Verdrängung von Laufzeitkeller-Segmenten und Adreßvariable-Segmenten zugelassen werden. In kleineren Betriebssystemen würde man darauf verzichten.

3.6.5 Beschreibung der gewählten Implementierung

Die Einteilung der Adreßvariablen in verschiedene Klassen, bei denen mehr oder weniger - sogar keine - Programmanalyse vom Compiler verlangt wird, hat - wie schon gesagt - keinen Einfluß auf die Strukturierung unseres Betriebssystems.
Ob Adreßvariable nur von ADBS angezeigt werden, die spezifisch für die Segmente sind, auf die die Adreßvariablen tatsächlich hinweisen oder ob alle Adreßvariablen von den ADBS aller Segmente, die der Prozeß irgendwann benutzen konnte, angezeigt werden, ist dem Betriebssystem gleichgültig.
Es wird nur angenommen, daß vom SKS über ADKS Adreßblöcke ADBS verkettet sind, von denen Adreßvariable angezeigt sind, die evtl. korrigiert werden müssen.
Segmente, die Adreßvariable beinhalten (Laufzeitkeller-Segmente und Segmente von Zeigerbereichen oder Markenbereichen) sind im gelieferten Betriebssystem unverschiebbar und unverdrängbar auf den Hintergrundspeicher. Sie werden als RESIDENT betrachtet.
Segmente von arithmetischen Daten, Zeitgrößen und Befehlscode sind verdrängbar, solange sie nicht vom Programmierer als RESIDENT spezifiziert sind. Die Vorgehensweise zur Verdrängung von Segmenten wird in 3.5 angegeben.

Die in den Aktivierungssätzen abgelegten ADBS enthalten physikalische Adressen von Adreßvariablen bzw. Adreßmengen.
Die Verkettung der ADBS miteinander und mit ihrem ADKS erfolgt auch physikalisch (also nicht virtuell) wie in 3.6.1 dargestellt.

Die Einrichtung von ADKS erfolgt durch die Prozeduren DCLSN und DCLSP, die in 3.5.5 beschrieben sind. Aufrufe dieser Prozeduren sind hinter CASINI, BASINI (siehe 3.4.2.2) abzulegen.

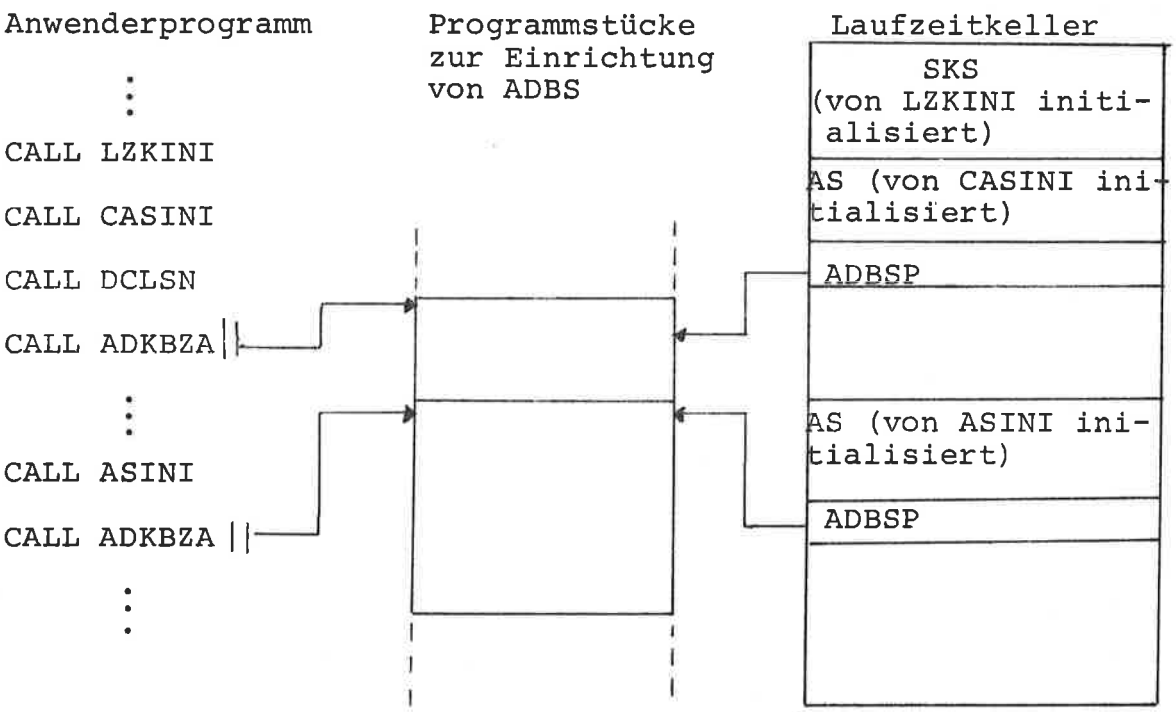
3.6.6 Einrichtung von ADBS

ADBS werden nur benutzt, wenn ein Segment verschoben oder auf den Hintergrund ausgelagert wird. Nun kann es passieren, daß ein Segment nie verschoben oder ausgelagert wird. Daher baut das System ADBS nur auf, wenn ein Segment, auf das ein Prozeß zugreift, tatsächlich verschoben oder ausgelagert und wieder in den Arbeitsspeicher gebracht worden ist. Das heißt, daß beim Aufbau eines Aktivierungssatzes kein ADBS eingerichtet wird. Es wird nur notiert, was zu tun wäre, wenn es notwendig wäre. Dazu dienen die Unterprogramme ADKBZA und ADKBZB. ADKBZA wird hinter DCLSN oder DCLSP oder beim Prozedureintritt oder Blockeintritt evtl. aufgerufen (siehe 3.4.2.2, 3.4.2.3.3 und 3.4.2.4):

```
CALL ADKBZA  || ADDR   Programm zur Einrichtung
                       von ADBS
```

Das Unterprogramm ADKBZA trägt in der Zelle AS.ADBSP den Hinweis auf das Programm zur Einrichtung von ADBS, das es als Parameter bekommen hat. Im übrigen könnte der Compiler die Befehlsfolge absetzen, die dasselbe ohne Aufruf von ADKBZA tut.
Hier wird also angenommen, daß der Compiler Programmstücke abgesetzt hat, die getrennt vom Befehlscode zur Einrichtung des Aktivierungssatzes sind. Diese Programmstücke für das ganze Anwenderprogramm werden als ein RESIDENT-Modul vom Binder zusammengefaßt.

Beispiel:



Am Ende jedes Programmstücks zur Einrichtung von ADBS muß ein Befehl zur Rückkehr in das Betriebssystem stehen. Desgleichen kopiert das Unterprogramm ADKBZB (siehe 3.4.2.4) bei Blockeröffnung den Hinweis auf das Programm zur Einrichtung von ADBS, den es als Parameter bekommen hat, in die ADBSP-Zelle in der blockspezifischen Struktur BD (siehe 3.4.2.1.2):

CALL ADKBZB || ADDR Programm zur Einrichtung von ADBS

Der Laufzeitkeller bleibt so stehen, solange kein beteiligtes Segment auf den Hintergrund ausgelagert wird. Wenn aber ein Segment, auf das der Prozeß zugreift, verdrängt und danach wieder in den Arbeitsspeicher eingelagert wird, werden alle ADBS im Laufzeitkeller eingerichtet und mit ihrer Hilfe die Referenzen auf das Segment korrigiert. Der Systemdienst ADVAB ruft die Programmstücke zur Einrichtung von ADBS anhand der in den ADBSP-Zellen angegebenen Zeiger auf.

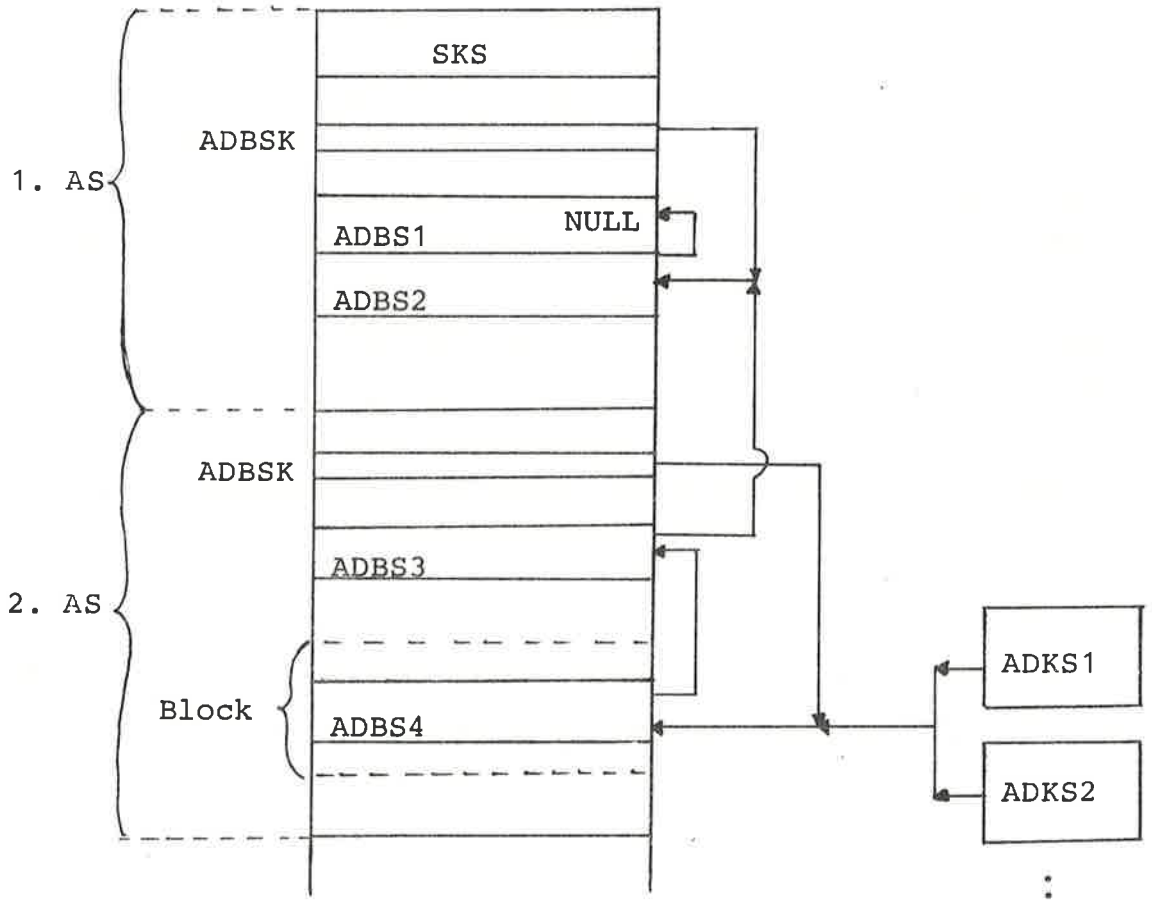
ADVAB führt folgendes aus:

. Der erste Aktivierungssatz im Laufzeitkeller, in dem evtl. schon ADBS früher eingerichtet wurden, wird vom aktuellen AS abwärts gesucht.

. Ab diesem AS, falls gefunden, oder sonst ab dem ersten AS im LZK werden in allen nachfolgenden Aktivierungssätzen die angezeigten Programmstücke zur Einrichtung von ADBS aufgerufen.

. Die VZADB in den ADKS, die an die SKS der Segmente gekettet sind, weisen auf den zuletzt eingerichteten ADBS.

In den Programmstücken zur Einrichtung von ADBS muß der Zeiger ADBSK des aktuellen AS jedesmal auf den letzten aufgebauten ADBS eingesetzt werden. Nachdem ADVAB ausgeführt worden ist, entsteht z.B. folgendes Bild des Laufzeitkellers:



3.7 Sicherung der Übertragung mit einer Datei gegen Verdrängung auf den Hintergrundspeicher

Im Abschnitt 3.5.4 wurde angedeutet, daß während einer Übertragung zwischen einer Datei auf einem Datenträger und dem Arbeitsspeicher die Quelle- oder Senke-Stelle (source oder sink) im Arbeitsspeicher nicht verdrängt werden darf. Dieses heißt, daß zwischen Übertragungsanfang und Übertragungsende das betroffene Segment des Arbeitsspeichers nicht ausgelagert werden darf.

Dazu dienen folgende Befehle vor und hinter einem Übertragungsbefehl:

```
CALL TRFBEG
Übertragungsbefehl
CALL TRFEND
```

Im Unterprogramm TRFBEG wird die Weiche IW3 (siehe 1.6.1) auf SYSF gestellt und die Zelle PKS.SPCA mit "CALL FL" initialisiert (siehe 3.5.4).
Die Weiche IW3 wird in der Systemfunktion zur Übertragung mit einer Datei GETR, PUTR, TAKR oder SENR (siehe 2.3.3) wieder auf ABFR gestellt. Damit wird erreicht, daß die Folge von TRFBEG und Übertragungsbefehl von einer anderen Systemfunktion nicht überholt wird. TRFBEG und Übertragungsbefehl werden also als eine Systemfunktion betrachtet.
Während der Übertragung steht der Befehl "CALL FL" in der PKS.SPCA-Zelle. Wenn jetzt (siehe 3.5.4) ein Prozeß Speicherplatz im Arbeitsspeicher anfordert, wird die Prioritätskette rückwärts durchsprungen und dabei das Unterprogramm FL für den übertragenden Prozeß aufgerufen.

FL führt folgendes aus:

- Die durch PKS.SP angezeigten ADKS werden aufgenommen. Für jedes entsprechende Segment wird in die Zelle K im SKS die Kennung "Nicht auslagerbar" eingetragen.

- Die Prioritätskette wird weiter rückwärts durchsprungen.

Zu bemerken ist hier, daß nicht nur das Quelle- oder Senke-Segment sondern die ganze Segmentumgebung des Prozesses nicht auszulagern ist. Dieses empfiehlt sich aus Vereinfachungsgründen der Verwaltung. Nachdem die Dateiübertragung beendet ist, wird TRFEND aufgerufen, in der die Sicherung aufgehoben wird.

TRFEND führt folgendes aus:

. In die SPCA-Zelle im PKS des Prozesses wird der Befehl "GOTO *PKS.PV" eingetragen.

