RECHENZENTRUM
DER TU MÜNCHEN

PS 440

SYSTEM TR 440

PS 440

Druck:
TELEFUNKEN COMPUTER GmbH
775 Konstanz
Max-Stromeyer-Str. 116

Bestell-Nr.: N31. ZZ. 10
Ausgabe: 0572

I. TEIL

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bericht 7002

von

G. Goos, K. Lagally, G. Sapper

I. TEIL
(Bericht 7002)

I N H A L T

Seite

1. EINFÜHRUNG

PS440 ist eine maschinennahe Programmiersprache, die für Zwecke der Implementierung von Betriebssystemen am TR440 entwickelt wurde. Ähnlich wie die Programmiersprachen PL360 [3], BCPL [2] oder SAL [1] soll PS440 dem Systemprogrammierer erlauben, sich der Hilfsmittel höherer Programmiersprachen zu bedienen. Die Vorteile höherer Programmiersprachen wie größere Übersichtlichkeit, verminderte Fehleranfälligkeit, höherer Dokumentationswert des Programms und größere Bequemlichkeit beim Programmieren werden damit auch beim Schreiben von Systemen nutzbar gemacht. Zusätzlich wird die in weiten Teilen von Systemprogrammen vorhandene Maschinenunabhängigkeit der Algorithmen freigelegt.

Es erscheint zweckmäßig, in einer solchen Programmiersprache allen Möglichkeiten des Maschinencodes spezielle Sprachkonstruktionen zuzuordnen. Dies erschwert das Erlernen der Sprache so sehr, daß letztlich doch wieder der Assemblercode benutzt würde. Außerdem würde die Implementierung so aufwendig, daß die für das Projekt gegebenen Termine nicht eingehalten werden könnten. Leider konnte der bei PL360 beschrittene Weg, die in der Sprache nicht direkt nutzbaren Maschinenfunktionen in Form von Prozeduraufrufen einzuführen, nicht begangen werden. Das Verfahren ist zu fehleranfällig wegen der Doppelcode-Befehle des TR440 und der zu hohen Anzahl verschiedenartiger Adreßteile. Stattdessen wird an verschiedenen Stellen das Einschieben von Informationseinheiten der Assemblersprache TAS gestattet. Dieses Vorgehen verlangt, daß der Übersetzer für PS440 einen Übersetzer für TAS als Bestandteil enthält. Dem wird dadurch Rechnung getragen, daß der Übersetzer PS440 in TAS übersetzt, wobei bereits in TAS geschriebene Teile unverändert übernommen werden. Der TAS-Übersetzer wird dann als zweiter Lauf der Implementierung verwendet und übernimmt die Umsetzung in den Montagecode des TR440. Leider impliziert dieses Verfahren, daß die TAS-Konventionen über Segmente als Gültigkeitsbereiche von Bezeichnungen übernommen werden müssen, wenn man in PS440 definierte Variablen auch in TAS-Einschüben verwenden will.

Der Auswahl der zulässigen Sprachkonstruktionen liegt die Vorstellung zugrunde, daß die Anweisungen eines PS440-Programms die auszuführenden Algorithmen in maschinenunabhängiger Form beschreiben sollen [1]. Hingegen sind bei den Deklarationen maschinenabhängige Konstruktionen möglich, wenn auch nicht erwünscht. Dem Programmierer wird dabei die Möglichkeit eröffnet, die Unterbringung seiner Daten im Speicher genau zu kontrollieren und die Effizienz seiner Programme, soweit sie von der Speicherverteilung abhängt, selbständig zu bestimmen.

---

1) Soweit das durch die Programmiersprache beeinflußbar ist.

Dies erscheint vor allem deshalb notwendig, weil bei der Konstruktion von Betriebssystemen, etwa im Gegensatz zum Übersetzerbau, die Art der Datenspeicherung die Ausnutzung der Arbeits- und Hintergrundsspeicher und damit die Leistung des Gesamtsystems erheblich beeinflußt, und daher dem BS-Konstrukteur, nicht aber dem Sprach-Designer vorbehalten bleiben muß. Zum zweiten ist der BS-Konstrukteur, namentlich im Zusammenhang mit EA-Transporten, oft an Hardware-Bedingungen gebunden, die in der Sprache formulierbar sein müssen, wenn Programme lesbar sein sollen.

Die ideenmäßige Trennung von Anweisungen und Deklarationen könnte sich im Programm widerspiegeln in Form einer Aufteilung Deklarationsteil - Anweisungsteil. Wegen des linearen - nicht geschachtelten - Aufbaus von Programmen aus einzelnen Segmenten ist es jedoch zweckmäßig, diese Aufteilung zu unterlassen. Sie äußert sich nur in der Forderung, daß alle Variablen und Prozeduren vor ihrer ersten Verwendung zu vereinbaren sind.

Die Speicherverteilung ist statisch: Jeder vereinbarten Variablen entspricht ein fester Speicherplatz. Durch Äquivalenzvereinbarungen ist es zwar möglich, auch auf dynamisch veränderlichen Plätzen unterzubringende Größen mit Bezeichnungen zu belegen; jedoch ist dann das Anschreiben der Deklarationen nicht so bequem wie in höheren Programmiersprachen; außerdem müssen die Anweisungen, etwa zur Organisation eines Kellers, explizit ins Programm geschrieben werden. Diese namentlich beim Übersetzerbau unbefriedigende Lösung hat zwei Gründe: Einerseits gehört die Kontrolle des dynamisch verwendeten Speichers zu den Aufgaben des Betriebssystems und kann nicht implizit durch die Sprache erfolgen. Andererseits sind die im Zusammenhang mit mehreren asynchron ablaufenden Prozessen zweckmäßigen Speicherstrukturen noch nicht so genau bekannt, daß die Festlegung eines bestimmten Schemas möglich wäre. Mit Hilfe der geplanten Makro-Vereinbarungen und geeigneter Programmierkonventionen sollte sich jedoch auch im Rahmen der gegebenen Sprache eine akzeptable Lösung finden lassen.

2. GRUNDBEGRIFFE

2.1. Programmaufbau

Ein PS440-Programm ist eine lineare Folge von Grundsymbolen (3.1.1). Die Worte "aufeinanderfolgen", "vorangehen" usw. beziehen sich stets auf diese lineare Anordnung.

Aufeinanderfolgende Grundsymbole werden zusammengefaßt zu Anweisungen. Anweisungen sind entweder "zusammengesetzt" (Prozedurvereinbarungen, Blöcke, bedingte Anweisungen, Laufanweisungen, Fallunterscheidungen, Prozeduraufrufe) oder "einfach". Zusammengesetzte Anweisungen enthalten selbst Anweisungen als Bestandteile. Weiterhin sind Anweisungen entweder (dynamisch) "ausführbar" (eigentliche Anweisungen, siehe 6) oder "nicht-ausführbar" (Deklaration, siehe 5). Eine Sonderstellung bezüglich dieser Unterscheidung nehmen Vereinbarungen mit dynamischer Vorbesetzung ein. Sie gehören zu den nicht-ausführbaren Deklarationen, enthalten jedoch eine ausführbare Wertzuweisung.

Aufeinanderfolgende Anweisungen werden zusammengefaßt zu Segmenten. Segmente definieren den Gültigkeitsbereich von Bezeichnungen und neuen Schlüsselworten (2.6). Segmente können nicht rekursiv geschachtelt werden.

2.2. Lauffähige Programme

Ein PS440-Programm wird zunächst vom PS440-Übersetzer in TAS übersetzt, dann vom TAS-Übersetzer in Montagecode umgesetzt und schließlich vom Montierer, eventuell unter Einschluß anderer Programme, in eine lauffähige Fassung gebracht. Die Zusatzprogramme können in PS440, TAS oder einer anderen Programmiersprache geschrieben sein. Für Zwecke dieser Beschreibung wird außer im Zusammenhang mit Eingangs-Spezifikationen und Extern-Vereinbarungen (siehe 5) angenommen, daß es keine Zusatzprogramme gibt. Im genannten Zusammenhang wird unterstellt, daß die Zusatzprogramme in PS440 geschrieben sind. Soll das lauffähige Programm als Operator verwendet werden, so müssen Startadresse, Alarmadresse, Indexbasisregister- und Unterprogrammzähler-Vorbesetzung mitgeteilt werden. Diese Angaben sind durch die entsprechenden Pseudo-Befehle in TAS-Einschüben (2.9) zu machen.

Ein übersetztes oder montiertes PS440-Programm ist durch einen "Programmnamen" gekennzeichnet. Er wird implizit definiert durch die Segmentbezeichnung (4.2) des ersten Segments des Programms. Der Programmname wird auch verwendet zur Kennzeichnung von Programmen, aus denen Bezeichnungen durch Extern-Vereinbarungen (5.1.13) zugänglich gemacht werden sollen.

2.3. Dynamischer Ablauf von Programmen

Der dynamische Ablauf eines PS440-Programms besteht in der Ausführung der ausführbaren Anweisungen (2.1) des Programms in der unten spezifizierten Reihenfolge. Die erste auszuführende Anweisung ist implizit definiert, z. B. durch den in einem TAS-Einschub angegebenen START-Befehl für einen Operatorlauf. 1) 2).

Die Ausführung einer Anweisung endet "normal", sie wird "terminiert" oder sie endet "irregulär". Eine Anweisung wird terminiert, wenn sie ein Sprung (6.1.13) ist. Der Sprung terminiert außerdem sämtliche zusammengesetzten Anweisungen, die den Sprung, nicht aber die als Sprungziel angegebene Programmstelle (2.4) enthalten. Die Ausführung endet irregulär, wenn sie einen Alarm (arithmetischer Alarm, Typenkennungsalarm usw.) auslöst. Die irreguläre Beendigung ist im allgemeinen gleichwertig einer Terminierung durch Sprung auf die durch Alarmadresse spezifizierte Programmstelle. Die Ausführung endet normal, wenn sie nicht terminiert wird oder irregulär endet.

Jede Anweisung definiert die als nächste auszuführende Anweisung. Endet die Ausführung normal, so handelt es sich um die im Sinne des Anschreibens nächste Anweisung. Andernfalls handelt es sich um die Anweisung, die bei der als Sprungziel spezifizierten Programmstelle beginnt.

Von der irregulären Ausführung ist die "undefinierte" Ausführung einer Anweisung zu unterscheiden. Sie tritt auf, wenn ein Programmfehler weder bei der Übersetzung noch während der Ausführung der fehlerhaften Operation entdeckt wird, z. B. bei Überschreitung der Indexgrenzen durch eine indizierte Variable.

2.4. Objekte, Bereiche und Zonen.

Die Ausführung einer Anweisung besteht aus einer Folge von Operationen (Maschinen-Befehlen), deren Operanden (interne) Objekte heißen. Objekte sind Marken und Prozeduren, die Inhalte von dem Programmierer zugänglichen Registern oder Registerteilen und "Datenobjekte".

---

1) Ein PS440-Programm könnte auch simultan oder quasi-simultan ablaufen. Die erste auszuführende Anweisung kann dabei für jeden Ablauf anders definiert sein. Auf mehrfache Abläufe und die hierbei zu erfüllenden Forderungen der eintrittsinvarianten Programmierung wird hier nicht eingegangen.

2) Keinesfalls darf angenommen werden, daß die im Sinne des Anschreibens erste ausführbare Anweisung auch als erste ausgeführt wird!

Marken kennzeichnen "Programmstellen", d.h. den Beginn "markierter Anweisungen". Prozeduren sind (meist zusammengesetzte)Anweisungen, deren Ausführung durch einen Prozeduraufruf veranlaßt werden kann, ohne daß an den Aufrufstellen die Prozedur niedergeschrieben werden muß. Marken und Prozeduren sind charakterisiert durch ihre (Anfangs-) Adressen (grundsätzlich 22-Bit-Adressen, sämtlich in einer Großseite), Prozeduren überdies durch die zum Aufruf bzw. Rücksprung verwendeten Befehle (SU- bzw. MU S 0 - Befehl oder SFB- und MABI S 0 - Befehl).

Datenobjekte sind charakterisiert durch die Art ihrer Unterbringung (z.B. schreibgeschützt oder nicht), ihre (Anfangs-) Adresse, ihren Umfang(z.B. Halbwort, Ganzwort, Doppelwort) und die Art der Adresse (16- oder 22-Bit-Adresse, Indexspeicheradresse). Im Gegensatz zu höheren Programmiersprachen wird bei Datenobjekten im PS440 nur hinsichtlich des Zugriffsverhaltens in Wertzuweisungen, nicht aber hinsichtlich der sonstigen auszuführenden Operationen (z.B. Festpunkt- oder Gleitpunktarithmetik) unterschieden. Die Unterscheidung, etwa zwischen Festpunkt- und Gleitpunktaddition, muß daher durch unterschiedliche Operationssymbole erfolgen (siehe 7).

Abgesehen von Registern werden Objekte in "Bereichen" des zu einem lauffähigen Programm gehörenden Adreßraums untergebracht. Die Adresse des Objekts ist die Anfangsadresse des Bereichs. Mit Ausnahme der Teilwort-Objekte (2.8.7) ist der Umfang des Objekts gleich der Länge des zugehörigen Bereichs [1]. Man unterscheidet selbständige und unselbständige Objekte. Selbständigen Objekten ist ein eigener Bereich mit unveränderlicher, statisch vorgegebener Anfangsadresse zugeordnet. Diese Bereiche überschneiden sich nicht. Für unselbständige Objekte sind die Anfangsadresse und der Bereich durch Rückgriff auf andere Objekte definiert: Der Bereich ist stets im Bereich eines selbständigen Objekts enthalten, die Anfangsadresse, also die Lage des Bereichs, kann bei unselbständigen Objekten dynamisch veränderlich sein.

Die Bereiche selbständiger Objekte werden gemäß der verlangten Unterbringung und Adreßlänge auf "Adreßzonen" verteilt.

Folgende Adreßzonen sind möglich:

K: Zone für schreibgeschützte und mit 16-Bit-Adressen ansprechbare "Konstanten".

V: Zone für nicht-schreibgeschützte und mit 16-Bit-Adressen ansprechbare "Variablen".

B: Zone für Befehle. Diese Zone ist schreibgeschützt und liegt vollständig innerhalb einer Großseite, die mit 22-Bit-Adresse ansprechbar ist.

DK: Zone für schreibgeschützte und mit 22-Bit-Adresse ansprechbare "Daten-Konstanten".

D: Zone für nicht-schreibgeschützte und mit 22-Bit-Adresse ansprechbare "Daten-Variablen".

---

1) Der Umfang einer Marke ist dabei formal mit 0 Halbworten anzusetzen.

Die Zonen K, V, B und D entsprechen den implizit definierten Adreßzonen K0, V0, B0, D0 des erzeugten TAS-Programms. Die Zone DK wird (unter einer dem Programmierer unzugänglichen Bezeichnung) explizit definiert. In diesen Zonen untergebrachte Bereiche belegen bereits zu Beginn des Programmlaufs physikalischen Speicher. Durch TAS-Einschübe können weitere Adreßzonen definiert werden. Dort untergebrachte Bereiche belegen eventuell erst nach dynamischer Speicheranforderung einen physikalischen Speicher (vgl. z.B. Freihaltezonen im Beispiel in 5.4.5).

Indexspeicher werden in Bereichen der Zonen V und D untergebracht, jedoch mit Indexspeicheradressen angesprochen. Demnach sind Indexspeicher unselbständige Objekte, deren Bereich mit dem Indexbasisregister dynamisch veränderlich in andere Bereiche abgebildet wird. Da dieser Rückgriff auf Bereiche der Zonen V oder D bei der Übersetzung nicht ersichtlich ist, werden Indexspeicher in PS440 formal als selbständige Objekte der Zone

I: Zone für nicht-schreibgeschützte und mit 8-Bit-Adresse ansprechbare "Indexspeicher"

betrachtet.

2.5. Externe Wiedergabe von Objekten, Werte von Datenobjekten

Objekte werden in einem PS440-Programm durch explizite Konstanten oder Benennungen wiedergegeben.

Explizite Konstanten (3.1.30) geben selbständige Objekte wieder und werden in einer Liste zusammengefaßt, in der jede Konstante genau einmal aufgenommen wird. Die Liste wird schreibgeschützt in der Zone K untergebracht. Treten explizite Konstanten jedoch als Bestandteile einer "konstanten Liste" (3.1.31) auf, so bestimmt der Verwendungszweck der konstanten Liste (5.1.19, 7.1.13) die Unterbringung der Konstanten.

Benennungen sind in erster Linie Bezeichnungen (3.1.5) sowie - im Fall von Registern - Schlüsselworte. Die Zuordnung zwischen Bezeichnungen und Objekten geschieht durch eine "Definition", nämlich eine Markendefinition (4.1.9, 4.2) im Fall von Marken oder eine Vereinbarung (siehe 5). Die Definition legt die Unterbringung, den Umfang und das Zugriffsverhalten des benannten Objekts fest (siehe 2.8). Durch Bezeichnungen können sowohl selbständige als auch unselbständige Objekte benannt werden. Durch Zufügung eines Wertoperators (8.1.6), eines Selektors (8.1.10) oder eines Index (8.1.7) zu einer Benennung erhält man weitere Benennungen für unselbständige Objekte. Die Zusätze spezifizieren nur den Umfang und teilweise das Zugriffsverhalten des Objekts. Die weiteren Eigenschaften ergeben sich aus der ursprünglichen Benennung.

Jedes Datenobjekt hat einen "Wert", wiedergegeben durch die Bitbesetzung des Bereichs für das Objekt. Die Interpretation der Bitbesetzung als ganze Zahl usw. ergibt sich aus dem Kontext, in dem das Objekt verwendet wird, ist aber keine Eigenschaft des Objekts. Während Binär-Angaben (3.1.12) die Bitbesetzung eines Datenobjekts unmittelbar wiedergeben, spezifizieren ganze Zahlen, Gleitpunktzahlen und Zeichenreihen die Bitbesetzung durch Angabe einer Interpretation des Wertes.

2.6. Größen und ihre Gültigkeitsbereiche

Neben den Grundsymbolen mit festgelegter Bedeutung kommen in PS440-Programmen vom Programmierer erfundene Grundsymbole vor, deren Bedeutung nur aus im Programm enthaltenen "Definitionen" ersichtlich ist. Diese Grundsymbole heißen "Größen". Größen sind entweder Bezeichnungen (3.1.5) oder "neue Schlüsselworte" (3.2.2). Bezeichnungen identifizieren Objekte (2.5), "Selektoren" (2.8.8, 9) und formale Makroparameter (5.3). Neue Schlüsselworte identifizieren "Befehle" (2.8.10) und "Makros" (5.3). Die Bedeutung von Größen wird im wesentlichen durch ihre "Attribute" charakterisiert, die in 2.7 zusammengestellt sind.

Jede Definition einer Größe und damit die Größe selbst hat einen "Gültigkeitsbereich", in dem die Größe "angewandt" werden darf. Der Gültigkeitsbereich besteht entweder aus einem einzigen Segment (4.1.2); in diesem Fall heißt die Größe "lokal". Oder er besteht aus dem gesamten Programm mit Ausnahme der Segmente, in denen die Größe durch weitere Definitionen als lokal definiert ist; in diesem Fall heißt die Größe "global". Oder die Größe identifiziert einen Makroparameter (siehe 5.3). In einem Segment darf es nur eine Definition für eine Größe geben; im ganzen Programm darf eine Größe nur durch eine Definition als global definiert sein. Ob eine Größe global ist, ist aus der Definition ersichtlich (vgl. 2.8).

Wird eine Größe angewandt, so müssen dem Übersetzer im allgemeinen die Attribute der Größe bekannt sein. Außer für Bezeichnungen, die Marken benennen, und bei Bezeichnungen in Referenzausdrücken (7.1.13) wird daher verlangt, daß die Definition der ersten Anwendung vorangeht. [1] Diese Forderung kann bei Bezeichnungen für selbständige Objekte umgangen werden, indem man der Definition eine "eigentliche Spezifikation" (5.4.1)vorausschickt. Durch die eigentliche Spezifikation werden die wesentlichen Attribute spezifiziert, es wird jedoch noch kein Bereich zugeordnet. Die durch eine eigentliche Spezifikation spezifizierten Attribute müssen mit den durch die (nachfolgende) Definition festgelegten Attributen übereinstimmen, insbesondere müssen die Spezifikation und die Definition den gleichen Gültigkeitsbereich festlegen. Eine Bezeichnung kann mehrmals spezifiziert werden. Man erhält daher für die Zulässigkeit der Anwendung einer Größe die Regel:

A) Die Anwendungsstelle muß im Gültigkeitsbereich einer Definition dieser Größe liegen. Diese Definition heißt die "zugehörige" Definition.

---

1) Andernfalls könnte man PS440 nicht in einem Lauf in TAS übersetzen.

B) a) Der Anwendungsstelle geht die zugehörige Definition voraus.

oder

b) Die Größe bezeichnet ein selbständiges Objekt und der Anwendungsstelle geht eine eigentliche Spezifikation der Größe voraus.

oder

c) Die Größe bezeichnet eine Marke.

oder

d) Die Größe bezeichnet ein selbständiges Objekt und die Anwendungsstelle ist ein Referenzausdruck.

2.7. Attribute von Größen

Die folgende Liste gibt die möglichen Attribute wieder. Die Attribute 2.7.2 - 2.7.5 heißen Hauptattribute und schließen sich gegenseitig aus. Eine Größe heißt selbständig und bezeichnet ein selbständiges Objekt, wenn sie eines der Hauptattribute 2.7.2 - 2.7.8 und nicht das Attribut "Äquivalenz" (2.7.19)hat. Die Attribute 2.7.2 - 2.7.8, 2.7.16 und 2.7.18 können auch durch eine eigentliche Spezifikation angegeben werden. Für weitere Einzelheiten über die Attribute siehe 2.8.

	Attribut	Bedeutung
2.7.1.	global	Die Größe hat globalen Gültigkeitsbereich.
2.7.2.	Marke	Die Größe identifiziert eine Marke.
2.7.3.	SU-Prozedur	Die Größe identifiziert eine mit SU-Befehl aufrufbare Prozedur.
2.7.4.	SFB-Prozedur	Die Größe identifiziert eine mit SFB-Befehl aufrufbare Prozedur.
2.7.5.	short	Die Größe identifiziert ein Halbwort oder einen Verbund von Halbworten.
2.7.6.	full	Die Größe identifiziert ein Ganzwort oder einen Verbund von Ganzworten.
2.7.7.	long	Die Größe identifiziert ein Doppelwort oder einen Verbund von Doppelworten.
2.7.8.	index	Die Größe identifiziert einen Indexspeicher oder einen Verbund von Indexspeichern.

	Attribut	Bedeutung
2.7.9.	part	Die Größe identifiziert einen durch Maske bestimmten Teil eines Ganzwortes.
2.7.10.	Selektor	Die Größe identifiziert einen Selektor,
2.7.11.	Teilwort-Selektor	Die Größe identifiziert einen Teilwortselektor.
2.7.12.	Makroparameter	Die Größe identifiziert einen Makroparameter.
2.7.13.	Befehl	Die Größe ist ein neues Schlüsselwort und identifiziert den Operationsteil eines Befehls.
2.7.14.	primär Makrokennzeichen	Die Größe ist ein neues Schlüsselwort und zwar das erste Schlüsselwort eines Makros.
2.7.15.	sekundäres Makrokennzeichen	Die Größe ist ein neues Schlüsselwort und zwar zweites oder weiteres Schlüsselwort eines Makros.
2.7.16.	Verbund	Die Größe identifiziert einen Verbund.
2.7.17.	konstant	Die Größe identifiziert ein selbständiges Datenobjekt im schreibgeschützten Bereich oder eine Prozedur.
2.7.18.	data	Die Größe identifiziert ein selbständiges Datenobjekt, das nur mit 22-Bit-Adresse ansprechbar ist.
2.7.19.	Äquivalenz	Die Größe identifiziert ein unselbständiges Datenobjekt.

2.8. Bemerkungen zu den Attributen

Bezüglich Makroparametern, primären und sekundären Makrokennzeichen sei auf 5.3 verwiesen. Die Definition einer Größe ist eine Segmentbenennung (4.1.4), eine Markendefinition oder eine Vereinbarung. Bei Vereinbarungen und eigentlichen Spezifikationen heißt die definierte bzw. spezifizierte Größe die "deklarierte Größe". Der Teil der Vereinbarung bzw. Spezifikation, welcher der ersten deklarierten Größe vorausgeht, heißt der "Kopf". Diese in 5.1.39, 41, 42 formal definierten Begriffe sind auf Makrovereinbarungen (5.1.35) nicht anwendbar. Die Angaben darüber, in welcher Weise das Attribut aus der Definition ersichtlich ist, gelten mit Abänderungen auch für Spezifikationen.

2.8.1.
global

Das Attribut liegt vor, wenn die Definition der Größe eine Extern-Vereinbarung (5.1.13) ist, mit dem Symbol global beginnt, oder, wenn die Größe eine Segmentbezeichnung (4.2) ist. Das Attribut kennzeichnet den Gültigkeitsbereich der Größe.

2.8.2.
Marken

Das Attribut liegt vor, wenn die Definition eine Markendefinition (4.1) oder eine Segmentbenennung ist, oder der Kopf das Symbol label enthält. Marken können außer mit Sprungoperatoren (goto, call, callsfb, 6.1.13, 14) nur in Referenzausdrücken vorkommen.

2.8.3.
SU-Prozeduren

Das Attribut liegt vor, wenn der Kopf der Definition das Symbol proc enthält. Die deklarierte Größe bezeichnet eine Anweisung, den "Prozedurrumpf" (5.1.25). Die Verarbeitung des Prozedurrumpfes kann veranlaßt werden durch einen "Prozeduraufruf" (6.1.14), der in einen SU-Befehl übersetzt wird. Dem Prozedurrumpf wird bei der Übersetzung der Befehl MU S 0 angefügt. Der Rücksprung kann auch durch eine im Prozedurrumpf enthaltene Rückkehranweisung "return" erfolgen, die ebenfalls in den Befehl MU S 0 übersetzt wird.

Wird der Prozeduraufruf terminiert, d.h. durch Sprung verlassen, so wird dabei der Unterprogrammzähler bu nicht verändert. Beim Aufruf und Rücksprung werden die Register ra, rq, rd, rh und bb nicht verändert. Prozeduren mit Parametern sind nicht möglich. Etwaige Versorgungsparameter oder Ergebnisse stehen entweder in den Registern oder in einem "Versorgungsblock", dessen Anfangsadresse in ra bereitgestellt ist (Programmierkonvention).

Rekursive Prozeduren sind bei Beachtung folgender Bedingungen möglich: Der Indexspeicherbereich für Rücksprünge muß ausreichen; lokale Größen und noch benötigte Versorgungsparameter sind über Äquivalenzvereinbarungen (5.1.32) in einem Keller unterzubringen; der rekursive Aufruf darf nicht aus einer Laufanweisung mit Zählung (6.1.6) heraus erfolgen. Man vergleiche das Beispiel in 5.4.2.

2.8.4.
SFB-Prozeduren

Das Attribut liegt vor, wenn der Kopf der Definition das Symbol procsfb enthält. Es gelten im wesentlichen die Ausführungen von 2.8.3 mit folgenden Abänderungen: Der Aufruf einer SFB-Prozedur erfolgt durch einen SFB-Befehl; der Rücksprung, der auch mit Rückkehranweisung veranlaßt werden kann, erfolgt durch den Befehl MABI S 0. Für die Sicherstellung der Rücksprungadresse aus dem bb und die Wiederbesetzung des bb vor dem Rücksprung hat der Programmierer (vorläufig) selbst zu sorgen. Das Register bb wird beim Aufruf verändert. Bei rekursiven SFB-Prozeduren ist auch die Rückkehradresse vom Programmierer in einem Keller unterzubringen.

2.8.5.
short, full, long

Eines dieser Attribute liegt vor, wenn der Kopf der Definition das Symbol short, full, bzw. long, nicht aber das Symbol select enthält. Die deklarierte Größe identifiziert ein (selbständiges oder unselbständiges) Datenobjekt. Der Umfang des Objekts ist entweder ein Halbwort, Ganzwort bzw. Doppelwort oder- für den Fall, daß zusätzlich das Attribut Verbund vorliegt - eine linear geordnete Menge von "Verbundelementen" (2.8.11), von denen jedes ein Halbwort, Ganzwort bzw. Doppelwort belegt.

Die Verteilung der selbständigen Datenobjekte mit diesen Attributen auf die einzelnen Adreßzonen (2.4) richtet sich nach den Attributen konstant und data:

vorhandenes Attribut	Adreßzone
konstant und data	DK
konstant	K
data	D
keines von beiden	V

Bei short-Größen ist die Unterbringung speziell im rechten oder linken Halbwort eines Ganzwortes nur durch Äquivalenzvereinbarung erreichbar.

Bei Wertzuweisungen von und an Register (speziell ra bzw. raq) werden die Befehle mit den Operationsteilen C2 und B2V (short-Größen), C und B (full-Größen), CZ und BZ (long-Größen) verwendet. Dies gilt auch, wenn die Größe als Verbund deklariert ist! In diesem Fall bezieht sich der Zugriff also nur auf das "erste" Verbundelement. [1]

Bei Sprüngen und Prozeduraufrufen (6.1.13, 14) sowie bei Anwendung von Wertoperatoren (8.1.6) auf Datenobjekte wird erwartet, daß das Datenobjekt rechtsbündig eine Adresse, d.h. das Ergebnis eines Referenzausdrucks, enthält. Bei Doppelworten muß die Adresse also im rechten Halbwort des zweiten Ganzwortes stehen. Für die Ausführung von Sprüngen und Prozeduraufrufen werden die Befehle mit den Operationsteilen SE, T, SUE bzw, SFBE verwendet.

Ist eine selbständige Größe mit den Attributen full oder long versehen, so besagt das noch nicht, daß man nur ganzwort- bzw. doppelwortweise auf den Bereich, der zu dieser Größe gehört, zugreifen kann. Unter Zuhilfnahme von Selektoren und Äquivalenzvereinbarungen kann man den Zugriff auf Halbworte, Ganzworte, Doppelworte oder durch Masken ausgeblendete Teile eines Ganzwortes erreichen. Voraussetzung ist, daß der veränderte Zugriff nicht ein Objekt größeren Umfangs unter-

[1] Das ist das Verbundelement mit Index 0! (2.8.11)

stellt als tatsächlich gegeben ist; beispielsweise ist der Zugriff auf ein Doppelwort undefiniert, wenn nur ein Bereich im Umfang eines Ganzwortes vorliegt. Die Veränderung des Zugriffs auf ein selbständiges Objekt mit dem Attribut short ist undefiniert, da hierzu bekannt sein müßte, ob der zugehörige Bereich mit gerader oder ungerader Adresse beginnt. Ob ein veränderter Zugriff in diesem Sinne undefiniert ist, kann vom Übersetzer nicht geprüft werden!

Man beachte, daß man Größen mit den Attributen full oder long auch dann indizieren (8.2.3) kann, wenn das Attribut Verbund nicht vorliegt.

2.8.6. [2]
index

Das Attribut liegt vor, wenn der Kopf der Definition das Symbol index enthält. Die deklarierte Größe identifiziert ein (selbständiges oder unselbständiges) Datenobjekt, dessen Umfang entweder ein Halbwort oder- für den Fall, daß zusätzlich das Attribut Verbund vorliegt - eine linear geordnete Menge von Verbundelementen, von denen jedes ein Halbwort belegt. Im Unterschied zu short-Größen wird auf diese Halbworte mit Indexspeicheradressen und -Befehlen zugegriffen, im Fall von Wertzuweisungen von und an Register mit den Befehlen mit den Operationsteilen TRX bzw. TXR.

Selbständige Objekte mit dem Attribut index werden in der formalen Adreßzone I (2.4) untergebracht. Damit ist die Indexspeicheradresse dieser Objekte, nicht aber der belegte Speicherbereich fixiert. Dieser ist vielmehr durch die dynamische Besetzung des Indexbasisregisters definiert und könnte sich durch Veränderung des Inhalts dieses Registers ändern.

Die Speicherbereiche, die für die Unterbringung von index-Größen vorgesehen sind, d.h. auf deren Anfangsadresse das Indexbasisregister eingestellt werden soll, müssen als Verbund von short-Größen speziell definiert werden. Wegen der technischen Besonderheiten beim Zugriff auf Indexspeicher sollte in diese Speicherbereiche stets nur mit Indexspeicheradressen, nicht aber mit den an sich ebenfalls möglichen, aus der Definition dieser Speicherbereiche resultierenden Kernspeicheradressen zugegriffen werden.

2.8.7.
part

Das Attribut liegt vor, wenn der Kopf der Definition das Symbol part, nicht aber das Symbol select enthält. Das Attribut kommt nur zusammen mit dem Attribut Äquivalenz (2.8.14) vor. Die deklarierte Größe identifiziert also stets ein unselbständiges Datenobjekt, nämlich einen Ausschnitt aus einem Ganzwort. Das Objekt wird charakterisiert durch ein Objekt Q mit dem Attribut full, nämlich dem Ganzwort aus dem ausgeschnitten wird, und einem Wert vom Umfang eines full-Objekts, der "Maske", die den Ausschnitt bestimmt.

---

[2] Man unterscheide zwischen dem Attribut index (mit kleinem i) und dem Wort Index (mit großem I). Letzteres kommt im Zusammenhang mit Selektoren und der Indizierung von Objekten speziell Verbunde, vor (2.8.8, 2.8.9).

Dem durch die part-Größe identifizierten Ausschnitt sind die Bits von Q zugeordnet, die den mit 0 besetzten Positionen der Maske entsprechen (die Typenkennung der Maske ist gleichgültig, die Typenkennung des ursprünglichen Ganzworts wird stets übernommen). Der Ausschnitt wird also durch die "Nullfelder" der Maske bestimmt. Von allgemeinem Interesse ist nur der Fall, daß die Maske genau ein (zusammenhängendes) Nullfeld enthält. Die weitere Beschreibung beschränkt sich auf diesen Fall.

Tritt eine part-Größe als Operand auf, so wird der Ausschnitt rechtsbündig als Objekt vom Umfang eines Ganzworts bereitgestellt. Die nicht zum Ausschnitt gehörenden Bits sind mit 0 besetzt. Umfaßt das rechte "Einsfeld" der Maske p Bits, so bedeutet das, daß nach der eigentlichen Maskieroperation noch eine Verschiebung um p Stellen nach rechts erfolgt. Wertzuweisungen an part-Größen verlaufen hierzu invers: Der Wert der rechten Seite wird um p Stellen nach links verschoben, bevor die dem Nullfeld der Maske entsprechenden Bits in das full-Objekt Q eingefügt werden.

Bei Wertzuweisungen und Operationen, die part-Größen betreffen, werden die Befehle mit den Operationsteilen BT, CT, AT und SBT verwendet. Objekte mit dem Attribut part können nicht zum Verbund zusammengefaßt werden und nicht in Referenzausdrücken auftreten.

2.8.8. Selektor

Das Attribut liegt vor, wenn der Kopf der Definition eines der Symbole short, full oder long, gefolgt vom Symbol select enthält. Die Größe identifiziert einen "Selektor", d.h. einen Operator, mit dem aus der Benennung eines selbständigen oder unselbständigen Datenobjekts Q die Benennung eines unselbständigen Objekts Q' erhalten werden kann. Die Angabe short, full oder long liefert das entsprechende Hauptattribut für das Objekt Q', definiert also den Umfang von Q'. Die Definition des Selektors enthält außerdem eine ganze Zahl, den "Index" des Selektors. [1] Der Index liefert die Relativadresse im Bereich des Objekts Q, an der das Objekt Q' beginnt. Die Zählung des Index beginnt mit 0 und läuft in Halbwortschritten, auch wenn full oder long als Attribut für Q' spezifiziert wird.

Definiert der Selektor für Q' das gleiche Hauptattribut, das für Q gültig ist, so ist die Anwendung des Selektors gleichwertig mit der entsprechenden Indizierung (8.2.3). Die Verwendung eines Selektors, der für Q' full oder long als Hautattribut definiert, ist nicht sinnvoll, wenn der Index ungerade ist oder das Hauptattribut von Q short war.

---

[1] Vergleiche die Fußnote 2 zu 2.8.6

2.8.9.
Teilwortselektor

Das Attribut liegt vor, wenn der Kopf der Definition die Symbolfolge part select enthält. Die Größe identifiziert einen "Teilwortselektor", d.h. einen Operator, mit dem aus der Benennung eines selbständigen oder unselbständigen Objekts Q die Benennung eines unselbständigen Objekts Q' mit dem Hauptattribut part erhalten werden kann. Die Definition des Teilwortselektors enthält eine ganze Zahl, den Index des Teilwortselektors und einen Wert vom Umfang eines full-Objekts, die Maske des Teilwortselektors. Die Anwendung des Teilwortselektors erfolgt in zwei Schritten: Zunächst wird wie bei einem Selektor (2.8.8) der Index verwandt, um aus Q ein Objekt Q" mit dem Hauptattribut full zu erhalten. Das Objekt Q" und die Maske bestimmen, wie in 2.8.7 beschrieben, ein Objekt mit dem Hauptattribut part.

Objekte mit dem Hauptattribut part können nur unter direkter oder indirekter Verwendung eines Teilwortselektors erhalten werden, da die Definition von Teilwortselektoren die einzige Programmstelle ist, an der Masken als explizite Konstanten angebbar sind. Zur Änderung der Aufteilung eines Objekts auf mehrere part-Größen ist daher nur der zugehörige Satz von Teilselektoren auszutauschen.

2.8.10
Befehl

Das Attribut liegt vor, wenn der Kopf der Definition das Symbol instruct enthält. Die deklarierte Größe ist ein neues Schlüsselwort (3.1.4), das einen Maschinenbefehl identifiziert. Der Operationsteil des Befehls ist in Form einer TAS-Sequenz (2.9) in der Definition angegeben. Schlüsselworte, die Befehle identifizieren, können als unäre Operatoren auf Benennungen (6.1.15) angewandt werden, sofern die Benennung kein Objekt mit dem Attribut part benennt. Die Operation kann als Primärausdruck verwandt werden, wenn in der Definition ein Ergebnisregister angegeben ist. Es wird dann so verfahren, als ob es sich um eine Wertzuweisung an dieses Register handelt.

Werden bei der Anwendung eines Befehls die folgenden Bedingungen nicht beachtet, so können sich Fehlermeldungen bei der TAS-Übersetzung ergeben:

a) Verlangt der Befehl eine Indexspeicheradresse, so darf er nur auf Benennungen von index-Objekten angewandt werden.

b) Verlangt der Befehl eine Kernspeicheradresse, so darf er nur auf explizite Konstanten und Benennungen von Objekten, die nicht das Attribut index oder part haben, angewandt werden.

c) Befehle, die nicht eine Kernspeicher- oder Indexspeicheradresse verlangen, sind verboten. (Dies schließt Operationsteile, wie LA, LZL, VLA von der Benutzung durch Befehlsdefinition aus).

d) Wird ein Befehl auf ein Register angewandt, so muß der Operationsteil als Zweitcode beim Befehl R zugelassen sein.

Es muß außerdem damit gerechnet werden, daß der Befehl mit mod2 modifiziert oder als Zweitcode beim Befehl E verwendet wird.

2.8.11. Verbund

Das Attribut liegt vor, wenn der Kopf der Definition eine Dimensionsangabe (5.1.6) enthält. Ist die deklarierte Größe selbständig, so identifiziert sie einen Bereich von n Halbworten oder Indexspeichern je nachdem, ob das Hauptattribut short, full bzw. long oder index ist. Bei unselbständigen Größen hat das Attribut Verbund keine Bedeutung; es ist höchstens als Hinweis für den menschlichen Leser interessant. Die Dimensionsangabe enthält entweder n als ganze Zahl, die auf das Symbol record folgt, oder es enthält (n-1) als zweite Zahl in der Indexklammer. Die beiden Formen der Dimensionsangabe sind semantisch gleichwertig; n heißt in jedem Fall die Länge des Verbunds. Bei Größen mit den Hauptattributen full bzw. long sind nur Längenangaben sinnvoll, die ein Vielfaches von 2 bzw. 4 sind.

Ein Objekt Q mit dem Attribut Verbund ist eine linear geordnete Menge von unselbständigen Objekten, den Verbundelementen. Diese Objekte haben mit Ausnahme des Attributs Verbund die gleichen Attribute wie das Objekt Q. Benennungen für die Verbundelemente erhält man, indem man einer Benennung für Q einen Index (8.1.7) zufügt. Der im Index enthaltene Indexausdruck gibt an, welche Relativadresse man auf die Adresse von Q aufaddieren muß, um die Adresse des gewünschten Verbundelements zu erhalten. Die möglichen Werte des Indexausdrucks sind daher 0, k, 2k,... , n-k, wobei k die Länge der Verbundelemente in Halbworten ist (k=1 für short und index, k=2 für full, k=4 für long). Man beachte, daß die Werte aktueller Indexausdrücke nicht auf Zulässigkeit, insbesondere nicht auf Überschreitung der Grenzen, geprüft werden.

Ist die Benennung des Objekts Q mit mehreren Indizes versehen, so werden die Werte der Indexausdrücke zusammengezählt. Fehlt der Index und ist auch kein Selektor oder Teilsektor angegeben, so wird der Index [0] ergänzt.

Die Verwendung von Indizes impliziert die semantische Einteilung in Verbundelemente gleichen, durch das Hauptattribut des Gesamtobjekts festgelegten Umfangs. Durch Verwendung von Selektoren und Teilwortselektoren läßt sich eine andere Einteilung erreichen , da dann der Umfang der resultierenden Teilobjekte durch den Selektor bzw. Teilwortselektor festgelegt wird. Indizierung und Selektoren können auch gekoppelt vorkommen.

2.8.12. konstant

Das Attribut liegt vor, wenn in der Definition auf die deklarierte Größe das Symbol is folgt. Die deklarierte Größe identifiziert entweder eine Prozedur(2.8.3, 2.8.4) oder ein selbständiges Datenobjekt. In beiden Fällen ist der zugehörige Bereich schreibgeschützt. Bei Datenobjekten folgt auf das Symbol is eine konstante Liste (3.1.31), welche die Besetzung des Bereichs wiedergibt.

Wertzuweisungen an Objekte mit dem Attribut konstant sind unzulässig und werden vom PS440-Übersetzer beanstandet. Man beachte jedoch, daß unselbständige Objekte schreibgeschützt untergebracht sein können, ohne daß das Attribut konstant vorhanden ist. In diesem Fall wird die Unzulässigkeit von Wertzuweisungen erst zur Laufzeit erkannt.

2.8.13.
data

Das Attribut liegt vor, wenn der Kopf der Definition das Symbol data enthält. Die deklarierte Größe identifiziert ein Datenobjekt, auf das nur mit 22-Bit-Adresse zugegriffen werden kann.

2.8.14.
Äquivalenz

Das Attribut liegt vor, wenn in der Definition auf die deklarierte Größe das Symbol = folgt und der Kopf der Definition das Symbol select nicht enthält. Die deklarierte Größe identifiziert ein unselbständiges Objekt Q. Die Adresse des Objekts ist durch die auf das Symbol = folgende Benennung B gegeben. Der Umfang des Objekts Q ist durch das in der Definition angegebene Hauptattribut short, full, long, index oder part gegeben. Die Hauptattribute index bzw. part dürfen genau dann vorkommen, wenn die Benennung B ein Objekt Q´ identifiziert, welches das gleiche Hauptattribut hat. In diesen Fällen ist Q=Q´. Wie bereits in 2.8.8, 2.8.9 und 2.8.11 ausgeführt, ist darauf zu achten, daß bei einer Änderung des Zugriffsverhaltens, z. B. von short auf full, die Benennung B keine ungerade Adresse liefert. Das Attribut Äquivalenz schließt die Attribute konstant und data aus. Das impliziert nicht, daß der Bereich des Objekts Q mit 16-Bit-Adresse erreichbar oder nicht schreibgeschützt ist. Da mit der Definition eines unselbständigen Objekts keine Speicherreservierung verbunden ist und bei Indizierung keine Prüfung auf Überschreitung der Indexgrenzen stattfindet, hat das Attribut Verbund keine Bedeutung.

Man beachte, daß die aus der Benennung B resultierende Adresse nicht, wie bei EQUIVALENCE-Anweisungen in FORTRAN, statisch, sondern dynamisch ausgewertet wird: Die Benennung B wird an jeder Stelle, an der die deklarierte Größe angewandt wird, in das Programm eingeordnet. Die sich ergebene Adresse könnte sich daher dynamisch ändern, z. B. wenn der Wert einer in B als Index vorkommenden Größe geändert wird.

2.9.
TAS-Sequenzen und TAS-Einschübe

In PS440 können zwischen den Zeichenfolgen */ und /* eingeschlossene "TAS-Sequenzen" vorkommen (3.1.25). TAS-Sequenzen erlauben die Ausnutzung von Maschineneigenschaften, denen keine sprachlichen Hilfsmittel in PS440 entsprechen. TAS-Sequenzen können in Definitionen anstelle von konstanten Listen (5.1.19)und zur Wiedergabe von Operationsteilen in Befehlsvereinbarungen (5.1.33) verwendet werden. Ansonsten kann man sie als eigentliche Anweisungen verwenden (6.1.1). TAS-Sequenzen, die anstelle von konstanten Listen oder eigentlichen Anweisungen stehen, heißen "TAS-Einschübe".

Für die Bedingungen für TAS-Sequenzen, die Operationsteile wiedergeben, siehe 2.8.10.

TAS-Einschübe müssen stets Folgen vollständiger TAS-Informationseinheiten sein. Hinter der letzten Informationseinheit wird ein Abgrenzungsteil ergänzt. Ansonsten werden TAS-Einschübe unverändert im erzeugten Programm abgelegt.

Steht der TAS-Einschub als konstante Liste in einer Definition, so soll er in der durch die An- oder Abwesenheit der Attribute data bzw. konstant gegebenen Adreßzone abgelegt werden. Die einzelnen Informationseinheiten müssen deshalb die entsprechenden Ablagekennungen enthalten, soweit sie nicht selbstdefinierende Konstanten sind.

TAS-Einschübe, die anstelle eigentlicher Anweisungen stehen, werden in der Zone abgelegt, die ihrem Inhalt entspricht.

Für alle TAS-Einschübe gelten folgende Bedingungen:

a) Sie sollen keine Benennungsteile enthalten.

b) Im PS440-Programm definierte selbständige Bezeichnungen (Marken, Prozeduren, selbständige Datenobjekte) dürfen in Adreßteilen und Adreßkonstanten vorkommen. Bezeichnungen für unselbständige Objekte können nicht verwandt werden.

c) Die folgenden Pseudobefehle dürfen nicht verwendet werden: ENDE-, SEGM-, STARR-, STEND-, EXTERN- und EINGG-Befehl. Durch die Pseudobefehle zur Vereinbarung von Adreßzonen und Gebieten im Zusammenwirken mit dem ABLAGE-Befehl kann die Verteilung der in PS440 definierten selbständigen Objekte auf Adreßzonen beeinflußt werden (vgl Beispiel in 5.4.5).

2.10. Zur Beschreibung der Syntax von PS440

In den Abschnitten 3-8 wird die Syntax in erweiteter Backus-Naur-Form beschrieben. Wo das nicht möglich ist, wird die Umgangssprache benutzt. Zusätzlich eingeführt werden "Wiederholungsklammern" und "Wiederholungsklammern mit Trennung" : $\{\alpha\}_n^m$ bedeutet, daß an dieser Stelle α mindestens n-mal ($n \geqq 0$) und höchstens m-mal ($m \geqq n$) vorkommt. Wird der untere Index n weggelassen, so wird 0 ergänzt. Wird der obere Index m weggelassen, so wird ∞ ergänzt (beliebig häufige Wiederholung). Die Wiederholungsklammer mit Trennung hat die Form $\{\alpha \,||\, \beta\}$ und bedeutet

$$\langle \text{leer} \rangle \;|\alpha|\; \alpha\, \{\beta\, \alpha\}$$

α :: = β $\{\gamma \,|\, \delta\}\, \varepsilon$ bedeutet

$$\alpha :: = \beta\, \{x\}\, \varepsilon$$

$$x :: = \gamma \,|\, \delta$$

2.11. Bemerkungen zu den Beispielen

Bei den in den nachfolgenden Abschnitten angegebenen Beispielen von aus PS440-Konstruktionen erzeugten Folgen von TAS-Befehlen ist zu beachten:

a) Als Abgrenzungsteil wird in Wirklichkeit nicht ein Komma, sondern ein Kommentar verwendet, der die Zeilennummer aus dem PS440-Programm enthält. Dies erleichtert die Lokalisierung des TAS-Übersetzers.

b) Etwaige Bezüge, die in Wirklichkeit auf die vom PS440-Übersetzer aufgebaute Konstantenliste (2.5) verweisen, werden in den Beispielen durch Literale erster Ordnung wiedergegeben.

c) Die in den Beispielen auftretenden, intern erzeugten Marken (** ....) gehorchen nicht dem gleichen Bildungsgesetz wie in Wirklichkeit.

d) Aus den Beispielen darf nicht geschlossen werden, daß irgendeine PS440-Konstruktion bestimmte Register des TR440 nicht verändert. Man vergleiche 8.4.

3. GRUNDSYMBOLE, EXPLIZITE KONSTANTEN, KONSTANTE LISTEN

3.1. Syntax

3.1.1. ⟨Grundsymbol⟩ ::= ⟨Sonderzeichen⟩|⟨Sonderzeichenkombination⟩|⟨Schlüsselwort⟩|⟨Bezeichnung⟩|⟨Wertangabe⟩|⟨TAS-Sequenz⟩|⟨Kommentar⟩ ⟨Grundsymbol⟩|⟨Zwischenraum⟩ ⟨Grundsymbol⟩|⟨Zeilenwechsel⟩ ⟨Grundsymbol⟩

3.1 2. ⟨Sonderzeichen⟩ ::= ⟨Zeichen ≠ Buchstabe, Ziffer, Zwischenraum, Zeilenwechsel, Punkt, tiefgestellte Zehn, Apostroph⟩

3.1.3. ⟨Sonderzeichenkombination⟩ ::= .=|:=|.=:|:=:|(/|/)|..|.,

3.1.4. ⟨Schlüsselwort⟩ ::=
abs | ac | and | aral | aut | bb |
begin | bf | bit | bodd | bu | by |
call | case | co | comment | cond |
continue | data | dmod | do | else |
elsf | end | entry | eq | esac | et |
exec | expr | fi | field | finis | for |
from | full | fval|
ge | glob | global | goto | gt|
if | index | instr | instruct | is |
ival |
label | le | left | leftc | long | lt |
lval |
macro | minus | ml | mll | mod |
ne | newid | not |
odd | of | or |
part | plus | proc | procedure |
procsfb |

ra | raq | rd | record | ref |
return | rh | rhq | right | rightc |
rm | rq | ry |
segm | segment | sel | select |
short | spec | stat | sval |
then | tk | tkal | tk0 | tk1 | tk2 |
tk3 | to |
until |
val | var | vel |
while | ws |

⟨neues Schlüsselwort⟩

3.1.5. ⟨Bezeichnung⟩ ::= ⟨Buchstabe⟩
{⟨Buchstabe⟩ | ⟨Ziffer⟩}

3.1.6. ⟨Buchstabe⟩ ::= A | B | C | D | E | F | G | H |
I | J | K | L | M | N | O | P |
Q | R | S | T | U | V | W | X |
Y | Z | &

3.1.7. ⟨Ziffer⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

3.1.8. ⟨Wertangabe⟩ ::= ⟨ganze Zahl⟩ | ⟨Gleitpunktzahl⟩ |
⟨Bits-Angabe⟩ | ⟨Zeichenreihe⟩

3.1.9. ⟨ganze Zahl⟩ ::= $\{\langle \text{Ziffer} \rangle\}_1$

3.1.10. ⟨Gleitpunktzahl⟩ ::= ⟨ganze Zahl⟩ ⟨Exponent⟩ |
$\{\langle \text{ganze Zahl} \rangle\}_0^1$.
⟨ganze Zahl⟩
$\{\langle \text{Exponent} \rangle\}_0^1$

3.1.11. ⟨Exponent⟩ ::= $\{ {}_{10} \mid {}_{10}{}^{+} \mid {}_{10}{}^{-} \mid$
E | E+ | E- $\}_1^1$
⟨ganze Zahl⟩

3.1.12. ⟨Bits-Angabe⟩ ::= ⟨Sedezimalzahl⟩ | ⟨Oktalzahl⟩ | ⟨Binärzahl⟩

3.1.13. ⟨Sedezimalzahl⟩ ::= '⟨TK-Angabe⟩ H ⟨lr-bündig⟩ {⟨Sedezimalziffer⟩}'

3.1.14. ⟨TK-Angabe⟩ ::= 0 | 1 | 2 | 3

3.1.15. ⟨lr-bündig⟩ ::= . | ⟨leer⟩

3.1.16. ⟨leer⟩ ::=

3.1.17. ⟨Sedezimalziffer⟩ ::= ⟨Ziffer⟩ | A | B | C | D | E | F

3.1.18. ⟨Oktalzahl⟩ ::= '⟨TK-Angabe⟩ O ⟨lr-bündig⟩ {⟨Oktalziffer⟩}' | '⟨TK-Angabe⟩ 0 ⟨lr-bündig⟩ {⟨Oktalziffer⟩}'

3.1.19. ⟨Oktalziffer⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

3.1.20. ⟨Binärzahl⟩ ::= '⟨TK-Angabe⟩ B ⟨lr-bündig⟩ {⟨Binärziffer⟩}'

3.1.21. ⟨Binärziffer⟩ ::= 0 | 1

3.1.22. ⟨Zeichenreihe⟩ ::= "⟨Zeichensequenz⟩"

3.1.23. ⟨Zeichensequenz⟩ ::= {⟨Zeichen⟩}

3.1.24. ⟨Zeichen⟩ ::= ⟨jedes Zeichen des TR440-Zentralcodes⟩

3.1.25. ⟨TAS-Sequenz⟩ ::= */ ⟨Zeichensequenz, welche /* nicht enthält⟩ /*

3.1.26. ⟨Kommentar⟩ ::= ⟨Kommentaranfang⟩ ⟨Zeichensequenz, welche kein Kommentarende enthält⟩ ⟨Kommentarende⟩

3.1.27. ⟨Kommentaranfang⟩ ::= co | comment

3.1.28. ⟨Kommentarende⟩ ::= ; | .,

3.1.29. ⟨Wortsymbol⟩ ::= '⟨Bezeichnung⟩'

3.1.30. ⟨explizite Konstante⟩ ::= {full} $^{1}_{0}$ ⟨Wertangabe⟩ |
long ⟨Wertangabe⟩ |
short ⟨ganze Zahl⟩ |
short ⟨Bits-Angabe⟩ |
short ⟨Zeichenreihe⟩

3.1.31. ⟨konstante Liste⟩ ::= ⟨explizite Konstante⟩ |
⟨konstante Referenz⟩ |
{short | full | long} $^{1}_{0}$
⟨Wiederholungszahl⟩ $^{1}_{0}$
({⟨konstante Liste⟩ || , } $_{1}$)

3.1.32 ⟨konstante Referenz⟩ ::= ref ⟨Benennung⟩ |
ref ⟨Marke⟩ |
ref ⟨Prozedurbezeichnung⟩

3.1.33. ⟨Wiederholungszahl⟩ ::= ⟨ganze Zahl⟩

3.2. Bemerkungen zur Syntax und zur Schreibweise

Die Syntax von 3.1 ist nicht eindeutig. Die nachfolgenden Bemerkungen geben Einschränkungen und Erweiterungen zu 3.1 wieder und gehören damit zur Syntax.

3.2.1. Grundsymbole. die selbst in anderen Grundsymbolen enthalten sind - z. B. Strichpunkt als Kommentarende - werden nicht als Grundsymbole betrachtet.

3.2.2. Schlüsselworte können als Bezeichnungen ("Schlüsselwortbezeichnungen") - z. B. ABS - oder als Wortsymbole - z. B. 'ABS' - geschrieben sein. Durch Befehlsvereinbarung (5.1.33) oder Makro-Vereinbarung (5.1.35)läßt sich der Bestand an Schlüsselworten durch "neue Schlüsselworte" erweitern.

Diese müssen von den vorhandenen Schlüsselworten verschieden sein. Auch wenn alle Schlüsselworte als Wortsymbole geschrieben sind, darf die gleichlautende Bezeichnung nicht anderweitig zur Identifizierung von Objekten, Selektoren oder formalen Makroparametern benutzt werden. Bei neuen Schlüsselworten bezieht sich das Verbot nur auf den Gültigkeitsbereich des neuen Schlüsselworts (2.6).

3.2.3. Bezeichnungen, Schlüsselwortbezeichnungen, ganze Zahlen und Gleitpunktzahlen dürfen keine Zeilenwechsel oder Zwischenräume enthalten. Sie enden vor dem ersten Zeichen, das gemäß der Syntax von 3.1.5, 3.1.9 oder 3.1.10 nicht mehr als zugehörig erkannt wird. Solche Trennzeichen sind insbesondere Zeilenwechsel und Zwischenraum. Die einzelnen Zeichen einer Sonderzeichenkombination sowie die beiden Apostrophe am Anfang bzw. Ende einer Zeichenreihe dürfen ebenfalls nicht durch Zeilenwechsel oder Zwischenraum getrennt werden.

3.2.4. An allen Stellen, an denen syntaktisch eine ganze Zahl verlangt ist - z.B. in 3.1.33 - kann auch eine Bit-Angabe mit der Typenkennungsangabe 1 stehen.

3.2.5. Apostroh ist als Zeichen in der Zeichensequenz einer Zeichenreihe erlaubt. Zwei aufeinanderfolgende Apotrophe sind jedoch durch vier Apostrophe wiederzugeben, um Verwechslung mit den abschließenden Apostrophen auszuschließen.

3.2.6. In Zeichenreihen steht

* ⟨Ziffer⟩ ⟨Ziffer⟩ ⟨Ziffer⟩

für das Zeichen, das im Zentralcode durch die auf * folgende dreistellige Dezimalzahl wiedergegeben wird. Führende Nullen müssen geschrieben werden. Der Stern kann in Zeichenreihen nur in diesem Zusammenhang verwendet werden.

3.2.7. Eine in einer konstanten Referenz auftretende Benennung (8.) muß eine dynamisch nicht veränderliche Adresse liefern. Das bedeutet, daß die Benennung keinen Wertoperator (8.1.6) oder Teilwortselektor (8.1.10) enthalten darf und daß als Indizes nur ganze Zahlen zulässig sind.

3.2.8. Für Spezialzwecke ist die Benutzung des Zeichens & (Oktade 119) als Buchstabe in Bezeichnungen zulässig. Der PS440-Übersetzer erzeugt intern Bezeichnungen der Form

$$\& \{ \langle \text{Sedezimalziffer} \rangle \}_{5}^{5}$$

3.2.9.

Die nachfolgenden Sonderzeichen, Sonderzeichenkombinationen bzw. Schlüsselworte sind gleichwertig:

∧	and	
co	comment	
=	eq	
≧	ge	> =
>	gt	
global	glob	
instruct	instr	
≦	le	< =
<	lt	
≠	ne	¬ =
¬	not	
v	or	
+ : =	+ . =	plus
- : =	- . =	minus
proc	procedure	
ra	ac	
segment	segm	
select	sel	
sval	val	
: =	. =	
: = :	. = :	
[	(/	
]	/)	
:	..	
;	.,	

Im Rest dieser Beschreibung wird im allgemeinen nur die in der ersten Spalte aufgeführte Schreibweise erwähnt.

3.3.
Semantik

Ein PS440-Programm ist eine Folge von Grundsymbolen (2.1). Kommentare, Zwischenräume und Zeilenwechsel haben keine semantische Bedeutung. Zwischenräume in Zeichenreihen und TAS-Einschüben sind jedoch bedeutungsvolle Zeichen.

Bezeichnungen und neue Schlüsselworte sind Größen (2.6). Für TAS-Einschübe vergleiche man 2.9.

Ganze Zahlen und Gleitpunktzahlen haben die übliche Bedeutung. Bitangaben geben einen Wert unmittelbar als Bitbesetzung wieder. Diese ergibt sich, indem man eine Folge von Drei- oder Vier-Bit-Gruppen bzw. eine Folge von Binärziffern als Folge von Bits auffaßt.

Die Bitfolge wird linksbündig bzw. rechtsbündig untergebracht, wenn die Bitangabe einen Punkt enthält bzw. nicht enthält. Bei in Ganz- oder Doppelworten unterzubringenden Bitfolgen charakterisiert die TK-Angabe die Typenkennung des bzw. der beiden Worte (s). Bei in Halbworten unterzubringenden Bitbesetzungen kann naturgemäß keine bestimmte Typenkennung (TK) gewährleistet werden; die TK-Angabe ist jedoch stets zu schreiben. Spielt die TK-Angabe keine Rolle - z. B. bei Halbworten oder bei Masken - so sollte die Typenkennung 2 verwendet werden (Programmierkonvention).

Der Zusatz von short, full oder long bei expliziten Konstanten charakterisiert den Umfang des wiedergegebenen Objekts (Halbwort, Ganzwort, Doppelwort). Aus dem Umfang ergeben sich die in 3. 3. 1 angegebenen Beschränkungen für ganze Zahlen, Gleitpunktzahlen und Bitangaben. Ist keiner dieser Zusätze vorhanden und ist die Konstante nicht Bestandteil einer konstanten Liste, so wird full implizit ergänzt.

Die in Zeichenreihen enthaltene Zeichensequenz wird gemäß dem TR440-Zentralcode in eine Folge von 8-Bit-Gruppen (Oktaden) übersetzt. Die so entstehende Bitfolge wird mit Typenkennung 3 in einer Folge von Halb-, Ganz- bzw. Doppelworten (je nach Umfangsspezifikation) abgelegt. Das letzte dieser Speicherelemente wird bei Bedarf rechts mit der Oktade NUL aufgefüllt. Beim Zugriff auf eine Zeichenreihe wird nur der Inhalt des ersten Speicherelements geliefert.

Konstante Listen sind eindimensionale Folgen von expliziten Konstanten und/oder konstanten Referenzen. Sie werden lückenlos in aufeinanderfolgenden Speicherzellen schreibgeschützt abgelegt. In Klammern eingeschlossene (Teil-) Listen werden dabei so oft wiederholt, wie die davor stehende Wiederholungszahl angibt; fehlt diese, so ist 1 zu ergänzen. Der Umfang der einzelnen Objekte ergibt sich wie folgt:

Handelt es sich um eine explizite Konstante, die eines der Schlüsselworte short, full oder long enthält, so ergibt sich der Umfang wie oben ausgeführt. Handelt es sich um eine sonstige explizite Konstante oder eine konstante Referenz, so bestimmt sich der Umfang aus der zur konstanten Liste gehörigen Voreinstellung, die sich aus dem Kontext ergibt (5. 2. 2, 7. 2. 3). Möglich sind die Voreinstellungen short, full oder long. Die Voreinstellung wird für die Elemente einer geklammerten Teilliste überschrieben, wenn dieser eines der Schlüsselworte short, full oder long vorausgeht.

Elemente einer konstanten Liste vom Umfang eines Ganzworts oder Doppelworts dürfen nicht mit ungerader Anfangsadresse abgelegt werden. Daraus ergeben sich analoge Einschränkungen, wie in 2. 8. 14 bezüglich der Veränderung des Zugriffsverhaltens ausgeführt.
Bei Unterbringung einer Referenz in einem Ganzwort oder Doppelwort wird diese rechtsbündig im letzten Halbwort als 22-Bit-Adresse gespeichert.

3.3.1.
Maximallängen bzw. -Größen von Wertangaben und Bezeichnungen

Ganze Zahlen:

short	0	≦	| i |	≦	$2^{22} - 1 = 4194303$
full	0	≦	| i |	≦	$2^{46} - 1 = 70368744177663$
long	0	≦	| i |	≦	$2^{92} - 1 =$ 4951760157141521099596496895

Gleitpunktzahlen:

x = 0 und

$0.745834 \ldots {}_{10}{}^{-156} \approx 16^{-128} \leqq |x| \leqq$

$(1-2^{-38}) \cdot 16^{127} \approx 0.837988 \ldots {}_{10}{}^{153}$

Dies gilt für full und long. Bei long ist der Maximalwert mit $(1-2^{-84}) \cdot 16^{127}$ geringfügig höher. Die Genauigkeit beträgt mindestens 10 bzw. 24 Dezimalstellen.

Bitangaben:

Maximale Ziffernanzahl = p/q

p = 24 für short, p=48 für full

p = 96 für long

q = 4 für Hexadezimalzahlen,

q = 3 für Oktalzahlen,

q = 1 für Binärzahlen

Zeichenreihen:

Ist n die Anzahl der Zeichen in der Zeichensequenz, so belegt diese

$\left[\frac{n+2}{3}\right]$ Halbworte bei short

$\left[\frac{n+5}{6}\right]$ Ganzworte bei full

$\left[\frac{n+11}{12}\right]$ Doppelworte bei long

Bezeichnungen:

Der PS440-Übersetzer verarbeitet Bezeichnungen beliebiger Länge, der TAS-Übersetzer jedoch nur Bezeichnungen bis zu 31 Zeichen. Daher dürfen selbständige Bezeichnungen (2.7) höchstens aus 31 Zeichen bestehen.

3.4.
Beispiele

```
neuesschluesselwort
BEZEICHNUNG

123
1E-1 1)        5E3    · 1      2.314_10 + 1
'2H.AFFE'      '1H 3E8'
'20.537 77'    '101750'
'2B.1010 1111 1111 111'        '1B 1 1110 1000'
" *207 *210 *180 *176"
*/-- TAS-SEQUENZ --/*
CO DAS IST EIN KOMMENTAR;
'WORTSYMBOL'
short 1 long 1.0
(full " ' ' ' ' KONSTANTE LISTE ' ' ' ' ' ' ,
      long 1 , short 16 ( 2 ),
      full 2   (ref X [4] , short (1,2 )))
```

Unzulässig sind

```
full ( 1, short 2,3 )
full 2 ( 1, short 2 )
```

---

1) Man beachte, daß E-1 einen Ausdruck, aber nicht die Gleitpunktzahl 1E-1 darstellt. Dementsprechend ist auch $_{10}$-1 anstelle von 1_{10}-1 verboten.

4. PROGRAMMAUFBAU

4.1. Syntax

4.1.1.	⟨Programm⟩	::=	{ ⟨Segment⟩ ||;}$_1$ finis
4.1.2.	⟨Segment⟩	::=	⟨Anweisungssegment⟩ | ⟨Prozedursegment⟩
4.1.3.	⟨Anweisungssegment⟩	::=	⟨Segmentbenennung⟩; ⟨Segmentrumpf⟩
4.1.4.	⟨Segmentbenennung⟩	::=	segment ⟨Bezeichnung⟩
4.1.5.	⟨Segmentrumpf⟩	::=	⟨Rumpf⟩
4.1.6.	⟨Rumpf⟩	::=	{⟨Anweisung⟩ || ; }
4.1.7.	⟨Anweisung⟩	::=	⟨markierte Anweisung⟩ | ⟨Deklaration⟩
4.1.8.	⟨markierte Anweisung⟩	::=	{⟨Markendefinition⟩} ⟨eigentliche Anweisung⟩
4.1.9.	⟨Markendefinition⟩	::=	{global} $^{1}_{0}$ ⟨Marke⟩ :
4.1.10.	⟨Marke⟩	::=	⟨Bezeichnung⟩
4.1.11.	⟨Prozedursegment⟩	::=	segment ⟨Prozedurvereinbarung⟩
4.1.12.	⟨Block⟩	::=	begin ⟨Rumpf⟩ end {⟨Bezeichnung⟩}

4.2.
Semantik

Segmente sind die Gültigkeitsbereiche lokaler Größen (2.6) [2]. Jedes Segment hat eine "Segmentbezeichnung", nämlich die in der Segmentbenennung vorkommende Bezeichnung bzw. bei Prozedursegmenten die zugehörige Prozedurbezeichnung (5.1.24). Die Segmentbezeichnung des ersten Segments liefert zugleich den Programmnamen (2.2).

Die Segmentbenennung definiert die Segmentbezeichnung als globale Marke und stellt somit eine andere Schreibweise einer Markendefinition dar. Bei Sprüngen auf diese Marke wird die erste ausführbare Anweisung (5.2, 6.2) des Segmentrumpfs als nächste Anweisung ausgeführt.

Markendefinitionen kennzeichnen Programmstellen (2.4, 2.8.2).

Blöcke sind zusammengesetzte Anweisungen. Die Ausführung eines Blocks besteht normalerweise in der Ausführung der im Block enthaltenen Anweisungen in der Reihenfolge des Anschreibens, beginnend mit der ersten Anweisung (man vergleiche jedoch 2.3.) Die auf end eines Blocks folgenden Bezeichnungen sind semantisch bedeutungslos und stellen eine spezielle Form eines Kommentars dar. Für die eigentlichen Anweisungen vergleiche man 6, für Prozedurvereinbarungen 5.1.23, 2.8.3, 4, für Benennungen 8.

4.3.
Beispiel

Das folgende Beispiel zeigt die Umsetzung von Segmenten in TAS-Segmente.

PS440

```
segment proc P is
begin ra : = 1 end

segment Q; full X;
M: P; X : = ra; goto M

finis
```

TAS

```
 P. = SEGM --- P --
 BA 1,
MU S 0,
Q. = SEGM --- Q --
X  = ASP  2/G,
M  = SU P,
     C X,
     S M,
ENDE,
```

[2] Die Blockstruktur hat keinen Einfluß auf die Bestimmung von Gültigkeitsbereichen!