. Die durch PKS.SP angezeigten ADKS werden aufgenommen. Für jedes entsprechende Segment wird in die Zelle K im SKS die Kennung "Nicht auslagerbar" gelöscht.

ANHANG A

SCHNITTSTELLEN ZUM COMPILER

A1 ECHTZEIT-BESTANDTEILE

Prozeßsteueranweisungen mit Startbedingungen (1.4.1)

Eine Anweisung der Form

Startbedingung,''Prozeßsteuerung [Taskname][Optionen]

wird intern durch folgende Befehlssequenz dargestellt:

(1) CALL SCHANF || ADDR Einplankontrollsatz EPS (bei statischer Speicherverwaltung)
|| oder
|| VALUE Länge des EPS in Vielfachen von 8 Wörtern (bei dynamischer Speicherverwaltung)

(2) In der Reihenfolge ihrer Niederschrift werden die Komponenten der Startbedingungen durch folgende Aufrufe ersetzt:

CALL AZ	|| ADDR Uhrzeit	für AT	Uhrzeit
CALL AI	|| ADDR Ereignisvariable ER	für WHEN	Interruptname
CALL WAL	|| ADDR Dauer	für ALL	Dauer
CALL WEV	|| ADDR Dauer	für EVERY	Dauer
CALL VERZ	|| ADDR Dauer	für AFTER	Dauer
CALL EUN	|| ADDR Uhrzeit	für UNTIL	Uhrzeit
CALL EDU	|| ADDR Dauer	für DURING	Dauer

Die Ausdrücke für Uhrzeiten und Dauern sind vor den Aufrufen zu berechnen. Der Funktion AI wird der Hinweis auf die Ereignisvariable ER übergeben, die bei Auswertung des Systemteils erzeugt wurde und Interruptname zugeordnet ist.

Die Folge der Aufrufe für eine Startbedingung, deren letzte Komponente nicht UNTIL Uhrzeit oder DURING Dauer ist, wird durch CALL LIM abgeschlossen.

(3) Falls als Prozeßsteuerung CONTINUE und in den Optionen eine Priorität angegeben ist:

```
CALL CNTMP  || ADDR   Wert von Priorität
```

(4) Falls in den Optionen EXCEPT Taskname,'' angegeben ist:

```
CALL EXCPT  || ADDR   Except-Liste
```

(5a) Falls als Prozeßsteuerung SUSPEND, CONTINUE, TERMINATE oder PREVENT spezifiziert ist, wird bei Angabe eines Tasknamens

```
CALL SCHMT  ||VALUE  Steuerungsart
            ||ADDR   PKS, der Taskname zugeordnet ist
```

und bei Fehlen eines Tasknamens

```
CALL SCHOT  || VALUE  Steuerungsart
```

abgesetzt. Dabei ist die Steuerungsart folgendermaßen verschlüsselt:

1 für SUSPEND
2 für CONTINUE mit Prioritätsangabe
3 für CONTINUE ohne Prioritätsangabe
4 für TERMINATE
6 für PREVENT

(5b) Falls als Prozeßsteuerung RESUME spezifiziert ist, wird bei Angabe eines Tasknamens

```
CALL RSUMTA  || ADDR   PKS, der Taskname zugeordnet ist
```

und bei Fehlen eines Tasknamens

CALL RSUMLT abgesetzt.

(5c) Falls als Prozeßsteuerung ACTIVATE spezifiziert ist:

```
CALL ACTNP  || ADDR   PKS, der Taskname zugeordnet ist
            || ADDR   Wert von Priorität
            || ADDR   Sema-Kontrollsatz, der Sema zuge-
            ||        ordnet ist bzw. NULL
            || ADDR   Segment
```

Für einen Prozeß mit kurzem PKS:

```
CALL ACTKP  || ADDR   PKS, der Taskname zugeordnet ist
            || ADDR   Wert von Priorität
            || ADDR   Segment
```

Trigger-, Induce-, Disable- und Enable-Anweisung (1.6.4)

Die Anweisungen

a) TRIGGER Interruptname

b) INDUCE Signalname

c) {DISABLE / ENABLE} Interruptname

werden intern dargestellt durch

a) CALL TRIGG || ADDR — Ereignisvariable ER, die Interruptname zugeordnet ist

b) CALL INDUCE || ADDR — Ereignisvariable ER, die Signalname zugeordnet ist

c) CALL {DISB / ENAB} || ADDR — Ereignisvariable ER, die Interruptname zugeordnet ist

On-Anweisung (1.5.1)

Die Anweisung

ON Signalname: Anweisung...

wird intern dargestellt durch

CALL RESP1 || ADDR — Ereignisvariable ER, die Signalname zugeordnet ist

GOTO Adresse hinter "CALL RESPND"

↑

Objektcode der Anweisungsfolge,
hierbei ist für eine Goto-Anweisung, deren Sprungziel außerhalb dieser Anweisungsfolge liegt, der Aufruf

CALL RSPAUS ∫| ADDR — Markenbeschreibung

abzusetzen

↓

CALL RESPND

Falls in der On-Anweisung mehrere Signalnamen spezifiziert sind, ist anstatt des Aufrufs von RESP1

```
CALL RESP  || Liste von:
           || ADDR   Ereignisvariable ER
```

abzusetzen.

Prozeßsteuerung

Taskvereinbarung (1.7.6)

Zur Auswertung aller Taskvereinbarungen eines Blocks ist im Objektcode zur Blockeröffnung (siehe 3.4.2.5) der Befehl

```
CALL PKSINI || Liste von:
            || VALUE  Art der Task
```

abzusetzen. Für jede vereinbarte Task ist eine der folgenden Zahlen als Parameter zu übergeben.

1 für TASKNAME, normaler PKS
2 für TASKNAME, kurzer PKS
3 für TASK, normaler PKS
4 für TASK, kurzer PKS

Für jede auf Modul-Ebene vereinbarte Task wird der Befehl

```
CALL MPINI || ADDR   Prozeßkontrollsatz
```

abgelegt. Dabei wird die Adresse des zur Bindezeit reservierten Prozeßkontrollsatzes als Parameter übergeben.

Prozeßsteueranweisungen ohne Startbedingungen

Activate-Anweisung (1.7.7)

Die Anweisung

ACTIVATE Taskname [PRIORITY Priorität {REL|[SYS]}] [USING Sema]

wird intern dargestellt durch

```
CALL ACTT || ADDR   PKS, der Taskname zugeordnet ist
          || ADDR   Wert von Priorität
          || ADDR   Sema-Kontrollsatz, der Sema zuge-
          ||        ordnet ist bzw. NULL
          || ADDR   Segment
```

Falls für den Prozeß ein kurzer PKS existiert, wird folgender Aufruf abgesetzt

```
CALL ACTKT  || ADDR   PKS, der Taskname zugeordnet ist
            || ADDR   Wert von Priorität
            || ADDR   Segment
```

Die Prioritätsangabe ist vor dem Aufruf ausgewertet. Dabei wird als Wert einer mit SYS spezifizierten Priorität der um 2**14 erhöhte Wert des Ausdrucks übergeben. Für eine fehlende Prioritätsangabe ist zur Übersetzungszeit ein geeigneter Wert einzusetzen.

Suspend-Anweisung (1.7.8)

Die Anweisungen

a) SUSPEND Taskname [EXCEPT Taskname,'']

b) SUSPEND [EXCEPT taskname,'']

werden intern dargestellt durch

```
a) CALL SUSPTA  || ADDR   PKS, der Taskname zugeordnet ist
                || ADDR   Except-Liste

b) CALL SUSPLT  || ADDR   Except-Liste
```

Falls die EXCEPT-Option benutzt wurde, ist als Parameter ein Hinweis auf die folgendermaßen aufgebaute Except-Liste zu übergeben, anderenfalls der Hinweis auf die Konstante 0.

. Anzahl (n ≥ 1) der in der EXCEPT-Option genannten Tasknamen
. Hinweis auf den PKS, der dem ersten Tasknamen zugeordnet ist
  ⋮
. Hinweis auf den PKS, der dem n-ten Tasknamen zugeordnet ist.

Continue-Anweisung (1.7.9)

Die Anweisungen

a) CONTINUE Taskname [EXCEPT Taskname,'']

b) CONTINUE Taskname PRIORITY Priorität {REL|[SYS]}
[EXCEPT Taskname,'']

c) CONTINUE PRIORITY Priorität {REL|[SYS]}
[EXCEPT Taskname,'']

werden intern dargestellt durch

a) CALL CONTTA || ADDR PKS, der Taskname zugeordnet ist
|| ADDR Except-Liste

b) CALL CPRIOT || ADDR PKS
|| ADDR Except-Liste
|| ADDR Wert von Priorität

c) CALL CPRIOL || ADDR Except-Liste
|| ADDR Wert von Priorität

Terminate-Anweisung (1.7.10)

Die Anweisungen

a) TERMINATE Taskname [EXCEPT Taskname,'']

b) TERMINATE [EXCEPT Taskname,'']

werden intern dargestellt durch

a) CALL TERMTA || ADDR PKS, der Taskname zugeordnet ist
|| ADDR Except-Liste

b) CALL TERMLT || ADDR Except-Liste

Prevent-Anweisung (1.7.12)

Die Anweisungen

a) PREVENT Taskname

b) PREVENT

werden intern dargestellt durch

a) CALL PREVTA || ADDR PKS, der Taskname zugeordnet ist

b) CALL PREVLT

Synchronisierung

Vereinbarung von Synchronisiervariablen (1.8.1)

Zur Auswertung aller Vereinbarungen von Sema- oder Boltvariablen eines Blocks sind im Objektcode zur Blockeröffnung (siehe 3.4.2.5) die Befehle

```
CALL SEMINI  || VALUE  Anzahl der Sema-Variablen
CALL BOLINI  || Liste von:
             || VALUE  Maximumwert einer Bolt-Variablen
```

Beim Aufruf von BOLINI ist, falls für eine Bolt-Variable keine obere Grenze spezifiziert wurde, die größte darstellbare Zahl als Parameter zu übergeben.

Request- und Release-Anweisung (1.8.3)

Die Anweisungen

a) { REQUEST / RELEASE } Semaname

b) { REQUEST / RELEASE } Semaname,..

werden intern dargestellt durch

a) CALL { REQ1 / REL1 } ||ADDR Sema-Kontrollsatz

b) CALL { REQSL / RELSL } || Liste von:
|| ADDR Sema-Kontrollsatz

Reserve-, Enter-, Free- und Leave-Anweisung (1.8.4)

Die Anweisungen

a) { RESERVE / ENTER / FREE / LEAVE } Boltname

b) { RESERVE / ENTER / FREE / LEAVE } Boltname,..

werden intern dargestellt durch

a) { RES1 / ENT1 / FREE1 / LEAV1 } || ADDR Bolt-Kontrollsatz

b) { RESBL / ENTBL / FREEBL / LEAVBL } || Liste von:
|| ADDR Bolt-Kontrollsatz

Prozeßdatenübertragung (2.2.2)

Die Übertragung zwischen einem Wort des Arbeitsspeichers und einem Gerät der Prozeßperipherie wird durch folgende Funktionsaufrufe realisiert:

Eingabe

Objektcode:

```
CALL  PDIN  ||  VALUE (virtuelle Kanalnummer)
            ||  ADDR (Ablagestelle im Arbeitsspeicher)
```

Ausgabe

Objektcode:

```
CALL  PDOUT ||  VALUE (virtuelle Kanalnummer)
            ||  ADDR (Ausgabewert)
```

Die 'virtuelle Kanalnummer' ist der Index, unter dem für das in der Anweisung genannte Gerät bei Auswertung des Systemteils ein Eintrag im Bereich BVK abgelegt wurde. Gehört das genannte Gerät nicht zur Prozeßperipherie, ist der Wert 1 zu übergeben.

Vereinbarung einer Variablen FILE (2.3.2.1)

Für jeden mit dem Attribut FILE vereinbarten Bezeichner wird bei Eröffnung des Blocks ein Filekontrollsatz abgelegt und initialisiert.

DECLARE···, filename FILE,···

Objektcode: CALL ALFKS

CREATE- und OPEN-Anweisung (2.3.2.2)

Die in den Create- und Open-Anweisungen angegebenen Dateibeschreibungsparameter sind in einem Parametersatz abzulegen und dessen Adresse beim Aufruf der Funktion als Argument zu übergeben.

Der *Parametersatz* PARCO hat folgende Form:

FIL	POINTER	Adresse des über file-name erreichten FKS
TIT	CHARACTER(8)	Die in der TITLE-Option angegebene Zeichenkette datei-name
DEV	POINTER	Adresse des über device-name erreichten GKS oder NULL für Geräte der Prozeßperipherie
ART	BIT(16)	Dateityp, siehe unten
TLG(1)	BIN FIXED(15)	Seitenzahl der Datei
TLG(2)	BIN FIXED(15)	Zeilen pro Seite
TLG(3)	BIN FIXED(15)	Informationseinheiten pro Zeile
TLG(4)	BIN FIXED(15)	Länge einer Informationseinheit in Vielfachen einer Systemgröße (z.B. Byte)
GTLG(4)	BIN FIXED(15)	Diese Größen sind nicht zu initialisieren, sie werden zur Zwischenspeicherung in CRLIST und OPLIST benutzt
SPBD	BIN FIXED(15)	
NPS	POINTER	

Das Bitmuster ART hat folgenden Wertebereich:

Bit	Werte
1 - 9	0
10	1 falls Standarddatei, 0 sonst
11 12	00 für ALPHA, 01 für BASIC
13 14	01 für INPUT, 10 für OUTPUT, 11 für UPDATE
15 16	00 für STREAM, 01 für SEQUENTIAL, 11 für DIRECT

{CREATE / OPEN} file-spezifikation1,···file-spezification n;

Objektcode:

{CALL CRLIST / CALL OPLIST} || ADDR (1. Parametersatz PARCO)
⋮
ADDR (n. Parametersatz PARCO)

In diesen Fällen müssen die Parametersätze bis einschließlich NPS abgelegt werden.

CREATE file-spezifikation;

Objektcode:

Falls die Dateigliederung angegeben ist,

CALL CRDE || ADDR (Parametersatz PARCO)

Der Parametersatz wird bis einschließlich TLG(4) abgelegt.