5. DEKLARATIONEN

5.1. Syntax

5.1.1. ⟨Deklaration⟩ ::= ⟨Spezifikation⟩ | ⟨Vereinbarung⟩

5.1.2. ⟨Spezifikation⟩ ::= ⟨Eingangs-Spezifikation⟩ |
⟨eigentliche Spezifikation⟩

5.1.3. ⟨Eingangs-Spezifikation⟩ ::= entry
{⟨Bezeichnung⟩ ||, }$_{1}$

5.1.4. ⟨eigentliche Spezifikation⟩ ::= spec {global}$_{0}^{1}$
⟨Spezifikationskopf⟩
{⟨Bezeichnung⟩ ||, }$_{1}$

5.1.5. ⟨Spezifikationskopf⟩ ::= {⟨Dimensionsangabe⟩}$_{0}^{1}$
⟨einfache Art⟩ |
⟨Prozedurart⟩ | label

5.1.6. ⟨Dimensionsangabe⟩ ::= record
⟨Länge⟩ | [0 : ⟨obere Grenze⟩]

5.1.7. ⟨Länge⟩ ::= ⟨ganze Zahl⟩

5.1.8. ⟨obere Grenze ⟩ ::= ⟨ganze Zahl⟩

5.1.9. ⟨einfache Art⟩ ::= ⟨Typ⟩ {data}$_{0}^{1}$ | index

5.1.10. ⟨Typ⟩ ::= short | full | long

5.1.11. ⟨Prozedurart⟩ ::= proc | procsfb

5.1.12. ⟨Vereinbarung⟩ ::= ⟨Extern-Vereinbarung⟩ |
{global}$_{0}^{1}$
⟨lokale Vereinbarung⟩

5.1.13. ⟨Extern-Vereinbarung⟩ ::= spec
⟨externer Programmname⟩
⟨Spezifikationskopf⟩
{⟨Bezeichnung⟩ ||, }$_1$

5.1.14. ⟨externer Programmname⟩ ::= ⟨Bezeichnung⟩

5.1.15. ⟨lokale Vereinbarung⟩ ::= ⟨selbständige Vereinbarung⟩ |
⟨Äquivalenzvereinbarung⟩ |
⟨Selektorvereinbarung⟩ |
⟨Teilwortselektorvereinbarung⟩ |
⟨Befehlsvereinbarung⟩ |
⟨Makrovereinbarung⟩

5.1.16. ⟨selbständige Vereinbarung⟩ ::= ⟨Typvereinbarung⟩ |
⟨Indexspeichervereinbarung⟩ |
⟨Prozedurvereinbarung⟩

5.1.17. ⟨Typvereinbarung⟩ ::= {⟨Dimensionsangabe⟩}$^1_0$
⟨Typ⟩ {data}$^1_0$
{⟨Typelement⟩ ||, }$_1$

5.1.18. ⟨Typelement⟩ ::= ⟨Bezeichnung⟩
{⟨statische Vorbesetzung⟩ |$_1$
⟨dynamische Vorbesetzung⟩}$_0$

5.1.19. ⟨statische Vorbesetzung⟩ ::= is ⟨konstante Liste⟩ |
is ⟨TAS-Sequenz⟩ |
preset ⟨konstante Liste⟩ |
preset ⟨TAS-Sequenz⟩

5.1.20. ⟨dynamische Vorbesetzung⟩ ::= := ⟨Ausdruck⟩

5.1.21. ⟨Indexspeichervereinbarung⟩ ::= {⟨Dimensionsangabe⟩}$^1_0$ index
{⟨Indexspeicherelement⟩ ||, }$_1$

5.1.22. ⟨Indexspeicherelement⟩ ::= ⟨Bezeichnung⟩
{⟨dynamische Vorbesetzung⟩}$^1_0$

5.1.23. ⟨Prozedurvereinbarung⟩ ::= ⟨Prozedurart⟩
⟨Prozedurbezeichnung⟩
is ⟨Prozedurrumpf⟩

5.1.24. ⟨Prozedurbezeichnung⟩ ::= ⟨Bezeichnung⟩

5.1.25. ⟨Prozedurrumpf⟩ ::= ⟨markierte Anweisung⟩

5.1.26. ⟨Selektorvereinbarung⟩ ::= ⟨Typ⟩
select {⟨Bezeichnung⟩ =
⟨konstanter Index⟩ ‖ , }$_1$

5.1.27. ⟨konstanter Index⟩ ::= ⟨bra⟩
⟨ganze Zahl⟩ ⟨ket⟩

5.1.28. ⟨bra⟩ ::= (| [

5.1.29. ⟨ket⟩ ::=] |)

5.1.30. ⟨Teilwortselektorvereinbarung⟩ ::= part select{⟨Bezeichnung⟩ =
{⟨konstanter Index⟩}$^1_0$
⟨Bits-Angabe⟩ ||, }$_1$

5.1.31. ⟨Äquivalenzvereinbarung⟩ ::= {⟨Dimensionsangabe⟩}$^1_0$⟨Typ⟩
{⟨Äquivalenz⟩||, }$_1$ |
{⟨Dimensionsangabe⟩}$^1_0$ index
{⟨Äquivalenz⟩||, }$_1$ |
part {⟨Äquivalenz⟩||, }$_1$

5.1.32. ⟨Äquivalenz⟩ ::= ⟨Bezeichnung⟩ = ⟨Benennung⟩

5.1.33. ⟨Befehlsvereinbarung⟩ ::= instruct
{(⟨Ergebnisregister⟩)}$^1_0$
⟨neues Schlüsselwort⟩=
⟨TAS-Sequenz⟩

5.1.34. ⟨Ergebnisregister⟩ ::= ra | rq | rd | rh | raq |
bb

5.1.35. ⟨Makrovereinbarung⟩ ::= macro {stat | ⟨Typ⟩ expr |
⟨einfache Art⟩ var}$^1_1$
⟨formales Makro⟩ is {newid
{⟨Bezeichnung⟩ ‖, }$_1$}$^1_0$
⟨Block⟩

5.1.36. ⟨formales Makro⟩ ::= {⟨Makroparameter⟩}$^1_0$
⟨neues Schlüsselwort⟩
{⟨Makroparameter⟩
⟨neues Schlüsselwort⟩}
{⟨Makroparameter⟩}$^1_0$

5.1,37. ⟨Makroparameter⟩ ::= ⟨Makrotyp⟩
⟨Bezeichnung⟩

5.1.38. ⟨Makrotyp⟩ ::= stat | expr | cond | var | newid

Für Beschreibungszwecke werden folgende syntaktischen Variablen definiert:

5.1.39. ⟨Kopf⟩ ::= entry | spec {global}$^{1}_{0}$ ⟨Spezifikationskopf⟩ |
spec ⟨externer Programmname⟩ ⟨Spezifikationskopf⟩ |
{global}$^{1}_{0}$ ⟨Spezifikationskopf⟩ |
⟨Typ⟩ select | part {select}$^{1}_{0}$ |
instruct {(⟨Ergebnisregister⟩)}$^{1}_{0}$

5.1.40. ⟨Deklarationselement⟩ ::= ⟨deklarierte Bezeichnung⟩
{⟨statische Vorbesetzung⟩ |
⟨dynamische Vorbesetzung⟩ |
is ⟨Prozedurrumpf⟩ |
= ⟨konstanter Index⟩ |
= {⟨konstanter Index⟩}$^{1}_{0}$
⟨Bits-Angabe⟩ |
= ⟨Benennung⟩}$^{1}_{0}$ |
⟨deklariertes Schlüsselwort⟩
= ⟨TAS-Sequenz⟩

5.1.41. ⟨deklarierte Bezeichnung⟩ ::= ⟨Bezeichnung⟩

5.1.42. ⟨deklariertes Schlüsselwort⟩ ::= ⟨neues Schlüsselwort⟩

5.2. Semantik

Für die Definition von Ausdrücken siehe 7, von Benennungen 8. Die Semantik von Makrovereinbarungen findet sich in 5.3.

Mit Ausnahme von Makrovereinbarungen besteht jede Deklaration aus einem Kopf und einem oder mehreren Deklarationselementen. Sind mehrere Deklarationselemente vorhanden, so ist die Deklaration semantisch gleichwertig mit der Folge von Deklarationen, die man erhält, indem man die Kommata zwischen den Deklarationselementen durch einen Strichpunkt, gefolgt vom Kopf der ursprünglichen Deklaration, ersetzt.

Eine Vereinbarung, die eine dynamische Vorbesetzung enthält (5.1.18, 22), ist semantisch gleichwertig mit der durch Strichpunkt getrennten Folge aus der Vereinbarung ohne die dynamische Vorbesetzung und der Wertzuweisung

⟨deklarierte Bezeichnung⟩

⟨dynamische Vorbesetzung⟩

Mit Ausnahme dieser aus dynamischer Vorbesetzung entstehenden Wertzuweisung sind Deklarationen nicht-ausführbare Anweisungen (2.1), die im erzeugten Programm höchstens zu Speicherbelegungen führen (vgl. die Beispiele in 5.4).

Im folgenden wird vorausgesetzt, daß die eben erwähnten Ersetzungen ausgeführt seien.

Eingangs-Spezifikationen kennzeichnen Bezeichnungen als "Kontaktnamen", die in getrennt übersetzten Programmen angesprochen werden können. Eigentliche Spezifikationen und Vereinbarungen spezifizieren die Attribute von Größen in der in 2.8 beschriebenen Weise. Vereinbarungen sind darüberhinaus die Definitionen der Größen und beschreiben die von den Größen identifizierten Objekte, Selektoren, Teilwortselektoren oder Befehle (vgl. 2.4-2.8). Für die erlaubten Reihenfolgen von Spezifikation, Vereinbarung und Anwendung einer Größe vergleiche man 2.6. Spezifiziert werden dürfen nur selbständige Bezeichnungen;die eigentliche Spezifikation und die Vereinbarung bzw. Markendefinition müssen die gleichen Attribute spezifizieren. Liegt das Attribut Verbund vor (2.8.11), so müssen in der Spezifikation und der Vereinbarung die Längenangaben übereinstimmen.

5.2.1. Eingangs-Spezifikation und Extern-Vereinbarung

Extern-Vereinbarungen definieren selbständige Objekte, ohne jedoch den zugehörigen Speicher zu reservieren. Vielmehr wird erwartet, daß diese Reservierung durch eine selbständige Vereinbarung bzw. Markendefinition in einem getrennt übersetzten Programm vorgenommen wurde. Dieses Programm ist durch einen Programmnamen (2.2) in der Extern-Vereinbarung zu kennzeichnen. Die durch Extern-Vereinbarung definierte Bezeichnung muß im anderen Programm in einer Eingangs-Spezifikation erscheinen.

Durch die Extern-Vereinbarungen werden im allgemeinen die gleichen Attribute spezifiziert wie durch die ursprüngliche Vereinbarung; der Umfang und der Zugriff auf das definierte Objekt kann jedoch auch verändert werden. Hierbei sind die gleichen Einschränkungen wie bei der Definition selbständiger Objekte durch Äquivalenz-Vereinbarung zu beachten. (2.8.14). Die Attribute part oder index sind in der Extern-Vereinbarung unzulässig.

Ein externer Programmname in einer Extern-Vereinbarung wird aufgrund seiner syntaktischen Stellung identifiziert und darf daher mit anderswo definierten Bezeichnungen übereinstimmen.

5.2.2. Statische Vorbesetzung

Statische Vorbesetzungen mit Verwendung des Schlüsselworts is spezifizieren das Attribut konstant (2.8.12), während eine Vorbesetzung unter Verwendung von preset nur die beim Laden des Programms vorzunehmende Besetzung des nicht-schreibgeschützten Bereichs wiedergibt. Bei Vorbesetzung mit konstanter Liste (3.3) ist die Voreinstellung für den Umfang der einzelnen Listenelemente durch das Hauptattribut der deklarierten Bezeichnung gegeben. Der Gesamtumfang der konstanten Liste muß bei Vorbesetzung mit is genau gleich dem Gesamtumfang des definierten Objekts sein; bei Verwendung von preset darf der Umfang der konstanten Liste den des definierten Objekts nicht überschreiten. Bei Vorbesetzung mit TAS-Sequenzen stehen diese als TAS-Einschübe. Es sind die in 2.9 angegebenen Bedingungen zu beachten. Der Umfang des TAS-Einschubs regelt sich wie für konstante Listen beschrieben, wird aber vom Übersetzer nicht kontrolliert.

5.3. Makrovereinbarungen und Makroaufruf

Makros sind vorläufig nicht implementiert. Die Grammatik 5.1.35-5.1.38 und die nachfolgenden Ausführungen könnten bei der Implementierung noch Änderungen erfahren.

Makroaufrufe sind im wesentlichen Prozeduraufrufe mit oder ohne Parameter, bei denen jedoch der Prozedurrumpf an der Aufrufstelle textlich in das Programm eingesetzt wird, nachdem zuvor die aktuellen Makroparameter in den Rumpf eingesetzt werden. Das Einsetzen kann auch rekursiv erfolgen: der Rumpf oder die aktuellen Makroparameter dürfen weitere Makroaufrufe enthalten. Jedoch muß das Einsetzen nach endlich vielen Schritten enden; daher darf z. B. der Rumpf keinen Aufruf des gleichen Makros enthalten.

Makros in PS440 dienen gleichzeitig der Einführung neuer sprachlicher Ausdrucksmittel, die es gestatten, häufig vorkommende Konstruktionen in übersichtlicher Form wiederzugeben. Makroaufrufe werden daher anders geschrieben als Prozeduraufrufe.

Ein Makro ist gegeben durch die im formalen Makro vorkommenden neuen Schlüsselworte. Das erste dieser Schlüsselworte hat das Attribut primäres Makrokennzeichen (2.7.14) und identifiziert das Makro eindeutig. Die weiteren Schlüsselworte haben das Attribut sekundäres Makrokennzeichen (2.7.15) und dienen lediglich als Begrenzer zwischen den Parametern; sie müssen nicht alle verschieden sein. Anstelle von Schlüsselworten mit dem Attribut sekundäres Makrokennzeichen dürfen auch Kommata und Doppelpunkte verwendet werden.

Ein Makroaufruf ersetzt syntaktisch eine eigentliche Anweisung (stat), einen Ausdruck (short expr, full expr, long expr) oder eine Benennung (short var, short data var, full var, full data var, long var, long data var, index). Dies wird durch eine der angegebenen Schlüsselwortkombinationen nach dem Symbol macro in der Vereinbarung angezeigt.

Ein Makroaufruf besteht aus einer Folge von aktuellen Makroparametern und Schlüsselworten in der durch das formale Makro (5.1.36) spezifizierten Reihenfolge. Die aktuellen Makroparameter haben den durch den Makrotyp angezeigten syntaktischen Aufbau: stat für Anweisung, cond für Bedingung, expr für Ausdrücke mit einem Ergebnis entsprechenden Umfangs, var für Benennungen, newid für Bezeichnungen. Dabei ist zu beachten, daß der erste Makroparameter keine "syntaktisch niedrigere" Position haben darf als das gesamte Makro, wenn dem Parameter kein Schlüsselwort vorausgeht: Ein Makro, das syntaktisch anstelle einer Benennung steht, darf nur mit einer Benennung oder einer Bezeichnung beginnen; steht es anstelle eines Ausdrucks, so darf es nur mit einer Bezeichnung, einer Benennung oder einem Ausdruck, nicht aber mit einer Bedingung oder Anweisung beginnen.

Die Bezeichnung eines Makroparameters heißt formaler Makroparameter. Die Bezeichnungen, die nach dem eventuell auf das Symbol is folgenden Schlüsselwort newid aufgelistet sind, heißen Leerparameter. Formale und Leerparameter müssen untereinander verschieden sein. Ihr Gültigkeitsbereich ist auf den "Makrorumpf", nämlich den auf is folgenden Block, beschränkt.

Bei Verarbeitung eines Makroaufrufs durch den Übersetzer werden die aktuellen Makroparameter an den durch die entsprechenden formalen Parameter gekennzeichneten Stellen in den Makrorumpf eingesetzt. Außerdem wird für jeden Leerparameter eine neue Bezeichnung erzeugt und an entsprechenden Stellen in den Rumpf eingesetzt. Der so abgeänderte Rumpf wird anstelle des ursprünglichen Aufrufs in das Programm eingesetzt und weiter verarbeitet.

Leerparameter und durch newid gekennzeichnete formale Makroparameter sind nur sinnvoll, wenn der Makrorumpf eine Deklaration oder Markendefinition enthält. Dies sind jedoch die einzigen Deklarationen oder Markendefinitionen, die in einem Makrorumpf vorkommen dürfen.

5.4.
Beispiele

5.4.1.

PS440

```
entry A, BC, D
spec proc Q
spec global [0:3]
  full x, y, z
spec RAHMEN label
  ERROR
spec RAHMEN full
  TIME
short S
global [0:999]
  full KELLER;
long data L
record 10 full R1
  is (short 4
  (''A'') , 1,1,1)


short data SA is
*/S A/K /*


long LA preset ref
  R1
global index I,
J:=0   K:=1
```

TAS

```
EINGG (A, BC, D)

EXTERN RAHMEN (ERROR)
EXTERN RAHMEN (TIME)
S = ASP 1
KELLER. = ASP 1000/G
L = ASP 4/DG

R1 = ''A''/HG,
     ''A''/H,
     ''A''/H,
     ''A''/H,
     1,1,1

ABLAGE **0 (K),
SA = S A/K,
AEND **0 (K)

LA = 0/V, 0/VH, R1/AV

INDEX (I.),
INDEX (J.),
ZX 0 J,
INDEX (K.),
ZX 1 K,
```

5.4.2.
Prozedurvereinbarung

Die nachfolgend angegebene Prozedur ist rekursiv aufrufbar:

```
global [0:499] full KELLER;
global index I:=-2;
co DIE VORBESETZUNG IST VOR DEM ERSTEN AUFRUF
   AUSZUFUEHREN;
segm proc FAC is
co FAKULTAET, PARAMETER UND ERGEBNIS IN ra MIT TK1;
begin I +:=2;  co DIESE ANWEISUNG VERAENDERT ra NICHT;
      full N  = KELLER [I];
           N: = ra;
      if ¬ tk 1 ra ∨ ra < 0 ∨ra > 16 then
           I-: = 2;
           ra: = ref "FAC: UNZULAESSIGER PARAMETER";
           goto ERROR;
           co bu MUSS BEI MARKE ERROR HERUNTERGE-
                 ZAEHLT WERDEN;
elsf  N = 0 then ra := 1 else ra := N-1;
FAC;
ra := ra x N fi;
I-:= 2
end
```

In TAS liefert das Beispiel

```
KELLER. = ASP 500/G,
INDEX (I.), ZX -2 I, S **5,
FAC. = SEGM,
HXP 2 I,
MF I, C KELLER,
STN **4 1A,
SK0 **4,
BAR 16,
SGG **1,
**4 = HXP -2 I,
BA  ("FAC: UNZULAESSIGER PARAMETER"),
S ERROR, S **2,
**1 = MF I, B KELLER,
SN0 **3,
BA 1, S **2,
**3 = MF I, B KELLER, SBA  1,
SU FAC,
```

```
MF I, ML KELLER, R B Q,
**2 = HXP -2 I,
MU S 0,
**5 = . . . .
```

5.4.3.
Selektor- und Äquivalenzvereinbarungen

```
full select PUFANF = [2]
part select BIT3BIS10 = '2H C03 FFF FFF FFF'
full PFA = PUFANF of PUFFER
full PFAA = PUFFER [2]
short KPEG = KELLER [PEGEL]
[0:29] index RSP = INDFREI [I+1]
part K10B3B10 = BIT3BIS10 of KELLER [I+10]
```

5.4.4.
Beispiel einer Makrovereinbarung

```
macro stat &for var P1 &from expr P2 &by expr P3 &to expr P4
&while cond P5 &do stat P6 &unless cond P7 &continue stat P8

is newid LP1, LP2, LP3;
  begin short LP1 := P3, LP2  := P4;
P1 := P2;
LP3: if (P1-LP2) x LP1 ≤ 0 ∧ (P5) then P6;
if ¬ (P7) then P8;
P1 + := LP1;
goto PL3 fi fi

end
```

Ein Aufruf wäre etwa:

```
&for I &from 0 &by 2 &to 200 &while 0=C &do
begin ra := PUF [I]; DRUCKE;
      ra := PUF [I+2]  end

&until¬ tk 3 ra
&continue begin ra := ZEILENWECHSEL;
                      ZEICHENAUS  end
```

5.4.5. Vereinbarung dynamischer Arbeitsspeicher

Benötigt ein Programm Arbeitsspeicher, der erst dynamisch angemeldet werden soll, so ist im Adressenraum eine Lücke freizulassen, die dem maximal zu erwartenden Speicherbedarf entspricht. Diese Lücke ist in PS440 z. B. auf folgende Weise über eine Bezeichnung erreichbar:

```
*/A = FZONE V; ABLAGE A (V0), ASP 0K /*;
record '1H5000' full A0;
*/AEND (V0) /*
```

Der Pseudobefehl ASP 0K gewährleistet, daß der Verbund A0 an einer Seitengrenze beginnt. Über Äquivalenzvereinbarungen können anschließend Adressen in der Lücke mit Bezeichnungen belegt werden.

5.5. Standardvereinbarungen

Die folgenden Größen sind in jedem PS440-Programm ohne Deklaration verfügbar:

HW, GW, DW, LEIT und LEITSM

Die Bedeutung entspricht den folgenden Vereinbarungen :

```
global short select HW = [0];
global full select  GW = [0];
global long select  DW = [0];
global record 256 full LEIT
       is */ASP 256/KG /*;
global record 256 full LEITSM;
```

Die Bezeichnungen LEIT und LEITSM bezeichnen den Leitblock. Der Zugriff auf den Verbund LEIT erfolgt durch den BLEI-Befehl; der Verbund hat daher das Attribut konstant. Der Zugriff auf den Verbund LEITSM erfolgt durch LEI-Befehle; die Verwendung von LEITSM setzt daher voraus, daß der Programmteil im System- oder Spezialmodus läuft.

Außerdem ist die Bezeichnung &HILF& mit Sonderbedeutung belegt: Sie bezeichnet den Hilfsspeicher für Zwischenspeicherungen bei Ausdrücken und für die Unterbringung von Schrittweite und Endwert bei Laufanweisungen. Vergleiche 6.2.3, 7.3.

6. EIGENTLICHE ANWEISUNGEN

6.1. Syntax

6.1.1. ⟨eigentliche Anweisung⟩ ::= ⟨bedingte Anweisung⟩ |
⟨Fall-Unterscheidung⟩ |
⟨Laufanweisung⟩ | ⟨Block⟩ |
⟨Sprung⟩ | ⟨Prozeduraufruf⟩ |
return |
⟨Befehlsaufruf⟩ |
⟨Makroaufruf⟩ | ⟨Tausch⟩ |
⟨Ausdruck⟩ | ⟨TAS-Sequenz⟩ |
⟨leere Anweisung⟩

6.1.2. ⟨bedingte Anweisung⟩ ::= if ⟨Bedingung⟩ ⟨Ja-Anweisung⟩
{⟨Nein-Anweisung⟩}$_0^1$ fi

6.1.3. ⟨Ja-Anweisung⟩ ::= then ⟨Rumpf⟩

6.1.4. ⟨Nein-Anweisung⟩ ::= else ⟨Rumpf⟩ |
elsf ⟨Bedingung⟩
⟨Ja-Anweisung⟩
{⟨Nein-Anweisung⟩}$_0^1$

6.1.5. ⟨Fall-Unterscheidung⟩ ::= case ⟨Ausdruck⟩ from
{⟨eigentliche Anweisung⟩ ‖ , }$_1$
esac

6.1.6. ⟨Laufanweisung⟩ ::= ⟨Zählung⟩
{while ⟨Bedingung⟩}$_0^1$ do
⟨laufende Anweisung⟩

6.1.7. ⟨Zählung⟩ ::= {for ⟨Laufvariable⟩}$_0^1$
{from ⟨Anfangswert⟩}$_0^1$
{by ⟨Schrittweite⟩}$_0^1$
{to ⟨Endwert⟩}$_0^1$

6.1.8. ⟨Laufvariable⟩ ::= ⟨Benennung⟩

6.1.9. ⟨Anfangswert⟩ ::= ⟨Ausdruck⟩

6.1.10. ⟨Schrittweite⟩ ::= ⟨Ausdruck⟩

6.1.11. ⟨Endwert⟩ ::= ⟨Ausdruck⟩

6.1.12. ⟨laufende Anweisung⟩ ::= ⟨eigentliche Anweisung⟩ |
⟨eigentliche Anweisung⟩ until ⟨Bedingung⟩ {continue ⟨eigentliche Anweisung⟩}$^{1}_{0}$

6.1.13. ⟨Sprung⟩ ::= goto ⟨Benennung⟩

6.1.14. ⟨Prozeduraufruf⟩ ::= ⟨Prozedurbezeichnung⟩ |
call ⟨Benennung⟩ |
callsfb ⟨Benennung⟩ |
exec ⟨Benennung⟩

6.1.15. ⟨Befehlsaufruf⟩ ::= ⟨neues Schlüsselwort⟩
⟨Benennung⟩

6.1.16. ⟨Makroaufruf⟩ ::= {⟨aktueller Makroparameter⟩}$^{1}_{0}$
⟨neues Schlüsselwort⟩
{⟨aktueller Makroparameter⟩
⟨neues Schlüsselwort⟩}
{⟨aktueller Makroparameter⟩}$^{1}_{0}$

6.1.17. ⟨aktueller Makroparameter⟩ ::= ⟨Anweisung⟩ |
⟨Ausdruck⟩ |
⟨Bedingung⟩ |
⟨Benennung⟩ |
⟨Bezeichnung⟩

6.1.18. ⟨Tausch⟩ ::= ⟨Benennung⟩ :=: ⟨Benennung⟩

6.1.19. ⟨leere Anweisung⟩ ::= ⟨leer⟩

6.2.
Semantik

Für Blöcke vergleiche 4, für Befehlsaufrufe 2.8.10, für Makroaufrufe 5.3 [1], für Ausdrücke und Bedingungen 7, für TAS-Sequenzen 2.9. Die Ausführung der leeren Anweisung ist leer.

6.2.1.
Bedingte Anweisung

Die Ausführung einer bedingten Anweisung besteht in der Verarbeitung der Bedingung, gefolgt von der Ausführung entweder der Ja-Anweisung oder der Nein-Anweisung. Die Ja-Anweisung wird gewählt, wenn der Wert der Bedingung "wahr" ist (7.2.5).

Die Nein-Anweisung

else ⟨leere Anweisung⟩

darf weggelassen werden. Hat die Nein-Anweisung die Form

else ⟨bedingte Anweisung⟩,

so kann das if dieser (inneren) bedingten Anweisung mit else zusammengezogen werden zum Symbol elsf; das abschließende fi der inneren bedingten Anweisung wird dann weggelassen (zweite Form der Nein-Anweisung in 6.1.4).

6.2.2.
Fall-Unterscheidung

Die Ausführung einer Fall-Unterscheidung besteht in der Verarbeitung des nach case folgenden Ausdrucks und der Ausführung von genau einer der n in der Fall-Unterscheidung enthaltenen eigentlichen Anweisungen. Ist x der Wert des Ausdrucks, so wird x bei Bedarf auf short "gekürzt" (7.2.1.1) und anschließend als ganze Zahl interpretiert. Für den gekürzten Wert muß gelten

$$0 \leqq x \leqq n-1.$$

Es wird dann die x-te Anweisung ausgeführt. Liegt der Wert von x nicht im angegebenen Bereich, so ist die Ausführung der Fall-Unterscheidung undefiniert.

6.2.3.
Laufanweisung

Die Ausführung einer Laufanweisung ohne Zählung ist semantisch gleichwertig mit einem Aufruf des folgenden Makros:

```
macro stat &while cond P5 &do stat P6 &until cond P7 &continue
stat P8 is newid LP3;
begin LP3; if P5 then P6;
if ¬ (P7) then P8;
goto LP3 fi fi
end
```

[1] Die Grammatik 6.1.17 für aktuelle Makroparameter ist aus semantischen Gründen mehrdeutig.

Die Angaben

while ⟨stets erfüllte Bedingung⟩

until ⟨stets unerfüllte Bedingung⟩

continue ⟨leere Anweisung⟩

dürfen weggelassen werden.

Enthält die Laufanweisung mindestens eine der Angaben: Laufvariable, Anfangswert, Schrittweite bzw. Endwert, so spricht man von einer Laufanweisung mit Zählung. Diese ist semantisch gleichwertig mit einem Aufruf des in 5.4.4 definierten Makros. Fehlt in der Zählung die Laufvariable, so wird vom Übersetzer intern ein Indexspeicher vereinbart und als Laufvariable verwendet. Fehlt in der Zählung die Angabe eines Anfangswertes bzw. einer Schrittweite, so wird

from 0

bzw.

by 1

ergänzt. Fehlt die Angabe eines Endwerts, so entfällt die Prüfung des Endwertes; ein Endwert wird also nicht ergänzt.

Aus 5.4.4 ergibt sich, daß eine Laufanweisung mit Zählung eventuell vom Übersetzer intern vereinbarte short-Größen benötigt. Hinzu kommt eventuell der oben erwähnte Hilfs-Indexspeicher. Man beachte auch die Ausführungen in 7.3.

6.2.4. Sprung

Die Ausführung eines Sprunges bewirkt die Fortsetzung der Programmausführung an der durch die Marke identifizierten Programmstelle, falls die Benennung eine Markenbezeichnung ist, bzw. an der Adresse, die der Wert der angegebenen Benennung ist. Die Benennung darf keine Prozedurbezeichnung sein.

6.2.5. Prozeduraufruf

Die Wirkung eines durch eine Prozedurbezeichnung verlangten Prozeduraufrufs ist in 2.8.3 bzw. 2.8.4 beschrieben.

Durch die Operatoren call bzw. callsfb wird ein Unterprogrammsprung auf die als Wert der Benennung bzw. als Marke spezifizierte Adresse veranlaßt. Der Unterprogrammsprung erfolgt bei call durch SUE- bzw. SU-Befehl und bei callsfb durch SFBE- bzw. SFB-Befehl. Ein Unterprogrammsprung auf eine Marke verlangt im allgemeinen, daß die Markendefinition in einem Prozedurrumpf steht, und daß call bzw. callsfb benutzt wird je nachdem, ob diese Prozedur SU- bzw. SFB-Prozedur ist. Auf diese Weise benutzte Marken definieren Nebeneingänge für die Prozedur.

Der Operator exec erwartet rechtsbündig im Wert der Benennung einen TR440-Befehl. Die Wirkung des Prozeduraufrufs besteht in der Ausführung dieses Befehls. Sollte es sich um einen Unterprogrammsprung handeln, so gilt die Folgeadresse des ursprünglichen Prozeduraufrufs als Rückkehradresse.

6.2.6. Die Rückkehranweisung "return"

Die Rückkehranweisung darf nur in Prozedurrümpfen vorkommen. Sie veranlaßt den Rücksprung aus der Prozedur. Man vergleiche 2.8.3.4.

6.2.7. Der Tausch

Durch einen Tausch werden die Werte der beiden Benennungen vertauscht. Die beiden Benennungen müssen das gleiche Hauptattribut short, full, long oder index tragen, das Attribut konstant darf nicht vorkommen.

6.3. Beispiele

```
PS440                                   TAS

if A=0 then A:=0;                       S A,
goto M else A:=1 fi                     SN0 **1,
                                        BA 0,
                                        C A,
                                        S M,
                                        S **2,
                                  **1=  BA 1,
                                        C A,
                                  **2=  ......

case I from A:=1                        TXR AN I
            goto M,                     SBA 1,
            A:=2,                       S **3,
            I:=1 esac             **1=  BA 1,
                                        C A,
                                        S **4,
                                  **2=  BA 2,
                                        C A,
                                        S **4,
                                        ZX 1 I,
                                        S **2,
                                        S M,
                                        S **1,
                                  **3=  RLR T A,
                                  **4=  ........

A:=:B                                   BC A, BC B, BC A,
```

Mit den Definitionen

index I; short N; [0:200] full A; full X

ergeben die folgenden Laufanweisungen nebenstehenden Code:

```
for I by 2 to N                    ZX -2 I,
do A [I] :=0                       B2V N,
                                   C2 &HILF&,
                              **1= HXP 2 I,
                                   VBC &HILF&,
                                   SXG **2,
                                   BA 0,
                                   MF I,
                                   C A,
                                   S **1,
                              **2= .......

while I≤N do I +:= A [I]      **1= XB 1,
until I=A [I]                      VBC N,
                                   SXG **2,
                                   MF I,
                                   B A,
                                   RX AC I,
                                   MF I,
                                   B A,
                                   TXR H I,
                                   SN **1,
                              **2= ........
```

PS440

```
for I by 2
do X:=A [I]++ A [I+2]
until X=0. 0
continue A[I] :=
         A [I]//X
```

TAS

```
      ZX -2 I,
**1=  HXP 2 I,
      MF I,
      B A,
      MF I,
      GA A+2,
      C X,
      SI0 **2,
      MF I,
      B A,
      GDV X,
      MF I,
      C A,
      S **1,
**2 = ........
```

Sind M1, M2, M3 Marken, so sind die folgenden Anweisungssequenzen gleichwertig:

```
index I;
case I from goto M1,
goto M2, goto M3 esac
```

```
index I;
[0:2] short M
is (ref M1,
ref M2,
ref M3) ;
goto M [I]
```

7. AUSDRÜCKE UND BEDINGUNGEN

7.1. Syntax

7.1.1.	⟨Ausdruck⟩	::=	⟨Wertzuweisung⟩ | ⟨Formel⟩ | ⟨Makroaufruf⟩
7.1.2.	⟨Wertzuweisung⟩	::=	⟨Benennung⟩ ⟨Zuweisungsoperator⟩ ⟨Ausdruck⟩
7.1.3.	⟨Zuweisungsoperator⟩	::=	:= |+:= |-:= |++:= |--:=
7.1.4.	⟨Formel⟩	::=	$\{\langle\text{Vorzeichen}\rangle\}_0^1$ ⟨Term⟩ {⟨Additionsoperator⟩ ⟨Term⟩}
7.1.5.	⟨Additionsoperator⟩	::=	+ | - |++ | --| vel | aut
7.1.6.	⟨Term⟩	::=	⟨Faktor⟩ {⟨Multiplikationsoperator⟩ ⟨Faktor⟩}
7.1.7.	⟨Vorzeichen⟩	::=	+ | -
7.1.8.	⟨Multiplikationsoperator⟩	::=	x | / | xx | // | xxx | mod | dmod | et
7.1.9.	⟨Faktor⟩	::=	⟨Primärausdruck⟩ {⟨Verschiebeoperator⟩ ⟨Primärausdruck⟩}
7.1.10.	⟨Verschiebeoperator⟩	::=	left | right | leftc | rightc
7.1.11.	⟨Primärausdruck⟩	::=	⟨Benennung⟩| ⟨explizite Konstante⟩| (⟨Ausdruck⟩)| ⟨Primäroperator⟩ ⟨Primärausdruck⟩| ⟨Referenzausdruck⟩| ⟨Befehlsaufruf⟩

7.1.12. ⟨Primäroperator⟩ ::= abs | tk0 | tk1 | tk2 | tk3

7.1.13. ⟨Referenzausdruck⟩ ::= ref ⟨Benennung⟩|
ref ⟨konstante Liste⟩

7.1.14. ⟨Bedingung⟩ ::= ⟨Disjunktion⟩

7.1.15. ⟨Disjunktion⟩ ::= ⟨Konjunktion⟩
{∨⟨Konjunktion⟩}

7.1.16. ⟨Konjunktion⟩ ::= ⟨Negation⟩
{∧⟨Negation⟩}

7.1.17. ⟨Negation⟩ ::= ⟨Test⟩|¬ ⟨Test⟩

7.1.18. ⟨Test⟩ ::= ⟨Relation⟩|
(⟨Bedingung⟩)|⟨Alarmtest⟩|
⟨TK-Test⟩|⟨ML-Test⟩|
⟨WS-Test⟩|⟨ODD-Test⟩|
⟨BIT-Test⟩|⟨Markentest⟩

7.1.19. ⟨Relation⟩ ::= ⟨Formel⟩
⟨Vergleichsoperator⟩
⟨Formel⟩

7.1.20. ⟨Vergleichsoperator⟩ ::= = | ≠ |< |≤ | > | ≥

7.1.21. ⟨Alarmtest⟩ ::= aral | tkal

7.1.22. ⟨TK-Test⟩ ::= tk ⟨TK-Angabe⟩
⟨Ausdruck⟩

7.1.23. ⟨BIT-Test⟩ ::= bit ⟨BIT-Nummer⟩
⟨Ausdruck⟩

7.1.24. ⟨ML-Test⟩ ::= ml {⟨ML-Nummer⟩|,}$_1$|
mll {⟨ML-Nummer⟩|,}$_1$

7.1.25. ⟨ML-Nummer⟩ ::= 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8

7.1.26. ⟨WS-Test⟩ ::= ws {⟨ML-Nummer⟩|,}$_1$

7.1.27. ⟨ODD-Test⟩ ::= odd | bodd

7.1.28. ⟨Markentest⟩ ::= rm

7.1.29. ⟨BIT-Nummer⟩ ::= 1 | 2 | 3 | ... | 48

7.2. Semantik

7.2.1. Wertzuweisung

Sei B eine Benennung, F eine Formel, dann sind die folgenden Wertzuweisungen gleichwertig:

B + := F	B := B + (F)
B - := F	B := B - (F)
B ++:= F	B := B ++(F)
B --:= F	B := B --(F)

Die linksstehende Form ergibt jedoch im allgemeinen einen günstigeren Code. Die einfache Wertzuweisung

B := F

bewirkt die Neubesetzung des durch B gegebenen Objekts mit dem Wert von F. Der neue Wert von B ist zugleich der Wert der gesamten Wertzuweisung. Die Benennung B muß ein Datenobjekt D mit einem der Hauptattribute index, full, long oder part oder ein Ergebnisregister (5.1.34) identifizieren. Wertzuweisungen an die Register rhq, ry, bu, bf sind unzulässig. Wie sich aus 7.2.2 ff ergibt, kann der Wert W der Formel F nur den durch die Attribute short, full oder long spezifizierten Umfang haben. Hat das durch B benannte Objekt D einen anderen Umfang, so gilt:

7.2.1.1. Ist D "kürzer" als W, so bezieht sich die Wertzuweisung nur auf den rechtsbündig stehenden Ausschnitt von W, der gleichen Umfang wie D hat.

7.2.1.2. Ist D "länger" als W und ist W vom Umfang short so wird W auf den Umfang full gebracht durch Linksanfügen vorzeichengleicher Stellen. Ist D vom Umfang long, W aber nicht, so wird das erste Wort von D mit der Typenkennung 1 und 0 besetzt.

Daraus ergibt sich, daß die Wertzuweisungen

FULL := -full 1 und FULL := -short 1

gleichwertig sind. Hingegen entspricht die Wertzuweisung

LONG := -full 1

nicht der Wertzuweisung

LONG := -long 1

sondern den beiden Wertzuweisungen

GW of LONG := full 0; GW of LONG [2] := -full 1.

7.2.2.
Formeln

Die Formeln dienen der Erarbeitung von Werten. Die Werte ergeben sich entweder als Werte "einfacher Operanden", nämlich von Benennungen, expliziten Konstanten, Referenzausdrücken bzw. Befehlsaufrufen, oder durch Verknüpfung solcher Operanden durch Operatoren. Das Ergebnis einer solchen Verknüpfung kann selbst wieder als Operand auftreten. Operatoren verlangen entweder einen Operanden (Vorzeichen, Primäroperatoren) oder zwei Operanden (Additions-, Multiplikations- und Verschiebeoperatoren). Sie werden gemäß dem in 7.2.7 zusammengestellten Prioritätsschema angewandt, sofern durch Klammerung nichts anderes verlangt ist. Bei gleicher Priorität werden die Operationen in der Reihenfolge von links nach rechts ausgeführt. Der Wert der Operanden bzw. der ganzen Formel hat einen bestimmten Umfang, nämlich ein Halbwort (short), ein Ganzwort (full) oder ein Doppelwort (long). Der Umfang ergibt sich rekursiv aus dem Umfang der Operanden und aus den angewandten Operatoren gemäß den Angaben in 7.2.3.4.

7.2.3.
Einfache Operanden

Eine als Primärausdruck auftretende Benennung liefert den Wert (=Inhalt) des durch sie identifizierten Objekts als Wert. Identifiziert die Benennung ein Teilwort, so ist der resultierende Wert ein Ganzwort, in dem das Teilwort rechtsbündig steht und mit Nullen aufgefüllt ist. Indexspeicher liefern ein Halbwort als Wert. Die Verwendung einer Marke oder Prozedurbezeichnung als Primärausdruck ist unzulässig.

Explizite Konstanten als Primärausdrücke liefern den durch sie gemäß 3.3 bezeichneten Wert. Bei Zeichenreihen handelt es sich also nur um das erste Speicherelement!

Referenzausdrücke liefern als Wert in einem Halbwort die (Anfangs-) Adresse des Speicherbereichs für das durch die Benennung bzw. die konstante Liste identifizierte Objekt. Die Benennung darf kein Register und kein Objekt mit dem Attribut part identifizieren. Es darf sich jedoch um Marken oder Prozedurbezeichnungen handeln. Die Voreinstellung für den Umfang der konstanten Liste ist full (vgl. 3.3).

Als Primärausdruck auftretende Befehlsaufrufe müssen in der zugehörigen Befehlsvereinbarung ein Ergebnisregister enthalten (vgl. 2.8.10, 5.1.33). Der Inhalt dieses Registers ist der Wert des Befehlsaufrufs.

7.2.4.
Operatoren

Die nachfolgenden Aufstellungen geben Auskunft über die Bedeutung der einzelnen Operatoren sowie über den Umfang der Operanden, für die sie definiert sind. Daraus ergibt sich dann der Umfang des Ergebnisses.

Aus 7.2.3 folgt, daß einfache Operanden stets einen durch short, full oder long spezifizierten Umfang haben (Teilworte werden auf full verlängert). Wird nun eine Operation auf einen Operanden bzw. auf ein Operandenpaar angewandt, für dessen Umfang sie nicht definiert ist, so wird der Umfang der Operanden gemäß den Regeln aus 7.2.1.2 erweitert: Aus short-Operanden wird durch vorzeichengleiche Verlängerung ein full-Operand, aus full-Operanden wird durch linksseitige Ergänzung von Nullen ein long-Operand.

Lassen sich durch diese Erweiterung keine Operanden zulässigen Umfangs gewinnen, so ist die gesamte Operation unzulässig. Eine automatische Verkürzung von Operanden findet nur für den zweiten Operanden einer Verschiebungsoperation statt (vgl. die Bemerkung in 7.2.4.2).

Abkürzungen: X Operand einer unären Operation; X1, X2 Operanden einer binären Operation; E Ergebnis der Operation. Die Befehlssequenzen sind so geschrieben, daß short- und full-Ergebnisse in ra anfallen.

7.2.4.1.
Unäre Operatoren

Operation	Umfang von X	Umfang von E	Bedeutung	typische, erzeugte Befehlssequenz
+X	short full long	short full long	keine	B2V X B X BZ X
-X	short full long	short full long	-X, auch auf Gleitpunkt-zahlen anwendbar	B2VN X BN X BZN X
abs X	short full	short full	|X|	B2V X, R BB A BB X
tk0 X, tk1 X, tk2 X, tk3 X,	full	full	X mit evtl. geänderter TK	B X, ZTR 0A usw.

7.2.4.2.
Binäre Operatoren

Operation	Umfang von X1, X2	Umfang von E	Bedeutung und Bedingungen	typische, erzeugte Befehlssequenz
X1 + X2	short-short	short	Festpunkt-addition	B2V X1, A2, X2
	full full	full		B X1, A X2,
	long long	long		BZ X1, A X2, AQ X2 + 2
X1 - X2	wie bei Festpunkt-additionen		Festpunkt-subtraktion	analog mit SB2, SB, SBQ
X1 * X2	short-short full-full	full full oder long	Festpunktmultiplikation (vgl. Bemerkung unten) nur für Operanden mit TK 1	B2V X1, M2 X2 B X1, ML X2, (RT AQ)

Operation	Umfang von X1, X2	Umfang von E	Bedeutung und Bedingungen	typische, erzeugte Befehlssequenz
X1 / X2	long-full	full	Festpunktdivision	BZ X1, DVD X2
X1 mod X2			Festpunktrestbildung nur für Operanden mit TK 1	BZ X1, DVD X2, RT AQ
X1 dmod X2	long-full	long	GW of E:=X1/X2 GW of E [2] := X1 mod X2 (vgl. Bemerkung unten)	BZ X1, DVD X2
X1 ++ X2	full-full	full	Gleitpunktoperationen hier für Operanden mit TK 0	B X1, GA X2
	long-long	long		BZ X1, DA X2
X1 -- X2 X1 ** X2 X1 // X2	wie bei Gleitpunktaddition			analog mit GSB, GML, GDV, DSB, DML, Die long-Division erfolgt durch Unterprogramm
X1 *** X2	full-full	long	Gleitpunktmultiplikation mit long-Ergebnis nur für Operanden mit TK 0	B X1, MLD X2
X1 vel X2	full-full	full	X1 ∨ X2	B X1, VEL X2
X1 aut X2			X1 ≠ X2 bitweise	B X1, AUT X2
X1 et X2			X1 ∧ X2	B X1, ET X2

Operation	Umfang von X1, X2	Umfang von E	Bedeutung und Bedingungen	typische, erzeugte Befehlssequenz
X1 left / X2	full-short	full	Verschiebung nach links bzw. rechts bei TK 0, TK 1 für X1 arithmetische Verschiebung, sonst logische Verschiebung bezüglich X2 vgl. Bemerkungen unten	B X1, MCF X2, SH AL 0
	long short	long		BZ X1, MCF X2, SH ZL 0
X1 right / X2	wie bei Linksverschiebung			analog mit SH A, SH Z
X1 leftc / X2	full-short	full	Verschiebung nach links bzw. rechts im Kreise, nur logische Verschiebung bezüglich X2, vgl. Bemerkung unten	B X1, MCF X2, SH ALK 0
	long-short	long		BZ X1, MCF X2, SH ZLK 0
X1 rightc / X2	wie bei Linksverschiebung im Kreise			

Bemerkungen:

(1) Als Ergebnis der Festpunktmultiplikation gilt im allgemeinen die zweite Hälfte des doppeltlangen internen Ergebnisses. Wird dieses jedoch zur unmittelbaren Weiterverarbeitung erweitert werden zu long, so wird auch die erste Hälfte des internen Ergebnisses mitgenommen und nicht durch Ergänzung von Nullen verlängert.

(2) Der Operator dmod liefert die beiden Teilergebnisse der Festpunktdivision zusammen in einem Doppelwort. Für $n = q \cdot m + r$ gilt also

$$E = q \cdot 2^{46} + r$$

(3) Der zweite Operand einer Verschiebeoperation darf auch den Umfang full oder long haben. In diesem Fall wird jedoch nur das letzte Halbwort von X2 verwendet.

Mit der Deklaration

```
full F;     long L;
```

ist also

```
X1 left F   bzw.  X1 left L
```

gleichwertig mit

```
X1 left HW of F [1]   bzw.
X1 left HW of L [3] .
```

Nach dieser eventuellen Verkürzung muß X2 der Bedingung

$$0 \leqq X2 \leqq 127$$

genügen, andernfalls wird die Operation falsch ausgeführt. Sinnvoll ist nur

$$0 \leqq X2 \leqq 96 .$$

7.2.5. Bedingungen

Bedingungen dienen der Erarbeitung von Wahrheitswerten. In PS440 werden solche Werte nur zur Entscheidung bei der bedingten Anweisung und als Fortsetzungs-bzw. Abbruchkriterium bei Laufanweisungen herangezogen.

Den einfachen Operanden in Formeln entsprechen die "einfachen Bedingungen", nämlich die Tests. Aus diesen können mit den Operatoren $\vee$, $\wedge$ und $\neg$ komplizierte Bedingungen aufgebaut werden. Durch Klammern wird die durch das Prioritätsschema aus 7.2.7 festgelegte Vorrangregel aufgehoben.

Die Operatoren X1 ∨ X2 und X1 ∧ X2 werden durch "Sprungkaskade" ausgewertet:

X1 ∨ X2 entspricht

if X1 then E := true else E := X2 fi [1)]

X1 ∧ X2 entspricht

if X1 then E := X2 else E := false fi [1)]

Der zweite "Operand" wird also nur verarbeitet, wenn der erste noch nicht zur Feststellung des Gesamtergebnisses ausreicht.

Die Operation ¬X ist die logische Negation:

¬X entspricht

if X then E := false else E := true fi

7.2.6. Tests

Von den in 7.1.18 ausgeführten Tests sind nur die Relationen, die geklammerten Bedingungen und der Test auf arithmetischen Alarm (aral) maschinenunabhängig. Alle anderen Tests beziehen sich auf spezielle Möglichkeiten des TR440 und sollten weitgehend vermieden werden.

7.2.6.1. Relationen

Die beiden Formeln in der Relation müssen ein full-Ergebnis aufweisen. short-Ergebnisse werden verlängert, long-Ergebnisse sind unzulässig. Haben beide Ergebnisse Typenkennung 0 bzw. Typenkennung 1, so entspricht die Relation einem Größenvergleich von Gleitpunkt- bzw. Festpunktzahlen. Andernfalls werden beide Ergebnisse als nicht-negative Festpunktzahlen von 48 Bit Länge interpretiert.

Siehe S. 130.1

Es gilt daher

+ 0 = - 0, aber tk2 (+0) ≠ tk2 (- 0)

- 1 ≦ 1, aber tk2 (- 1) > tk2 (1) .

7.2.6.2. Alarmtests

Der Alarmtest aral trifft zu, wenn die vorangehende Rechenwerksoperation einen arithmetischen Alarm (Zahlüberlauf, Division durch 0) verursachte; der Alarmtest tkal trifft zu, wenn die vorangehende Rechenwerksoperation einen Typenkennungsalarm (z. B. Gleitpunktoperationen mit Operanden, deren Typenkennung ≠ 0 ist) verursacht.

[1)] Hier bedienen wir uns der in ALGOL üblichen Schreibweise in der Bedingungen und Wahrheitswerte als Ergebnisse vor Formeln auftreten dürfen.

Beide Alarmtests verlangen, daß zwischen der den Alarm verursachenden Operation und der Abfrage keine weitere Rechenwerksoperation, insbesondere keine Abspeicherung aus einem Rechenwerksregister, stattfand. Die Sequenz

```
X := X / Y; if aral then ...
```

ist daher sinnlos, hingegen ist

```
ra := X / Y; if aral then ...
```

korrekt.

7.2.6.3. Sonstige Tests

Die TK- und BIT-Tests prüfen Typenkennung oder einzelne Bits des angegebenen Ausdrucks, dessen Wert, soweit nötig, in ein Ganzwort-Register geholt wird.

Die Merklicht- und Wahlschalter-Tests prüfen, ob mindestens eines der durch ihre Nummer angegebenen Merklichter bzw. einer der Wahlschalter gesetzt ist. Die Wahlschalter-Tests (ws ...) dürfen nur im System- oder Spezialmodus durchgeführt werden. Merklicht-Tests, die nicht mit ml , sondern mit mll beginnen, löschen zusätzlich die abgefragten Merklichter. (Ansonsten ist das Setzen und Löschen von Merklichtern nur durch TAS-Einschub zu erreichen).

Die Tests odd bzw. bodd prüfen, ob das letzte Bit in ra bzw. bb besetzt ist. Im Fall positiver Festpunktzahlen ist das der Test, ob der Registerinhalt ungerade ist. Bei nicht-positiven Zahlen sind die Eigenheiten der Zahldarstellung zu beachten.

Der Test rm prüft, ob das Markenregister besetzt ist. Dieser Test kann praktisch nur im Anschluß an TAS-Einschübe gebraucht werden.

7.2.7. Prioritäten

Die verschiedenen Operatoren werden nach dem folgenden (aus 7.1 ableitbaren) Prioritätsschema verarbeitet. Operatoren mit niedrigerer Prioritätsnummer binden schwächer.

1 :=, + :=, - :=, ++ :=, -- :=	Zuweisungsoperatoren
2 ∨	Operatoren in Bedingungen kommen zusammen mit Zuweisungsoperatoren nicht ohne Klammern vor.
3 ∧	
4 ¬	
5 <, ≦, =, ≠, ≧, >	

6	+, -, ++, --, vel, aut	Additionsoperatoren, Vorzeichen (unär)
7	x, /, mod, dmod, xx, xxx, //, et	Multiplikationsoperatoren
8	left, right, leftc, rightc	Verschiebeoperatoren
9	abs, tk0, tk1, tk2, tk3	Primäroperatoren
10	ref, ival, sval, fval, lval	Referenzausdruck, Wertoperatoren

7.3. Zwischenspeicherungen und eintrittsinvariante Programmierung

Bei der Verarbeitung von Formeln, wie etwa

A x B + C x D,

fallen Zwischenergebnisse an, die temporär zwischengespeichert werden müssen. Zur Unterbringung solcher Zwischenergebnisse und der Schrittweiten und Endwerte von Laufanweisungen vereinbart implizit der Übersetzer einen globalen selbständigen Verbund mit der Bezeichnung &HILF&. Die Anzahl der in einem Segment in diesem Verbund benötigten Plätze wird in einem speziellen Übersetzermodus ausgedruckt.

Die statische Unterbringung dieser Werte macht bei eintrittsinvarianter Programmierung Schwierigkeiten: Bei rekursiven Prozeduren ergeben sich Schwierigkeiten bezüglich der Laufanweisung (vgl. 6.2.3). Bei im Systemmodus ablaufenden Unterprogrammen, die von mehreren asynchronen Prozessen in Anspruch genommen werden, ist auch für Zwischenergebnisse die statische oder kellerartige Unterbringung unzulässig.

Die Schwierigkeiten kann der Programmierer beseitigen, indem er die Bezeichnung &HILF& in den einzelnen Segmenten seines Programms lokal durch Äquivalenzvereinbarungen definiert. Ab der durch diese Vereinbarung definierten Ganzwortadresse muß Speicher für die gemeldete Anzahl von Ganzworten bereitstehen. Dies kann vom Übersetzter naturgemäß nicht nachgeprüft werden. Der Programmierer muß daher anläßlich der Neu-Übersetzung nach Programmveränderungen die Speicherlänge mit dem gemeldeten Bedarf vergleichen.

7.4. Beispiele

Die nachfolgenden Beispiele setzen die Vereinbarungen

```
index I;  short M;  full X, Y;  long A, B;
part  select  P = '2HF';  co irgendeine Maske;
```

voraus

(Diese Beispiele erläutern vor allem das Verhalten bei Längenänderung).

```
A   := I x M + B          TXR A I, M2 M,
                          C &HILF&, BZ B,
                          AQ &HILF&, CZ A,
X   := X + Y + M          B X, A Y, C &HILF&,
                          B2V M, A &HILF&, C X,
X   := M + X + Y          B2V M, A X, A Y, C X,
X   := X x Y + A          B X, ML Y, A A,
                          AQ, A + 2, CQ X,
P   of X :=               BQ ('F'/2), BT X,
P   of X + P of Y         C &HILF&, BT Y,
                          A &HILF&, CT X,
A   := X / Y              BQ X, BA 0, DVD Y,
                          C A + 2, BA 0, C A,
A   := A // B ++ B        SFB **DDV, A/AB,
                          B/AB, DA B, CZ A,
```

(Diese Beispiele zeigen die Behandlung verschiedener Tests).

```
if  ¬ aral ∧ tkal         SAA **2, SAT **1,
    then goto M fi        S **2,
                          **1 = SE M,
                          **2 = ...
if  0 ≦ X ∧               B X, SK0 **2,
    (ml 1,2 ∨ mll 3)      SL **1 12, SNL **2 3
    then goto M fi        **1 = SE M,
                          **2 = ...
if  odd ∧ ws 2 ∧          SRN **2, SW **1 2,
    X ≧ 0                 S **2
    then goto M fi        **1 = B X, SK0 **2, SE M,
                          **2 = ...
```

Anmerkung:

Bei TK-, ML- und WS-Tests wird eventuell die Ja-Anweisung -bei Laufanweisungen: die laufende Anweisung - durch einen Relativsprung übersprungen. Der PS440-Übersetzer prüft nicht, ob die Adreßdifferenz die zulässigen Grenzen von 127 überschreitet. Dementsprechend können in solchen Fällen Fehlermeldungen aus der TAS-Übersetzung kommen.

8. BENENNUNGEN

8.1. Syntax

8.1.1.	⟨Benennung⟩	::=	⟨einfache Benennung⟩| ⟨selektierte Benennung⟩| ⟨Makroaufruf⟩
8.1.2.	⟨einfache Benennung⟩	::=	⟨Hauptbenennung⟩| ⟨dereferenzierte Benennung⟩| ⟨einfache Benennung⟩ ⟨Index⟩
8.1.3.	⟨Hauptbenennung⟩	::=	⟨Bezeichnung⟩|⟨Register⟩
8.1.4.	⟨Register⟩	::=	ra | rq | rd | rh | raq | rhq | bb | ry | bu | bf
8.1.5.	⟨dereferenzierte Benennung⟩	::=	⟨Wertoperator⟩ ⟨Hauptbenennung⟩| ⟨Wertoperator⟩ ⟨deferenzierte Benennung⟩| ⟨Wertoperator⟩⟨Index⟩
8.1.6.	⟨Wertoperator⟩	::=	val | sval | fval | lval | ival
8.1.7.	⟨Index⟩	::=	⟨bra⟩ ⟨Indexausdruck⟩ ⟨ket⟩
8.1.8.	⟨Indexausdruck⟩	::=	$\{\langle\text{Vorzeichen}\rangle\}_0^1$ ⟨Indexterm⟩ $\{\{+\|-\}_1^1\ \langle\text{Indexterm}\rangle\}$
8.1.9.	⟨selektierte Benennung⟩	::=	⟨Selektor⟩ of ⟨Benennung⟩
8.1.10.	⟨Selektor⟩	::=	⟨Bezeichnung⟩
8.1.11.	⟨Indexterm⟩	::=	$\{\langle\text{ganze Zahl}\rangle \times\}_0^1$ ⟨Benennung⟩ | $\{\langle\text{Benennung}\rangle \times\}_0^1$ ⟨ganze Zahl⟩

8.2.
Semantik

Benennungen benennen Register, Datenobjekte, Marken und Prozeduren. Ist die Hauptbenennung eine Marke oder eine Prozedurbezeichnung, so bildet sie bereits die gesamte Benennung. Weitere Zusätze sind in diesem Fall nicht erlaubt. Ein Register kann nicht indiziert werden, wohl aber kann ein Selektor darauf angewandt werden. An die Register rhq, ry, bu, bf kann keine Wertzuweisung erfolgen (vgl. 7.2.1).

Außer für den Fall, daß die Hauptbenennung ein selbständiges Objekt identifiziert und als Benennung in einem Referenzausdruck (7.1.13) auftritt, muß jede als Hauptbenennung auftretende Bezeichnung vorher deklariert sein (vgl. 2.6). Als Selektoren auftretende Bezeichnungen müssen zuvor durch eine Selektor- oder Teilwortselektorvereinbarung deklariert sein.

Sofern durch Klammern nichts anderes vorgeschrieben ist, werden, wie die Grammatik von 8.1.2, 5 und 9 zeigt, die zur Hauptbenennung möglichen Zusätze:Selektor, Wertoperator und Index, ausgehend von der Hauptbenennung in der folgenden Reihenfolge verarbeitet:

Die der Hauptbenennung folgenden Indizes, dann die Selektoren in der Reihenfolge von rechts nach links, schließlich die der Hauptbenennung unmittelbar vorausgehender Wertoperatoren in der Reihenfolge von rechts nach links.

Die Adresse und der Umfang des identifizierten Objekts ergeben sich rekursiv wie folgt:

8.2.1.

Ist die Benennung ein Register, so ist der Umfang durch die Hardware festgelegt. Eine Adresse existiert nicht.

Ist die Benennung eine selbständige Bezeichnung (2.7), so sind Adresse und Umfang durch die zugehörige Vereinbarung (5.1.16) bzw. Markendefinition (4.1.9, 4.2) und das hierdurch definierte Hauptattribut festgelegt.

Ist die Benennung eine unselbständige Bezeichnung, so unterteile man die in der zugehörigen Äquivalenzvereinbarung enthaltene Benennung anstelle der Bezeichnung [1] und beginne von vorn.

---

[1] Geht der Bezeichnung ein Wertoperator voraus oder folgt eine Indizierung nach, so wird die zu unterteilende Benennung geklammert. Auf diese Weise können abweichend von 8.1.2 auch geklammerte, insbesondere aber selektierte Benennungen indiziert werden. Jedoch darf eine solche Benennung kein Teilwort benennen.

8.2.2. Besteht die Benennung B aus einem Wertoperator und einer Benennung B', so muß das durch B' benannte Objekt rechtsbündig eine (22-Bit-) Adresse oder eine Indexspeicheradresse enthalten. Dies ist die Adresse des durch B benannten Objekts. Der Umfang dieses Objekts sowie die Unterscheidung zwischen Kernspeicher- und Indexspeicheradresse ergibt sich aus dem Wertoperator:

ival	: Indexspeicheradresse, Umfang ist ein Halbwort (Attribut index)
sval oder val	: Kernspeicheradresse, Umfang ist ein Halbwort (Attribut short)
fval	: Kernspeicheradresse, Umfang ist ein Ganzwort (Attribut full)
lval	: Kernspeicheradresse, Umfang ist ein Doppelwort (Attribut long).

Benennt B' ein Teilwort, so bedeutet "rechtbündig": rechtsbündig im Teilwort.

8.2.3. Besteht die Benennung B aus einer Benennung B' und einem Index, so ist der Umfang der durch B und B' benannten Objekte der gleiche. Die Adresse für B ergibt sich, indem man zur Adresse für B' den Wert des Indexausdrucks hinzuzählt. Die Indizierung läuft also in Halbwortschritten. Benennt B ein full- oder long-Objekt, so muß die resultierende Adresse gerade sein (vgl. 2.8.11).

Zur Berechnung des Indexausdrucks wird die Arithmetik für short-Objekte benutzt. Benennt eine im Indexausdruck vorkommende Benennung ein full- oder long-Objekt, so wird folglich nur das letzte Halbwort verarbeitet.

Aus der rekursiven Anwendung von 8.2.3 folgt, daß die Werte aufeinanderfolgender Indizes zusammengezählt werden.

Register und Benennungen von Teilworten können nicht indiziert werden.

8.2.4. Besteht die Benennung B aus einem Selektor S und einer Benennung B', so erhält man die Adresse für B, indem man auf die Adresse für B' den Index addiert, der sich aus der Vereinbarung von S ergibt (vgl. 2.8.8, 9 und 5.1.26, 30). Der Umfang des durch B benannten Objekts ergibt sich entsprechend den Angaben short, full, long bzw. part in der Vereinbarung von S.

Im letzten Fall ist der genaue Umfang durch die von der Vereinbarung von S festgelegte Maske definiert. Außer für short-Objekte muß die resultierende Adresse gerade sein.

Benennt B' bereits ein part-Objekt (Teilwort), so ist die Anwendung weiterer Selektoren unzulässig. Auf die Register ra, rq, rd und rh dürfen Halbwortselektoren (short select ...) und Teilwortselektoren angewandt werden, wenn die resultierende Benennung nicht als linke Seite von Wertzuweisungen, in Referenzausdrücken oder als rechte Seite von Äquivalenzvereinbarungen benutzt wird. Als Indizes in der Vereinbarung des Selektors sind bei Halbwortselektoren 0 (linke Registerhälfte) und 1 (rechte Registerhälfte), bei Teilwortselektoren nur 0 zulässig.

8.3. Beispiele

Gegeben seien die Vereinbarungen

```
[0:3] index I; index J; [0:99] full X;
full Y = X [J+2]; short M; long A;
short select HW2 = [2]; full select GW6 [6];
part select P1 = [2] '2HF', P2 = [0] '2HF'; co irgendeine
Maske ;
```

Anmerkung:

Die erzeugten Befehlssequenzen sind abhängig von den Optimierungsmaßnahmen der jeweiligen Übersetzerversion und müssen nicht in allen Fällen den obigen Beispielen entsprechen.

PS440	TAS
I [2] := J	TXX I+2 J
Y := GW6 of X	B X+6, MF J, C X+2
X [I] [J+2] :=	BQ ('F'/2), MF J,
P1 of GW6 of Y	BT X+10, M I,
	MF J, C X+2
X [2xJ] :=	BQ ('F'/2), MF J,
P1 of fval Y	MCF X+3, BT 2,
	XBA X, HBPX 2 J,
	MAB C 0
P1 of X [Y[I]] :=	BQ ('F'/2), R BT A
P2 of ra	M I, MF J,
	MCF X+3, CT X+2
HW2 of A := HW of ra	R B2V AL, C2 A+2
ival lval fval A :=	MCF M, MCF 0, B2V 0,
val val M	MCF A+3, MCF 1,
	MCF 3, TRX A 0

8.4. Zur Benutzung von Registern

Die Verwendung von Registern sollte soweit wie irgendmöglich vermieden werden. Sie kann als sinnvoll angesehen werden:

im Zusammenhang mit vor- oder nachfolgenden TAS-Einschüben oder Befehlsaufrufen, zur Bereitstellung von Werten für TK-Tests und ODD-Tests (7.1.18), zur Übermittlung von Parametern bei Prozeduraufrufen und bei der Rückkehr aus Prozeduren.

Außer im ersten Fall sollte man sich auf die Verwendung von ra beschränken.

Zwischen der Zuweisung an ein Register und der Verwendung des Inhalts muß garantiert werden, daß der Registerinhalt nicht durch vom Übersetzer erzeugte Operationen verändert wird. Die Erhaltung von Registerinhalten wird in folgenden Fällen gewährleistet [1]:

(1) Bei Prozeduraufrufen wird höchstens bb und bu verändert.

(2) Bei der Berechnung der Adresse von Benennungen wird höchstens bb verändert (Bei Verwendung von Teilwortselektoren wird rq und eventuell ra zerstört).

(3) Bei Wertzuweisungen an sowie Erhöhungen und Erniedrigungen des Inhalts von Indexspeichern wird höchstens bb verändert, sofern die Adressen sämtlicher vorkommender Indexspeicher statisch berechenbar sind und auf der rechten Seite der Zuweisung nur genau eine vorzeichenlose Zahl oder genau eine Benennung eines Indexspeichers steht.

(4) Bedingungen verändern keine Registerinhalte, solange nur Relationen zwischen ra und rh, ra und 0 oder bb und 0 vorkommen.

(5) In der auf eine Wertzuweisung an ein Register statisch unmittelbar folgenden Bedingung, Formel oder sonstigen nicht-zusammengesetzten Anweisung darf der Inhalt des betreffenden Registers verwendet werden. Die Erhaltung des Registerinhalts wird notfalls durch Zwischenspeicherung gewährleistet.

Bei Nichtbeachtung dieser Regeln muß mit Fehlern bei neuen Übersetzer-Versionen gerechnet werden.

Literatur:

[1] Ch. A. Lang — SAL-Systems Assembly Languages, Proc. SJCC, pp. 543-555, AFIPS, 1969.

[2] M. Richards — BCPL: A tool for compiler writing and system programming, Proc. SJCC, pp. 557-566, AFIPS, 1969.

[3] N. Wirth — PL 360, A Programming Language for the 360 Computer, J, ACM vol. 15, Nr. 1, Januar 1968.

[1] Trivialerweise wird bf laufend verändert.

STICHWORTVERZEICHNIS

II. TEIL

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bericht 7106
von
H. Wich

II. TEIL
(Bericht 7106)

I N H A L T

Seite

1. ÜBERSETZUNG EINES PS440-PROGRAMMS

Der PS 440-Übersetzer wird wie die anderen Übersetzer des Programmiersystems mit dem UEBERSETZE-Kommando gestartet, als Sprachspezifikation ist "PS 440" anzugeben, z. B.

◇ UEBERSETZE, SPRACHE = PS440, PROTOKOLL=-STD-,
QUELLE=/

Der PS440-Übersetzer verarbeitet ein PS440-Programm sequentiell, das heißt, er hat nur einen Lauf. Er übersetzt es in eine Folge von TAS-Befehlen, in ein TAS-Programm, das er in eine Texthaltungsdatei ablegt. Diese Datei wird vom PS440-Übersetzer selbst eröffnet und hat den Namen DPS440. Zur Erzeugung eines Montageobjekts muß eine TAS-Übersetzung folgen, der TAS-Übersetzer übernimmt also die Rolle eines zweiten Übersetzerlaufs, z. B.

◇ UEBERSETZE, SPRACHE=TAS, PROTOKOLL=-,
MO=MONTOBJEKT, QUELLE=DPS440

Nun muß das Programm nur noch montiert und gestartet werden, z. B.

◇ MONTIERE, MO=MONTOBJEKT, PROGRAMM=PROGR

◇ STARTE, PROGRAMM=PROGR

Der Inhalt der Datei DPS440 wird bei jeder PS440-Übersetzung ersetzt. Möchte man also mehrere PS440-Programme hintereinander übersetzen, so muß man dafür sorgen, daß die in der Datei DPS440 enthaltene Information nicht verloren geht. Am besten läßt man jedesmal sofort die TAS-Übersetzung folgen und erzeugt dabei Montageobjekte mit verschiedenen Namen oder man kopiert die Datei in eine andere um (mit dem TKOPIERE-Kommando).

2. AUFBAU EINES PS440-PROGRAMMS

Ein PS440-Programm ist eine Folge von Segmenten (entsprechend den Segmenten in TAS), es wird abgeschlossen durch das Wortsymbol FINIS.

Ein Segment ist eine Folge von Deklarationen, Spezifikationen und Statements, die fast beliebig gemischt sein können; es beginnt mit der Angabe SEGMENT ⟨Segmentname⟩; und endet mit der nächsten derartigen Angabe oder mit dem Programmende.

Trennzeichen ist das Semikolon.

3. DIE ELEMENTE DER SPRACHE ALLGEMEIN

3.1. Wortsymbole

Wortsymbole können in ALGOL-Schreibweise geschrieben werden, z. B. 'FINIS', Zwischenräume haben hier keine Bedeutung. Bequemer ist aber die Schreibweise ohne Apostroph, z. B. FINIS, hier gilt der Zwischenraum als Trennzeichen.

3.2. Bezeichnungen

Eine Bezeichnung muß mit einem Buchstaben beginnen, der dann von einer beliebigen Folge von Buchstaben und Ziffern gefolgt sein kann. Zwischenräume sind nicht erlaubt. Der Übersetzer verwendet als Sonderbuchstabe das "kommerzielle Und" (&), es sollte aber vom Programmierer nicht verwendet werden, um Kollisionen mit den vom PS440-Übersetzer erzeugten Marken zu vermeiden.

PS440-Bezeichnungen können beliebig lang sein. Es ist aber zu beachten, daß der TAS-Übersetzer nur 32 Zeichen zur Unterscheidung heranzieht.

3.3. Wertangaben

a) Ganze Zahl (Festpunktzahl):

Eine Folge von Ziffern (0, 1, 2, ... 8, 9), z. B. 987654

Es wird ein Wort mit Typenkennung 1 erzeugt.

b) Gleitpunktzahl:

ALGOL-Schreibweise, statt der Basis-Zehn kann auch E geschrieben werden.

Es wird ein Wort mit Typenkennung 0 erzeugt.

c) Bitangabe:

Hier muß die Typenkennung und ein Kennzeichen angegeben werden, mit dem unterschieden werden kann, ob es sich um eine Sedezimalzahl (Kennzeichen : H), um eine Oktalzahl (Kennzeichen: Buchstabe O) oder um eine Binärzahl (Kennzeichen: B) handelt. Die Angaben werden folgendermaßen gemacht: '⟨Typenkennung⟩ ⟨Kennzeichen⟩ ⟨Wert⟩'

Das Wort wird rechtsbündig besetzt, es sei denn, man macht die Angabe, daß es linksbündig abgelegt werden soll. Zu diesem Zweck schreibt man einen Punkt vor die Wertangabe. Gibt man weniger Zeichen an, als die Länge des Wortes beträgt, so wird mit Nullen aufgefüllt. Zwischenräume sind bedeutungslos.

Sedezimalzahl (hexadekadische Zahl):

Zeichenvorrat:	die Ziffern 0, 1, ... 9, die Buchstaben A, B, ... F
z. B. '1 H 0'	hexadekadische Zahl mit Typenkennung 1, Wert Null
'2H. BF'	linksbündig, Typenkennung (TK) 2
'3H. AFFE'	linksbündig, TK 3

Oktalzahl:

Zeichenvorrat: die Ziffern 0, 1, ... 7

z. B. '1 O 77333'

Buchstabe O für oktal

Binärzahl:

Zeichenvorrat: die Ziffern 0, 1

z. B. '3B1100110011001000110100101101001 1'

d) Zeichenreihe:

Zum Zeichenvorrat gehören sämtliche Zentralcodezeichen, auch die, welche extern nicht darstellbar sind. Für diese gibt es eine Ersatzdarstellung:

* ⟨Zentralcodezeichen (dezimaler Wert)⟩

(siehe Zentralcodetabelle, Anhang 3)

Es ist zu beachten, daß der dezimale Wert des Zentralcodezeichens immer 3stellig angegeben werden muß, z. B.

*037 für das Textende-Zeichen

*021 für Vorschub auf neue Zeile

*120 für Stern, der immer in Ersatzdarstellung angegeben werden muß.

Bei Fernschreiberbetrieb am LRZ muß statt des Sterns ein x (Mal-Kreuz) stehen, da der Stern hier Fluchtsymbol ist.

Zeichenreihen werden immer linksbündig abgelegt und, falls nötig, mit Ignoriere-Zeichen (Oktade Null) aufgefüllt; sie haben TK 3,

z. B. "ANTON"

"ANTON * 037"

"*120 *120 *120" (ergibt: * * *) +)

+) Man beachte: Ein Zwischenraum in einer Zeichenreihe entspricht der Oktade 'AF' (Zentralcode-Zeichen, dezimaler Wert 175).

3.4. Kommentare

Kommentare sind beliebige Zeichenfolgen, die jedoch kein Semikolon oder dessen Ersatzdarstellung . , (Punkt-Komma) enthalten dürfen. Sie können in einem PS440-Programm an beliebiger Stelle eingefügt werden. Sie werden eingeleitet durch das Wortsymbol C0 und abgeschlossen durch Semikolon.

Außerdem gibt es noch die Möglichkeit, hinter dem Wortsymbol END eine Bezeichnung (siehe 3.2) als Kommentar einzufügen, in der Form END ⟨Bezeichnung⟩;.

3.5. TAS-Einschübe

Es ist möglich, in einem PS440-Programm TAS-Sequenzen zu benutzen. TAS-Einschübe haben grundsätzlich folgende Form:

/⟨TAS-Einschub⟩/;

(an LRZ-Fernschreibern Mal-Kreuz statt Stern).

Sie werden vom PS440-Übersetzer unbesehen übernommen, Fehler werden also nicht bemerkt (erst später, bei der TAS-Übersetzung).

a) Befehlssequenzen als TAS-Einschübe

Ein TAS-Einschub kann aus beliebig vielen TAS-Befehlen bestehen. Man sollte aber nur in Notfällen davon Gebrauch machen, da dabei erfahrungsgemäß zu viele Fehler gemacht werden und das Programm außerdem unleserlich wird.

Zu beachten ist, daß Namen, die in einem TAS-Einschub definiert werden, dem PS440-Übersetzer nicht bekannt sind, also außerhalb von TAS-Sequenzen nicht verwendet werden dürfen. Namen, die im PS440-Teil erklärt werden, können dagegen in TAS-Einschüben benützt werden.

Ausnahme: Selektoren, Teilwortselektoren, PART-Größen und Namen, die durch Äquivalenzdeklaration definiert wurden (siehe 4.5, 4.6, 4.7). Diese kommen in dem vom PS440-Übersetzer erzeugten TAS-Code nicht mehr vor, sie sind dort bereits ersetzt.

Beispiel:

```
*/ZI  (A1/A)/*;
*/SHB R 1, WTV X1 X2/*;
```

b) Befehlsvereinbarungen (siehe 4.8)

Hier verwendet man nur den Operationsteil eines TAS-Befehls (siehe Große Befehlsliste [2]),

z. B.

```
INSTRUCT (RA) BHW = */B2/*;
```

c) Konstantenlisten als TAS-Sequenz (siehe 4.3.2.3)

Hier wird die TAS-Schreibweise von Konstanten verwendet, z. B.