Falls die Dateigliederung nicht spezifiziert ist, d.h. eine gerätespezifische Standardgröße und -gliederung gewünscht ist,

CALL CRSE || ADDR (Parametersatz PARCO)

Der Parametersatz wird bis einschließlich ART abgelegt.

OPEN file-spezifikation;

Objektcode:

CALL OPEN || ADDR (Parametersatz PARCO)

Bei Angabe der Dateigliederung ist der Parametersatz bis einschließlich TLG(4) anderenfalls bis einschließlich ART abzulegen.

CLOSE- und DELETE-Anweisung (2.3.2.3 u. 2.3.2.4)

```
{CLOSE }   {FILE (file-name)}
{DELETE}   {file-name       };
```

Objektcode:

```
CALL  CLOSE  ||  ADDR (der über file-name
CALL  DELETE ||  erreichte FKS)
```

LOCK- und UNLOCK-Anweisung (2.3.2.5)

```
{LOCK  }   {FILE (file-name)} ... ;
{UNLOCK}   {file-name       }
```

Objekt-Code:

```
CALL LOCK   ||  ADDR (der über den 1.file-name erreichte FKS)
            ||   :
CALL UNLOCK ||  ADDR (der über den n.file-name erreichte FKS)
```

Ein-Ausgabeanweisungen für Dateien (2.3.3)

Wenn eine Get-, Put-, Take- oder Sendanweisung die externe Endstelle in der Form

$$\left\{\begin{matrix}\text{FILE(file-name)}\\ \text{file-name}\end{matrix}\right\}\left[\left\{\begin{matrix}\text{ADVANCE}\\ \text{POSITION}\end{matrix}\right\}\left(\left\{\begin{matrix}\text{seite [,zeile,einheit]}\\ \text{[seite],[zeile],[einheit]}\end{matrix}\right\}\right)\right]$$

spezifiziert, wird im Rahmen ihrer Ausführung eine der folgenden Funktionen aufgerufen, die eine Übertragung zwischen der externen Endstelle und einem durch Anfangsadresse und Länge spezifizierten Bereich des Arbeitsspeichers einleitet.

Sprachform	Objektcode	
GET-Anweisung	CALL GETR	ADDR (Parametersatz PARS)
PUT-Anweisung	CALL PUTR	
TAKE-Anweisung	CALL TAKR	
SEND-Anweisung	CALL SENR	

Diese Funktionen setzen voraus, daß ein Parametersatz der folgenden Form existiert und initialisiert ist.

Parametersatz PARS

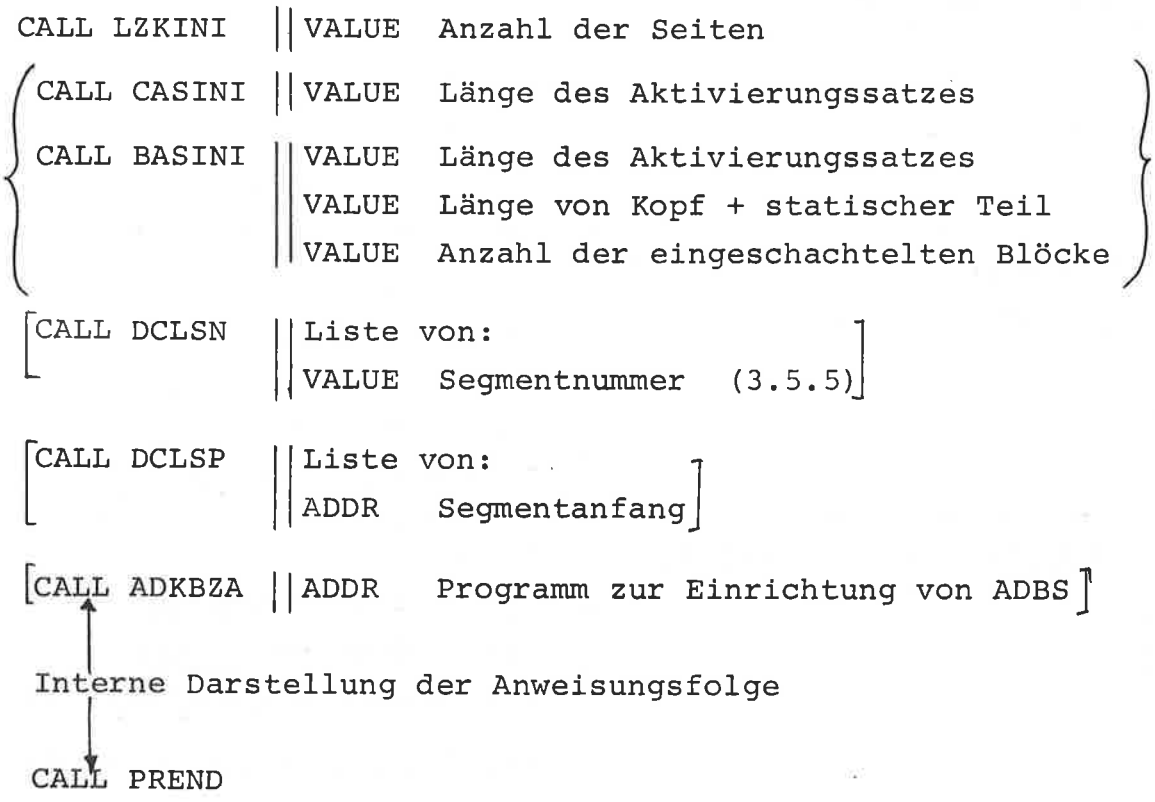
FIL	POINTER	Adresse des über file-name erreichten FKS
BA	POINTER	Adresse eines Bereiches im Arbeitsspeicher
BL	BIN FIXED(15)	Länge des Bereiches in Vielfachen einer Systemgröße (z.B. Byte)
T	BIN FIXED(15)	Positionierungstyp
P(2)	BIN FIXED(15)	Seitenangabe seite
P(3)	BIN FIXED(15)	Zeilenangabe zeile
P(4)	BIN FIXED(15)	Einheitenangabe einheit

Positionsparameter

Sprachform	Parameter
Keine Positionsangabe	T = 0
{ADVANCE / POSITION} (seite)	{T = 1 / T = 4} P(2) = Wert von seite
{ADVANCE / POSITION} (seite,zeile, einheit)	{T = 2 / T = 5} P(2) = Wert von seite P(3) = Wert von zeile P(4) = Wert von einheit
POSITION ([seite],[zeile], [einheit])	T = 6 P(2) = Wert von seite, -1 P(3) = Wert von zeile, -1 P(4) = Wert von einheit, -1 Für fehlende Angaben ist der Wert -1 zu übergeben

In den Fällen T = 0 und T = 1,4 wird der Parametersatz nur bis einschließlich T bzw. P(2) abgelegt.

Befehlsfolge einer Task (3.4.2.2)



Im Code des übergeordneten Prozesses steht vor dem Segment ein Sprungbefehl auf den ersten Befehl hinter dem eingeschachtelten Segment.

Begin-Blöcke (3.4.2.4)

```
CALL BLKINI

 [CALL PKSINI     Liste von:                                    ]
                  VALUE  Taskart                                ]

 [CALL SEMINI     VALUE  Anzahl der Sema-Variablen]

 [CALL BOLINI     Liste von:                                    ]
                  VALUE  Maximalwert der Bolt-Variablen         ]

 [CALL ALFKS      für jede File-Variable]

 [CALL ASN        VALUE  Anzahl der Seiten                      ]
                  VALUE  Typ des Segments                       ]
                für jeden Bereich                               ]

 [CALL ADKBZB     ADDR   Programm zur Einrichtung von ADBS]
       ↑
  Objektcode der Anweisungsfolge
       ↓
 {CALL SBLEND   falls mindestens eine Task-, Sema-, Bolt- oder  }
                File-Variable vereinbart ist                    }
 {CALL BLEND    sonst                                           }
```

Eine Goto-Anweisung mit einem Sprungziel außerhalb des Blocks und innerhalb der Prozedur ist durch

```
CALL BLAUS || ADDR   Markenbeschreibung des
                     Sprungziels
```

zu übersetzen.

Prozeduren

Prozedurvereinbarung (3.4.2.3.3)

CALL ASINI	||	VALUE	Länge des Aktivierungssatzes AS
	||	VALUE	Schachtelungsstufe
	||	VALUE	Länge von (Kopf + statischer Teil)
	||	VALUE	Anzahl der eingeschachtelten Blöcke
CALL PARUEB	||	VALUE	(n+1) n = Anzahl der Parameter
	||	ADDR	Ablagestelle
[CALL ADKBZA	||	ADDR	Programm zur Einrichtung von ADBS]

Befehlsfolge zur Initialisierung der im statischen Teil abgelegten Programmdaten.
Normale Übersetzung des Prozedurkörpers.

Hier werden folgende Fälle für das Verlassen einer Prozedur unterschieden (3.4.2.6):

(i) Die Return-Anweisung im äußeren Block der Prozedur:

CALL PEND	||	ADDR	Parameterposition der Rückkehradresse

(ii) Die Return-Anweisung in einem eingeschachtelten Block der Prozedur:

CALL PRET	||	ADDR	Parameterposition der Rückkehradresse

(iii) Die Goto-Anweisung:

CALL PRAUS	||	ADDR	Zeiger auf Markenbeschreibung des Sprungziels

Prozeduraufruf (3.4.2.3.2)

Hier werden drei Fälle unterschieden:

(i) Normalfall:

Der Prozeduraufruf steht weder in einer On-Anweisung noch ist die aufgerufene Prozedur den aufrufenden Prozeduren als Parameter übergeben.

```
CALL Prozedureingang ||  ADDR          Rückkehradresse
                     ||  [Liste von:                 ]
                     ||  [ADDR          Argument     ]
```

(ii) Aufruf innerhalb einer On-Anweisung:

```
CALL RESPRO  ||  {ADDR    Prozedureingang  }
             ||  {-ADDR   Parameterprozedur}
             ||   ADDR    Rückkehradresse
             ||  [Liste von:              ]
             ||  [ADDR    Argument        ]
```

(iii) Aufruf einer als Parameter übergebenen Prozedur:

```
CALL PARPRO  ||  ADDR    Parameterprozedur
             ||  ADDR    Rückkehradresse
             ||  [Liste von:              ]
             ||  [ADDR    Argument        ]
```

ANHANG B

SCHNITTSTELLEN ZUM RECHNER

Das PEARL-Betriebssystem ist in GBL1, einem PL/1-Subset geschrieben unter zusätzlicher Benutzung von IBM-PL/1-Preprozessor-Anweisungen.

Durch die Preprozessor-Anweisungen ist es möglich, aus dem allgemeinen PEARL-Betriebssystem bestimmte Teile zu entfernen oder Alternativen auszuwählen, so daß einerseits Betriebssysteme für verschiedene PEARL-Subsets extrahiert werden können und andererseits auf die vorhandene Hardware Rücksicht genommen werden kann.

Wo im vorliegenden PEARL-Betriebssystem hardware- bzw. implementationsabhängige Operationen auszuführen sind, ist ein sogenannter "MACRO-Aufruf" geschrieben. Im Teil B2 sind die Funktionen der einzelnen "MACROS" logisch beschrieben.

B1 Generierung mit Hilfe des Preprozessors

Durch die entsprechende Setzung der folgenden 13 Preprozessor-Variablen kann der Implementator mit Hilfe des IBM-PL/1-Preprozessors verschiedene Versionen des PEARL-Betriebssystems generieren. Jede der Variablen darf mit "YES" oder "NO" gesetzt werden. Im folgenden ist für jede Variable angegeben, welcher Betriebssystemmodul im Falle "NO" weggelassen wird.

PREPW1:	Unterbrechungsbearbeitung
PREPW2:	Speicherplatzverwaltung
PREPW3:	Sema- und Bolt-Bearbeitung
BV:	Bolt-Bearbeitung
PREPW4:	E/A-Verwaltung
FV:	File-Bearbeitung
HINTW:	Hintergrundverwaltung
PDEAW:	Prozeßdaten-Ein/Ausgabe
RELP:	Bearbeitung von Relativprioritäten im Anwenderprogramm
UVZW:	"NO": Es wird angenommen, daß der Dauer- bzw. Uhrzeitinterrupt in festen Zeitabständen (DELTAD bzw.DELTAU) auftritt. "YES": Es wird angenommen, daß die externen Zeitgeber für Dauer- bzw. Uhrzeitverfolgung in festen Zeitabständen die Zahlen ZKD.D bzw. ZKU.D erhöhen und der entsprechende Interrupt nur auftritt, wenn bei der Erhöhung das Vorzeichen wechselt.
UADT:	Es wird angenommen, daß in der Hardware kein Uhrzeitinterrupt vorhanden ist; die Uhrzeitbearbeitung wird aus dem Dauertakt hergeleitet.
TEST:	"YES" bewirkt, daß das PL/1-Testsystem mitgeneriert wird
OC:	"YES" Generierung für IBM-PL/1-F-Compiler "NO" Generierung für IBM-PL/1-Optimizing Compiler und GBL1-Compiler.

B2 Implementationsabhängige Betriebssystemteile (MACROs)

In diesem Kapitel sind die Betriebssystemteile logisch beschrieben, die wegen ihrer Implementationsabhängigkeit nicht im vorliegenden Betriebssystem formuliert sind (MACROs).

Die meisten MACROs werden als Prozeduren aufgerufen und sind dementsprechend zu formulieren.

Nicht als Prozeduren aufgerufen werden können die MACROs, die den Übergang vom Anwenderprogramm zum Betriebssystem und umgekehrt organisieren; der Anstoß dieser MACROs ist durch spezielle Formulierungen (die Erweiterungen zu PL/1 sind) angedeutet. Diese Erweiterungen sind in /*/ eingeschlossen, was bedeutet, daß sie von IBM-PL/1-Compilern als Kommentare überlesen werden, von jedem anderen Compiler jedoch ausgewertet werden können, der /*/ als abgeschlossenen Kommentar ansieht.

Die Beschreibung der MACROs geht von folgenden Voraussetzungen aus:
Es wird angenommen, daß das PEARL-Betriebssystem in einem zusammenhängenden Bereich des Hauptspeichers liegt, so daß bei Eintritt einer Unterbrechung durch einen Vergleich des Befehlszählers des unterbrochenen Programms mit den Grenzen des Betriebssystembereiches festgestellt werden kann, ob eine Systemfunktion unterbrochen wurde (siehe MACRO TSYSF).

Es existiert im Betriebssystem eine Struktur AST (= Anwenderstatus), in der während der Ausführung einer Systemfunktion der gesamte Status des aufrufenden Anwenderprogramms aufgehoben werden kann (alle Register, insbesondere AST.LF der Befehlszähler und AST.X das Basisadreßregister) für die Rückkehr hinter den Aufruf.

Entsprechend ist auch der in PKS ablegbare Status erweitert, der im PL/1-Programm nur durch PKS.LF und PKS.AC vertreten ist, und ebenso der ablegbare Status im RSPEL, der im PL/1-Programm nur durch RSPEL.RAS (entsprechend PKS/AST.LF) und RSPEL.AS (entsprechend PKS/AST.X) vertreten ist.

B2.1 Organisation der logischen Ununterbrechbarkeit der BS-Funktionen

B2.1.1 Die vom Anwenderprogramm aus aufrufbaren Eingänge haben das Zusatzattribut /*/INDIVISIBLE/*/. Dieses Attribut gibt an, daß beim Aufruf dieser Eingangsstelle zu allererst, also noch vor der Parameterübernahme, folgende Anweisungsfolge auszuführen ist:

INH (Verbiete Unterbrechung)
Lege Status des aufrufenden Programms in AST ab.
Setze eine Weiche, an der der Macro PAL erkennen kann, daß er bei diesem BS-Aufruf zum ersten Mal aufgerufen wird.
ENB (Gib Unterbrechung frei).

B2.1.2 Alle RETURN-Anweisungen in der Prozedur PEARL (nicht jedoch die von eingeschachtelten Prozeduren) müssen zur Ausführung folgender Anweisungsfolge führen:

Nimm den Status aus AST auf.

B2.1.3 Einige Eingangsstellen haben als Parameter eine Liste von gleichartigen Variablen; dies ist in PL/1 so formuliert:
E_name: ENTRY (/*/(*)/*/P_name) /*/INDIVISIBLE/*/;

In diesem Fall erfolgt beim Eingang keine Parameterübernahme; die Elemente der Liste werden einzeln durch Aufruf des Macros PAL vom Betriebssystem abgerufen, wodurch das jeweils abgerufene Listenelement unter P_name bereitsteht.

Ist die Parameterliste erschöpft, so wird zu der Marke, die als Parameter an PAL übergeben wurde, verzweigt, nachdem vermerkt wurde, daß bei einem eventuell folgenden PAL-Aufruf wieder die Parameterliste von Anfang an bearbeitet werden soll.

B2.1.4 Der Macro AGAIN wird angewendet, wenn ein Prozeß zur Wiederholung des aktuellen Systemaufrufs veranlaßt werden soll. Dies ist immer der Fall, wenn er ein belegtes Betriebsmittel anfordert (SEMA, BOLT, Speicherplatz, Gerät u.ä.). AGAIN hat folgende Aufgabe:

INH
Trage AST als Status im laufenden PKS ein,
wobei der laufende Befehlszähler vermindert wird, so daß der Systemaufruf wiederholt wird.
Fahre fort beim Macro SUWI.

B2.1.5 Der Macro WAITS2 wird beim Warten auf Unterprozesse am Blockende verwendet. Hier tritt der Fall ein, daß in einer Systemfunktion gewartet wird.
Es muß daher folgendes ausgeführt werden:

Rette die zur Weiterführung der Systemfunktion "Blockende" benötigten Daten (A, PBL, PPKS1, PAS, PBD, ZWIP, BSTZL, PMB und PRA) und AST in den Laufzeitkeller des laufenden Prozesses.
INH
Trage im laufenden PKS den Status zur Fortführung bei Marke WAP ein (PKS.LF = WAP).
Deaktiviere den laufenden Prozeß (PKS.CA = "GOTO *PKS.PN")
Fahre fort beim Macro SUWI

WAP: Mache die im ersten Schritt dieses Macros ausgeführte Rettung rückgängig.

B2.1.6 WAITS1 wird bei "ACTIVATE proc USING Sema" benutzt, wenn die angegebene Sema-Variable gesperrt ist. In diesem Fall wird die Arbeit, auf die Freigabe der Sema-Variablen zu warten und die Aktivierung zu Ende zu führen, vom aktivierenden an den aktivierten Prozeß (proc) delegiert.
Auszuführende Anweisungsfolge:

INH
Deaktiviere den zu aktivierenden Prozeß
(PPKS1->PKS.CA = "GOTO *PPKS1->PKS.PN")
Trage in PPKS1->PKS den Status ein, die Aktivierung zu Ende zu führen.
(PPKS1->PKS.LF = Rückkehradresse von WAITS1)
IF PPKS1 ≠ PPKS THEN DO;
ENB; RETURN (d.h. nimm AST auf); END;
ELSE fahre fort beim Macro SUWI.

B2.1.7 PSKEND beendet den Prozeß der Sekundärreaktion:

INH
Deaktiviere den Sekundärreaktionsprozeß
(SEK.CA = "GOTO *SEK.PN")
Notiere: keine weitere Sekundärreaktion auszuführen (IW2 = NSR)
Trage in SEK den Status ein, daß die Sekundärreaktion am Anfang aufgenommen wird
(SEK.LF = SEKI)
Fahre fort beim Macro SUWI.

B2.1.8 UEBERS definiert einen Punkt, an dem ein Systemprozeß überholbar ist. Dieser Macro wird nur im Programm zur Freigabe von Einplankontrollsätzen verwendet, das unter der Regie des unwichtigsten Prozesses PRIOKE läuft.
Anweisungsfolge:

INH
Trage im laufenden PKS als Status ein, daß hinter dem Aufruf von UEBERS fortzufahren ist. (PKS.LF = Rückkehradresse von UEBERS)
Fahre fort mit dem Macro SUWI.

B2.1.9 NEXTP wird am Ende einer Systemfunktion aufgerufen, wenn entweder der aufrufende Prozeß nicht fortgesetzt werden kann (blockiert) oder ein möglicherweise wichtigerer Prozeß aktiviert wurde; nachdem der Status zur Fortsetzung hinter dem Systemaufruf im PKS abgelegt ist, muß der wichtigste Prozeß gesucht werden. Dieser Status liegt normalerweise in AST, kann aber, falls die Systemfunktion durch Interrupt unterbrochen wurde (erkenntlich an IW1 = SFRETT, siehe B2.3.8), bereits im PKS abgelegt sein:

```
INH
IF ÑMRKVGL (IW1,SFRETT) THEN DO;
   Übertrage AST in den laufenden PKS; END;
Fahre fort beim Macro SUWI.
```

B2.1.10 RRETT wird im Rahmen von TRIGGER benutzt.
Da hier eine Programmunterbrechung simuliert wird, muß der Status in den PKS ausgelagert werden, falls dies noch nicht geschehen ist. RRETT führt das gleiche aus wie NEXTP,kehrt aber hinter den Aufruf zurück, anstatt bei SUWI fortzufahren.

B2.2 Organisation zur Ausführung von eingeplanten Prozeßsteueranweisungen

Prozeßsteueranweisungen, die mit einer Einplanung (Schedule) versehen sind, werden nicht beim Überlaufen unter der Regie des steuernden Prozesses sondern beim Eintritt des eingeplanten Ereignisses unter Regie des Sekundärreaktionsprozesses ausgeführt.
Die Programme zur Abwicklung der Prozeßsteueranweisung sind in beiden Fällen die gleichen, nur werden sie im Rahmen der Sekundärreaktion nicht über die INDIVISIBLE-Eingänge aufgerufen, sondern über die Macros OPAUSF und OPAUSD werden Eingänge mit dem Zusatzattribut BASED aufgerufen:
E_name: ENTRY (paraliste) /*/BASED/*/;

Beim Aufruf der Macros OPAUSF und OPAUSD weist der Zeiger auf ein Einplanelement; OPAUSF führt nun folgendes aus:

OPAUSF wird im Rahmen der Sekundärreaktion zur Ausführung von Prozeßsteueranweisungen aufgerufen und führt folgendes aus:

- Aus PEL->STR.KP wird der Zeiger auf den Kopf des Einplankontrollsatzes (EPS.OP) entnommen.
- Entsprechend der Parameterliste des in EPS.OP angegebenen BASED-Einganges werden die Inhalte der Zellen EPS.ZTV, EPS.PEO und EPS.P als Parameter bereitgestellt.
- In AST wird ein Status eingestellt zum Fortfahren hinter dem Aufruf von OPAUSF (AST.LF = Rückkehradresse von OPAUSF). Da die Anweisungsfolge zur Ausführung der Prozeßsteueranweisung mit RETURN (d.h. nimm AST auf) endet, wird so die Rückkehr in das Sekundärreaktionsprogramm organisiert.
- Aufnehmen des Programms des in EPS.OP angegebenen BASED-Eingangs.

OPAUSD wird beim Überlaufen einer Prozeßsteueranweisung mit Einplanung (Schedule) in der letzten Anweisung der bearbeitenden Betriebssystemfunktion aufgerufen, falls die Steueranweisung schon beim Überlaufen zum erstenmal auszuführen ist. Es wird das gleiche ausgeführt wie bei OPAUSF, nur daß der dritte Schritt (Status in AST bereitstellen) entfällt. RETURN (d.h. nimm AST auf) am Ende des aufgerufenen Prozeßsteuerprogramms führt so zur Fortsetzung des Programms, das die Prozeßsteueranweisung überlaufen hat.

B2.3 Programmunterbrechungsorganisation

B2.3.1 An Stellen, wo der Eintritt einer Programmunterbrechung zu Störungen des Systems führen kann, werden durch die Macros INH und ENB Programmunterbrechungen verboten bzw. wiederzugelassen.
Diese Macros wurden schon bei vorher beschriebenen Macros in diesem Sinne verwendet.

B2.3.2 Bei Eintritt einer Programmunterbrechung muß zuerst unter Verbot von weiteren Programmunterbrechungen ein Sprung über die Weiche IW1 ausgeführt werden:

GOTO IW1

Dies führt zur dem Systemzustand entsprechenden Statusrettungsaktion.

B2.3.3 STRETT ist die Statusrettungsaktion im Normalfall; der Status des unterbrochenen Programms wird im PKS des laufenden Prozesses abgelegt.

B2.3.4 IDINT identifiziert, welche Meldung (Programmunterbrechung) gerade eingetroffen ist und weist die Adresse der zugehörigen Unterbrechungsbeschreibung (Kopf einer Zeitkette, Ereignisbeschreibung oder Übertragungskontrollsatz) dem Zeiger PI zu.

B2.3.5 GKILL wird bei unerwarteter Rückmeldung aus einem Peripheriegerät verwendet, und unterbindet weitere Unterbrechungen von diesem Gerät.

B2.3.6 Ist nach der Primärreaktion auf eine Unterbrechung keine Sekundärreaktion nötig, wird PRFORT aufgerufen zur Fortsetzung der unterbrochenen Arbeit.

Wurde das System beim Suchen des wichtigsten Prozesses unterbrochen (siehe B2.4.3), wird diese Arbeit neu begonnen.

(IF MRKVGL (IW1,NRETT) THEN setze fort bei SUWI)

PRFORT1:

Aufnehmen des Status aus dem laufenden PKS.

B2.3.7 TSYSF prüft, in welchem Zustand das System unterbrochen wurde und führt die entsprechende Reaktion aus:

Wurde während des Suchens des wichtigsten Prozesses unterbrochen (siehe 3.2.4.3), wird diese Arbeit neu begonnen.
(IF MRKVGL (IW1,NRETT) THEN fahre fort bei SUWI)

Ist der Befehlszähler des unterbrochenen Programms (PKS.LF) eine Adresse im Hauptspeicherbereich, in dem das PEARL-Betriebssystem liegt, kehrt TSYSF hinter den Aufruf zurück.

Ansonsten (Unterbrechung eines Anwenderprogramms) wird zur Marke LSKE verzweigt.

B2.3.8 SFFORT wird aufgerufen zur Fortsetzung einer unterbrochenen Systemfunktion vor Einleitung der Sekundärreaktion:
Ist die unterbrochene Systemfunktion die Sekundärreaktion, wird einfach der Sekundärreaktionsprozeß fortgesetzt.
Anderenfalls muß ebenfalls der laufende Prozeß fortgesetzt werden, wobei sichergestellt werden muß, daß, falls die Systemfunktion mit RETURN (d.h. nimm AST auf) endet, hinter dem Aufruf von SFFORT fortgesetzt wird, wo ein Aufruf an den Macro SUWI das Aufnehmen des Sekundärreaktionsprozesses gewährleistet.
Zu diesem Zweck wird AST in den Status im PKS übertragen und in AST der Status zur Fortsetzung hinter SFFORT eingetragen.
Außerdem wird durch Umstellen der Weiche IW1 sichergestellt, daß bei erneuter Unterbrechung nicht der bereits im PKS eingetragene Status zur Rückkehr ins Anwenderprogramm überschrieben wird.

Die Anweisungsfolge von SFFORT ist also:

IF PPKS = PSEK THEN fahre fort bei PRFORT1
(siehe B2.3.6)
INH
Übertrage den Status im PKS (unterbrochene Systemfunktion) in ein Hilfsfeld (HSTAT)
Übertrage AST in den Status im PKS
Setze in AST den Status zur Fortsetzung hinter SFFORT ein (AST.LF = Rückkehradresse von SFFORT)
IW1 = SFRETT
Nimm den Status HSTAT auf und ENB

B2.3.9 SRAF wird unter der Marke SFRETT aufgerufen und macht die unter B2.3.8 gemachte Übertragung des Status aus AST in den PKS rückgängig:

Übertrage den Status aus PKS nach AST.