RECORD 4 SHORT A PRESET */A1/A, A2/A, A3/A, A4/A/*;

Konstantenliste

3.6. Standardnamen

Man kann in PS440 auf bestimmte Rechenwerks- und Befehlswerk-Register zugreifen. Als Bezeichnung für diese Register gibt es folgende Standardnamen:

RA für Akkumulator (Länge 1 Ganzwort)

RQ für Quotientenregister (Länge 1 Ganzwort)

RH für Hilfsregister (Länge 1 Ganzwort)

RD für Multiplikandenregister (Länge 1 Ganzwort)

RAQ für ein doppelt langes Register, das sich aus RA und RQ zusammensetzt (Länge 2 Ganzworte)

BB für Bereitadressenregister (Länge 1 Halbwort)

Diese Register dürfen verändert werden.

Drei weitere Register können außerdem verwendet werden, man darf sie aber nicht überspeichern:

BU für Unterprogrammzähler (Länge 1 Oktade)

RY für Shiftzähler (Länge 1 Oktade)

BF für Befehlsfolgeregister (Länge 1 Halbwort)

Die Verwendung von Registern sollte, soweit möglich, vermieden werden, denn diese werden durch vom Übersetzer erzeugte Operationen verändert. Man sollte Register also nur dann benutzen, wenn man sie direkt davor explizit besetzt hat, und dann möglichst nur RA.

Sinnvolle Verwendungsmöglichkeiten:

a) Im Zusammenhang mit vorhergehenden oder nachfolgenden TAS-Einschüben

b) Für Typenkennungstests (siehe 6)

c) Für ODD- bzw. BODD-Tests (siehe 6)

d) Zur Übermittlung von Parametern bei Prozeduraufrufen und bei der Rückkehr aus Prozeduren (siehe 4.4.2.7).

Weitere Standardnamen sind LEIT und LEITSM (siehe 4.3.2.4).

4. VEREINBARUNGEN

Bevor ein Name verwendet wird, muß er deklariert oder zumindest spezifiziert werden; der Übersetzer muß das Objekt "kennen" . +)

a) Deklaration:

Sie definiert den Typ (z. B. FULL, PROC usw.) und schafft Platz für das Objekt.

b) Spezifikation:

Sie definiert nur den Typ des Objekts.

Anwendung:

Will man einen Namen verwenden, bevor man ihn deklariert hat, so muß man ihn spezifizieren, damit der Übersetzer weiß, welchen Code er erzeugen soll.

Möchte man sich auf Objekte in einem anderen, getrennt übersetzten Programm beziehen, so muß man dem Übersetzer mitteilen, um welchen Typ es sich handelt und wo (in welchem Montageobjekt) das betreffende Objekt zu finden ist.

Sollen Objekte einem anderen Montageobjekt zugänglich gemacht werden, so muß man sich als Eingänge spezifizieren (ENTRY ⟨Liste von Objekten⟩;).

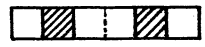
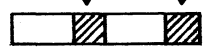
Diese Eingänge müssen globalen Geltungsbereich haben (siehe 4.1.2).

+) Ausnahmen:

Namen von Prozeduren und Marken können - allerdings nicht in allen Fällen - vor ihrer Deklaration verwendet werden, sofern der Typ des Objekts aus der Art der Verwendung eindeutig hervorgeht.
Adreßkonstanten dürfen sich auch auf Namen von noch nicht vereinbarten Objekten beziehen.

4.1. Allgemeines

4.1.1. Objekt - Typen

Typ	Bedeutung
SHORT	Halbwort
FULL	Ganzwort mit Typenkennung
LONG	Zwei aufeinanderfolgende Ganzworte, jedes mit Typenkennung
INDEX	Indexzelle
PART	Teilfeld eines Ganzwortes
	PART  ergibt 
LABEL	Marke
PROC	Prozedur, die mit SU angesprungen wird (oder mit SUE, falls der Sprung über eine Großseite hinausgeht), Rückkehr mit MU S 0
PROCSFB	Prozedur, die mit SFB angesprungen wird (oder mit SFBE, falls Sprung in andere Großseite → 22-Bit-Adressierung), Rückkehr mit MABI S 0
INSTRUCT	neu definierte Operation
MACRO	nicht implementiert

4.1.2. Geltungsbereich von Namen

Namen gelten nur lokal in dem Segment, in dem sie vereinbart sind. Soll der Name im ganzen Programm gelten - also auch in anderen Segmenten, so muß bei der Spezifikation bzw. Deklaration das Attribut GLOBAL hinzugefügt werden, z. B.

```
GLOBAL FULL x;          (Deklaration)

SPEC GLOBAL PROC p;     (Spezifikation)
```

4.1.3. Adressierung über Großseitengrenze

Im allgemeinen liegen die Zonen Konstantenbereich, Variablenbereich, Befehlsbereich (siehe TAS-Handbuch [3]) eines PS440-Programms innerhalb einer Großseite; das bedeutet, daß man mit 16-Bit-Adressen auskommt (der Adreßteil eines TAS-Befehlswortes besteht aus 16 Bit !).

Manchmal ist jedoch eine Adressierung über Großseitengrenze hinaus notwendig; das bedeutet, daß andere Befehlssequenzen erzeugt werden müssen. Zu diesem Zweck muß man bei der Deklaration bzw. Spezifikation vor dem Namen das Attribut DATA angeben:

{SPEC} {GLOBAL} $\left\{\begin{matrix}\text{SHORT}\\\text{FULL}\\\text{LONG}\end{matrix}\right\}_1^1$ DATA x;

Es ist auch möglich, Prozeduren als DATA zu vereinbaren, worauf hier jedoch nicht näher eingegangen wird.

4.2. Spezifikationen

Spezifikationen werden eingeleitet mit dem Wortsymbol SPEC, gefolgt von Typ und Namen des Objekts,

z. B.

```
SPEC FULL x;
SPEC PROC p;
```

Handelt es sich um Objekte in einem anderen Montageobjekt, so muß der betreffende Montageobjektname angegeben werden,

z. B.

```
SPEC Montageobjektname LABEL m;
SPEC IOC PROC IOC1, IOC2, IOC3, IOC4;
```

Diese Objekte sind aber nur dann zugänglich, wenn im Montage-Objekt mit dem Namen "Montageobjektname" folgende Spezifikation gemacht wurde: ENTRY m; und im Montageobjekt IOC: ENTRY IOC1, IOC2, IOC3, IOC4;. Die betreffenden Objektnamen müssen global sein (siehe 4.1.2).

4.3. Deklarationen

4.3.1. Konstanten und Variablen

Man unterscheidet Konstanten (im schreibgeschützten Bereich) und Variablen (nicht schreibgeschützt).

a) Variablen,

z. B.
```
SHORT s;
FULL f1, f2, f3;
LONG l1, l2;
INDEX x;
```

Anmerkung:

LONG-Größen sollte man, außer wenn sie für Festpunktrechnung nötig sind, nur verwenden, wenn man von der Möglichkeit Gebrauch machen will, daß eine Verschiebung innerhalb von Doppelworten gemacht werden kann (siehe 5.4.6 und 5.4.7).

Vorzuziehen ist die Vereinbarung eines Verbundes von 4 Halbworten (siehe 4.3.2).

Bei der Deklaration einer Variablen kann eine statische Vorbesetzung (durch PRESET) oder eine dynamische (durch Wertzuweisung) erfolgen. Die statische Vorbesetzung wird vom PS440-Übersetzer ausgeführt, also schon vor der Laufzeit des Programms, während für die dynamische Vorbesetzung eine TAS-Befehlssequenz erzeugt wird, die mit der Befehlssequenz für eine entsprechende Wertzuweisung identisch ist. Programme, die Vorbesetzungen mit PRESET enthalten, können jedoch mehrfach hintereinander gestartet werden, da die PRESET-Größen vom Betriebssystem bei jedem Start neu initialisiert werden,

z.B.

```
FULL f1 PRESET 1;
SHORT s PRESET "ABC";
```

} statische Vorbesetzung

```
INDEX x1:=1, x2:=2;
FULL f:=f1;
```

} dynamische Vorbesetzung

Deklarationen, die dynamische Vorbesetzungen enthalten, müssen im Programm so angeordnet sein, daß sie dynamisch durchlaufen werden, bevor die vorzubesetzende Größe verwendet wird.

b) Konstanten

Hier erfolgt ausschließlich eine statische Besetzung mit Hilfe des Wortsymbols IS, da Konstanten im schreibgeschützten Bereich abgelegt werden und nicht besetzt werden können.

z.B.

```
FULL kf IS 25;
SHORT s1 IS '1H0';
SHORT s2 IS p; (Vorbesetzung mit Adreßkonstante)
```

s2 erhält als Wert die Adresse p.

Gleichbedeutend ist folgende Schreibweise:

```
SHORT s2 IS REF p;
```

(siehe 5.3.2)

4.3.2,
Verbund

Ein Verbund ist ein Bereich von n Halbworten, er kann vom Typ SHORT, FULL, LONG oder INDEX sein. n muß eine ganze Zahl sein; es gibt also keinen Verbund von variabler Länge. Bei Typ SHORT handelt es sich um eine Folge von n Halbworten, bei Typ FULL um $\frac{n}{2}$ Ganzworte (beginnend auf Ganzwortadresse), bei Typ LONG um $\frac{n}{4}$ Doppelworte und bei INDEX um eine Folge von n Indexzellen. Es gibt Verbünde von Variablen und von Konstanten, sie können auch in einer anderen Großseite liegen (Attribut DATA).

Die Deklaration eines Verbundes von Variablen bzw. die Spezifikation eines beliebigen Verbundes sieht folgendermaßen aus:

$$\{\underline{\text{SPEC}}\}_0^1\ \underline{\text{RECORD}}\ n \left\{ \begin{array}{l} \underline{\text{SHORT}} \\ \underline{\text{FULL}} \\ \underline{\text{LONG}} \\ \underline{\text{INDEX}} \end{array} \right\}^1_1 \text{Vname;}$$

n ist die Dimensionsangabe. Mit Vname wird immer das nullte Halbwort, Ganzwort oder die nullte Indexzelle des Verbundes bezeichnet. Auf die weiteren Worte greift man durch Indizierung oder auf dem Umweg über Äquivalenzen (siehe 4.5) zu. Zugriff durch Indizierung: Vname [j] oder Vname (j), wobei j ein Ausdruck folgender Form ist:

$$\{\Sigma \pm \{c_i \times\} v_i\}\ \{\pm c\}$$

Beispiel:

```
RECORD 500 FULL a;
INDEX i, k;
        .
        .
        .
        .
a [i] := a [i+4*k];
a [6*k-2] := a [2*a[2*k]-2*i+4];
```

Die Indizierung bedeutet einen Zugriff auf das Objekt mit der Adresse >Vname< +j. Die Verwendung von eckigen oder runden Klammern ist dabei gleichbedeutend.

Zur Indizierung sollten nach Möglichkeit Indexzellen verwendet werden, da dies die Effizienz erhöht.

Vorsicht:

Bereichsüberschreitungen werden vom Übersetzer nicht bemerkt!

4.3.2.1.
Verbund im Variablenbereich

Die Deklaration kann in der oben gezeigten Form (ohne Vorbesetzung) erfolgen,

z. B.

RECORD 10 FULL vbl;

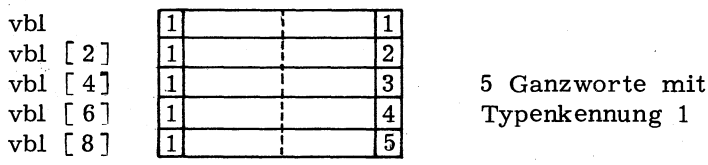
vbl ist ein Verbund von 5 Ganzworten.

Außerdem ist es möglich, einen Verbund bei der Deklaration mit PRESET mit einer Konstantenliste (siehe 4.3.2.3) vorzubesetzen,

z. B.

RECORD 10 FULL vbl PRESET (1, 2, 3, 4, 5);
Konstantenliste

Bedeutung:

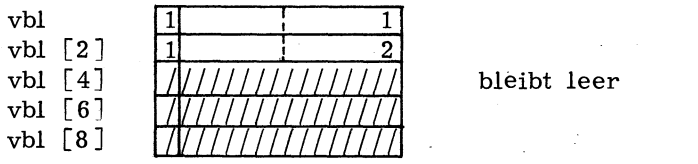


5 Ganzworte mit Typenkennung 1

Man muß einen Verbund nicht unbedingt vollständig vorbesetzen,

z. B. RECORD 10 FULL vbl PRESET (1, 2);

Bedeutung:



bleibt leer

Man bekommt hierbei eine Warnung vom Übersetzer, da er nicht weiß, ob man die Vorbesetzung in dieser Form wirklich wollte oder ob man sich beim Aufbau der Konstantenliste verzählt hat oder ob man eine falsche Dimensionsangabe gemacht hat.

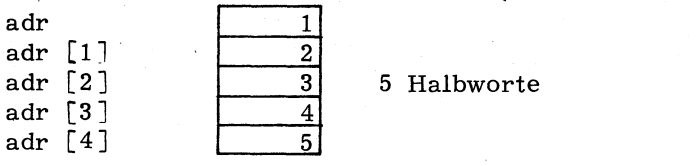
4.3.2.2.
Verbund im Konstantenbereich

Hier erfolgt die Besetzung und Deklaration wie bei einfachen Konstanten mit IS,

z. B.

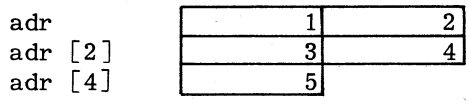
RECORD 5 SHORT adr IS (1, 2, 3, 4, 5);
Konstantenliste

Bedeutung:

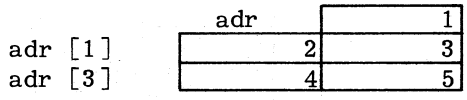


5 Halbworte

Da ein Verbund vom Typ SHORT auf Halbwortadresse beginnt, kann er folgendermaßen im Speicher liegen:



oder:



Falls die Dimension des Verbunds nicht mit der Länge der Konstantenliste übereinstimmt, erfolgt eine Warnung durch den Übersetzer. In diesem Fall (Verbund wird schreibgeschützt abgelegt!) liegt ein echter Programmier- oder Schreibfehler vor.

4.3.2.3.
Konstantenliste

Eine Konstantenliste ist eine Folge von Konstanten - deren Typ verschieden sein kann und angegeben werden muß - und Adreß-Konstanten (siehe auch 5.3.2).

Eine Konstantenliste kann selbst wieder Konstantenlisten enthalten. Um solche Unterlisten zu kennzeichnen, kann man beliebig Klammernpaare einfügen.

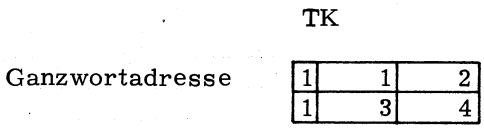
Soll eine Konstantenliste mehrmals hintereinander abgelegt werden, so setzt man sie in Klammern und gibt davor die Anzahl der Wiederholungen an.

Außerdem ist es möglich, eine Konstantenliste als TAS-Sequenz zu schreiben (siehe 2.5).

Beispiele:

SHORT (1, 2, 3, 4)

ergibt, abgelegt auf einen beliebigen Verbund:



Im Fall, daß der Verbund vom Typ SHORT ist, wird die Konstantenliste möglicherweise wie folgt abgelegt:

Halbwortadresse		1
1	2	3
1	4	

TK

FULL (1, SHORT (2,3) ,2 ("4"), SHORT 2 (5, 6)) ergibt, abgelegt auf einen Verbund vom Typ FULL oder LONG:

1		1
1	2	3
3	B4igigigigig	
3	B4igigigigig	
1	5	6
1	5	6

6 Oktaden im Zentral-Code (ig = Oktade Null)

TK

Läßt man die Typangabe weg, so wird angenommen, daß die Konstantenliste vom Typ FULL ist; bei Ablage auf einen Verbund erhält sie den Typ des Verbundes.

Beispiel 1:

((1, 2), FULL 3, LONG 4, SHORT 4(5), (6, 7))

bei Typ SHORT: Fehlerquelle, da auf Halbwortadresse begonnen wird, d.h. unter Umständen auf einer ungerader Adresse.

bei Typ FULL:

1		1	Ganzwort
1		2	
1		3	
1		0	} Doppelwort
1		4	
1	5	5	} 4 Halbworte
1	5	5	
1		6	
1		7	

TK

bei Typ LONG:

1		0	} Doppelwort
1		1	
1		0	} Doppelwort
1		2	
1		3	Ganzwort
1		0	} Doppelwort
1		4	
1	5	5	} 4 Halbworte
1	5	5	
1		0	
1		6	
1		0	
1		7	

TK

Beispiel 2:

("ABCDEFG", '1H1')

bei Typ SHORT:

Halbwortadresse

A B C
D E F
Gigig
1

Vorsicht:

Wird diese Konstanten-Liste mit einer geraden Anfangs-Adresse (= Ganzwortadresse) abgelegt, so steht der Einser im zweiten Halbwort des zweiten Ganzwortes. Da Halbworte keine Typenkennung haben, richtet sich diese nach der Typenkennung des Ganzwortes (in diesem Fall TK = 3),

bei Typ FULL:

3	A B C D E F
3	Gigigigigig
1	1

bei Typ LONG:

3	A B C D E F
3	Gigigigigig
1	0
1	1

(ig=Oktade Null)

Beispiel für Adreß-Konstanten:

SHORT 2 (A, B [3], REF A, REF B)

abgelegt auf einen Verbund vom Typ FULL:

1	>A< >B [3] <
1	>A< >B<
1	>A< >B [3] <
1	>A< >B <

Es wird jeweils die Adresse der betreffenden Größe abgelegt; der Operator REF kann weggelassen werden, weil in Konstanten-Listen sowieso nur Adreßkonstanten zugelassen sind. Bei A kann es sich zum Beispiel um eine Konstante, eine Variable, eine Prozedur oder um ein Label handeln.

Beispiel für die Schreibweise einer Konstantenliste als

TAS-Sequenz:

```
*/0/1HG, A1/A, 'C0C1C2C3C4C5'/3, '("ANTON*037")'/*;
```

Es ist zu beachten, daß eine Konstantenliste dieser Art nicht in Klammern eingeschlossen werden darf.

Die Schreibweise von Konstanten in TAS siehe TAS-Handbuch [3].

Konstantenlisten müssen nicht unbedingt auf einen Verbund mit einem Namen abgelegt werden.

Beispiel:

```
x := REF (1, 2, 3);
```

Die Konstantenliste wird in den Konstantenbereich abgelegt, der Zugriff auf sie erfolgt über ihre Adresse (die in x steht). Da keine andere Typ-Angabe gemacht wurde, wird der Typ der Konstantenliste als FULL angenommen (siehe hierzu 5.3.2: Der Operator REF).

4.3.2.4.
Standard-Verbund

Es gibt in PS440 zwei Standard-Verbünde: LEIT und LEITSM. Es handelt sich hier um Verbund von 256 Halbworten vom Typ FULL, die den Leitblock enthalten.

LEIT ist schreibgeschützt abgelegt, es ist also nur ein lesender Zugriff erlaubt, dieser erfolgt über BLEI-Befehle.

LEITSM liegt im Variablenbereich. Es kann auf ihn nur im System- oder Spezialmodus zugegriffen werden, da der Zugriff über LEI-Befehle erfolgt (es ist zu beachten, daß es in PS440 den Begriff "Modus" nicht gibt und auch eine Modus-Umschaltung von der PS440-Ebene aus nicht möglich ist). Leitblockverlängerungen müssen über eine Ortsangabe im Leitblock indirekt adressiert werden, da nur 8-Bit-Adressen zur Verfügung stehen.

4.4. Marken und Prozeduren

4.4.1. Marken

Marken kennzeichnen "Programmstellen", d.h. den Beginn von Anweisungen. Vor Deklarationen sind Marken nur dann sinnvoll, wenn gleichzeitig mit der Deklaration eine dynamische Vorbesetzung gemacht wird.

Die Markendefinition sieht folgendermaßen aus:

$$\{\underline{\text{GLOBAL}}\}_{0}^{1}\ \langle \text{Name der Marke} \rangle :$$

Statt des Doppelpunktes können auch zwei unmittelbar aufeinanderfolgende Punkte geschrieben werden (Zwischenräume verboten!).

Marken brauchen im allgemeinen weder spezifiziert noch deklariert zu werden, sofern es sich um lokale Marken handelt und aus ihrer Verwendung eindeutig hervorgeht, daß es sich um Objekte vom Typ LABEL handelt.

Eine Marke wird durch folgende Anweisung angesprungen:

GOTO ⟨Marke⟩;

"Marke" kann hier direkt der Name der Marke sein,

z. B.

```
m: x := y; ..... GOTO m;
```

es wird dann ein S-Befehl erzeugt, oder der Name einer Variablen, die die Adresse einer Marke enthält,

z. B.

```
SHORT x := REF m;  GOTO x;
```

hier wird ein SE-Befehl erzeugt.

Allgemein ist zur Verwendung von GOTO folgendes zu sagen: GOTO ist im Prinzip entbehrlich, wenn man die Möglichkeiten der bedingten Anweisungen (siehe 7.1), Prozeduren (siehe 4.4.2), Laufanweisungen (siehe 7.3) und Fallunterscheidungen (CASE-Anweisung) (siehe 7.2) voll ausnützt; man sollte es möglichst nicht verwenden, weil die Lesbarkeit des Programms darunter leidet . Man kann bei GOTO auch den Namen einer Prozedur angeben; dadurch wird das Programm jedoch unübersichtlich und es schleichen sich leicht Fehler in bezug auf die Unterprogrammhierarchie ein. Ebenso ist es möglich, statt GOTO m; nur m; und statt GOTO x; nur x; zu schreiben, dadurch wird das Programm aber meist unleserlicher (siehe Prozeduraufruf, 4.4.2.5).

Beispiel:

```
INDEX i;
RECORD 3 SHORT  switch IS (m1, m2, m3);

                    .
                    .
                    .
i := . . . ; CO möglicher Wert: 0/1/2;
```

```
GOTO switch [i];
                .
                .
                .
m1: . . . . . .
                .
                .
                .
m2: . . . . . .
                .
                .
                .
m3: . . . . . .
                .
                .
                .
```

Je nachdem, welchen Wert i an der Aufrufstelle hat, wird nach m1 (i=0), m2 (i=1) oder nach m3 (i=2) gesprungen. Eine eventuelle Bereichsüberschreitung muß der Programmierer gegebenenfalls selbst prüfen.

Folgendes ist auch zulässig:

```
RA := switch [i];
GOTO RA;
```

Das bedeutet, daß die Adresse des Sprungziels rechtsbündig in RA steht.

4.4.2. Prozeduren

Eine Prozedur ist eine (meist zusammengesetzte) Anweisung, die durch einen Prozeduraufruf zur Ausführung gebracht werden kann, ohne daß an der Aufrufstelle die Prozedur selbst niedergeschrieben werden muß. Die Prozedur muß vereinbart werden, der Übersetzer erzeugt dann ein TAS-Unterprogramm, d.h. eine Befehlssequenz, die mit einer Marke mit dem Namen der Prozedur beginnt und mit einem Rücksprung hinter die Aufrufstelle endet.

Es gibt zwei Arten von Prozeduren in PS440, die den beiden Arten von Unterprogrammen in TAS entsprechen und auch so gehandhabt werden.

4.4.2.1.
SU-Prozeduren

Vereinbarung: {GLOBAL}$^{1}_{0}$ PROC ⟨Name⟩ IS ⟨Rumpf⟩;

Aufruf: CALL ⟨Name⟩;

Hier wird ein SU-Befehl oder SUE-Befehl, falls eine Adressierung über Großseitengrenze notwendig ist, erzeugt, wobei der Unterprogrammzähler BU erhöht und die Rückkehradresse in einer Indexzelle vermerkt wird. Nach Ausführung der Anweisung wird mit MU S 0 zurückgesprungen und dabei der Unterprogrammzähler BU wieder erniedrigt.

4.4.2.2.
SFB-Prozeduren

Vereinbarung: {GLOBAL}$^{1}_{0}$ PROCSFB ⟨Name⟩ IS ⟨Rumpf⟩;

Aufruf: CALLSFB ⟨Name⟩;

Hier wird ein SFB-Befehl oder bei Adressierung über Großseitengrenze ein SFBE-Befehl erzeugt, der Unterprogrammzähler BU bleibt unverändert; die Rückkehradresse wird im Bereitadressen-Register BB mitgeliefert und muß vom Programmierer selbst "gerettet" werden. Nach Ausführung der Anweisung wird mit MABI S 0 zurückgesprungen, das bedeutet, daß die Rückkehradresse vor Ausführung dieses Befehls im Bereitadressen-Register BB bereitgestellt werden muß. Diese Art von Prozeduren ist etwas unbequemer zu handhaben als SU-Prozeduren, aber man kann hierbei Indexzellen sparen, was besonders bei tieferer Unterprogramm-Verschachtelung interessant ist.

4.4.2.3.
Der Prozedurrumpf

Der Prozedurrumpf kann aus einer einzigen Anweisung bestehen, z.B. x := y;, meistens handelt es sich jedoch um eine zusammengesetzte Anweisung, die mit BEGIN und END geklammert wird,

z.B.

```
BEGIN FULL a := 1;
        .
        .
        .
m:  SHORT b := 2;
        .
        .
        .
    CALL p;
        .
        .
        .
END;
```

Da dies kein Block (wie in ALGOL), sondern nur eine Zusammenfassung von mehreren Anweisungen zu einer einzigen ist, sind auch hier vereinbarte Größen im ganzen Segment gültig, und man kann beliebig in diese Klammern hineinspringen, sollte davon aber möglichst wenig Gebrauch machen (vgl. 4.4.1), in obigem Beispiel z.B. mit GOTO m; oder CALL m.

4.4.2.4. Rücksprung aus einer Prozedur

Wird aus einer Prozedur mit GOTO ⟨Marke⟩; herausgesprungen, so wird der Unterprogrammzähler BU nicht verändert, sondern nur das Befehlsfolge-Register BF. Ein Rücksprung aus der Prozedur heraus erfolgt mit der Anweisung RETURN;. Im Falle einer SU-Prozedur wird vom Übersetzer an dieser Stelle ein MU S 0-Befehl, im Falle einer SFB-Prozedur ein MABI S 0-Befehl erzeugt. Am Ende einer Prozedur erscheint automatisch ein Rücksprungbefehl (siehe 4.4.2.1 und 4.4.2.2), er muß an dieser Stelle also nicht explizit niedergeschrieben werden.

4.4.2.5. Aufruf einer Prozedur

Der Aufruf einer Prozedur erfolgt - je nach Art der Prozedur - mit CALL ⟨Name⟩, oder CALLSFB ⟨Name⟩;. Dabei kann für ⟨Name⟩wie beim Sprung auf Marken, statt des Namens selbst der Name einer Variablen oder Konstanten angegeben werden, deren Inhalt die Adresse der Prozedur ist. In diesem Fall wird je nach Art der Prozedur entweder ein SUE- oder ein SFBE-Befehl erzeugt.

Es ist möglich, eine Prozedur nur mit ihrem Namen aufzurufen, also CALL und CALLSFB wegzulassen. Außerdem sind Unterprogrammsprünge auf Marken mit CALL oder CALLSFB zugelassen. Vor diesen Verwendungsmöglichkeiten muß aber gewarnt werden, da sie leicht zu Fehlern führen und das Programm unübersichtlich machen.

Werden Prozeduren mit CALL bzw. CALLSFB aufgerufen, so müssen sie nicht unbedingt vorher erklärt werden, da der Objekt-Typ durch die Art des Aufrufs erkannt wird. Das gleiche gilt für Marken, die mit GOTO angesprungen werden, da sonst Vorwärtssprünge ohne vorhergehende Spezifikation oder Deklaration nicht möglich wären.

Der Aufruf exec ⟨Benennung⟩ bewirkt die Ausführung des rechtsbündig im Wert der Benennung stehenden TR440-Befehls.

4.4.2.6. Prozeduren als Segmente

Man kann bei der Vereinbarung von Prozeduren ein neues Segment eröffnen, d.h. Prozeduren in eigene Segmente legen. Der Prozedurname ist dann gleichzeitig der Segmentname und ist vom Typ GLOBAL. Auf diese Weise erreicht man, daß lokale Hilfsgrößen, die nur innerhalb der Prozedur benötigt werden, wirklich nur lokal in dieser Prozedur gelten. Andere in der Prozedur verwendete Objekte müssen dann aber GLOBAL vereinbart sein.

Z. B.

```
GLOBAL FULL x;

SEGMENT PROC P1 IS
BEGIN RA := x;

CALL P2;

END P1;

SEGMENT PROC P2 IS
BEGIN INDEX i;
        .
        .
        .
END P2;

SEGMENT Q;

FULL y;
    .
    .
    .
CALL P1;
    .
    .
    .
FINIS
```

4.4.2.7. Parameterübergabe

PS440-Prozeduren haben keinen festen Parametermechanismus, man kann sie aber mit Parametern versorgen.

Dafür gibt es folgende Möglichkeiten:

a) Parameterübergabe über globale Variable

b) Parameterübergabe über Register (z. B. RA, RAQ usw.)

c) Parameterübergabe mit Hilfe eines Versorgungsblocks:

Der Versorgungsblock ist ein Verbund, er wird mit den Adressen der Objekte besetzt, die als Parameter übergeben werden sollen. Der Prozedur muß bekannt sein, auf welchen Positionen die Adressen der Objekte im Versorgungs-Block stehen und welchen Typ das jeweilige Objekt hat. Die Übergabe der Adresse des Versorgungsblocks erfolgt dann in einem der Register.

```
z.B.    RA := REF v;    (siehe 5.3.2)
        CALL up;
```

up sei eine Prozedur mit 3 Parametern. Der erste habe den Typ SHORT, der zweite den Typ FULL und der dritte den Typ INDEX:

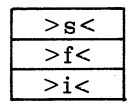
```
SHORT s; FULL f; INDEX i;
```

Der Versorgungsblock sieht folgendermaßen aus:

```
RECORD 3 SHORT v PRESET (s, f, i);
```

oder

```
RECORD 3 SHORT v IS (s, f, i); (schreibgeschützt)
```



Adresse des 1. Parameters P1
Adresse des 2. Parameters P2
Adresse des 3. Parameters P3

Die Prozedur up soll P1 + P3 → P2 bilden und das Ergebnis nach RA liefern.

```
PROC up IS
BEGIN SHORT vbladr := RA; CO Adresse von v retten;
      SHORT vbl  = SVAL vbladr;
      SHORT P1   = SVAL vbl [0];
      FULL  P2   = FVAL vbl [1];
      INDEX P3   = IVAL vbl [2];
      CO siehe 4.5 und 5.6;
      P2 := P1 + P3;
      RA := P2;
END up;
```

4.5. Äquivalenzvereinbarung

Bei einer Äquivalenzvereinbarung wird einem bereits erklärten Objekt ein anderer Name zugeordnet. Diesen Namen kann man nur auf PS440-Ebene verwenden - also nicht in TAS-Einschüben - da er bei der Code-Erzeugung durch die dieses Objekt darstellende Befehlsfolge ersetzt wird . Er muß vor der Verwendung bekannt sein.

Die Äquivalenzdeklaration hat folgende Form:

$$\{\underline{\text{GLOBAL}}\}_{0}^{1}\ \langle\text{Typ}\rangle\ \langle\text{Bezeichnung}\rangle = \langle\text{Benennung}\rangle;$$

Zwischen folgenden Objekttypen sind Äquivalenzvereinbarungen möglich:

			INDEX	=	INDEX			
			PART	=	PART			
LABEL	PROC	PROCSFB	=	LABEL	PROC	PROCSFB		
SHORT	FULL	LONG	=	SHORT	FULL	LONG		

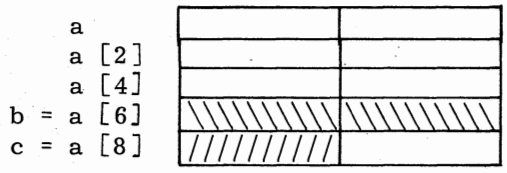
Äquivalenzvereinbarungen mit dem Attribut GLOBAL dürfen sich nicht auf lokale Objekte beziehen, dagegen kann über lokale Äquivalenzvereinbarungen auf Objekte vom Typ GLOBAL zugegriffen werden.

Für globale Äquivalenzdeklarationen gilt noch eine Einschränkung, die wegen des PS440-Übersetzers gemacht werden muß: Sie müssen am Anfang eines Segments vor den Deklarationen irgendwelcher lokaler Größen stehen.

Beispiele:

Beispiel 1:

```
RECORD 10 FULL a;
FULL  b =  a [6];
SHORT c =  a [8];
```



Auf das Ganzwort a [6] kann man nun mit dem Namen b, auf das Halbwort a [8] mit dem Namen c zugreifen,

z. B. b := 7; hat die gleiche Wirkung wie a [6] := 7;

Beispiel 2:

```
RECORD 10 FULL a;
INDEX i;
SHORT d = a [i];
```

Bei Verwendung von d wird jeweils auf das i. Halbwort von a zugegriffen, wobei für i jeweils der aktuelle Wert eingesetzt wird,

z. B. FULL summe := 0;

```
FOR i FROM 0 BY 1 TO 9 DO
summe + := d;
```

Bedeutung: summe := $\sum_{i=0}^{9}$ a [i] (siehe 5.1 u. 7.3)

Beispiel 3:
```
RECORD 10 FULL a;
FULL SELECT sc = [6];
FULL c = sc OF a;
```

das bedeutet: FULL c = GW OF a [6];

das 4. Ganzwort von a wird mit c bezeichnet (siehe 4.6).

Beispiel 4:
```
PROC p IS
BEGIN   .
        .
        .
   P1:  .
        .
        .
END p;
PROC NEBEN = P1;
CALL NEBEN;
```

Durch die Definierung des Namens NEBEN hat man bei P1 einen Nebeneingang in die Prozedur p geschaffen; es ist nun der Prozeduraufruf CALL NEBEN; möglich.

Beispiel 5:
```
RECORD 10 FULL a;
RECORD  2 SHORT adra IS (a [6], a [8]),
FULL f  =  FVAL adra [1];
```

adra besteht aus 2 Halbworten, von denen das erste mit der Adresse von a [6], das zweite mit der Adresse von a [8] besetzt ist. Beim Zugriff auf f erhält man das Ganzwort a [8] (siehe 5.6).

Beispiel 6:
```
FULL f;
SHORT f1 = f [1];
```

Es wird das zweite Halbwort aus dem Ganzwort f mit f1 benannt!

Beispiel 7:
```
RECORD 10 FULL a;
PART SELECT ops = [8] '2H FF FFFF 00 FFFF';
PART op = ops OF a;
PART op1 = op;
```

Der Operationsteil des zweiten Halbworts aus dem Ganzwort a [8], um 16 Bits nach rechts verschoben (also rechtsbündig), erhält die Namen op und op1 (siehe 4.7).

4.6. Selektoren

Eine Selektorvereinbarung ist eine Operatorvereinbarung. Durch sie wird ein Operator definiert, mit dessen Hilfe man auf ein bestimmtes Element in einem beliebigen Datenobjekt zugreifen kann. Es muß angegeben werden, von welchem Typ dieses Element sein soll und an welcher Stelle relativ zum Anfang des Daten-Objekts es sich befindet (welchen Index es hat). Selektoren können auch auf Register angewendet werden, sofern es sich um einen lesenden Zugriff auf das betreffende Register handelt.

Die Selektorvereinbarung hat folgende Form:

⟨Typ⟩ SELECT ⟨Selektorname⟩ = [⟨konstanter Index⟩];

Beispiele:
```
RECORD 6 SHORT s;
RECORD 8 FULL f;
RECORD 8 LONG l;
```

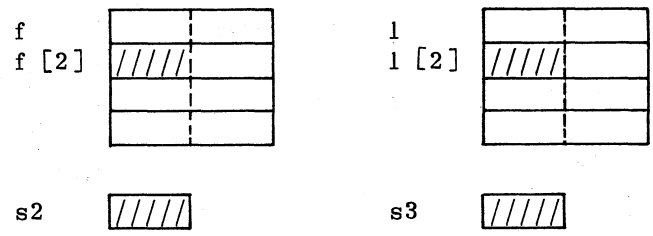
Selektorvereinbarungen:
```
SHORT SELECT sks = [2];
FULL  SELECT skf = [2];
LONG  SELECT skl = [2];
```

Verwendung in Wertzuweisung:

```
SHORT s1 := sks OF s;
```
Der Inhalt des Halbwortes s [2] wird nach s1 gebracht.

```
sks OF s := s1;
```
Der Inhalt von s1 wird in das Halbwort s [2] gebracht.

```
SHORT s2 := sks OF f;
s3 := sks OF l;
```



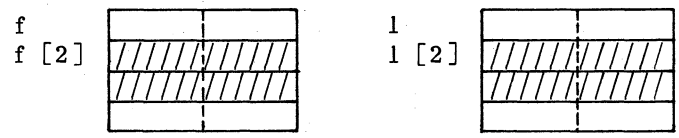
Auf Datenobjekte vom Typ SHORT darf man nur Selektoren vom Typ SHORT anwenden, weil SHORT-Objekte auf Halbwortadressen beginnen.