B2.4 Organisation der Prozeßwarteschlange

Die Prozeßwarteschlange, an deren Beginn der Prozeß SEK und an deren Ende der Prozeß PRIOKE stehen, ist zwecks schnellen Durchsuchens so organisiert, daß in der CA-Zelle jedes PKS steht:

Ein indirekter Sprung über die PN-Zelle, falls der Prozeß nicht lauffähig ist,
ein Unterprogrammaufruf des Macros SETZEP, falls der Prozeß lauffähig ist oder
ein Unterprogrammaufruf an EVRUN oder WSP zum Prüfen, ob der Prozeß lauffähig ist.

B2.4.1 Durch STPZ(P) wird in P—>PKS.CA der indirekte Sprung über die Zelle P—>PKS.PN eingetragen.

B2.4.2 Durch INIPKE wird die CA-Zelle des unwichtigsten Prozesses (PRIOKE.CA) mit dem Befehl "GOTO PRKEND" initialisiert.

B2.4.3 SUWI sucht den wichtigsten lauffähigen Prozeß, dazu wird in der Weiche IW1 vermerkt, daß bei Unterbrechung kein Status zu retten ist und in die Sprungkette, die sich über alle PKS in der Reihenfolge ihrer absteigenden Priorität zieht, eingesprungen:

```
INH
IW1 = NRETT
GOTO SEK.CA
```

B2.4.4 Ein lauffähiger Prozeß wird beim Durchspringen der Prioritätskette gefunden, wenn in der CA-Zelle seines PKS ein "CALL SETZEP" steht.

SETZEP führt folgendes aus:

INH

Ermitteln der Anfangsadresse des PKS aus der Rückkehradresse und Eintragen in PPKS (laufender Prozeß)

IW1 = RETTEN

Aufnehmen des Status aus dem PKS und ENB.

B2.4.5 Ist ein Prozeß zwar logisch lauffähig, aber noch zu prüfen, ob alle seine Betriebsmittel vorhanden sind, wird beim Durchspringen der Prioritätskette in der CA-Zelle seines PKS ein "CALL WSP" oder "CALL EVRUN" ausgeführt.
Die Ausführung eines solchen Prüfprogramms ist so zu organisieren, wie wenn es direkt vor der Stelle im Anwenderprogramm, auf die PKS.LF zeigt, aufgerufen worden wäre; dazu wird in diesen Prüfprogrammen zuerst der Macro FAUSF aufgerufen:

INH

Ermitteln der Anfangsadresse des PKS aus der Rückkehradresse der aufrufenden Prozedur (WSP bzw. EVRUN) und Eintragen in PPKS (laufender Prozeß)

Übertragen des Status aus PKS nach AST

IW1 = RETTEN

ENB

FEND ist das Gegenstück zu FAUSF und wird als letzte Anweisung der beiden Prüfprogramme aufgerufen.
FEND ist identisch mit NEXTP (siehe B2.1.9).

B2.5 Macros für die RESPONSE-Bearbeitung

B2.5.1 BGAP wird für die Bearbeitung einer SIGNAL-Meldung eines Peripheriegeräts benutzt. Das Gerät, das die SIGNAL-Meldung ausgibt, arbeitet im Auftrag eines Prozesses. Dem Zeiger PPKS2 wird durch BGAP die Adresse des PKS dieses Prozesses zugewiesen.

B2.5.2 DRSPAD wird im Rahmen der Systemfunktion RESP und RESP1 angesprochen. In die systembekannte Markenvariable RSPAD soll die Adresse der über RESP oder RESP1 im Programm eingeführten ON-Anweisung übertragen werden.
Die Adresse der ON-Anweisung ist die Rückkehradresse der RESP- bzw. RESP1-Funktion um eins erhöht:

```
INH
IF ÑMRKVGL (IW1, SFRETT)
THEN RSPAD = AST.LF+1;
ELSE RSPAD = PKS.LF+1;
ENB
```

B2.5.3 LERS wird beim Auftreten eines Signals aufgerufen und führt die Umschaltung zur Bearbeitung der Response-Anweisung durch. Der Status, in dem das Anwenderprogramm unterbrochen wurde, wird in das RSPEL übertragen und zur direkten Fortsetzung ein Status bereitgestellt, der die Ausführung der Response-Anweisung bewirkt. LERS hat einen Parameter (PPKSR), den Zeiger auf den PKS für den die Response-Anweisung auszuführen ist.
Unter der Voraussetzung, daß die Struktur "STATUS" in AST, PKS und RSPEL gleich aufgebaut ist (siehe B2), kann der Macro LERS wie folgt aussehen:

```
        INH
        IF PPKSR = PPKS THEN IF ÑMRKVGL (IW1,SFRETT) THEN DO;
           PA = ADDR(AST); GOTO LERSN; END;
        PA = ADDR(PKS.Status);
LERSN:  RSPEL.Status = PA->Status
        PA->Status.LF = PRSP1->RSPB.AD
        PA->Status.X, PPKSR->PKS.AS = PRSP1->RSPB.AS
        ENB
```

B2.5.4 BRPGRQ(P,Q) wird beim Suchen der passenden Response-Anweisung verwendet. P und Q sind POINTER.
Es ist auszuführen:

```
IF P > Q THEN GOTO ALZKE;
```

B2.6 Macros für Prozeduraufruforganisation

Beim Aufruf einer Parameterprozedur und beim Aufruf einer Prozedur aus einer Response-Anweisung ist vor dem Eintritt in die aufgerufene Prozedur einige Organisation auszuführen, deshalb sind solche Aufrufe so übersetzt:

```
CALL {PARPRO}  ||  Identifizierung der Prozedur
     {RESPRO}  ||  Parameter
               ||     .
               ||     .
               ||     .
```

Nachdem die Organisation in PARPRO bzw. RESPRO abgewickelt ist, wird durch RUPAN bzw. RUOAN der Eintritt in die angegebene Prozedur durchgeführt, wobei insbesondere die Rückkehradresse so bereitgestellt werden muß, als ob der Prozeduraufruf direkt im Anwenderprogramm gestanden hätte, so daß in der aufgerufenen Prozedur wie bei einem normalen Prozeduraufruf verfahren werden kann.

B2.7 Macros für die Speicherplatzverwaltung

B2.7.1 LXR wird benutzt, um den in PKS.AS bereitgestellten Aktivierungssatzzeiger als Basisregister im Status zur Fortsetzung nach Beendigung der Systemfunktion einzutragen:

```
INH
IF MRKVGL (IW1, SFRETT)
THEN PKS.Status.X = PKS.AS
ELSE AST.X = PKS.AS
ENB
```

B2.7.2 LACCU und STACCU organisieren die Übergabe des Zeigers auf einen SKS zwischen den Systemfunktionen ASN und CASI N I oder BASINI
LACCU führt aus:

```
INH
IF MRKVGL (IW1, SFRETT)
THEN PKS.AC = PSKS
ELSE AST.AC = PSKS
ENB
```

und STACCU:

```
INH
IF MRKVGL (IW1, SFRETT)
THEN PSKS = PKS.AC
ELSE PSKS = AST.AC
```

B2.7.3 REFS ermittelt eine neue freie Seite im Arbeitsspeicher und liefert ihre Anfangsadresse in STNFS.

B2.7.4 WTRAND und WTRE sind die Transfermacros zu und vom Hintergrundspeicher für die Segmentverdrängung. WTRAND überträgt mehrere Segmente zum Hintergrund; die Anzahl ist aus ASAL ersichtlich, das nach jedem Segment um 1 vermindert wird,bis 0 erreicht ist.

WTRE überträgt ein Segment vom Hintergrund in den Arbeitsspeicher.
Da beim Zugriff auf den Hintergrundspeicher Verzögerungen entstehen, geht das Betriebssystem an diesen Stellen in einen Wartezustand und kann durch neue Betriebssystemaufrufe überholt werden. Es ist daher darauf zu achten, daß alle Größen, die über diese Stelle hinweg benötigt werden, solange in den Laufzeitkeller des bearbeitenden Prozesses ausgelagert werden.

B2.7.5 Um bei Arbeitsspeichermangel möglichst schnell die Prozesse ausfindig zu machen,die sich in einem längerdauernden Wartezustand befinden und daher ihren Speicherplatz abgeben können, ist ähnlich der Prioritätssprungkette eine zweite Sprungkette über alle Prozesse in der umgekehrten Reihenfolge ihrer Priorität angelegt.
Zu diesem Zwecke steht in der SPCA-Zelle eines Prozesses:
Ein indirekter Sprung über die PV-Zelle, falls der Prozeß seine Segmente nicht abgeben soll,
ein Unterprogrammaufruf an NW, falls der Prozeß länger wartet und daher seine Segmente abgeben darf oder
ein Unterprogrammaufruf an FL, falls der Prozeß eine E/A-Operation ausführt und daher seine Segmente auf keinen Fall ausgelagert werden dürfen.

Durch INSPCA(P) wird in der SPCA-Zelle des durch P angezeigten PKS der indirekte Sprung über die Zelle P—>PKS.PV eingetragen.

Durch PRKZ wird in die Sprungkette über die SPCA-Zellen der PKS eingesprungen:

```
INH
IW1 = NRETT
GOTO *PRIOKE.PV
```

Hat ein Prozeß eine Aussage über seine Segmente zu machen, wird in seiner SPCA-Zelle ein "CALL NW" oder "CALL FL" ausgeführt. Die Ausführung dieser beiden Programme ist (ebenso wie bei EVRUN und WSP, siehe B2.4.5) so zu organisieren, wie wenn sie direkt vor der Stelle im Anwenderprogramm, auf die PKS.LF zeigt, aufgerufen worden wären.
Zu diesem Zweck enthalten diese Programme als erste Anweisung den Aufruf des Macros SPPKS und enden mit dem Aufruf des Macros BRVP.

SPPKS ist identisch mit FAUSF (siehe B2.4.5)
BRVP führt das gleiche aus wie FEND (bzw. NEXTP, siehe B2.1.9), endet aber mit weiterem Durchspringen der Rückwärtskette; falls inzwischen kein Interrupt aufgetreten ist:

```
INH
IF MRKVGL (IW1, SFRETT) THEN fahre fort bei SUWI
Übertrage AST in den laufenden PKS
IW1 = NRETT
GOTO *PKS.PV
```

B2.8 Optimierbare Macros

Die hier aufgeführten Macros sind nur aus dem Grund als Macros formuliert, weil ihr Code optimierbar ist, also evtl. als Assemblerbefehlsfolge dem Betriebssystem angeschlossen werden kann.

CHANGE vertauscht die Werte der beiden Parameter, die BIN FIXED(28) vereinbart sind.

ERS2 hat 3 Parameter vom Typ POINTER; dem ersten Parameter wird der Wert des zweiten und dann dem zweiten der Wert des dritten zugewiesen.

BFERS2 wirkt wie ERS2, jedoch sind die 3 Parameter vom Typ BIN FIXED(15).

XCHP wirkt wie CHANGE; jedoch sind die beiden Parameter vom Typ POINTER.

TRF3 hat als Parameter einen Index(X); es ist auszuführen:

```
OPER = ARTST(X).OP;
OPER1 = ARTST(X).OP1;
MUSTER = ARTST(X).MUST;
```

Da im Strukturbereich ARTST die Elemente OP, OP1 und MUSTER direkt hintereinander liegen, ist der Code optimierbar.

PVGL ist eine Funktion mit 3 Parametern (P,Q,R) vom Typ POINTER; PVGL wirkt wie:

```
RETURN (P = Q | P = R);
```

VGL ist äquivalent PVGL, jedoch sind die 3 Parameter vom Typ BIT(16).

B2.9 Macros für Anweisungsfolgen, die nicht in PL/1 formulierbar sind

MRKVGL vergleicht zwei als Parameter übergebene Marken; sind sie identisch, wird der Wert '1'B, sonst 'O'B geliefert. Dieser Macro wurde schon mehrfach bei der Beschreibung anderer Macros verwendet.

SNUM ist eine Funktion; sie extrahiert aus der als Parameter übergebenen Adresse (Typ POINTER) die Seitennummer (höchstwertige 8 Bits) und gibt sie als BIN FIXED(15)-Wert zurück.

PMQ ermittelt die Differenz der beiden Parameter vom Typ POINTER und gibt sie als BIN FIXED(15)-Wert zurück.

PINCR ist eine Pointerinkrementierung und hat 3 Parameter (P,Q,X), die ersten beiden (P,Q) sind POINTER, der letzte BIN FIXED(15). PINCR weist P den um X erhöhten Wert von Q zu.

Die folgenden vier Macros sind Pointerdekrementierungen:

PKSEMA hat 2 Parameter (P,Q) vom Typ POINTER.
Vorausgesetzt ist, daß Q auf die SN-Zelle eines PKS zeigt.
Der Wert von Q wird um so viel vermindert, daß er auf den Beginn (CA-Zelle) desselben PKS zeigt und P zugewiesen.

PVNCA hat einen Parameter P vom Typ POINTER.
Vorausgesetzt ist, daß P auf die VN-Zelle eines PKS zeigt.
P wird um so viel vermindert, daß er auf die CA-Zelle desselben PKS zeigt.

PML1 vermindert den global bekannten Zeiger PEL um so viel,daß er auf STR.CA zeigt, vorausgesetzt, daß er vorher auf STR.U gezeigt hat.

PMSTR vermindert ebenfalls den Zeiger PEL um so viel, daß er auf STR.CA zeigt, vorausgesetzt, daß er vorher auf STR.E gezeigt hat.

PVAL erlaubt die Benutzung der BIN FIXED(31)-Größe, die als Parameter übergeben wurde, als POINTER.

BRIP wird benutzt zum Aufruf von Betriebssystemprozeduren, deren Einsprungadressen in Betriebssystemlisten vermerkt sind. Der Parameter vom Typ BIT(16) beinhaltet die Einsprungadresse der aufzurufenden Prozedur. Die Rückkehr aus der aufgerufenen Prozedur hat hinter dem Aufruf von BRIP zu erfolgen.

GENCA und GENOP werden in der Initialisierungsphase benutzt, um in BIT(16)-Größen Prozeduraufrufe abzulegen, die später in verschiedenen Betriebssystemlisten eingetragen werden. Sie besitzen zwei Parameter; der erste ist die BIT(16)-Größe und der zweite eine Betriebssystemprozedur und zwar bei GENOP einer der BASED-Eingänge der Prozedur PEARL, bei GENCA eine interne Prozedur.

B3 Anpassung des Systems an Rechner

Wir haben schon gesehen, daß ein Teil der Anpassung des Betriebssystems an einen Rechner durch Codierung der Macros erfolgt.
Die Treiber der Geräte, die mit der Umwelt verkehren, sind ebenfalls bei einer Implementierung des Betriebssystems zu schreiben; dabei kann eine gewisse Anpassungsarbeit bei den Schnittstellen des Betriebssystems zu den Treibern erforderlich sein.
Es ist möglich, daß bei einem ziemlich "intelligenten" Rechner die Hardware einen großen Teil der Arbeit, die wir mit Primärreaktion bezeichnet haben, beim Eintritt einer Meldung übernimmt oder zumindest stark unterstützt; dadurch wäre unter Umständen ein Teil der Primärreaktionsprogramme zu modifizieren. Solche Modifizierungen könnten evtl. ein bißchen weiter führen: Nehmen wir z.B. den Fall, in dem bei einer Unterbrechung der ganze Registersatz (und Befehlszähler) von der Hardware automatisch in einen Keller gespeichert werden. Es wäre in diesem Fall günstig, den Status aus dem PKS eines Prozesses zu eliminieren und ihn durch eine Zelle ZAB zu ersetzen, in der der Hinweis auf den Auslagerungsblock in dem Keller eingetragen wäre. Dadurch würden einige Stellen im Betriebssystem, in denen z.B. PKS.LF angesprochen wird, auch zu modifizieren.

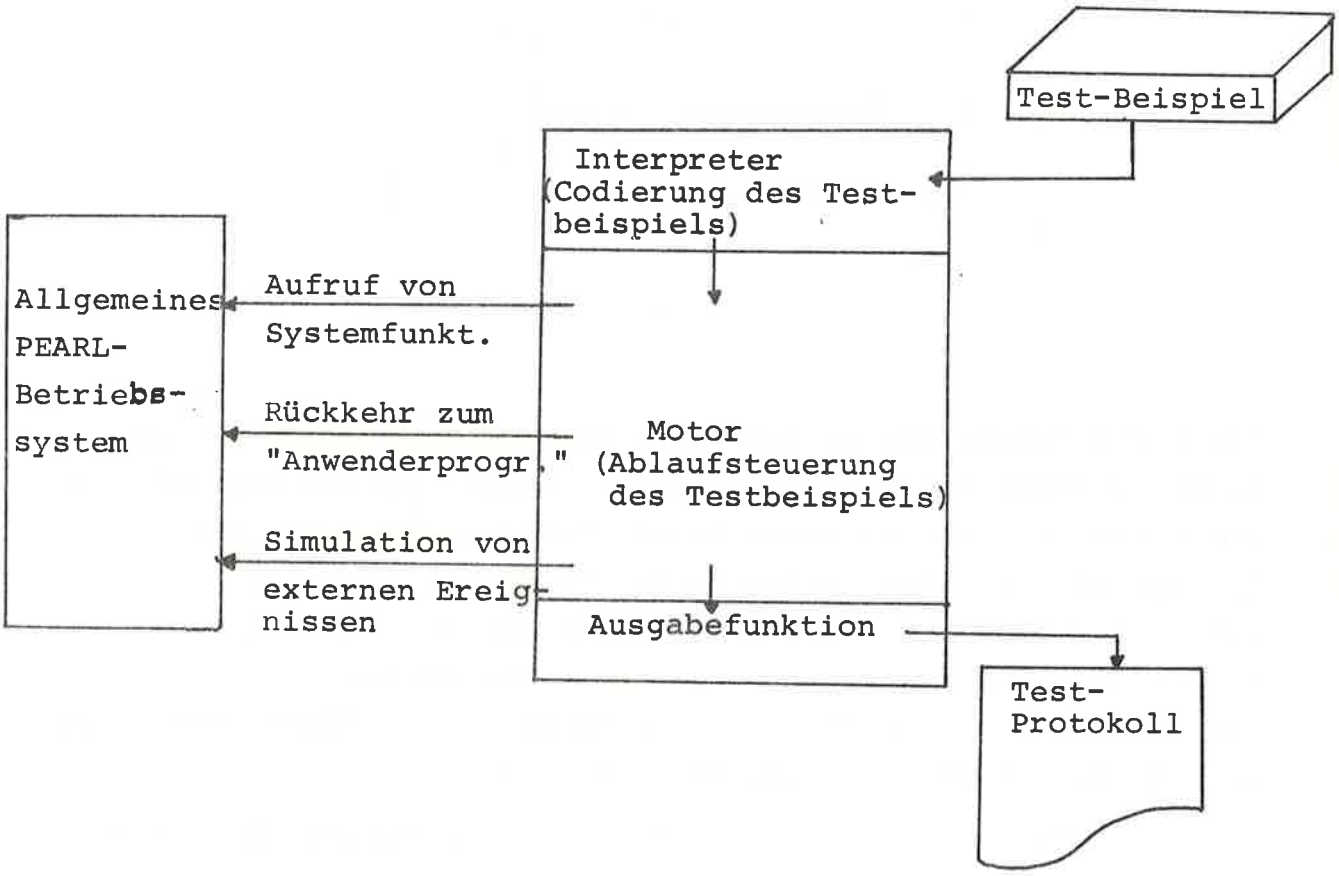
Falls aus organisatorischen Gründen Befehle in Datenstrukturen wie in PKS.CA, EPS.OP usw. nicht möglich sind, können diese durch Bitmuster ersetzt werden; die ursprünglichen Befehle werden nicht mehr ausgeführt, sondern die Bitmuster werden interpretiert.

Da Befehle in Datenstrukturen in unserem Betriebssystem per Macro behandelt sind, braucht der Implementator nur einen neuen Code für diese Macros zu definieren.

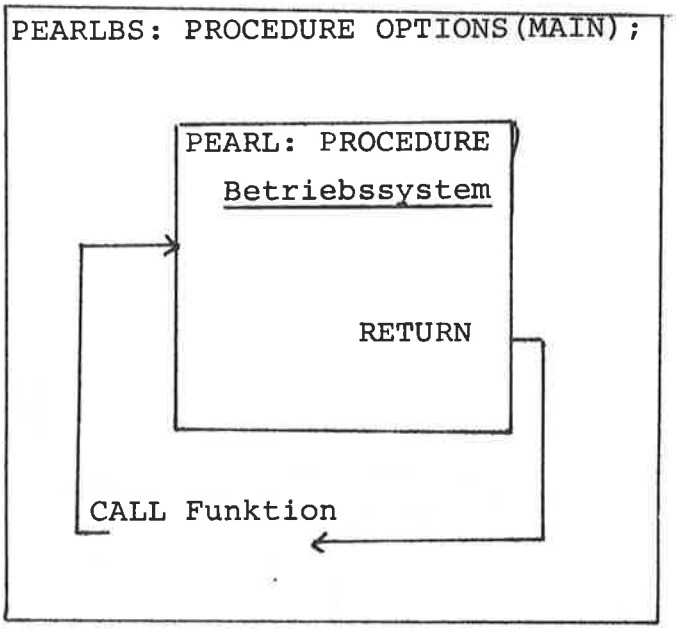
ANHANG C

ÜBERBLICK ÜBER DAS TESTSYSTEM

Das Testsystem hat uns die Möglichkeit gegeben, alle Funktionen des Betriebssystems unter verschiedenen Prozeßzuständen aufzurufen und die Verwaltungslisten zu protokollieren. Da zur Zeit kein PEARL-Compiler vorliegt, haben wir ein geschlossenes Testsystem eingerichtet, das uns erlaubt, freie Programmierung von Testbeispielen sowie Simulation externer Ereignisse und flexible Ausgabe von wichtigen Daten zu handhaben.
Man kann das Testsystem so darstellen:



Das Testsystem ist so an das PEARL-Betriebssystem angepaßt, daß der Zugang vom Testsystem zum Betriebssystem und zurück nur durch Prozeduraufrufe und Rückkehr von Prozeduren erfolgt. Das PEARL-Betriebssystem ist nämlich als Unterprozedur namens PEARL mit vielen Eingängen im Testsystem (MAIN-Prozedur namens PEARLBS) eingebettet.



Sämtliche Daten, die im Betriebssystem benutzt werden, sind in der äußeren Prozedur PEARLBS vereinbart. Betriebssystem und Testsystem können diese Daten direkt ansprechen. Dem Testsystem sind dazu Testdaten zur Verfügung gestellt, die die Testumgebung des Programms bilden und die dem Betriebssystem nur als Parameter übergeben werden dürfen.
Die Macros, die im Betriebssystem benutzt sind (siehe Anhang B), werden durch kleine Prozeduren simuliert.

Selbstverständlich ist das ganze Testsystem abschaltbar, indem die Prozessorweiche TEST auf "NO" gesetzt wird (siehe Anhang B).

Ein Testlauf wird folgendermaßen ausgeführt:

1. Initialisierung der Daten, die im Betriebssystem benutzt sind; Initialisierung der Testdaten.
2. Eingabe des Testbeispiels, Codierung durch den Interpreter.
3. Anlauf des "Motors" des Testsystems; die Testanweisungen werden ausgeführt solange der Testendebefehl nicht erreicht ist; nach jeder Anweisung wird geprüft, ob besondere Protokollierung erwünscht ist und ggf. ausgegeben.

C1 Programmierung von Testbeispielen; der Interpreter

Ein Testbeispiel ist eine Menge von Testanweisungen, die zwischen den beiden Interpreter-Kommandos:

ABW am Anfang und
END am Ende liegen.

Beide Interpreter-Kommandos müssen auf der Spalte 1 anfangen.

Das allgemeine Format einer Testanweisung ist

Prozeß ␣ Anweisung [(Parameter)] ␣ [Kommentar | Ausgabefunktion [(Parameter)]] ␣ ...

↑ Spalte1 ↑ Spalte5

␣ steht für das Leerzeichen.

Prozeß ist der Name des Prozesses, unter dessen Regie die Anweisung auszuführen ist. Dieser Name besteht aus drei Buchstaben, ab der Spalte1 abgelegt und kann nur einer der 6 Namen sein, die dem Interpreter bekannt sind und zwar:

HPT: Name des Hauptprozesses
TVN: TASK-Variable mit normalem PKS
TVK: TASK-Variable mit kurzem PKS

TKN: TASK-Konstante mit normalem PKS

TKK: TASK-Konstante mit kurzem PKS

HWT: Name des PRIOKE-Prozesses, der im Testsystem Befehle zur Simulation von externen Ereignissen ausführen darf.

Anweisung ist der Name der Systemfunktion oder der Hardware-Simulationsanweisung, die unter Regie von Prozeß auszuführen ist. Dieser Name besteht aus 2 bis 6 Buchstaben und ist ab der Spalte5 abgelegt. Wenn Anweisung eine Systemfunktion ist, ist ihr Name anzugeben (z.B. ACTKT, LEAVBL, usw.), mit der Ausnahme von PEND, die LPEND genannt werden muß. Wenn Anweisung eine Hardware-Simulationsanweisung ist, muß sie wie folgt genannt werden:

DAUER: Erhöhen des Dauer-Zählers

KGS: Simulation einer Geräterückmeldung

ZEIT: Erhöhen des Zeit-Zählers.

Meldungen, die als Interrupt bzw. Signal spezifiziert sind, werden selbstverständlich durch TRIGG und INDUCE simuliert.

Wenn beim Aufruf der Anweisungs-Funktion Parameter anzugeben sind, sind diese zwischen Klammern und durch Kommas getrennt nach Anweisung anzugeben.

Parameter ist die Liste (evtl. aus einem Element) der Anweisungs-Parameter. Ein Name besteht aus 1 bis 3 Buchstaben und darf entweder eine Zahl oder nur einer der dem Interpreter bekannten Namen sein und zwar:

EI : Meldung als Interrupt spezifiziert

ES : Meldung als Signal spezifiziert

ESI : Meldung als Interrupt und Signal spezifiziert

O : Leere EXCEPT-Liste

NIL : Zeiger = NULL

S1
:
S5 : SEMA-Variable

B1
:
B5 : BOLT-Variable

HPT
TVN
TVK
TKN
TKK
: Prozesse (siehe oben)

F1
:
F5
: FILE-Variable

D1
D2
: DKS

U1
:
U3
: UKS

EX1 : EXCEPT-Liste (TVN)
EX2 : " (TVK)
EX3 : " (TKN)
EX4 : " (TKK)
EX5 : " (TVN,TVK,TKN,TKK)

MB1
:
MB4
: Markenbeschreibungen

TWE : Wert für Prozeßdaten-Ein/Ausgabe

P1
:
P5
: Parametersätze PARCO für OPEN und CREATE

Q1
:
Q5
: Parametersätze PARS für Übertragungsbefehle

Die Anzahl der Parameter ist spezifisch für die aufgerufene Funktion; manche, wie PKSINI, REQSL usw. können aber eine unspezifizierte Anzahl von Parametern bearbeiten.
Bei DAUER und ZEIT ist eine ganze Zahl als Parameter anzugeben. Parameter für KGS ist U1 oder U2 oder U3.

Kommentar steht für eine beliebige in /* und */ eingeschlossene Zeichenkette. Kommentare werden überlesen.

Ausgabefunktion [(Parameter)] gibt an, welche Datenstrukturen nach Ausführung der Anweisung auszugeben sind. Diese Ausgabefunktionen sind folgende:

Z1PKS(Prozeß): Veränderungen im Prozeß-zugeordneten PKS

ZAPKS: Stand aller PKS

ZVPKS: Veränderungen in allen PKS

Z1AS(Prozeß): Veränderungen im aktuellen AS des Prozesses

ZAAS: Stand der aktuellen AS aller Prozesse

ZVAS: Veränderungen in den aktuellen AS aller Prozesse

ZEPS1(Prozeß): Veränderungen in allen EPS des Prozesses

ZEPSA: Stand aller EPS von allen Prozessen

ZEPSV: Veränderungen in allen EPS von allen Prozessen

ZVSKS: Veränderungen in allen SKS und zugehörigen ADKS

Z1FKS(FILE-Variable): Veränderungen in dem FILE-Variable zugeordneten FKS

ZAFKS: Stand aller FKS

ZVFKS: Veränderungen in allen FKS

Z1DKS(Datei): Veränderung im Datei-zugeordneten DKS

ZADKS: Stand aller DKS

ZVDKS: Veränderungen in allen DKS

Die Ausgabefunktionen werden im Abschnitt C3 dieses Anhangs beschrieben.