Verwendung in Äquivalenzvereinbarungen:

FULL f1 = skf OF f, f2 = skf OF l.

Das Ganzwort f [2] aus dem FULL-Objekt f bekommt den Namen f1, das Ganzwort l [2] aus dem LONG-Objekt l den Namen f2.

LONG l1 = skl OF f, l2 = skl OF l.



Es gibt in PS440 drei Standard-Selektoren:

HW bezieht sich auf das nullte Halbwort,

GW bezieht sich auf das nullte Ganzwort,

DW bezieht sich auf das nullte Doppelwort eines Datenobjekts.

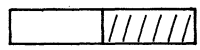
Man könnte obige Beispiele also auch folgendermaßen schreiben:

SHORT s1 := HW OF s [2],
HW OF s [2] := s1,
SHORT s2 := HW OF f[2], s3 := HW OF l [2],
FULL f1 = GW OF f [2], f2 = GW OF l [2],
LONG l1 = DW OF f [2], l2 = DW OF l [2],

(siehe auch Beispiel 3 in Abschnitt 4.5).

Man unterscheide folgendes:

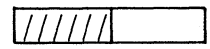
```
SHORT a;  FULL b;
a := b;
b := a;
a := RA;
RA := a;
```



Hier wird auf das rechte Halbwort zugegriffen (siehe 5.1).

```
SHORT a;  FULL b;
a := HW OF b;
HW OF b := a;
a := HW OF RA;
```

(dagegen folgendes nicht zulässig: HW OF RA := a;).



Hier wird auf das linke Halbwort zugegriffen.

4.7. Teilwort-Selektoren

Die Vereinbarung eines Teilwort-Selektors ist eine Operator-Vereinbarung, wobei ein Operator definiert wird, mit dessen Hilfe man auf ein bestimmtes Feld in einem beliebigen Ganzwort eines Datenobjekts zugreifen kann (das kleinste Teilwort ist ein Bit).

Es handelt sich hierbei um eine Erweiterung der Selektorvereinbarung (siehe 4.6): Man kennzeichnet die normale Selektorvereinbarung mit dem Attribut PART und gibt zusätzlich zu dem konstanten Index eine Bitkonstante (die Maske) an, deren "Nullfeld" das Teilwort des Ganzwortes bezeichnet, auf das man mit dem Teilwort-Selektor zugreifen möchte. Es ist zu beachten, daß dieses Teilwort nach dem Zugriff im Ergebnis so weit wie möglich nach rechts verschoben ist (BT-Befehl in TAS).

Teilwort-Selektoren können nur auf FULL-Größen angewendet werden und ergeben auch wieder ein Wort dieses Umfangs. Die angegebene Maske muß also auch ein Wert vom Umfang FULL sein.

Bei Anwendung eines Teilwort-Selektors erhält man ein Objekt vom Typ PART . Dieses Attribut wird für sich allein nur bei Äquivalenzvereinbarungen verwendet (siehe 4.5, Beispiel 7) und muß dann angegeben werden, wenn es sich bei dem betreffenden Objekt um ein PART-Objekt handelt - dies ist nur im Zusammenhang mit Teilwort-Selektoren möglich.

Die Vereinbarung eines Teilwort-Selektors hat folgende Form:

PART SELECT ⟨Selektorname⟩ =[⟨konstanter Index⟩] ⟨Maske⟩;

Die Angabe ⟨konstanter Index⟩ kann entfallen, falls er Null ist, der Teilwort-Selektor sich also auf das nullte Ganzwort eines Datenobjekts bezieht.

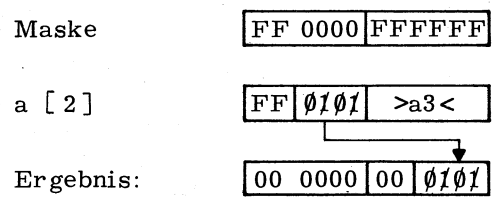
Anmerkungen:

a) Die Verwendung von mehreren PART-Größen innerhalb eines Ausdrucks ist im allgemeinen ineffizient.

b) Die Anwendung von Schiebeoperationen (siehe 5.4.6 und 5.4.7) auf Objekte, die direkt als PART-Größen angesprochen sind, ist zu unterlassen, denn Teilwortbefehle verändern die zum Schieben nötigen Register.

Beispiel 1:
```
RECORD 4 FULL a IS SHORT (a1, a2,
                          '2H FF0101', a3);
PART SELECT opsadr = [2]'2HFF0000FFFFFF';
PART adr = opsadr OF a;
PART adra = adr;
```

Bedeutung:



PART-Größe mit den beiden Namen adr und adra.

Eine andere Schreibweise dieses Beispiels:

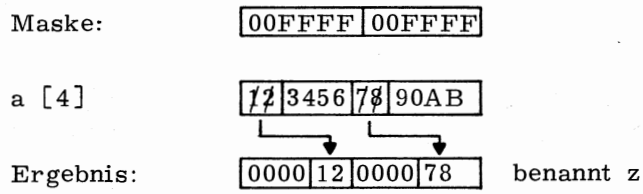
```
PART SELECT opsadr0 = '2 H FF0000FFFFFF',
PART adr = opsadr0 OF a [2];
```

Der Teilwort-Selektor ist in diesem Fall allgemeiner gehalten, sein konstanter Index ist Null, deshalb wurde er weggelassen. adr hat die gleiche Bedeutung wie oben.

Es sind beliebige Wertzuweisungen und Operationen möglich:

z. B.
```
FULL f := opsadr OF a;
SHORT s; ... s+ := adr; (siehe oben)
```

Beispiel 2:
```
RECORD 6 FULL a;
a [4] := '1H 1234567890AB';
PART SELECT xy = [4]'2H FFFF 00 FFFF';
PART z = xy OF a;
```



Beispiel 3:
```
PART SELECT bit40 = '2H FFFFFF FFFEFF';
RA := bit40 OF Ganzwort;
```

Das 40. Bit eines Ganzwortes wird rechtsbündig in das Register RA gebracht.

4.8. Befehlsvereinbarung

Durch Befehlsvereinbarungen kann man Operatoren definieren, die TAS-Befehle erzeugen, welche der PS440-Übersetzer selbst nicht kennt.

Die Befehlsvereinbarung hat folgende Form:

INSTRUCT (⟨Ergebnisregister⟩) ⟨Name⟩ = ⟨TAS-Sequenz⟩

In der TAS-Sequenz ist der Befehlsteil eines TAS-Befehls anzugeben (siehe Große Befehlsliste [2]). Es ist zu beachten, daß nur Ein-Adreß-Befehle zugelassen sind, dazu gehören auch die Doppelcode-Befehle. Der Operator darf nur so verwendet werden, wie es in TAS bei dem betreffenden Befehl verlangt wird.

"Ergebnisregister" ist die Bezeichnung für ein Register, das nach Ausführung des betreffenden Befehls einen definierten Wert enthalten soll; diese Angabe kann weggelassen werden.

Beispiele:
```
INSTRUCT (RQ) BB3 = */BQB/*;
a := BB3 b;
```

Aus b wird durch den BQB-Befehl ein Wort nach RA gebracht, RQ erhält den gleichen Wert, und dieser wird von RQ nach a abgespeichert.

INSTRUCT (RH) BC3 = */CR/*;

a := BC3 b;

Es wird aus dem RA ein Wort nach b gespeichert, und der Inhalt des Hilfsregisters nach RA und nach a gebracht.

INSTRUCT ZT1 = */ZT1/*;

ZT1 a;

Das Ganzwort a erhält Typenkennung 1 im Speicher.

INSTRUCT EZC = */EZ C/*;

EZC ix;

Dies ist ein Beispiel für einen Doppelcode-Befehl: Erhöhe den Index ix um 2 und speichere den Inhalt von RA in das Ganzwort, dessen Adresse in ix steht.

5. OPERATOREN

5.1. Zuweisungsoperatoren

Mit Hilfe von Zuweisungsoperatoren können Variablen Werte zugewiesen werden. Die großen Rechenwerksregister RA, RH, RQ und RD gelten als Variable vom Typ FULL, das Bereitadressenregister BB als Variable vom Typ SHORT und RAQ als Variable vom Typ LONG.

5.1.1. Wertzuweisung

Eine einfache Wertzuweisung hat folgende Form: v := e;
Der angegebene Ausdruck e hat einen Wert, und dieser wird der betreffenden Variablen v zugewiesen.

e selbst kann wieder eine Wertzuweisung sein, die den Wert e1 hat:

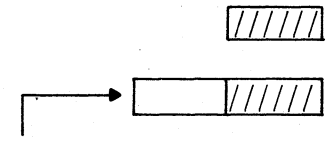
```
v := v1 := v2 := e2;
     v1 :=    e1
v :=   e
```

Wertzuweisungen werden immer von rechts nach links abgearbeitet. Die betreffenden Variablen können von verschiedenem Typ sein. In diesem Fall wird der Wert als Festpunktzahl interpretiert und durch Verlängern oder Verkürzen dem Typ der Variablen angeglichen, der er dann zugewiesen wird. Dieser neue Wert ist dann der Wert der Wertzuweisung.

Der Typ INDEX wird dabei wie der Typ SHORT behandelt.

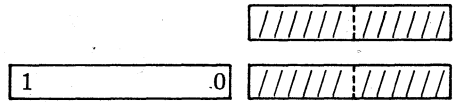
Verkürzen: Beim Verkürzen ist jeweils das rechte Halb- oder Ganzwort der neue Wert.

Verlängern: von SHORT auf FULL:



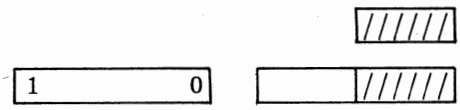
Es wird vorzeichengleich aufgefüllt

von FULL auf LONG:



Ins 1. Ganzwort wird Null mit Typenkennung 1 gesetzt.

von SHORT auf LONG:



Null mit TK 1 | vorzeichengleich aufgefüllt

5.1.2.
Erhöhungen

Erhöhungen zählen zu den Wertzuweisungen, ihre Schreibweise ist folgende:

v + := e;	für v := v + e;
v - := e;	für v := v - e;
v ++ := e;	für Gleitpunkt-Erhöhung v := v ++ e;
v -- := e;	für Gleitpunkt-Verminderung v := v -- e; (siehe 5.4.1 und 5.4.4).

Die Verwendung der Zuweisungsoperatoren ergibt im allgemeinen einen günstigeren Code als die explizite Schreibweise als arithmetischer Ausdruck, da es in TAS die entsprechenden Befehle gibt,

Mehrfachzuweisungen sind wie bei einfachen Wertzuweisungen möglich, da auch die Erhöhung einen Wert hat,

z. B.

```
INDEX i1, i2; SHORT s; FULL f;

                .
                .
                .
            i1 := 6;
            f := 10;
                .
                .
                .
f - := s := i1 + := i2 := 2;
```

i2 erhält den Wert 2; i1 wird um diesen Wert erhöht, der neue Wert ist 8; mit diesem wird s besetzt (s := 8); nun muß auf den Umfang "Ganzwort" verlängert werden, da f vom Typ FULL ist, und f wird um 8 vermindert.

Der Wert dieser Mehrfachzuweisung ist 2, der Umfang des Ergebnisses FULL.

5.2. Der Tauschoperator

Bei einem Tausch werden die Werte von zwei Variablen gleichen Typs vertauscht: v1 :=: v2;

Beispiele:

```
SHORT s1, s2; INDEX i1, i2 ; FULL f1, f2;
RECORD 10 SHORT s;
        .
        .
        .
s1 :=: s2;
i1 :=: i2;
f1 :=: f2;
HW OF f1 :=: HW OF f2;
RA :=: f1;
RA :=: RQ;
s1 :=: s [s1];
s [s1] :=: s1;
```

5.3. Unäre Operatoren

5.3.1. Unäre Operatoren, deren Operand ein Ausdruck ist

Operatoren dieser Art werden folgendermaßen verwendet:

⟨Operator⟩ e

e kann ein Ausdruck sein, der selbst mit einem unären Operator gebildet ist.

5.3.1.1. Vorzeichen

\+ positives Vorzeichen, keine Bedeutung

\- negatives Vorzeichen (auch auf Gleitpunktzahlen anwendbar)

5.3.1.2. Bildung des Absolutbetrags

ABS Bildung von |e|

5.3.1.3. Setzen der Typenkennung

TK0	für Typenkennung 0
TK1	für Typenkennung 1
TK2	für Typenkennung 2
TK3	für Typenkennung 3

5.3.1.4.
Beispiele

RQ := TK3 (a+b)

Es wird der arithmetische Ausdruck a+b ausgewertet, Typenkennung 3 gesetzt und das Ergebnis in das Quotientenregister gebracht.

a := - ABS TK1 RA;

Der Akkumulator erhält Typenkennung 1, es wird der Absolutbetrag gebildet und der negative Wert nach a gebracht.

a := - (- RA + TK1 x - (- ABS y));

Um Mehrdeutigkeiten zu vermeiden, müssen Klammern gesetzt werden.

5.3.2.
Der Operator REF

Bei Anwendung des Operators REF auf ein adressierbares Objekt (REF v) erhält man als Wert in einem Halbwort die Anfangs-Adresse des betreffenden Objekts. Adressierbare Objekte sind z. B. Prozeduren, Variablen, auch explizit angegebene Konstanten und Konstantenlisten, nicht dagegen Register, PART-Größen und Äquivalenzen.

Beispiele:

```
SHORT x := REF y;          (x:= Adresse der Variablen oder Konstanten y)

x := REF a [4]             (x:= Adresse des 3. Ganzwortes oder 5. Halbwortes von a)

x := REF (1, 2, 3);        (x:= Anfangsadresse der Konstantenliste, die in Ganzworten im Speicher abgelegt ist)

x := REF 1;                (x:=Adresse der Konstanten 1 im Speicher)

x := REF "DIES IST EIN TEXT *037";
                           (x:= Anfangsadresse des Textes im Speicher)
```

5.3.3.
Der Operator NOT

Der Operator NOT bewirkt die logische Negation und benötigt daher als Operanden einen Wahrheitswert. Ein Wahrheitswert ist das Ergebnis einer Vergleichsoperation oder eines Tests (siehe 5.5 und 6). Dieser wird durch NOT negiert,

z. B. NOT x = y (bei Vergleich)

NOT TK 3 RA (bei Typenkennungstest)

Anmerkung:

Das Zeichen ¬ kann statt des Wortsymbols NOT verwendet werden.

5.4. Binäre Operatoren

Mit binären Operatoren werden Ausdrücke der Form

e1 ⟨Operator⟩ e2

gebildet, wobei e1 und e2 selbst wieder Ausdrücke dieser Art sein können.

5.4.1. Festpunkt-Addition und -Subtraktion

e1 + e2 Addition

e1 - e2 Subtraktion

Sind die Werte von e1 und e2 von verschiedener Länge, so entspricht der Umfang des Ergebnisses einer dieser Operationen dem Umfang des längeren Wertes.

5.4.2. Festpunkt-Multiplikation

e1 x e2

Die Operanden müssen Typenkennung 1 haben, sie dürfen unterschiedlich lang sein.

LONG-Größen werden auf die Länge FULL verkürzt, und man bekommt dann eine Meldung vom Übersetzer.

Das Ergebnis der Festpunktmultiplikation ist doppelt so lang wie der größere Operand.

5.4.3. Festpunkt-Division

e1 / e2

Die Operanden müssen Typenkennung 1 haben. Der Dividend e1 wird auf den Umfang LONG, der Divisor e2 auf den Umfang FULL gebracht. Das Ergebnis hat die Länge FULL.

5.4.4. Gleitpunkt-Arithmetik

Gleitpunkt-Operationen können nur durchgeführt werden, wenn die Operanden Typenkennung 0 haben. Außerdem müssen die Operanden entweder beide vom Typ FULL oder beide vom Typ LONG sein. Das Ergebnis hat entsprechend die Länge FULL oder die Länge LONG.

e1 ++ e2 Addition

e1 -- e2 Subtraktion

e1 xx e2 Multiplikation

e1 // e2 Division

5.4.5. Logische Operationen

e1 VEL e2 Wirkung:

VEL	0	1
0	0	1
1	1	1

e1 AUT e2 Wirkung:

AUT	0	1
0	0	1
1	1	0

e1 ET e2 Wirkung:

ET	0	1
0	0	0
1	0	1

Die Abarbeitung geschieht bitweise, für die Durchführung der Operationen werden die in TAS gleichlautenden Maschinen-Befehle verwendet (siehe Große Befehlsliste [2]).

Die Operanden müssen vom Typ FULL sein und Typenkennung 2 oder 3 haben, da Typenkennung 0 oder 1 unerwünschte Effekte ergeben können. Das Ergebnis ist ein Bitmuster der Länge FULL.

Falls die Operanden verschiedene Typenkennung haben, bekommt das Ergebnis die höhere Typenkennung.

5.4.6. Längsshifts

e1 LEFT e2 (Shift nach links)

e1 RIGHT e2 (Shift nach rechts)

Der Wert von e1 wird bitweise um den Wert von e2 geshiftet, dabei gilt für den Wert von e2: $0 \leq e2 \leq 127$ (sinnvoll ist allerdings nur $e2 < 96$).

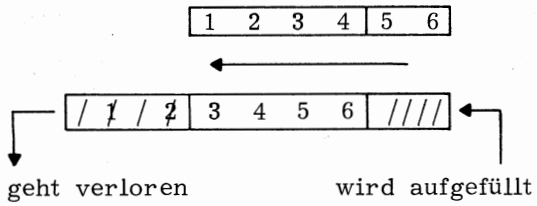
e2 kann beliebige Länge haben, es wird nur das letzte Halbwort verwendet.

Die Ausführung erfolgt unterschiedlich, je nachdem, welche Länge und Typenkennung e1 hat. Ist e1 vom Typ FULL, so werden die 48 Bits des Ganzwortes mit einem Kurzschift in einem der großen Rechenwerks-Register geshiftet, ist e1 vom Typ LONG, so werden die 96 Bits des Doppelwortes mit einem Langshift in dem doppelt langen Register RAQ geshiftet. Bei Typenkennung 0 oder 1 wird ein arithmetischer Shift vorgenommen, und dabei werden vorzeichengleiche Bits nachgezogen.

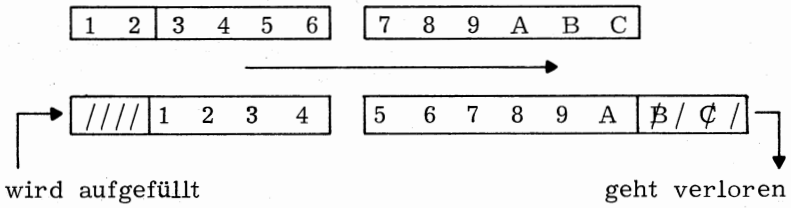
Bei Typenkennung 2 oder 3 entsteht ein logischer Schift und es werden Nullen nachgezogen. +) Die aus dem Register hinausgelaufenen Stellen gehen verloren,

z. B.

Linksschift bei FULL-Größen: RA LEFT 16



Rechtsschift bei LONG-Größen RAQ RIGHT 16



5.4.7. Kreisshifts

e1 LEFTC e2 (Kreisschift nach links)

e1 RIGHTC e2 (Kreisschift nach rechts)

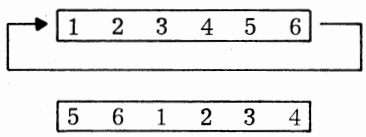
Für e2 gilt das gleiche wie bei den Längsschifts, sinnvoll ist nur e2 < 96.

e1 kann vom Typ FULL sein, das ergibt einen kurzen Kreisschift, oder vom Typ LONG, dann wird die Operation im doppelt langen Register RAQ ausgeführt. Die Typenkennung ist bei einem Kreisschift bedeutungslos.

Die aus dem Register hinausgelaufenen Bits laufen auf der anderen Seite wieder hinein,

z. B.

kurzer Kreisschift nach rechts:



RA RIGHTC 16

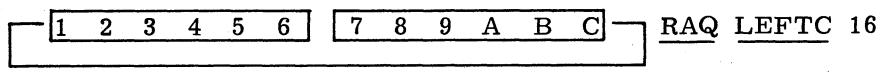
Ergebnis

+) Anmerkung:

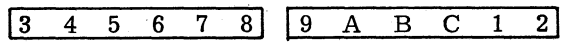
Beim Linksschift mit Typenkennung 0 oder 1 tritt ein arithmetischer Alarm auf, wenn die ersten beiden Bits des Wortes (das Marken- und das Vorzeichenbit) im Lauf der Schiebeoperation ungleich besetzt werden!

langer Kreisschift

nach links:



Ergebnis



5.5. Vergleichsoperatoren

Vergleichsoperatoren dienen der Erarbeitung von Wahrheitswerten. Solche Werte werden in PS440 nur zur Entscheidung bei bedingten Anweisungen und als Abbruch- oder Fortsetzungs-Kriterium bei Laufanweisungen herangezogen (siehe auch 6) Explizit gibt es diese Werte nicht.

Ist der Vergleich richtig, so ist sein Wert "true", andernfalls "false".

5.5.1. Einfache Vergleichsoperationen

Vergleichsoperationen sehen folgendermaßen aus:

e1 ⟨Vergleichsoperator⟩ e2

e1 und e2 können Ausdrücke sein, deren Ergebnis vom Typ FULL oder SHORT ist. LONG-Größen sind nicht zulässig. SHORT-Ergebnisse werden auf FULL-Länge erweitert. Haben beide Ergebnisse Typenkennung 0 bzw. 1, so entspricht der Vergleich einem Größenvergleich von Gleitpunkt- bzw. Festpunkt-Zählen. Bei anderer Typenkennung werden alle 48 Bits des Ganzwortes untersucht, und das Ganzwort wird als nichtnegative Festpunktzahl interpretiert; das bedeutet, daß die negative Null dabei die größte darstellbare positive Zahl wird!

Siehe dazu S. 130.1

Operatoren (nebeneinanderstehende Angaben sind gleichwertig)			Bedeutung
=	EQ	EQUAL	"gleich"
≠ +)	NE	NOTEQUAL ¬ =	"ungleich"
< +)	LT	LESS	"kleiner"
> +)	GT	GREATER	"größer"
≦ +)	LE	NOTGREATER < =	"kleiner-gleich"
≧ +)	GE	NOTLESS > =	"größer-gleich"

+) Anmerkung:

Die Zeichen ≠, ≦ und ≧ sind im Zeichensatz der LRZ-Drucker nicht enthalten, an ihrer Stelle erscheint! auf dem Protokoll. Am Fernschreiber sind zudem >und<nicht darstellbar. Um einen lesbaren Protokolldruck zu bekommen, muß man eine andere Schreibweise wählen.

Hinweis zu 5.5.1 Einfache Vergleichsoperationen

Die Vergleichsoperatoren arbeiten wie erwartet, wenn beide Operanden die gleiche Typenkennung haben:

- haben beide Operanden die TK 0, so werden sie als Gleitkommazahlen unter Berücksichtigung von Exponent und Vorzeichen verglichen;
- Haben beide Operanden die TK 1, so werden sie als Festkommazahlen unter Berücksichtigung des Vorzeichens verglichen;
- haben beide Operanden die TK 2 oder 3, so werden sie als positive Festkommazahlen verglichen.

Besondere Vorsicht ist jedoch geboten, wenn nicht garantiert werden kann, dass beide Operanden die gleiche Typenkennung haben. In diesem allgemeinen Fall sind die Größer- und Kleiner-Vergleiche **nicht transitiv**, d. h. es gibt Tripel von Operanden, die von diesen Operatoren nicht geordnet werden können, zum Beispiel `+1`, `–1` und `''1''`:

- `+1` und `–1` haben beide TK 1, also gilt `+1>–1`;
- `''1''` hat TK 3, also wird `–1` als `'FFFFFFFFFFFE'` mit `''1''` als `'B10000000000'` verglichen, also `–1>''1''`;
- `+1` wird als `'000000000001'` mit `''1''` als `'B10000000000'` verglichen, also `''1''>+1`.

Mit *a* = `+1`, *b* = `–1` und *c* = `''1''` ergibt sich also die widersprüchliche Situation, dass *a*>*b* und *b*>*c* und *c*>*a!* Eine Sortierung von beliebigen `FULL`-Operanden ist also mit Hilfe des `>`-Operators unmöglich, auch nicht mit <, ≤ oder ≥.

Um Werte unbekannter TK (z. B. von externen Unterprogrammen gelieferte) zu vergleichen, schreibt man statt `A > B` besser.

- wenn man auf jeden Fall einen Binärvergleich will:
  `A > TK3 B`;
- wenn man einen Gleitkomma-Vergleich will (mit TK- bzw. BÜ-Alarm bei unzulässigen Werten der Operanden):
  $A{+}{+}0_{10}0 > B{+}{+}0_{10}0$.

Einen Festkomma-Vergleich zwischen beliebigen Operanden kann man nicht als Ausdruck formulieren, da die Addition von `0` Typenkennungen ≠ 1 toleriert und die Multiplikation mit `1` ein `LONG`-Ergebnis liefert, das als Operand des Vergleichs nicht zulässig ist. Man muss also vor dem eigentlichen Vergleich die beteiligten Operanden aufgrund ihrer jeweiligen Typenkennung unterschiedlich behandeln.

Otto Stolz

Beispiele:

```
INDEX i, j; SHORT s; FULL f;

i + j ≦ s + f
x > (j + := 1)

REF s = i
f ≠ RA

i = i
NOT i ≠ i        } immer "true"
```

5.5.2.
Verknüpfungen von Vergleichsoperationen

Mit Hilfe von Verknüpfungsoperatoren können einfache Vergleichsoperatoren zusammengesetzt werden. Dies sind Boolesche Operationen, ihr Ergebnis ist wieder ein Wahrheits-Wert. Sie können somit selbst auch wieder verknüpft werden.

Es gibt zwei Verknüpfungsoperatoren:

AND Wirkung:

AND	"true"	"false"
"true"	"true"	"false"
"false"	"false"	"false"

OR Wirkung:

OR	"true"	"false"
"true"	"true"	"true"
"false"	"true"	"false"

siehe hierzu Beispiel 2 im Abschnitt 5.7.

5.6.
Wertoperatoren

Ein Wertoperator ist ein unärer Operator. Sein Operand ist eine Variable, die rechtsbündig die 22-Bit-Adresse oder die Indexspeicheradresse des Objekts enthält, auf das zugegriffen werden soll (die also auf das Objekt zeigt). Durch den Wertoperator wird angegeben, ob es sich um ein Objekt im Indexspeicherbereich oder um ein Objekt im Kernspeicher handelt, und, falls es im Kernspeicher liegt, welchen Umfang es hat:

⟨Wertoperator⟩ v (v kann auch eine Adreß-Konstante sein).

Es gibt vier Wertoperatoren:

IVAL — v enthält eine Indexspeicheradresse, zeigt also auf ein Halbwort vom Typ INDEX

SVAL — v enthält eine Kernspeicheradresse, es zeigt auf ein Halbwort vom Typ SHORT

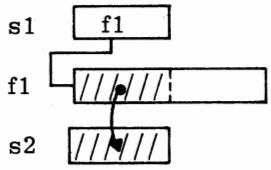
FVAL — v zeigt auf ein Ganzwort vom Typ FULL im Kernspeicher

LVAL — v zeigt auf ein Doppelwort vom Typ LONG im Kernspeicher

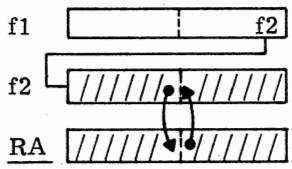
Beispiele:

```
FULL f1, f2;
s1 := REF f1;
s2 := SVAL s1;
```

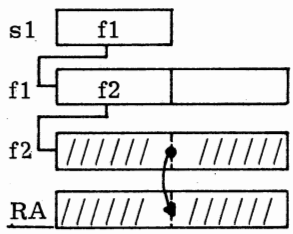
```
SHORT s1, s2;
```




```
f1 := REF f2;
RA := FVAL f1;
FVAL f1 := RA;
```




```
s1 := REF f1;
HW OF f1 := REF f2;
RA := FVAL SVAL s1;
```



5.7. Prioritäten der Operatoren

Beim Bilden von Ausdrücken ist darauf zu achten, nach welchem Prioritätsschema die verschiedenen Operatoren abgearbeitet werden. Operatoren mit niedrigerer Priorität binden schwächer als Operatoren mit höherer Priorität.

Priorität	Operatoren
10	REF, IVAL, SVAL, FVAL, LVAL
9	ABS, TK0, TK1, TK2, TK3
8	LEFT, RIGHT, LEFTC, RIGHTC
7	×, /, ××, //, ET
6	+, -, ++, --, VEL, AUT
5	<, ≤, =, ≠, ≥, >
4	NOT
3	AND
2	OR
1	:=, +:=, -:=, ++:=, --:=

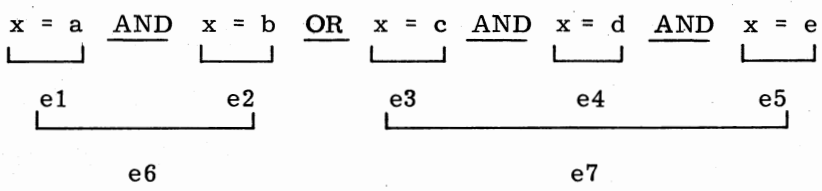
Soll ein Ausdruck anders abgearbeitet werden, als es durch das Proiritätsschema vorgegeben ist, so müssen Klammern gesetzt werden.

Beispiel 1:

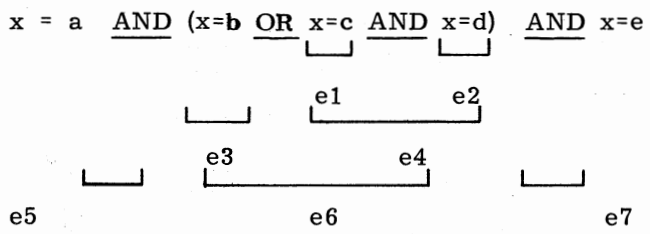
```
x := x1 + x2 × x3 LEFT x4 + x5;
                └─────────────┘
                      e1
          └────────────┘
               e2                       x := x1 +e2 + x5;
```

```
x := (x1 + x2 × x3) LEFT (x4 + x5);
           └──────┘       └──────┘
└─────────e1┘                e3
      e2                                x := e2 LEFT e3;
```

Beispiel 2: Verknüpfung von Vergleichsoperationen



e6 OR e7



e5 AND e6 AND e7

6. BEDINGUNGEN

Bedingungen sind Ausdrücke, deren Wert ein Wahrheitswert ist. Dazu gehören Vergleichsoperationen (siehe 5.5) und Tests, die im folgenden beschrieben werden:

Tests entsprechen bestimmten bedingten, Sprungbefehlen in TAS. Diese verwendet der PS440-Übersetzer, ohne daß dabei berücksichtigt wird, daß es sich hierbei in manchen Fällen um lokale Sprungbefehle handelt, die eine maximale Reichweite von ± 127 haben. Bei Überschreitung dieser Reichweite wird der Fehler erst vom TAS-Assembler erkannt.

Typenkennungs-Test: TK ⟨Typenkennungsangabe⟩ ⟨Variable⟩

Der Typenkennungstest kann auch auf eines der großen Rechenwerksregister RA, RQ, RH oder RD angewendet werden,

z. B.

TK 1 RA

Der Test hat den Wert "true", wenn im Akkumulator ein Wort mit Typenkennung 1 steht.

Anmerkung: Zwischen dem Wortsymbol TK und der Typenkennungangabe muß unbedingt ein Zwischenraum stehen (nicht zu verwechseln mit den unären Operatoren TK0, TK1, TK2 und TK3, mit denen man Typenkennungen setzen kann !).

Test, ob Bit gesetzt ist: BIT ⟨Bitnummer⟩ ⟨Variable⟩

Dieser Test kann ebenfalls nur auf eines der großen Rechenwerks-Register angewendet werden. Die Bitnummer bezeichnet das Bit, das man abfragen möchte. Ist es besetzt, so ist der Wert des Tests "true". Die Numerierung der Bits beginnt links im Wort mit Eins, das am weitesten rechts stehende Bit hat die Nummer 48,

z. B.

BIT 24 RH

Anmerkung: Der Wert des Ausdrucks wird für die Prüfung, falls nötig, in ein Ganzwort-Register geholt, bei short- und index-Größen werden dabei die linken 24 Bit gleich dem 25. gesetzt, die TK wird 1!

Test, ob rechtes Bit in RA gesetzt ist: ODD

Bei diesem Test ist keine weitere Angabe nötig. Er hat den Wert "true", wenn das Bit besetzt ist. Man verwendet diesen Test meistens, um zu prüfen, ob eine Zahl gerade oder ungerade ist. Dabei ist aber zu beachten, daß die negativen geraden Zahlen als ungerade Zahlen betrachtet werden.

Test, ob rechtes Bit in BB gesetzt ist: BODD

Hier gilt das gleiche wie bei ODD: Vorsicht mit der negativen Null!

Alarmtests: Die betreffenden Alarme werden durch die Abfrage unwirksam, sofern der Test durchgeführt wird, bevor das Rechenwerk nach Auftreten des Alarms wieder verwendet wird.

Die betreffenden Tests haben den Wert "true", wenn ein entsprechender Alarm vorgelegen hat.

ARAL Prüfung auf arithmetischen Alarm

TKAL Pürfung auf Typenkennungs-Alarm

Allgemein gilt für Vergleichsoperationen und Tests, daß ihr Wert durch die Anwendung des Operators NOT (siehe 5.3.3) negiert werden kann.

7. ANWEISUNGEN

Unter dem Begriff der Anweisung versteht man folgendes:

a)	Bedingte Anweisungen	(s. 7. 1)
b)	Fall-Unterscheidungen	(s. 7. 2)
c)	Laufanweisungen	(s. 7. 3)
d)	Sprunganweisungen	(s. 4. 4. 1)
e)	Prozeduraufrufe	(s. 4. 4. 2. 1 u. 4. 4. 2. 2 u. 4. 4. 2. 5)
f)	RETURN	(s. 4. 4. 2. 4)
g)	Befehlsaufrufe	(s. 4. 8)
h)	Tausch	(s. 5. 2)
i)	TAS-Sequenz	(s. 3. 5)
j)	Zusammengesetzte Anweisungen	(s. 4. 4. 2. 3)
k)	Ausdrücke (alle mit Hilfe von Operatoren gebildete Formeln, nicht jedoch Bedingungen)	

7. 1. Bedingte Anweisungen

Bedingte Anweisungen haben folgende Form:

IF ⟨ Bedingung ⟩ THEN ⟨Anweisungsfolge ⟩ FI;
(einfache bedingte Anweisung)

IF ⟨Bedingung ⟩ THEN ⟨Anweisungsfolge ⟩
ELSE ⟨ Anweisungsfolge ⟩ FI;
(bedingte Anweisung m. Alternative)

Eine bedingte Anweisung wird folgendermaßen ausgeführt:

Zunächst wird die Bedingung verarbeitet; ist ihr Wert "true", so wird die durch THEN eingeleitete Anweisungsfolge durchgeführt, die aus einer oder mehreren Anweisungen besteht. Ist ihr Wert "false", so wird entweder nichts getan (im Falle der einfachen bedingten Anweisung) oder die durch ELSE eingeleitete Anweisungsfolge ausgeführt.