Beispiele von Testanweisungen:

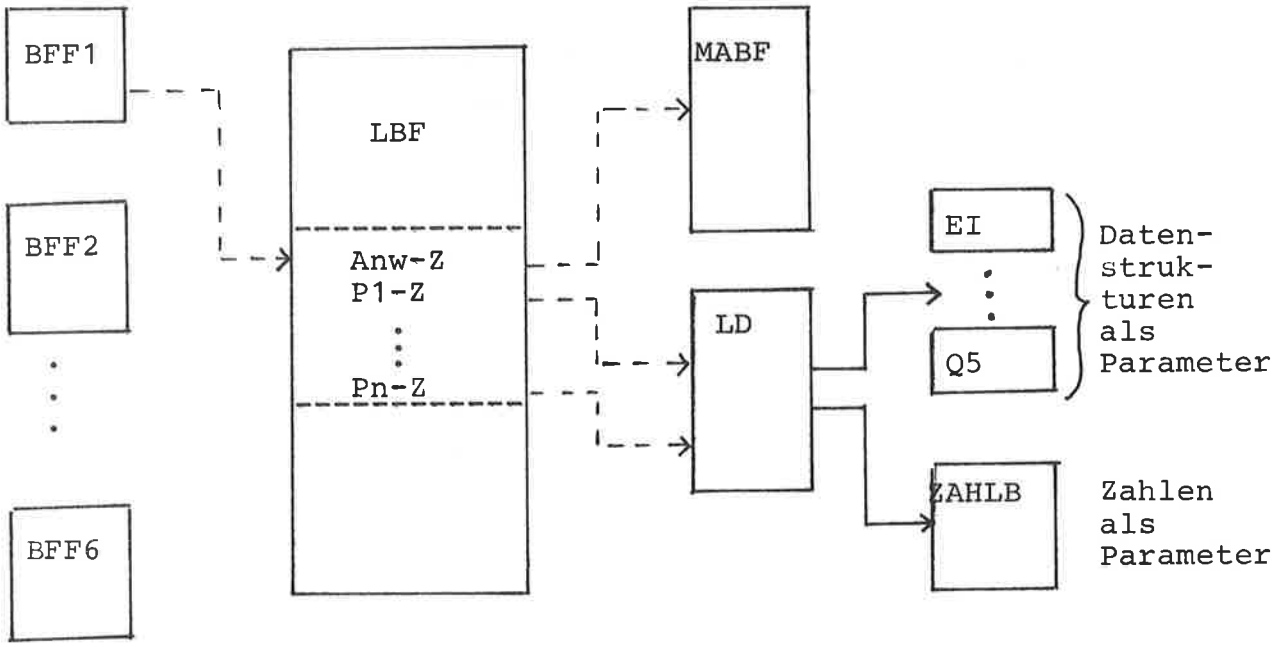
```
HPT PKSINI(1,3)  /* ANLEGEN VON TVN, TKN */ Z1AS(HPT)
TVN ACTT(TKN,2,S3,1) /* ACTIVATE TKN PRIORITY 2 REL USING S3 */
TKK SUSPTA(HPT,Ø)  /* SUSPEND HPT */ ZVPKS
HWT DAUER(2OO)  ZEPSA  ZVPKS
TKN BLKINI
```

Der Interpreter:

Der Interpreter liest das Test-Beispiel ein und übergibt dem "Motor" des Testsystems die umcodierten Testanweisungen in Datenbereichen. Jedem Prozeß des Testsystems ist ein Zahlenbereich zugeordnet, der den Code des Prozesses darstellt:

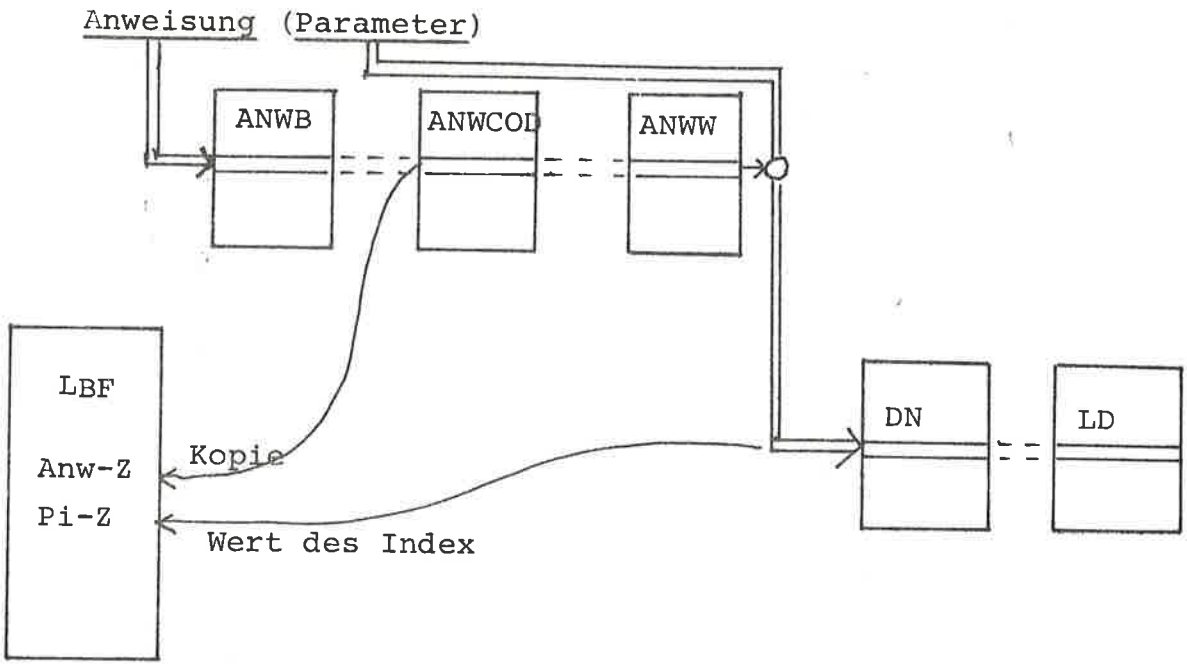
BFF1 für TVN
BFF2 für TVK
BFF3 für TKN
BFF4 für HPT
BFF5 für HWT
BFF6 für TKK.

Zahlen aus diesen Bereichen dienen als Index in einem anderen Zahlenbereich LBF, in dem die Testanweisungen und ihre Parameter verschlüsselt sind. Eine Anweisung evtl. mit Parametern ist im LBF als eine Folge von Zahlen abgebildet.
Die Zahl (im folgenden Bild als Anw-Z dargestellt), die eine Anweisung identifiziert, dient als Index im Markenbereich MABF, der im Abschnitt C2 dieses Anhangs beschrieben wird.
Die Zahl (im folgenden Bild als Pi-Z dargestellt), die einen Parameter identifiziert, dient als Index im Zeigerbereich LD, in dem die Adressen der Parameter abgelegt sind. Wenn der Parameter eine ganze Zahl ist, wird der Wert im Bereich ZAHLB abgelegt.



Der Interpreter liest die zu codierende Testanweisung ein. Wenn sie das Interpreter-Kommando ABW ist, werden die Datenstrukturen, die zum Test dienen, initialisiert. Wenn sie eine normale Testanweisung ist, wird anhand der Prozeß-Angabe der Bereich BFFi bestimmt, der dem Prozeß zugeordnet ist. In die erste freie Zelle des Bereiches BFFi wird die Indexnummer der ersten freien Zelle des Bereiches LBF eingetragen.

Danach wird die Anweisung erkannt und deren Code in der ersten freien Zelle von LBF abgelegt, indem der Character-Bereich ANWB solange durchgesucht wird, bis die Anweisungs-Zeichenkette gefunden ist und der Code dem parallelen Bereich ANWCOD entnommen wird. Parallel zu den Bereichen ANWB und ANWCOD ist noch der Bereich ANWW vorgesehen, der angibt, wieviele Parameter von welchen Arten der Anweisung zugeordnet sind. Anhand dieser Information werden die Parameter erkannt und deren Code jeweils in der ersten freien Zelle von LBF abgelegt. Wenn der Parameter ein Name ist, wird der Character-Bereich DN solange durchsucht, bis die Parameter-Zeichenkette gefunden ist; der Wert des Index in DN dient als Code für den Parameter. Wenn der Parameter eine Zahl ist, wird im Zahlenbereich ZAHLB gesucht, ob die Zahl schon vorhanden ist; falls nein, wird sie in ZAHLB neu eingetragen und ihre Adresse in LD abgelegt; ihr Index in LD gilt als Code für den Parameter.



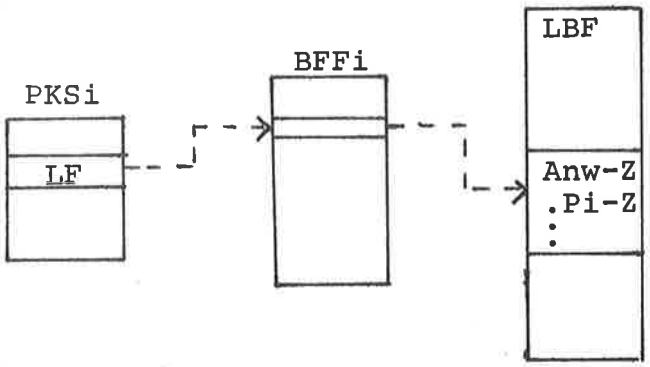
Nachdem die Anweisung mit sämtlichen Parametern umcodiert wurde, wird gesucht, ob Ausgabefunktion(Parameter) vorhanden ist. Falls ja, wird dasselbe Verfahren zur Umcodierung angewendet. Wenn keine Ausgabefunktion mehr vorhanden ist, werden die nächsten Testanweisungen aufgenommen, solange bis das Interpreter-Kommando END gelesen ist. Wenn END erreicht ist, wird der "Motor" des Testsystems zum Laufen gebracht.

C2 Ausführung von Testbeispielen; der Motor

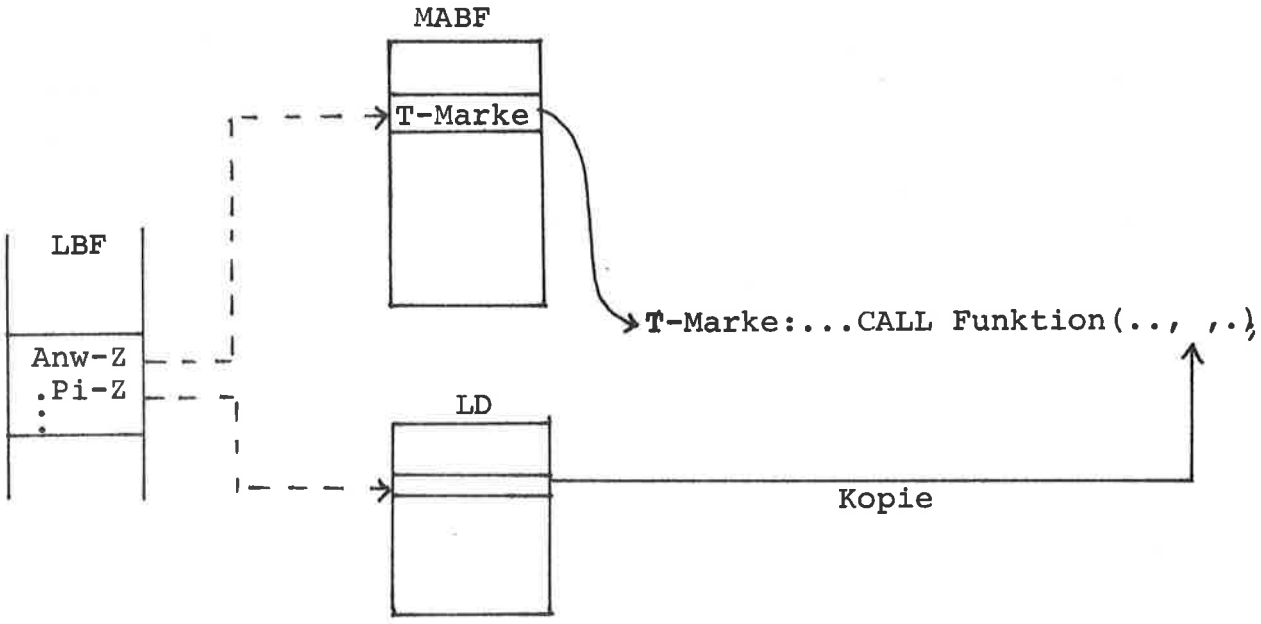
Während der Anlaufphase des Motors wird der Hauptprozeß HPT aktiviert. Ihm ist die Prioritätszahl 18000 zugeordnet (also Systempriorität). Am Testanfang ist er der einzige lauffähige Prozeß im Testsystem. Unter seiner Regie werden die ersten Testanweisungen ausgeführt.

Im Motor des Testsystems wird die Prioritätskette solange durchgesucht, bis der PKS eines lauffähigen Prozesses gefunden ist. In dessen PKS.AC ist eine Nummer eingetragen, die bestimmt, welcher Bereich BFFi ihm zugeordnet ist. Dieser Bereich wird betrachtet; in der PKS.LF-Zelle, die für Testzwecke als Binary-Fixed-Größe behandelt wird, ist die Anwei-

sungsnummer eingetragen, die als Index in BFFi dient. Aus BFFi wird also der Index entnommen, der in LBF die Stelle bestimmt, ab der die Anweisung und ihre Parameter codiert sind.