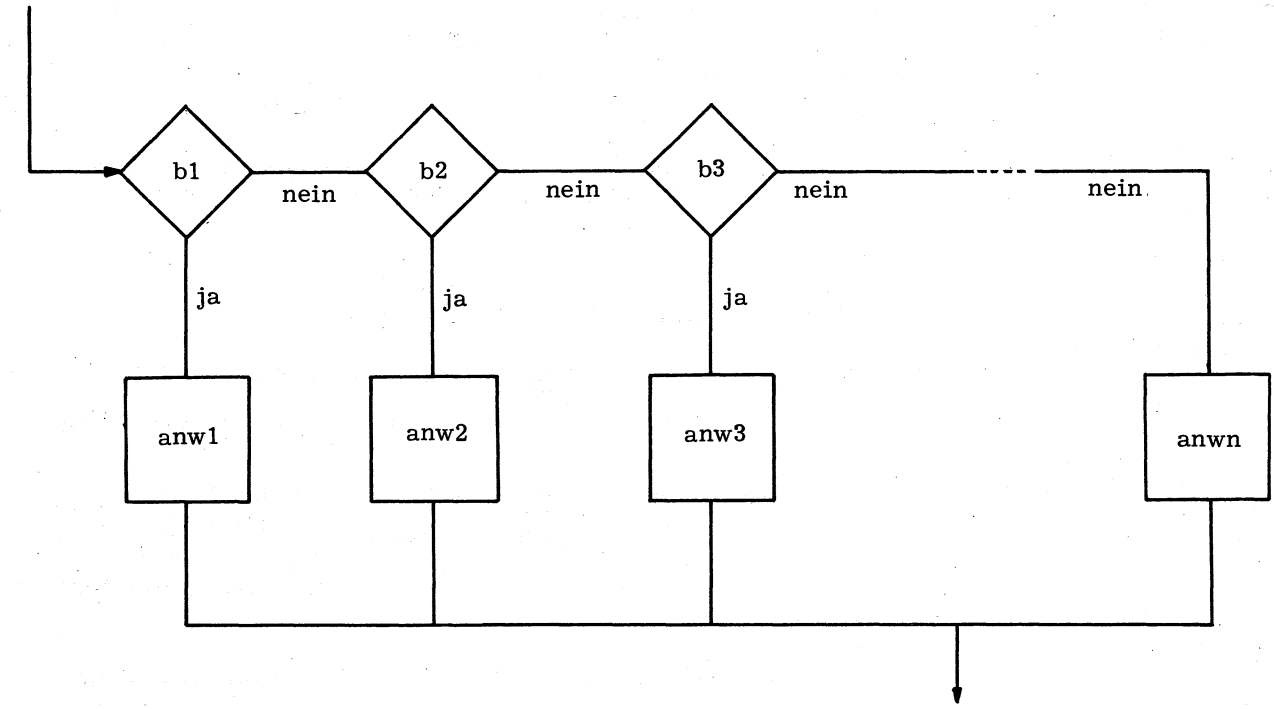
Die bedingte Anweisung wird abgeschlossen durch FI. Dieses explizit angegebene Ende der bedingten Anweisung macht es überflüssig, in ihr enthaltene Folgen von Anweisungen nochmals in irgendeiner Form zu klammern.

Bedingte Anweisungen können selbst wieder bedingte Anweisungen enthalten. Für den Fall, daß man nacheinander mehrere Bedingungen abfragen muß, kann man statt ELSE IF verkürzt ELSF schreiben, man benötigt dann am Ende nur ein FI:

```
IF  b1  THEN  anw1
      ELSE  IF   b2  THEN  anw2
                     ELSE  IF   b3  THEN  anw3
                                    :
                                  ELSE  anwn
                           FI;
            FI;
FI;
```

```
IF    b1  THEN  anw1
ELSF  b2  THEN  anw2
ELSF  b3  THEN  anw3
                  :
ELSE  anwn
FI;
```

Flußdiagramm:



7.2. Die Fall-Unterscheidung

Eine Fall-Unterscheidung ist eine Anweisung folgender Form:

CASE ⟨Ausdruck⟩ FROM ⟨Liste von Anweisungen⟩ ESAC;

Bei der Ausführung dieser Anweisung wird zunächst der auf CASE folgende Ausdruck verarbeitet. Sein Wert x wird bei Bedarf auf SHORT-Länge verkürzt und als ganze Zahl interpretiert. In Abhängigkeit von diesem Wert x wird genau diejenige Anweisung aus der Liste der Anweisungen ausgeführt, die an der x-ten Stelle in der Liste steht.

Die Liste der Anweisungen enthält n Elemente, die Zählung beginnt mit Null. Deshalb muß für den Wert **x** gelten:

$$0 \leqq x \leqq n - 1$$

Es ist zu beachten, daß der Übersetzer keine Prüfung dafür erzeugt, ob **x** im zulässigen Bereich liegt.

Eine Liste von Anweisungen sieht folgendermaßen aus:

anw0, anw1, anw2, ...

Trennzeichen ist das Komma.

Beispiele:

```
CASE s FROM CALL p0,   s: = 1, CALL p2  ESAC;
```

falls s = 0 s = 1 s = 2

```
CASE s1: = s2 FROM                                              falls
     s2: = 0,                                                     | s1 = 0
     IF s2 = 0 THEN s2: = 1; CALL p1;  ELSE CALL p2; FI,          | s1 = 1
     CASE s2-1 FROM s1: = 0, s2: = 2, s1-: = s2 ESAC,             | s1 = 2
ESAC;
```

7.3. Die Laufanweisung

Die einfachste Form der Laufanweisung ist die endlose Schleife:

DO ⟨Anweisung⟩;

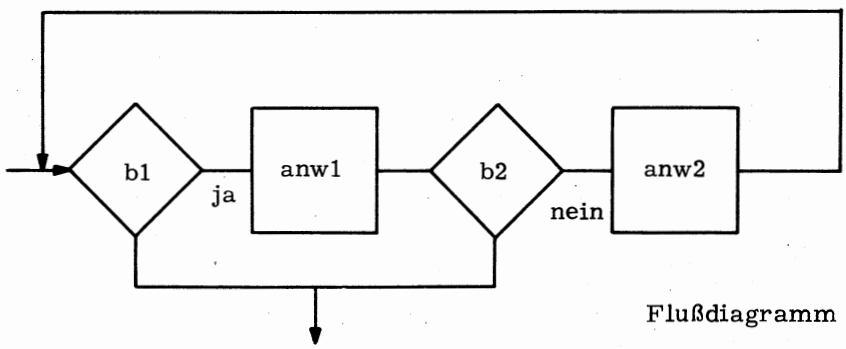
die allgemeinste Form folgende:

{FOR v}$_{0}^{1}$ {FROM e1}$_{0}^{1}$ {BY e2}$_{0}^{1}$ {TO e3}$_{0}^{1}$ {WHILE b1}$_{0}^{1}$ DO anw1

{UNTIL b2 {CONTINUE anw2}$_{0}^{1}$}$_{0}^{1}$;

Bedeutung:

v	Laufvariable	
e1	Anfangswert	Ausdrücke, die am Anfang einmal ausgewertet werden; später wird nur der Wert verwendet.
e2	Schrittweite	
e3	Endwert	
b1	Bedingung, die einen Abbruch vor dem Durchlaufen der Schleife bewirkt, falls sie nicht erfüllt ist.	
anw1	Anweisung, die bei jedem Durchlaufen der Schleife ausgeführt wird.	
b2	Bedingung, die zu einem Abbruch nach der Ausführung von anw1 führt, falls sie erfüllt ist.	
anw2	Anweisung, mit der die Schleife fortgesetzt wird, falls b2 nicht erfüllt ist.	



Flußdiagramm

anw1 kann selbst wieder eine Laufanweisung sein. Bei zwei ineinandergeschachtelten Laufanweisungen bezieht sich deswegen ein abschließendes UNTIL ... CONTINUE auf die innere Laufanweisung. In zweifelhaften Fällen verwende man eine zusätzliche BEGIN ... END - Klammerung.

Alle Angaben, die geklammert sind, können weggelassen werden, wenn man sie nicht braucht.

Es können die Abbruch-Bedingung { und / oder } die Fortsetzungsanweisung { und / oder } die gesamte Zählung FOR v FROM e1 BY e2 TO e3 entfallen.

Werden nur Teile der Zählung weggelassen, so geschieht bei der Übersetzung folgendes, je nachdem, welche Angaben fehlen: Für die Laufvariable wird vom Übersetzer eine interne Zählgröße erzeugt. Als Anfangswert wird Null angenommen, als Schrittweite Eins und als Endwert ∞.

Beim Heraussprung aus einer Laufanweisung hat die Laufvariable den letzten Wert, bei normalem Ende ist ihr Wert undefiniert.

8. DIE VERWENDUNG VON REGISTERN*)

Wenn in PS440 Register explizit verwendet werden sollen, muß folgendes beachtet werden:

PS440-Ausdrücke werden vom Übersetzer in eine Folge von TAS-Befehlen übersetzt; welche Befehlssequenz dabei im einzelnen erzeugt wird, und welche Register dabei verwendet werden, kann sich von Übersetzerversion zu Übersetzerversion ändern.

Es gibt jedoch bestimmte Fälle, bei denen festgelegt ist, welche Registerinhalte erhalten bleiben müssen und welche zerstört werden können, da sonst keine sinnvolle Verwendung der Register möglich wäre.

a) Adressierung von

- einfachen Objekten durch direkte Angabe des Namens: sämtliche Register bleiben erhalten.
- indizierten Objekten:
  BB kann zerstört werden.
- PART-Größen:
  RA und RQ werden verändert.
- Äquivalenzen:
  falls es sich bei dem durch die Äquivalenz benannten Objekt um eine einfache oder indizierte Variable handelt, gilt obiges, andernfalls können sämtliche Register zerstört werden.

b) Sprungbefehle und Prozeduraufrufe

- Bei einem Prozeduraufruf mit CALLSFB wird BB immer verändert.
- Bei Verwendung von CALL bzw. GOTO kann BB zerstört werden, wenn eine indizierte Variable angegeben wird (siehe oben).

c) Transporte

- Zuweisungsoperationen:
  Folgende Wertzuweisungen verändern nur die angegebenen Register:

FULL	→	RA, RQ, RD, RH
LONG	→	RAQ
SHORT	→	RA
INDEX	→	BB

*) Dieser Abschnitt bezieht sich nur auf die Register RA, RQ, RH, RD, BB.

Die entsprechenden Wertzuweisungen in umgekehrter Richtung auf FULL-, LONG- oder INDEX-Größen verändern keine Register, bei der Zuweisung des Inhalts von RA an eine SHORT-Größe enthält RA anschließend den auf SHORT-Länge verkürzten Wert.

Folgende Wertzuweisungen zerstören nur BB:

$$\text{INDEX} \left\{ \begin{array}{l} := \\ +:= \\ -:= \end{array} \right\} \text{INDEX}$$

$$\text{INDEX} \left\{ \begin{array}{l} := \\ +:= \\ -:= \end{array} \right\} \text{ganze Zahl} < 2^{16}$$

Folgende Wertzuweisungen verändern das angesprochene Register und BB:

$$\text{INDEX} \rightarrow \begin{array}{l} \text{RA} \\ \text{RQ} \\ \text{RD} \\ \text{RH} \end{array}$$

Tauschoperationen:

Werden zwei der großen Rechenwerksregister RA, RQ, RD, RH miteinander vertauscht, so werden nur diese beiden Register verändert.

Wird ein Register mit einer FULL-Größe vertauscht, so wird RA zerstört und das betreffende Register erhält einen neuen Wert.

Werden zwei Indexspeicher vertauscht. so wird BB zerstört.

d) Bedingungen:

Bei Tests bleiben die Registerinhalte erhalten, bei Vergleichsoperationen dagegen werden sie im allgemeinen zerstört mit einer Ausnahme: Ein Vergleich zwischen RA und RH verändert keine Register

e) Wertoperationen:

Bei Verwendung von Wertoperationen kann BB verändert werden.

In allen anderen Fällen kann über den Zustand der Register nach der Auswertung eines Ausdrucks keine Aussage gemacht werden.

ANHANG 1

STANDARDRAHMEN FÜR PS440-PROGRAMME

I N H A L T Seite

STANDARDRAHMEN FÜR PS440-PROGRAMME

Um dem PS440-Programmierer den Umgang mit dem Betriebssystem BS3 des TR440 zu erleichtern, wurde ein Satz von Unterprogrammen geschaffen, der die am häufigsten benötigten Dienste in komfortabler Form erledigt.

1. Allgemeines

Der Standardrahmen liegt als Montageobjekt mit dem Namen IOC in der Standard-Datenbasis des Betriebssystems. Er hat die nachfolgend beschriebenen Eingänge, die bei Bedarf wie folgt zu spezifizieren sind:

SPEC IOC PROCSFB IOC0, KGZ;

SPEC IOC PROC IOC1, IOC2, ... IOC21, IOC22, UMTZ;

Der Aufruf der einzelnen IOC-Dienste erfolgt mit CALL bzw. CALLSFB (Parameterübergabe siehe Beschreibung der einzelnen IOC-Dienste).

Der Standardrahmen benutzt mit dem PS440-Programm gemeinsame Indexspeicher; die Indexspeicher Null bis einschließlich 7. Diese werden beim Aufruf eines IOC-Dienstes verändert und in undefiniertem Zustand hinterlassen, darin enthaltene Information aus dem PS440-Programm geht also dabei verloren. Es ist daher zweckmäßig, am Anfang eines PS440-Programmes 8 Indexspeicher zu vereinbaren, die im PS440-Programm selbst nicht verwendet werden, z. B. RECORD 8 INDEX dummy;.

Die dynamisch ersten Befehle eines Programms müssen der Aufruf von IOC0 mit seiner Versorgung sein, wobei die gesamte Anfangsorganisation erledigt wird.

Die Angabe der Startadresse des Programms muß im Hauptprogramm liegen und kann im Gegensatz zu allen anderen statischen Voreinstellungen nicht in IOC0 angegeben sein. Diese Angabe muß mit einem TAS-Einschub erfolgen:

/START ⟨Anfangsadresse⟩/;

Anfangsadresse ist der Name einer Marke im PS440-Programm, auf der das fertig übersetzte und montierte Programm (der Operator) später gestartet werden kann.

2. Liste der IOC-Dienste

Falls nichts anderes angegeben worden ist, handelt es sich um SU-Prozeduren.

IOC0	SFB-Prozedur Anfangsorganisation	(3.1.1)
IOC1	Arbeitsspeicher-Reservierung	(3.5.5)
IOC2	Adresse Startsatz	(3.5.3)
IOC3	Leseprogramm	(3.4.1)
IOC4	Vormerken einer Dumpausgabe im Alarmfall	(3.5.4)
IOC5	Ausgabe Fehlermeldung	(3.2.8)
IOC6	Druck Zeilenvorschub, Text, Zeilenvorschub	(3.2.7)
IOC7	Druck Text, Zeilenvorschub	(3.2.6)
IOC8	Ausgabe Zeilenvorschub	(3.2.3)
IOC9	Druck Text	(3.2.5)
IOC10	Druck Festpunktzahl	(3.2.9)
IOC11	Umwandeln Festpunktzahl in druckfertige Form	(3.3.3)
IOC12	Normalende	(3.1.2)
IOC13	Ausgabe Einzelzeichen	(3.2.1)
IOC14	Ausgabe eines Wortes aus RA	(3.2.2)
IOC15	Vorschub auf nächste Seite	(3.2.4)
IOC16	Start Protokolloperator	(3.5.2)
IOC17	Operatorende mit Fehlermeldung	(3.1.3)
IOC18	Löschprogramm	(3.5.6)
IOC19	Kopftexterweiterung	(3.5.1)
IOC20	Informiere über EA-Zähler	(3.4.2)
IOC21	Anmelden Fortsetzungs-Adresse Alarmfeld	(3.5.7)
IOC22	Hexadekadischer Druck eines Ganzwortes	(3.2.10)
KGZ	SFB-Prozedur konvertiere ganze Zahl	(3.3.1)
UMTZ	Umschlüsseln Tetraden in Zentralcode-Zeichen	(3.3.2)

3. Beschreibung der einzelnen IOC-Dienste

3.1. Anfang und Ende Operatorlauf

Ein PS440-Programm muß statisch und damit dynamisch mit einem Aufruf der Anfangsorganisation beginnen, da sonst wegen fehlender Voreinstellungen undefinierte Resultate auftreten.

Ebenso ist für einen ordnungsgemäßen Abschluß der Ein/Ausgabetätigkeiten und die Abmeldung beim Betriebssystem ein entsprechender IOC-Aufruf am dynamischen Ende des Objektprogramms unbedingt erforderlich.

3.1.1. Anfangsorganisation

Aufruf: CALLSFB IOC0;

Versorgung: In RA ist die Anfangsadresse der Kopfzeile mitzuliefern, in RQ ein Wort mit TK3, das am Anfang und am Ende des Operatorlaufs zusammen mit dem Operatornamen gedruckt wird, z. B. die Maintenancenummer.

Beispiel:

```
RA := REF "PS440-PROGRAMM *037";
RQ := "006203";
CALLSFB IOC0;
```

Zur Anfangsorganisation gehören statisch erforderliche Angaben, z. B. Setzen des Unterprogramm-Ordnungs-Zählers und der Indexbasis, Anmeldung der Alarmadresse für Hardware- und SSR-Fehler. Es werden Voreinstellungen für alle anderen IOC-Dienste vorgenommen, so daß dort später keine weiteren Anfangsaufrufe mehr nötig sind, und es wird der Speicher mit Null, TK3 vorgelöscht. Außerdem wird ins Ablaufprotokoll eine Kopfzeile gedruckt und der Startsatz des Operators auf eventuelle Eingabedaten untersucht.

3.1.2. Normalende

Aufruf: CALL IOC12;

Versorgung: keine

Wirkung: Auf dem Ablaufprotokoll wird ein Schlußtext ausgegeben, der die vom Operator benötigte Rechenzeit enthält, und der Operatorlauf wird beendet.

3.1.3. Irreguläres Programmende

Aufruf: CALL IOC17;

Versorgung: Fall 1:

RA = Anfangsadresse eines Textes

(RA darf nicht Typenkennung 3 haben, siehe Fall 2!)

Wirkung: Im Ablaufprotokoll wird der angegebene Text - eingeleitet mit "FEHLER:" und abgeschlossen mit "ABBRUCH" - ausgegeben, anschließend wird wie bei Normalende verfahren, jedoch wird zusätzlich dem Betriebssystem die Tatsache des irregulären Endes mitgeteilt.

Versorgung: Fall 2:

RA hat Typenkennung 3, die Informationsbits sind ohne Bedeutung.

Wirkung: Der gesamte Abschnitt wird sofort abgebrochen. Es erfolgen keine weiteren Druckausgaben. War das dynamisch zuletzt ausgegebene Zeichen kein Zeilenwechsel, so geht die letzte Druckzeile verloren (s. 3.2).

Von diesem Aufruf sollte nur im Notfall oder bei speziellen organisatorischen Operatoren Gebrauch gemacht werden.

3.2. Druckprogramme

Druckprogramme geben Texte in das Ablaufprotokoll aus. Als Text wird hier eine im Speicher hintereinanderliegende Folge von Worten mit Typenkennung 3 aufgefaßt, die jeweils 6 Zentralcode-Zeichen enthalten. Abbruchkriterium ist entweder ein Wort mit Typenkennung ungleich 3 oder das Zeichen "Textende", dezimal 37; beides gehört dann nicht mehr zum Text.

Die von den verschiedenen Aufrufen der Druckprogramme stammenden Texte werden zunächst aufgesammelt, bis entweder die Maximalzahl der Zeichen in einer Zeile erreicht ist oder ein Vorschubzeichen im Text auftritt. Erst dann wird eine ganze Zeile ausgegeben und das Vorschubzeichen für die nächste Ausgabe vorgemerkt.

Die Reihenfolge der Druckausgaben entspricht dann nicht der Reihenfolge der Texte im Ablaufprotokoll, wenn dazwischen von anderen Programmen als den IOC-Diensten gedruckt wird und vorhergehende IOC-Ausgaben nicht mit Zeilenvorschub abgeschlossen wurden.

3.2.1.
Ausgabe Einzelzeichen

Aufruf: CALL IOC13;

Versorgung: RA enthält in den letzten 8 Bits ein Zentralcode-Zeichen, Typenkennung von RA kann 1, 2 oder 3 sein.

Wirkung: Das betreffende Zeichen wird zur Ausgabe vorgemerkt. Vorschubzeichen werden entsprechend interpretiert.

3.2.2.
Ausgabe eines Wortes

Aufruf: CALL IOC14;

Versorgung: RA enthält mit Typenkennung 3 ein Wort, dessen 6 Zentralcodezeichen ausgegeben werden sollen.

Wirkung: Die 6 Zeichen werden, soweit sie ungleich Null (Ignoriere-Zeichen) sind, zur Ausgabe vorgemerkt. Keine Wirkung ergibt sich, wenn die Typenkennung in RA nicht 3 ist.

3.2.3.
Vorschub um eine Zeile

Aufruf: CALL IOC8;

Versorgung: keine

3.2.4.
Vorschub auf neue Seite

Aufruf: CALL IOC15;

Versorgung: keine

3.2.5.
Ausgabe Zeichenreihe

Aufruf: CALL IOC9;

Versorgung: RA = Anfangsadresse eines Drucktextes

Wirkung: Der Text wird in Einzelzeichen zerlegt, auf Vorschub- und Ignoriere-Zeichen (= Null) geprüft und vom Einzel-Zeichen-Ausgabeprogramm weiter verarbeitet. Die Zeichenreihe muß mit einem Wort anderer Typenkennung oder dem Textende-Zeichen abgeschlossen sein.

3.2.6.
Ausgabe Zeichenreihe mit abschließendem Zeilenvorschub

Aufruf: CALL IOC7;

Versorgung: RA = Anfangsadresse Text

Wirkung: Dieses Unterprogramm entspricht der Ausgabe Zeichenreihe, nur daß anschließend noch alle aufgesammelten Zeichen ausgegeben sind und ein Zeilenvorschub vorgemerkt ist.

3.2.7.
Ausgabe Zeichenreihe mit einleitendem und abschließendem Zeilenvorschub

Aufruf: CALL IOC6;

Versorgung: RA = Anfangsadresse Text

Wirkung: Dieses Unterprogramm entspricht ebenfalls der Ausgabe Zeichenreihe, nur daß der Text auf eine eigene Zeile abgesetzt wird.

3.2.8.
Ausgabe Fehlermeldung

Aufruf: CALL IOC5;

Versorgung: RA = Anfangsadresse eines Drucktextes

Wirkung: Im Ablaufprotokoll wird eine neue Zeile begonnen, der Text "FEHLER:" ausgegeben, anschließend der gewünschte Text.

3.2.9.
Druck Festpunktzahl

Aufruf: CALL IOC10;

Versorgung: RA = Festpunktzahl mit Typenkennung 1 und $0 \leqq RA < 10^{12}$. Es ist zu beachten, daß Typenkennung, Bereichsüberschreitung und Vorzeichen nicht geprüft und nicht berücksichtigt werden, was zu einem Alarm führen kann.

Ergebnis: Ist der Betrag der zu druckenden Zahl kleiner als 10^6, so werden 6, sonst 12 Zeichen in das Ablaufprotokoll abgesetzt (s. 3.3.3).

3.2.10.
Hexadekadischer Druck eines Ganzwortes

Aufruf: CALL IOC22;

Versorgung: BB = Adresse eines Ganzwortes

Ergebnis: Im Ablaufprotokoll werden ohne weitere Zwischenräume oder Zeilenvorschübe abgesetzt:

Typenkennung, 1 Zwischenraum, 12 Tetraden.

3.3.
Konvertierungsprogramme

3.3.1.
Konvertieren ganze Zahl

Aufruf: CALLSFB KGZ;

Versorgung: RA = Festpunktzahl mit Typenkennung 1 und $0 \leqq RA \leqq$ '09184E729FFF', andernfalls tritt ein Alarm auf.

Ergebnis: RAQ enthält rechtsbündig in 13 Tetraden mit Typenkennung 1 die in das Dezimalsystem konvertierte Zahl.

3.3.2.
Umwandeln Tetraden in Zentralcode-Zeichen

Aufruf: CALL UMTZ;

Versorgung: beliebiges Bitmuster in RA

Ergebnis: RAQ enthält 12 Zentralcode-Zeichen (Typenkennung 3), die die druckfertige Form des Bitmusters in RA darstellen. 0 bis 9 werden in die Zentralcode-Zeichen (hexadekadisch) B0 bis B9, A bis F in C0 bis C5 umgeschlüsselt.

3.3.3.
Umwandeln Festpunktzahl in druckfertige Form

Aufruf: CALL IOC11;

Versorgung: RA = Festpunktzahl mit Typenkennung 1 und $0 \leqq RA < 10^{12}$, andernfalls tritt ein Alarm auf, da Typenkennung, Vorzeichen und Bereichsüberschreitung nicht geprüft werden.

Ergebnis: Fall 1:

Die Zahl ist kleiner als 10^6
RA wird mit TK3, 0 gelöscht;
RQ enthält in Zentralcode-Zeichen die Zahl; führende Nullen sind durch Zwischenraum ('AF') ersetzt, nur die letzte Null bleibt, falls die Zahl = 0 ist.

Fall 2:

Die Zahl ist größer als in Fall 1.
Außer RQ enthält RA ebenfalls Zentralcode-Zeichen, die druckfertige Zahl steht also rechtsbündig in RAQ. Führende Nullen sind durch Zwischenraum ersetzt.

3.4. Eingabeprogramme

3.4.1.
Leseprogramm

Aufruf: CALL IOC3;

Versorgung: keine

Ergebnis: Aus den Eingabedaten wird das nächste Zeichen im Zentralcode mit Typenkennung 1 nach RA und RH geliefert. Am Ende einer Zeile wird ein Vorschubzeichen geliefert. Sind die Eingabedaten erschöpft, so wird - unter Umständen beliebig oft - das Zeichen EM ("End of Medium", dezimal 33) abgegeben, ohne daß eine Fehlermeldung erfolgt.

Fehlerausgang: Die Behandlung eventueller Versorgungsfehler liegt beim Unterprogramm S&GZF, einem Montageobjekt in der Standarddatenbasis, das für das Lesen der Eingabe herangezogen wird. Bei der Anfangsorganisation sind im Zusammenhang hiermit folgende Fehler möglich:

Keine Daten vorhanden
oder
Eingabedatensatz vorhanden aber leer.

In diesen irregulären Fällen erfolgt ein Programmabbruch erst dann, wenn der erste Aufruf für das Leseprogramm gegeben wurde.

3.4.2.
Informiere über
EA-Zähler

Aufruf: CALL IOC20;

Versorgung: keine

Ergebnis: RA = Positionszähler für Eingabe; er steht nach Zeilenwechsel auf +Null und gibt die Anzahl der bereits gelesenen Zeichen einer Zeile an.

RQ = Zeilennummer, wie sie von S&GZF geliefert wurde.

TK = 3: Zeilennummer liegt in druckfertiger Form vor;

TK = 1: Zeilennummer wurde als hexadekadische Zahl geliefert, nach einer Konvertierung ergibt sich eine sechsstellige Zahl!

RH = Positionszähler für die Ausgabe; er steht am Zeilenanfang auf +Null und gibt die Anzahl der zum Druck aufgesammelten Zeichen einer Zeile an.

3.5. Programme für spezielle Systemdienste

3.5.1. Kopftexterweiterung

Aufruf: CALL IOC19

Versorgung: RA = Anfangsadresse eines Textes.

Ergebnis: Der angegebene Text wird an das System weitergegeben und erscheint so lange am Anfang jeder Seite unter der Systemkopfzeile, bis eine andere Kopftexterweiterung erfolgt oder ein anderer Operator gestartet wird.

3.5.2.
Start Protokolloperator

Aufruf: CALL IOC16;

Versorgung: RA = Adresse Versorgungsblock oder +Null.

Ergebnis: Vorhandene Eingabedaten werden an den Protokolloperator übergeben. Ist das Hauptprogramm mit dem UEBERSETZE-Kommando gestartet worden, so wird die Spezifikation PROTOKOLL ausgewertet; andernfalls wird ein Standardprotokoll erzeugt. Ein etwa angegebener Versorgungsblock dient zum Einmischen von Fehlermeldungen in das Protokoll gemäß Statusdokumentation IV, 4.9.2. [4]. Er besteht aus vier Ganzworten:

1. GW: Gebietskennzeichen (Nummer oder Name) eines Gebietes mit festen Fehlertexten;

2. GW: Kennzeichen eines Gebietes mit variablen Fehlertexten;

3. GW: Kennzeichen eines Gebietes mit Fehlerelementen;

4. GW: linkes Halbwort:
gebietsrelative Anfangsadresse der variablen Texte,

rechtes Halbwort:
gebietsrelative Anfangsadresse der Fehlerelemente.

Mehrere der angegebenen Gebiete können identisch sein, sie müssen im Kernspeicher liegen. Fehlt ein Gebiet, so ist +Null anzugeben. Die Versorgungsinformation wird ungeprüft weitergegeben!

3.5.3.
Adresse Startsatz

Aufruf: CALL IOC2;

Versorgung: keine

Ergebnis: RA = Anfangsadresse des Speicherbereichs, in dem der eigene Startsatz steht.

3.5.4.
Vormerken Dumpausgabe für Alarmfall

Aufruf: CALL IOC4;

Versorgung: linkes und rechtes Halbwort in RA je eine Adresse - die Anfangs- und Endadresse eines Speicherbereichs, Reihenfolge beliebig.

Ergebnis: Der durch das Adressenpaar gekennzeichnete Speicherbereich wird im Falle eines späteren Alarms binär gedumpt. Darüber hinaus erhält man im Alarmfall immer einen Binärdump

über alle adressierten Laufzeitgebiete.

Fehlerausgang: Beim Eintrag der Adressen wird keine Prüfung auf eventuellen Speicherschutz-Alarm bei späterem Zugriff vorgenommen. In diesem Fall beendet später das Alarmprogramm seine Ausgaben vorzeitig mit der Meldung "SSR-FEHLER IM ALARMPROGRAMM" oder "ALARM IM ALARMPROGRAMM".

3.5.5. Anmelden Arbeitsspeicher

Aufruf: CALL IOC1;

Versorgung: keine

Ergebnis: RA enthält mit Typenkennung 2 im linken Halbwort die End-, im rechten Halbwort die Anfangsadresse des dynamisch verfügbaren Arbeitsspeichers als absolute (nicht gebietsrelative) Adressen. Mehrfacher Aufruf ist zulässig und liefert immer das gleiche Resultat. Es wird höchstens soviel Arbeitsspeicher geliefert, wie in IOC0 voreingestellt wurde.

Fehlerausgang: Steht zu wenig Arbeitsspeicher zur Verfügung, da dem System eine gewisse Reserve belassen werden muß, so endet das Programm mit der Fehlermeldung: "FEHLER: ZU WENIG KERNSPEICHER. ABBRUCH".

3.5.6. Löschprogramm

Aufruf: CALL IOC18;

Versorgung: linkes und rechtes Halbwort von RA enthalten Anfangs- und Endadresse des zu löschenden Bereichs (geradzahlig) in beliebiger Reihenfolge;
RQ enthält das Muster, mit dem der Bereich vorgelöscht werden soll.

Ergebnis: Der Speicherbereich wird mit dem angegebenen Muster vorbesetzt.

3.5.7. Anmelden der Fortsetzungsadresse für den Alarmfall

Aufruf: CALL IOC21;

Versorgung: rechtes Halbwort von RA enthält die Fortsetzungsadresse.

Ergebnis: Die angegebene Adresse wird als Fortsetzungsadresse vorgemerkt, falls nach einem Alarm oder SSR-Fehler die interne Alarmsperre zu löschen und die Verarbeitung an der angegebenen Stelle fortzusetzen ist.

Bei Ansprung der Fehleradresse werden mitgeliefert:

RAQ = Alarminformation vom Abwickler

RD = Gebietsnummer des "internen Gebietes" (vgl. Status IV, 4.1.3.2).

RH = Anfangsadresse des Alarmkellers (besetzt durch SSR 4 8)

ANHANG 2

BEISPIEL FÜR EIN PS440-PROGRAMM

I N H A L T Seite

BEISPIEL FÜR EIN PS440-PROGRAMM

Im folgenden PS440-Programm wird ein Leseprogramm getestet, das (in erweiterter Form) als Teil eines Übersetzers geplant ist und im Hinblick darauf den Aufbau bestimmter Listen enthält, die der Übersetzer später benötigt.

1.
Programmanfang
(s. hierzu auch Anhang 1)

```
000010  SEGMENT LESEPROGRAMM;
000020  SPEC IOC PROCSFB IOC0;
000030  SPEC IOC PROC IOC3,IOC6,IOC8,IOC9;
000040  SPEC IOC PROC IOC10,IOC12,IOC13,IOC14;
000050  RECORD 8 INDEX DUMMY;
000060T */START ANF/*;
000070
000080
000090C CO STARTADRESSE DES PROGRAMMS;
000100S ANF: RQ:=''000000'';
000110S      RA:= REF ''LESEPROGRAMM*037'';
000120       CALLSFB IOC0;
000130       CALL IOC8;
000140       CALL IOC8;
000150
```

Die hier und im folgenden aufgeführten Programmteile wurden dem vom PS440-Übersetzer erzeugten Standardprotokoll entnommen.

Dazu folgende Bemerkungen:

Am Anfang einer Zeile steht die Nummer der Quellzeile (evtl. die betreffende Satznummer aus der Datei, die die Quelle enthält), und zwar 6stellig mit führenden Nullen. Nach zwei Zwischenräumen folgt der Ausdruck der Quellzeile. Enthält die Zeile einen Kommentar (beginnend mit CO), einen TAS-Einschub oder eine Zeichenreihe (= String), so erscheint unmittelbar hinter der Zeilennummer entsprechend ein C, ein T oder ein S statt des ersten Zwischenraums.

2.
Aufgabe des Leseprogramms

Es soll ein Text eingelesen werden, der aus einer Folge von bestimmten Textelementen besteht.

a) Identifier
(eine beliebige Folge von Buchstaben und Ziffern, beginnend mit einem Buchstaben)

b) Positive ganze Zahlen
(eine Folge von höchstens 12 Ziffern)

c) Sonderzeichen

Die einzelnen Textelemente werden vom Leseprogramm erkannt und entsprechend weiterverarbeitet:

Identifier werden in eine Identifierliste mit Namen IDLIST, Zahlen in eine Zahlliste mit Namen ZL abgelegt; das Leseprogramm gibt die Adresse aus der entsprechenden Liste ab, auf der das Textelement dann zu finden ist.

Sonderzeichen werden direkt übergeben mit 3 Ausnahmen: das ignoriere-Zeichen (Oktade Null) und der Zeilenvorschub werden ignoriert, der Zwischenraum wirkt als Trennzeichen, wird aber nicht weitergegeben.

Sind die Eingabedaten erschöpft, wird das Endezeichen EM (Oktade "21") geliefert.

3.
Lesematrix und
Matrixroutinen

Aus der Aufgabenstellung ergibt sich ein Problem:
Das Leseprogramm liest einzelne Zeichen, die Textelemente "Identifier" und "Zahl" bestehen aber meistens aus mehreren Zeichen, die erst gesammelt werden müssen, bevor das Textelement weiterverarbeitet werden kann. Das Leseprogramm wird sich also längere Zeit in einem bestimmten "Zustand" befinden (Zustand "Aufreihen Zahl" oder Zustand "Aufreihen Identifier"), bis ein Trennzeichen folgt, das wieder einen dem Anfang gleichen Zustand herstellt. In jedem dieser Zustände muß damit gerechnet werden, daß das nächste eingelesene Zeichen irgend ein beliebiges Zeichen ist. Damit ist bereits ein Schema (= die Lesematrix) gegeben, aus dem klar hervorgeht, welche verschiedenen Fälle auftreten können, und danach ist eindeutig zu entscheiden, was dann getan werden muß:

Eingabe-Zeichen / Zustand	Buchstabe	Ziffer	Sonderzeichen	Zwischenraum	Neue Zeile ignorieren
leer	L2	L5	L8	L1	L1
Identifier	L3	L3	L4	L4	L1
Ganze Zahl	L7	L6	L7	L7	L1

```
000160
000170  RECORD 15 SHORT MATRIX IS
000180        (( L2, L5, L8, L1, L1 ),
000190         ( L3, L3, L4, L4, L1 ),
000200         ( L7, L6, L7, L7, L1 ));
```

In jedem Feld der Matrix steht (als Adreßkonstante) der Name des Unterprogramms (= Name der Matrixroutine), das in dem betreffenden Fall ausgeführt werden muß. Welcher Fall gerade vorliegt, ist durch den Zustand (die Zeile der Matrix) und durch ein gelesenes Zeichen (die Spalte der Matrix) festgelegt. Damit sind bereits die beiden Zeiger bekannt, mit deren Hilfe man im PS440-Programm auf diese Matrix zugreifen kann:

ZEILE — der die Anfangsadresse der Zeile enthält, die dem aktuellen Zustand entspricht (Voreinstellung "leer");

ZIND — (Zeilenindex), der die Nummer der Spalte der Matrix enthält, die dem gelesenen Zeichen zugeordnet ist.