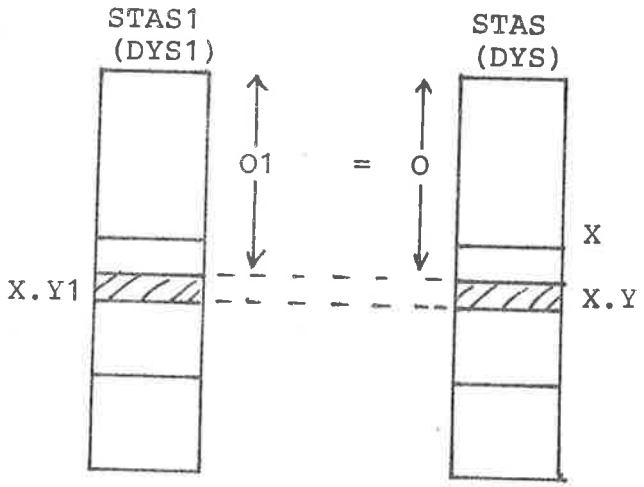
Der Anweisungscode Anw-Z bezeichnet die Stelle im Markenbereich MABF, über die gesprungen wird. Am Sprungziel T-Marke werden die Parameter aus LD(Pi-Z) übernommen.



Die Funktion im PEARL-Betriebssystem wird aufgerufen. Nach der Ausführung wird hinter die Aufrufstelle zurückgesprungen. Da steht der Befehl "GOTO SYFEND", der in den Motor des Testsystems zurückführt. Dort wird der Befehlszähler PKS.LF um eins erhöht und getestet, ob jetzt eine Ausgabefunktion aufzurufen ist. Falls ja, wird sie aufgerufen. Wenn keine Ausgabefunktion mehr aufzurufen ist, wird der wichtigste lauffähige Prozeß aufgenommen.

C3 Interpretierte Ausgabe der Datenstrukturen

Die Datenstrukturen PKS, AS, EPS, FKS und DKS werden in AREA STAS und die SKS in AREA DYS abgelegt. In AREA STAS1 und AREA DYS1 wird nach der Ausgabe eines Strukturelements dessen Wert bei gleichem OFFSET übertragen. Damit ist es möglich, den Wert eines Strukturelements X.Y zusammen mit dem letzten Ausgabewert X.Y1 zu drucken. Die Ausgabe kann unterdrückt werden, wenn X.Y1 = X.Y gilt.



In welcher Form die Exemplare eines Strukturtyps X ausgegeben werden, ist in einem Beschreibungsbereich BX festgelegt. Für jedes Element X.Y, das bei der Ausgabe berücksichtigt werden soll, existiert in BX ein Element der Form

```
1  B  Elementbeschreibung
2  N  CHAR(8)     Name des Elements
2  T  BIN FIXED   Datentyp des Elements
2  L  BIN FIXED   Inkrement zum nächsten Strukturelement,
                  das auszugeben ist.
```

Die Preprocessor-Prozedur $B erzeugt eine Anweisungsfolge, in der BX mit der benötigten Länge vereinbart wird und durch Aufruf der Prozedur ES jedes Element aus BX initialisiert wird.

Aufruf von $B:

$B(X, Y.Z...Y.Z#)

mit

X Strukturname

Y Elementname

Z Datentyp (= 1 Zeiger, = 2 Zahl, = 3 Binärgröße)

Struktur- und Elementnamen müssen mit der Vereinbarung übereinstimmen. Hinter dem Komma muß das erste Strukturelement genannt werden, danach können in der Reihenfolge wie sie ausgegeben werden sollen, andere Strukturelemente angegeben werden. Zwischen zwei Angaben Y.Z können n≥0 Leerzeichen stehen.

Beispiel:

```
In DCL 1 X BASED(PX), 2(A BIT(16), B PTR, C BIN FIXED(15),
                                   D BIN FIXED(31));
```

sei die Struktur X vereinbart. Von Exemplaren dieser Struktur sollen die Elemente D und B in dieser Reihenfolge ausgegeben werden. Dazu ist

```
$B(X,A.0 D.2 B.1 #)
```

zu programmieren. Hierbei ist ausgenutzt, daß die Ausgabe eines Strukturelements X.Y durch Angabe von Y.0 unterdrückt werden kann.

Verständliche Ausgabe von Adressen

Im Testsystem ist AREA ARB reserviert, um eine Liste von Adreßbeschreibungen aufzunehmen.

```
1  C  Adreßbeschreibung
2  N  CHAR(8)  frei gewählte Bezeichnung
2  P  POINTER
```

Bei Ausgabe eines als Zeiger vereinbarten Strukturelements X.Y (in $B(X,...Y.1...)) wird zunächst die Liste der Adreßbeschreibungen durchsucht und, falls für ein C gilt X.Y = C.P, wird als 'Wert' von X.Y die Zeichenkette C.N ausgedruckt, anderenfalls wird der Wert von X.Y als Zahl ausgegeben.
Für jede Adresse, die in verständlicher Form ausgegeben werden soll, ist

```
CALL ED (Name, Adresse);
```

zu programmieren, die Prozedur ED erweitert die Liste der Adreßbeschreibungen in ARB um ein Element und initialisiert es mit C.N = Name, C.P = Adresse.

Beispiel: Das n-te Exemplar der oben vereinbarten Struktur X wird allokiert, die Adresse von X.B sei im Wartebereich einiger Zeigervariablen, die auszugeben sind. Dann wird durch CALL ED (BVONXN, ADDR(PX->X.B)); eine Adreßbeschreibung erzeugt und bei Ausgabe einer Zeigervariablen P mit P = ADDR(Xn.B) wird die Zeichenkette BVONXN gedruckt.

Protokoll eines Testbeispiels

Die Gesamtausgabe hat folgende Form:
Die Anweisungen des Testbeispiels werden in der Zeichenfolge ihrer Ausführung gelistet. Enthält eine Zeile eine oder mehrere Ausgabefunktionen, folgen in den nächsten Zeilen die Ergebnisse dieser Ausgabefunktionen in der Reihenfolge ihrer Niederschrift in dem Format

Strukturname.Elementname alter Wert neuer Wert .

Dabei ist 'Strukturname' ein Bezeichner für ein Exemplar einer Struktur und 'Elementname' der Bezeichner, mit dem das Element vereinbart und nach Angabe in der Prozedur $B in einer Elementbeschreibung B.N abgelegt wurde.

Unter 'alter Wert' erscheinen die in STAS1(DYS1) und unter 'neuer Wert' die in STAS(DYS) abgelegten Daten bzw. die Namen oder Zahldarstellungen von Adressen.

ANHANG D

INDEX

DAT 2 - 6,2 - 45
DATE 2 - 43
DBE 2 - 44
DCLSN 3 - 27,3 - 49,3 - 60,3 - 61,3 - 73,A - 18
DCLSP 3 - 27,3 - 49,3 - 60,3 - 61,3 - 73,A - 18
DELETE 2 - 37,A - 15
DFA 2 - 44
DFE 2 - 44
DISB 1 - 48,A - 3
DKS 2 - 21,2 - 24,2 - 25,2 - 30,2 - 33,2 - 34,
2 - 36,2 - 37,2 - 43,2 - 45,3 - 14
DRSPAD B - 14
DSBL 1 - 36,1 - 48
DSP 2 - 13,2 - 44
DVS 2 - 19,2 - 24,2 - 28,2 - 29,2 - 30,2 - 33,
2 - 36,2 - 38

EALANG 2 - 14,2 - 15
EDU 1 - 11,A - 1
EIND 1 - 25
EINER 1 - 26
EINK 1 - 84
EINØRD 1 - 60,1 - 70
EINU 1 - 25
EINUK 2 - 13,2 - 14
ENAB 1 - 48,A - 3
ENB B - 9
ENTBL 1 - 85,1 - 86,A - 10
ENT1 1 - 85,1 - 86,A - 10
EPD 2 - 6,2 - 16
EPS 1 - 14,1 - 27,1 - 39,1 - 64,1 - 75,1 - 76,
3 - 12,3 - 13,B - 8
EPSFZ 3 - 12
ER 1 - 22,1 - 29,1 - 33,1 - 36,1 - 40,1 - 41,
1 - 43,1 - 47,1 - 48,1 - 66
EREND 1 - 41
ERS2 B - 20
ESY 2 - 13,2 - 14

NEXTP	B - 7,B - 13
NIMM	3 - 53,3 - 56,3 - 59
NMVGL	2 - 29,2 - 30
NØPD	1 - 37
NØPE	1 - 38,1 - 42
NØPG	1 - 37
NØPU	1 - 38
NØTIER	1 - 36,1 - 42,1 - 48
NSSU	3 - 53,3 - 54,3 - 58
NUK	2 - 14,2 - 16
NW	3 - 51,3 - 59,B - 19

ØFFSB	3 - 47,3 - 48,3 - 59
ØPAUSD	B - 8
ØPAUSF	1 - 39,1 - 41,B - 8
ØPDØ	2 - 34,2 - 35
ØPEN	2 - 33,A - 14
ØPLIST	2 - 35,A - 13
ØPNØP	1 - 72
ØPTEST	2 - 33,2 - 35

PAL	B - 4
PARCØ	2 - 26,A - 12
PARMIN	2 - 28,2 - 30,2 - 33
PARPRØ	3 - 30,A - 21,B - 16
PARS	2 - 39,2 - 42,A - 16
PDIN	2 - 13,A - 11
PDØUT	2 - 13,A - 11
PEND	3 - 41,3 - 42,A - 20
PINCR	B - 21
PKS	1 - 44,1 - 49,1 - 51,1 - 55 BIS 1 - 79, 2 - 43,3 - 14,3 - 17,3 - 54 BIS 3 - 58, 3 - 64,3 - 65,3 - 76,3 - 77,B - 3,B - 12, B - 14,B - 18,B - 23
PKSEMA	B - 21
PKSINI	1 - 62,3 - 34,A - 5,A - 19

PML1	B - 21
PMQ	B - 21
PMSTR	B - 21
PØSBE	2 - 42
PRAUS	3 - 41,3 - 42,A - 20
PREND	1 - 74,3 - 62,A - 18
PRET	3 - 41,3 - 42,A - 20
PRETER	1 - 72,1 - 73,1 - 75
PREVLT	1 - 75,A - 8
PREVTA	1 - 75,A - 8
PRFØRT	B - 9
PRIMD	1 - 35
PRIMU	1 - 35
PRIØKE	1 - 55,1 - 76,B - 6,B - 12,B - 18
PRKEND	B - 12
PRKZ	B - 18
PSIG	1 - 41,1 - 43
PSKEND	B - 6
PUF	1 - 36,1 - 41,1 - 42
PUTR	2 - 39,2 - 41,3 - 76,A - 16
PVAL	B - 21
PVGL	B - 20
PVNCA	B - 21

RADKS	3 - 14
REBØLT	3 - 14
REFS	B - 17
REKPKS	3 - 14
RELFKS	2 - 38
RELSL	1 - 83,1 - 84,A - 9
REL1	1 - 83,1 - 84,A - 9
RENPKS	3 - 14
REQSL	1 - 83,1 - 84,A - 9
REQ1	1 - 83,A - 9
RESBL	1 - 85,1 - 86,A - 10
RESDKS	2 - 29,3 - 14
RESEMA	3 - 14
RESFKS	3 - 14
RESP	1 - 30,1 - 31,A - 4,B - 14
RESPND	1 - 30,1 - 45,A - 3

RESPRØ	3 - 30,A - 21,B - 16
RESP1	1 - 29,1 - 31,A - 3,B - 14
RESRSP	3 - 14
RES1	1 - 85,A - 10
RHSKS	3 - 14
RRETT	1 - 47,B - 7
RSPAUS	1 - 30,1 - 45,A - 3
RSPB	1 - 31,1 - 43,1 - 44,1 - 47,3 - 39
RSPEL	1 - 43,1 - 45,1 - 46,3 - 14,B - 3,B - 14
RSUMLT	1 - 12,1 - 28,A - 2
RSUMTA	1 - 12,1 - 28,A - 2
RUØAN	B - 16
RUPAN	B - 16

SBLEND	3 - 40,3 - 41,3 - 42,A - 19
SCAKP	1 - 63,1 - 66
SCANP	1 - 63,1 - 66
SCHANF	1 - 11,1 - 14,A - 1
SCHMT	1 - 12,1 - 27,A - 2
SCHØT	1 - 12,1 - 27,A - 2
SCØNT	1 - 68,1 - 69
SCPREV	1 - 75
SCSUSP	1 - 67,1 - 68
SCTERM	1 - 72
SCTPR	1 - 68,1 - 70
SEK	1 - 55
SEKE	1 - 38,1 - 41
SEKEND	1 - 41,1 - 42
SEKG	1 - 37,2 - 6
SEKRD	1 - 37,1 - 39
SEKRE	1 - 37,1 - 40,1 - 42
SEKRU	1 - 38,1 - 39
SEMA	1 - 80,3 - 14
SEMEND	1 - 82
SEMINI	1 - 80,3 - 34,3 - 35,A - 9,A - 19
SENR	2 - 39,2 - 41,3 - 76,A - 16
SETZEP	B - 12
SFFØRT	B - 10

SG	1 - 29,1 - 43,1 - 48
SGNRBL	3 - 46,3 - 47,3 - 55
SKS	3 - 15,3 - 44 BIS 3 - 47,3 - 49,3 - 52,
	3 - 54 BIS 3 - 56,3 - 59,3 - 62 BIS 3 - 65,
	3 - 72,3 - 76,3 - 77
SNUM	B - 21
SPPKS	B - 19
SRAF	B - 11
STACCU	B - 17
STARB	2 - 43
STPEPS	3 - 13
STPZ	B - 12
STRETT	1 - 33,B - 9
SUSPEN	1 - 68
SUSPLT	1 - 67,1 - 68,A - 6
SUSPTA	1 - 67,1 - 68,A - 6
SUWI	B - 5,B - 6,B - 7,B - 9,B - 10,B - 12,
	B - 19
SWAF	3 - 52,3 - 58

TAKR	2 - 39,2 - 41,3 - 76,A - 16
TERMIN	1 - 72,1 - 73
TERMLT	1 - 72,A - 7
TERMPR	1 - 72,1 - 73,1 - 74,3 - 62
TERMTA	1 - 72,A - 7
TESTA	1 - 27,1 - 64
TRAUS	3 - 55
TRFBEG	3 - 76
TRFEND	3 - 76,3 - 77
TRF3	B - 20
TRIGG	1 - 47,A - 3
TSYSF	B - 10

UEBERS	B - 6
UEFHL	2 - 13,2 - 41
UKK	2 - 14,2 - 15
UKS	1 - 33,1 - 37,2 - 3 BIS 2 - 17,2 - 24,
	2 - 44,2 - 45
UNLØCK	2 - 38, A - 15