Der Zugriff auf das Leseprogramm erfolgt dann später mit
CALL ZEILE [ZIND];

Dies bewirkt einen Unterprogrammsprung auf die entsprechende Matrixroutine:

```
000210
000220  SHORT LEER IS MATRIX,
000230          ID IS MATRIX[5],
000240          GZ IS MATRIX[10];
000250  INDEX ZIND := 4;
000260  SHORT ZUSTAND := LEER,
000270        ZEILE = SVAL ZUSTAND;
000280

000290C PROC L1 IS CALL LESEN; CO LESEN NAECHSTES ZEICHEN;
000300
000310  PROC L2 IS
000320C BEGIN ZUSTAND:=ID;      CO EINSTELLEN ZUST. IDENTIFIER;
000330C       CALL IDANF;       CO ANFANGSBEH. IDENTIFIER;
000340  END L2;
000350
000360  PROC L3 IS
000370C BEGIN CALL AUFID;       CO AUFREIHEN IDENTIFIER;
000380C       CALL LESEN;       CO LESEN NAECHSTES ZEICHEN;
000390  END L3;
000400
000410  PROC L4 IS
000420C BEGIN ZUSTAND:=LEER;    CO EINSTELLEN ZUSTAND LEER;
000430C        CALL IDEND;      CO ENDEBEHANDLUNG IDENTIFIER;
000440C        CALL HP;         CO ID. ABLIEFERN AN HP;
000450  END L4;
000460
000470  PROC L5 IS
000480C BEGIN ZUSTAND:=GZ;      CO EINSTELLEN ZUST. GANZE ZAHL;
000490C       CALL ZLANF;       CO ANFANGSBEH. GANZE ZAHL;
000500  END L5;
000510
000520  PROC L6 IS
000530C BEGIN CALL AUFZL;       CO AUFREIHEN GANZE ZAHL;
000540C       CALL LESEN;       CO LESEN NAECHSTES ZEICHEN;
000550  END L6;
000560
000570  PROC L7 IS
000580C BEGIN ZUSTAND:=LEER;    CO EINSTELLEN ZUSTAND LEER;
000590C       CALL ZLEND;       CO ENDEBEHANDLUNG GANZE ZAHL;
000600C       CALL HP;          CO GZ ABLIEFERN AN HP;
000610  END L7;
000620
000630  PROC L8 IS
000640C BEGIN CALL SZEIN;       CO SONDERZEICHENBEHANDLUNG;
000650C       CALL HP;          CO GZ ABLIEFERN AN HP;
000660C       CALL LESEN;       CO LESEN NAECHSTES ZEICHEN;
000670  END L8;
000680
```

In den einzelnen Matrixroutinen wird jeweils der Zustand (die Zeile) und der Zeilenindex für den nächsten Zugriff auf das Leseprogramm eingestellt.

4. Unterprogramme zu den Matrixroutinen

4.1. Lesen nächstes Zeichen

Hier wird mit Hilfe von IOC3 (siehe Anhang 1) das nächste Zeichen gelesen, und in Abhängigkeit davon der Zeilenindex ZIND eingestellt:

```
000690
000700  PROC LESEN IS
000710  BEGIN CALL IOC3;
000720        FULL ZEI:=RA;
000730        IF ZEI=0 OR ZEI='1H15' THEN ZIND :=4;
000740        ELSF ZEI='1HAF' THEN ZIND:=3;
000750        ELSF ZEI LE '1HB9' AND ZEI GE '1HB0'
000760                              THEN ZIND:=1;
000770        ELSF ZEI GE '1HC0' THEN ZIND:=0;
000780        ELSE ZIND:=2;
000790        FI;
000800  END LESEN;
000810
000820
```

4.2. Behandlung des Textelements "Ganze Zahl"

Die Festpunktzahl muß zunächst aufgebaut werden; das geschieht im Ganzwort ZAHL mit Hilfe der Prozeduren ZLANF (Voreinstellung) und AUFZL (Aufreihen der Zahl). Dabei wird vorausgesetzt, daß die Zahl den Bereich für Festpunktzahlen nicht überschreitet (man sollte später noch eine Prüfung auf "Zahl-Überlauf" einbauen). Die fertig aufgereihte Zahl soll mit Hilfe der Prozedur ZLEND in eine Zahlliste ZL abgelegt werden, die aus Ganzworten mit Typenkennung 1 (= Festpunktzahl) besteht und mit einem Wort mit Typenkennung Null abgeschlossen ist, auf das der Zahllisten-Index ZLIND zeigt (Voreinstellung: ZLIND zeigt auf das erste Wort der Zahlliste, die Liste enthält also nur das Abschlußwort). Bevor die Zahl jedoch in die Zahlliste eingetragen wird, muß geprüft werden, ob sie nicht schon vorhanden ist (Durchsuchen einer Liste mit dem TLI-Befehl siehe Große Befehlsliste [2]). Wird die Zahl neu eingetragen, so muß das Schlußwort der Liste wieder neu besetzt werden; das geschieht durch Abspeichern von '0H0' auf EWORT, das in Abhängigkeit von ZLIND immer das Schlußwort der Liste benennt. Als Ergebnis der Prozedur ZLEND wird eine Adresse nach RA geliefert; entweder die Adresse, auf der die Zahl eingetragen wurde oder die Adresse, auf der sie gefunden wurde.

```
000830  FULL ZAHL;
000840  RECORD 500 FULL ZL;
000850  INDEX ZLIND:=REF ZL;
000860  FULL EWORT= FVAL ZLIND:='0H0';
000870
000880  PROC AUFZL IS ZAHL:=ZAHL*10+TK1 ZEI ET '1HF';
000890
000900  PROC ZLANF IS ZAHL:=0;
000910
000920  PROC ZLEND IS
000930  BEGIN RD:=ZAHL;
000940T       */TLI ZL/*;
000950        IF TKAL THEN EWORT:=ZAHL;
000960                     ZLIND+:=2;
000970                     EWORT:='0H0';
000980                     RA:=ZLIND-2;
000990                ELSE RA:=BB;
001000        FI;
001010  END ZLEND;
001020
001030
```

4.3. Behandlung des Textelements "Identifier"

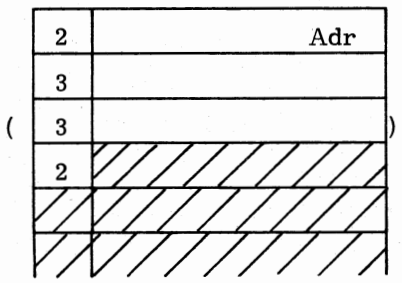
Ein Identifier kann aus beliebig vielen Zeichen bestehen. Diese werden zunächst oktadenweise (als Zentralcode-Zeichen) von rechts nach links in das Aufreihwort IWORT hineingeschoben, das mit Null, Typenkennung 3 vorgelöscht ist und 6 Zentralcode-Zeichen aufnehmen kann. Zur Zählung der Schifts wird der Zähler IDZ verwendet, der von 48 in Schritten von -8 läuft und die Anzahl der noch freien Stellen in IWORT angibt. Sobald IDZ gleich Null ist, wird IWORT in den Lesekeller LK eingetragen. Dieser Lesekeller ist für den Fall notwendig, daß der Identifier aus mehr als 6 Zeichen besteht. In diesem Fall ist IWORT voll besetzt, bevor der Identifier fertig aufgereiht ist. Der Inhalt von IWORT wird also in den Lesekeller transportiert, und zwar auf die Adresse, die im Lesekeller-Index LKIND steht (LKIND zeigt immer auf das letzte Wort im Lesekeller, auf das Schlußwort mit Null, Typenkennung 2). Daraufhin wird LKIND um 2 erhöht und ein neues Schlußwort eingetragen, da das alte überspeichert wurde. IWORT wird wieder neu vorgelöscht, IDZ neu eingestellt, und die Identifier-Aufreihung wird fortgesetzt (die Prozedur IDANF1).

Die Voreinstellungen zu Beginn der Identifier-Aufreihung werden von der Prozedur IDANF erledigt, die Identifier-Aufreihung geschieht in der Prozedur AUFID.

Die Endebehandlung des Identifiers wird von der Prozedur IDEND durchgeführt. Hier wird gegebenenfalls das letzte Teilwort des Identifiers (oder der Identifier selbst, falls er aus weniger als 6 Zeichen besteht) um die in IDZ angegebenen Stellen nach links geschiftet und in den Lesekeller eingetragen. Dann wird die Prozedur IDEINTRAG aufgerufen, mit deren Hilfe der Identifier in die Identifierliste IDLIST eingetragen wird (siehe unten). Als Ergebnis von IDEND steht in RA die Adresse, auf der der Identifier in IDLIST zu finden ist.

```
001040  RECORD 20 FULL LK;
001050  INDEX LKIND;
001060
001070  PROC IDANF1 IS
001080  BEGIN FULL IWORT:='3H0';
001090        SHORT IDZ:=48;
001100  END IDANF1;
001110
001120  PROC IDANF IS
001130  BEGIN CALL IDANF1;
001140        LKIND:=REF LK;
001150        FVAL LKIND:='2H0';
001160  END IDANF;
001170
001180  PROC AUFID IS
001190  BEGIN IWORT:=IWORT LEFT 8 VEL ZEI;
001200        IDZ-:=8;
001210        IF IDZ LE 0 THEN FVAL LKIND:=IWORT;
001220                         LKIND+:=2;
001230                         FVAL LKIND:='2H0';
001240                         CALL IDANF1;
001250        FI;
001260  END AUFID;
001270
001280  PROC IDEND IS
001290  BEGIN IF IDZ NE 48 THEN FVAL LKIND:=IWORT LEFT IDZ;
001300                          LKIND+:=2;
001310        FI;
001320        FVAL LKIND:='2H0';
001330        CALL IDEINTRAG;
001340  END IDEND;
001350
```

Die Identifier-Liste IDLIST setzt sich aus (maximal 64) Teillisten zusammen, deren Elemente in beliebiger Reihenfolge stehen. Ein Teillisten-Element hat folgende Struktur:



TK

← Folgeadresse der Teilliste bzw. Null

← entfällt, falls der Identifier weniger als 7 Zeichen hat

← nächstes Listenelement (oder Schlußwort mit TK 2)

Die Zugehörigkeit eines Identifiers zu einer bestimmten Teil-Liste wird vor dem Eintrag durch eine "Kennzahl"-Berechnung (mit Hilfe eines Hash-codes) ermittelt. Diese Kennzahlen können - möglichst gleichverteilt - Werte von 0 bis 63 annehmen (s.i. Programm Verwendung der Maske IDVLA). Der Identifier kann ein oder zwei Ganzworte lang sein, er wird mit einem Wort mit Typenkennung 2 abgeschlossen (entweder durch das Schlußwort der Liste oder durch das erste Wort des nächsten Listenelements; s. oben). Auf das letzte Wort der Liste, das Schlußwort, zeigt der Pegel IDIND.

Die Folgeadresse einer Teilliste weist auf die Anfangsadresse des Identifier-Strings im nächsten Element der Teilliste, nicht auf das Wort mit Typenkennung 2, das die nächste Folgeadresse der Teilliste enthält. Beim letzten Element einer Teilliste steht als Folgeadresse Null. Die Anfangsadressen der Teillisten sind im Vektor IDVEKT zu finden entsprechend den berechneten Kennzahlen.

Jeder Identifier soll in IDLIST nur einmal vorhanden sein; zu diesem Zweck wird für jeden gelieferten Identifier die betreffende Teilliste durchsucht. Gegebenenfalls wird der Identifier dann am Ende der Identifier-Liste (auf das IDIND zeigt) eingetragen. Dabei werden in dieser Version des Leseprogramms höchstens 2 Ganzworte aus dem Lesekeller übernommen, längere Identifier werden also auf 12 Zeichen verkürzt.

Die Adresse, auf der der Identifier eingetragen wurde, wird im vorhergehenden Element der betreffenden Teilliste (oder in IDVEKT, falls es sich um das erste Element einer Teilliste handelt) vermerkt, und ein neues Schlußwort wird eingetragen.

Zur Steuerung dieses Such- und Eintrag-Mechanismus gibt es zwei Zeiger, die voneinander abhängig sind:

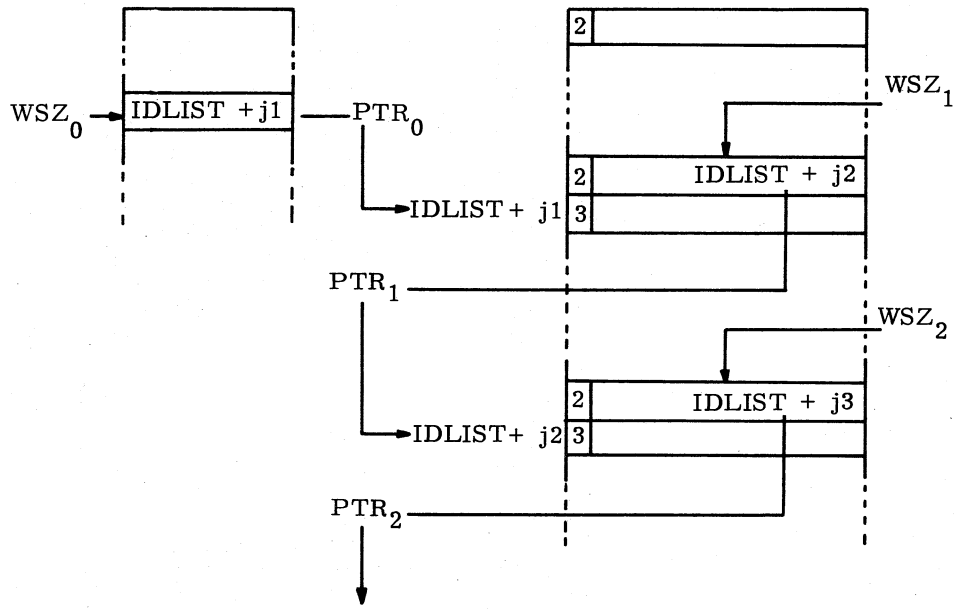
WSZ der auf ein Wort, das die Folgeadresse einer Teilliste enthält, zeigt (unter Umständen auf ein Halbwort in IDVEKT, in dem die Anfangsadresse der Teilliste steht)

PTR der auf den Anfang des Identifiers in diesem folgenden Teillistenelement zeigt (also die Folgeadresse "enthält").

PTR ist durch folgende Äquivalenzvereinbarung von WSZ abhängig:

```
SHORT PTR = SVAL WSZ;
```

Die Elemente einer Teilliste sind folgendermaßen verkettet:



Bei Durchsuchen der Teilliste nehmen die beiden Zeiger folgende Werte an:

WSZ_0 : enthält REF IDVEKT + Kennzahl

PTR_0 : "enthält" REF IDLIST + j1

WSZ_1 : enthält REF IDLIST + j1 -1 (= PTR_0 -1)

PTR_1 : "enthält" REF IDLIST + j2

usw.

```
001360
001370  RECORD 64 FULL IDVEKT;
001380  FULL IDVLA IS '1H3F';
001390  RECORD 5000 FULL IDLIST;
001400  IDLIST:='2H0';
001410  INDEX IDIND:=REF IDLIST;
001420
001430  PROC IDEINTRAG IS
001440  BEGIN FULL HASH IS '1H.3E7F';
001450        RAQ:=HW OF LK*HASH;
001460        INDEX WSZ:=RA ET IDVLA+REF IDVEKT;
001470        SHORT PTR=SVAL WSZ;
001480        WHILE PTR NE 0 DO
001490        BEGIN RAQ:=LVAL PTR;
001500              RH:=LK;
001510              IF RA=RH THEN RH:=LK[2];
001520                        IF TK 3 RQ AND RQ=RH OR
001530                           NOT TK 3 RQ AND NOT TK 3 RH
001540                        THEN RA:=PTR;
001550                             RETURN;
001560                        FI;
001570               FI;
001580               WSZ:=PTR-1;
001590        END;
001600        PTR:=IDIND+:=2;
001610        RAQ:=LVAL IDIND:=DW OF LK;
001620        IF TK 3 RQ THEN IDIND+:=4;
001630                   ELSE IDIND+:=22
001640        FI;
001650        FVAL IDIND :='2H0';
001660        RA:=PTR;
001670  END IDEINTRAG;
001680
```

4.4.
Sonderzeichen-Behandlung

Die Sonderzeichen-Behandlung wird von der Prozedur SZEIN erledigt. Als Ergebnis der Prozedur wird das Sonderzeichen mit Typenkennung 3 in RA geliefert.

```
001690
001700  PROC SZEIN IS RA:=TK3 ZEI;
001710
```

4.5.
Übergabe der Textelemente an das Hauptprogramm

Das eigentliche Hauptprogramm, das die einzelnen Textelemente zur Weiterverarbeitung entgegennimmt, ist hier als Prozedur geschrieben (PROC HP). Das hat den Vorteil, daß sich das Leseprogramm selbst steuern kann, ein expliziter Wiederholungs-Mechanismus bei Sonderzeichen - da Sonderzeichen zunächst als Trennzeichen und dann noch als Textelemente verarbeitet werden müssen - entfällt, und daß auf das Hauptprogramm nur dann zugegriffen werden muß, wenn es ein fertiges Textelement entgegennehmen soll.

Um welche Art von Textelement es sich dabei handelt, ist aus der von den anderen Matrixroutinen in RA gelieferten Information leicht zu entscheiden. Falls das Sonderzeichen EM (hexadekadisch '21') geliefert wird, also die Eingabedaten erschöpft sind, wird der Lesevorgang beendet.

Die in diesem Programm verwendete Prozedur HP dient nur Testzwecken, die Beschreibung dazu siehe Test-Teil des Programms.

5.
Lesen eines Textes

Um einen Text vollständig einzulesen, ist nur folgende Anweisung notwendig, alles andere steuert das Leseprogramm selbst:

```
001720
001730  DO CALL ZEILE[ZIND];
001740
```

6.
Test-Teil des Programms

In der Prozedur HP werden Druckprogramme aufgerufen, die die Textelemente in der Reihenfolge drucken, in der sie eingelesen und vom Leseprogramm angeliefert werden. Nach Identifiern und Zahlen wird ein Zwischenraum, nach Sonderzeichen ein Zeilenvorschub ausgegeben.

Sonderzeichen werden mit der Prozedur SZDRUCK ausgedruckt, ganze Zahlen mit der Prozedur ZAHLDRUCK, Identifier mit der Prozedur IDDRUCK.

Außerdem werden, wenn die Eingabedaten erschöpft sind, von der Prozedur ENDEEINGABE der Text "ENDE EINGABE" ausgegeben, und dann werden alle Listen ausgedruckt, die das Leseprogramm aufgebaut hat.

a) ZL: Hier werden die Zahlen in der Reihenfolge ausgegeben, wie sie in der Zahlliste stehen; hintereinander, ohne Rücksicht auf Zeilenüberlauf im Druckprotokoll.

b) IDVEKT: Jeweils zwei Anfangsadressen von Teillisten aus IDLIST, die in einem Ganzwort stehen, erscheinen nebeneinander, jedes Ganzwort auf einer neuen Zeile.

c) IDLIST: Jeweils die Adresse und der Inhalt eines Ganzwortes aus IDLIST werden nebeneinander gedruckt, dann erfolgt Zeilenvorschub.

d) Die einzelnen Teillisten von IDLIST:
Am Anfang einer Zeile erscheint die Nummer des Halbwortes von IDVEKT (die Adresse relativ zur Anfangsadresse von IDVEKT), das die Anfangsadresse der betreffenden Teilliste enthält. Dann folgen die Identifier, die zu dieser Teilliste gehören, ohne Rücksicht auf Zeilenüberlauf.

Dann wird der Programmlauf mit einem Aufruf von IOC12 beendet.

```
001750  PROC HP IS
001760  BEGIN FULL HRA:=RA;
001770        IF RA='3H21' THEN CALL ENDEEINGABE;
001780        ELSF TK 3 RA THEN CALL SZDRUCK;
001790        ELSE RA:=FVAL RA;
001800             IF TK 1 RA THEN RA:=HRA;CALL ZAHLDRUCK;
001810                        ELSE RA :=HRA;CALL IDDRUCK;
001820                        FI;
001830             FI;
001840  END HP;
001850
001860  PROC ZAHLDRUCK IS
001870  BEGIN RA:=FVAL RA;
001880        CALL IOC10;
001890        RA:='3HAF';
001900        CALL IOC13;
001910  END ZAHLDRUCK;
001920
001930  PROC IDDRUCK IS
001940  BEGIN SHORT IX:=RA;
001950        RA:=FVAL IX;
001960        IF TK 3 RA THEN CALL IOC14;
001970        FI;
001980        RA:=FVAL(IX+2);
001990        CALL IOC14;
002000        RA:='3HAF';
002010        CALL IOC13;
002020  END IDDRUCK;
002030
002040  PROC SZDRUCK IS
002050  BEGIN CALL IOC13;
002060        CALL IOC8;
002070  END SZDRUCK;
002080
002090  PROC ENDEEINGABE IS
002100  BEGIN CALL IOC8;
002110S       RA:=REF ''ENDEEINGABE*037'';
002120        CALL IOC6;
002130        CALL IOC8;
002140S       RA :=REF''ZAHL - LISTE*037'';
002150        CALL IOC6;
002160        INDEX I;
002170        FOR I FROM REF ZL BY 2 TO ZLIND -2 DO
002180        BEGIN RA:=FVAL I;
002190              CALL IOC10;
002200              RA:='3HAF';
002210              CALL IOC13;
002220        END I;
```

```
002230          CALL IOC8;
002240S         RA:=REF'' IDVEKT*037'';
002250          CALL IOC6;
002260          FOR I FROM REF IDVEKT BY 2 TO REF IDVEKT+62 DO
002270          BEGIN RA:=SVAL I;
002280                CALL IOC10;
002290                RA:=SVAL (I+1);
002300                CALL IOC10;
002310                CALL IOC8;
002320          END I;
0023305         RA:=REF '' IDLIST*037'';
002340          CALL IOC6;
002350          FOR I FROM REF IDLIST BY 2 TO IDIND -2 DO
002360          BEGIN RA:=I;
002370                CALL IOC10;
002380                RA:='3HAFAF';
002390                CALL IOC14;
002400                RA:=FVAL I;
002410                IF TK 2 RA THEN CALL IOC10;
002420                           ELSE CALL IOC14;
002430                FI;
002440                CALL IOC8;
002450          END I;
002460          CALL IOC8;
002470S         RA:=REF'' TEILLISTEN VON IDLIST*037'';
002480          CALL IOC9;
002490          SHORT K;
002500          FOR I FROM REF IDVEKT BY 1 TO REF IDVEKT+63 DO
002510          BEGIN K:=SVAL I;
002520                IF K NE 0 THEN CALL IOC8;
002530                               RA:=I-REF IDVEKT;
002540                               CALL IOC10;
002550                               RA:='3HAFAF';
002560                               CALL IOC14;
002570                FI;
002580                WHILE K NE 0 DO
002590                BEGIN RA:=FVAL K;
002600                      CALL IOC14;
002610                      RA:=FVAL(K+2);
002620                      IF TK 3 RA THEN CALL @OC14;
002630                      FI;
002640                      RA:= '3HAFAF';
002650                      CALL IOC14;
002660                      K:=SVAL(K-1);
002670                END ;
002680          END I;
002690          CALL IOC12;
002700    END ENDEEINGABE;
002710
002720    FINIS
```

7.
Auszüge aus dem Protokoll eines Testlaufs des Programms

Im folgenden Testlauf des Programms wurde als Eingabetext die PS440-Quelle des Programms selbst verwendet.

Zunächst Auszüge aus dem Protokoll der eingelesenen Textelemente:

```
LESEPROGRAMM


SEGMENT LESEPROGRAMM ;
SPEC IOC PROCSFB IOC0 ;
SPEC IOC PROC IOC3 ,
IOC6 ,
IOC8 ,
IOC9 ;
SPEC IOC PROC IOC10 ,
IOC12 ,
IOC13 ,
IOC14 ;
RECORD      8 INDEX DUMMY ;
*
/
START ANF /
*
;
CO STARTADRESSE DES PROGRAMMS :
      .
      .
      .

PROC L3 ISBEGIN CALL AUFID ;
CO AUFREIHEN IDENTIFIER ;
CALL LESEN ;
CO LESEN NAECHSTES ZEICHEN ;
END L3 ;
PROC L4 ISBEGIN ZUSTAND :
=
LEER ;
CO EINSTELLEN ZUSTAND LEER ;
CALL IDEND ;
CO ENDEBEHANDLU IDENTIFIER ;
CALL HP ;
CO ID .
ABLIEFERN AN HP ;
END L4 ;
PROC L5 ISBEGIN ZUSTAND :
```

Bemerkung: IS und BEGIN wurden zu ISBEGIN zusammengezogen, da der dazwischenliegende Zeilenvorschub nicht als Trennzeichen gilt.

```
   .
   .
   .
CALL IOC8 ;
END I ;
CALL IOC8 ;
RA :
=
REF .
.
TEILLISTEN VON IDLIST *
    37 .
.
;
CALL IOC9 ;
SHORT K ;
FOR I FROM REF IDVEKT BY      1 TO REF IDVEKT +
    63 DO BEGIN K :
=
SVAL I ;
IF K NE      0 THEN CALL IOC8 ;
RA :
=
I -
REF IDVEKT ;
CALL IOC10 ;
RA :
=
.
     3 HAFAF .
;
CALL IOC14 ;
FI ;
WHILE K NE      0 DO BEGIN RA :
=
FVAL K ;
CALL IOC14 ;
RA :
=
FVAL (
K +
     2 )
;
IF TK      3 RA THEN CALL IOC14 ;
FI ;
RA :
=
.
     3 HAFAF .
;
CALL IOC14 ;
K :
=
SVAL (
K -
     1 )
;
END ;
END I ;
CALL IOC12 ;
END EINGABE ;
FINIS

ENDEEINGABE
```

Eingabedaten erschöpft

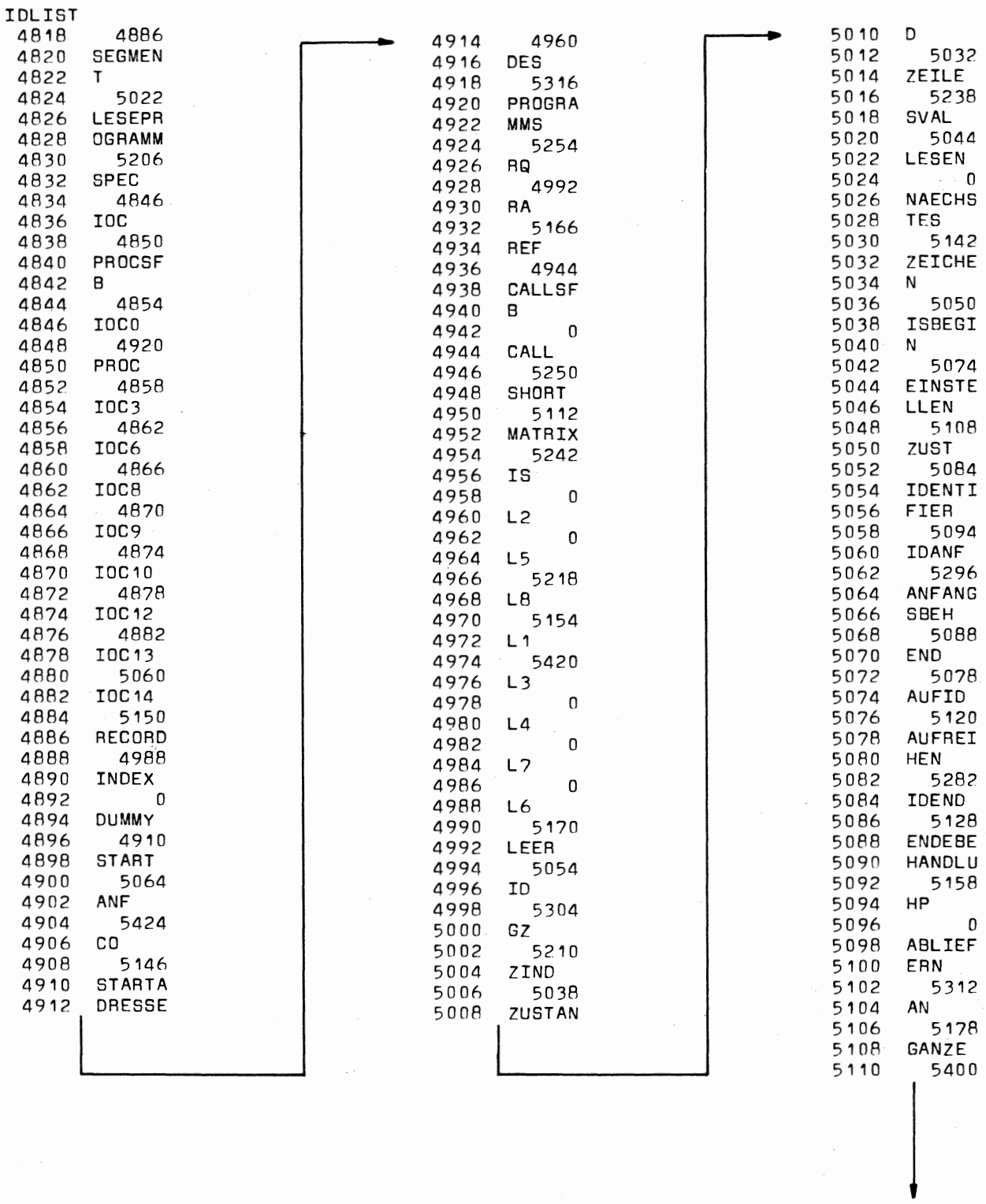
Protokoll der Zahlliste ZL:

```
ZAHL - LISTE
       8        0       37       15        5       10        4        1        3        2
     500       20       48       64     5000       62       63
```

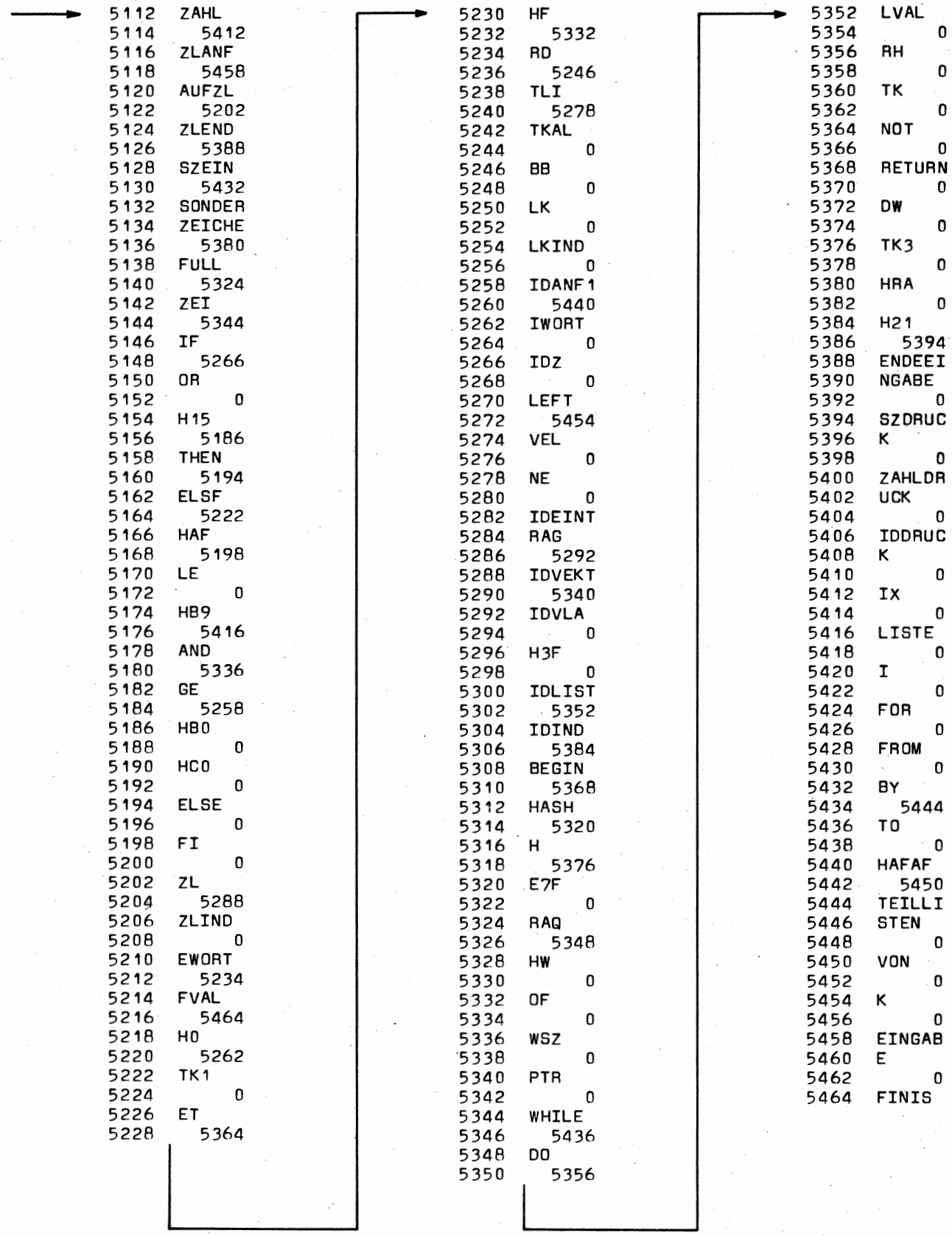
Protokoll von IDVEKT, in dem die Anfangsadressen der Teillisten von IDLIST vermerkt sind, jeweils zwei Adressen, die in einem Ganzwort stehen, erscheinen nebeneinander:

```
IDVEKT
 4894  5190
 4968  5230
    0  5014
 4826  4836
 4984  5026
 5406  4996
 5428  5182
 4890  5000
 5174     0
 5300  5116
 4964  4948
    0  5124
 5274  4926
 4980  4832
 4952  5328
 5070  4820
 4976  4906
 4934  5308
 4840     0
 4916  5214
    0  5132
    0     0
 4972  5372
 5008  5104
 4902  4956
 5004  5018
    0  5360
    0  5138
 5162  4930
 5270  5226
 4938     0
 5098  4898
```

Protokoll der Identifierliste IDLIST (die Adresse und der Inhalt eines Ganzwortes erscheinen jeweils nebeneinander):



```
IDLIST
 4818    4886
 4820  SEGMEN
 4822  T
 4824    5022
 4826  LESEPR
 4828  OGRAMM
 4830    5206
 4832  SPEC
 4834    4846
 4836  IOC
 4838    4850
 4840  PROCSF
 4842  B
 4844    4854
 4846  IOC0
 4848    4920
 4850  PROC
 4852    4858
 4854  IOC3
 4856    4862
 4858  IOC6
 4860    4866
 4862  IOC8
 4864    4870
 4866  IOC9
 4868    4874
 4870  IOC10
 4872    4878
 4874  IOC12
 4876    4882
 4878  IOC13
 4880    5060
 4882  IOC14
 4884    5150
 4886  RECORD
 4888    4988
 4890  INDEX
 4892       0
 4894  DUMMY
 4896    4910
 4898  START
 4900    5064
 4902  ANF
 4904    5424
 4906  CO
 4908    5146
 4910  STARTA
 4912  DRESSE
 4914    4960
 4916  DES
 4918    5316
 4920  PROGRA
 4922  MMS
 4924    5254
 4926  RQ
 4928    4992
 4930  RA
 4932    5166
 4934  REF
 4936    4944
 4938  CALLSF
 4940  B
 4942       0
 4944  CALL
 4946    5250
 4948  SHORT
 4950    5112
 4952  MATRIX
 4954    5242
 4956  IS
 4958       0
 4960  L2
 4962       0
 4964  L5
 4966    5218
 4968  L8
 4970    5154
 4972  L1
 4974    5420
 4976  L3
 4978       0
 4980  L4
 4982       0
 4984  L7
 4986       0
 4988  L6
 4990    5170
 4992  LEER
 4994    5054
 4996  ID
 4998    5304
 5000  GZ
 5002    5210
 5004  ZIND
 5006    5038
 5008  ZUSTAN
 5010  D
 5012    5032
 5014  ZEILE
 5016    5238
 5018  SVAL
 5020    5044
 5022  LESEN
 5024       0
 5026  NAECHS
 5028  TES
 5030    5142
 5032  ZEICHE
 5034  N
 5036    5050
 5038  ISBEGI
 5040  N
 5042    5074
 5044  EINSTE
 5046  LLEN
 5048    5108
 5050  ZUST
 5052    5084
 5054  IDENTI
 5056  FIER
 5058    5094
 5060  IDANF
 5062    5296
 5064  ANFANG
 5066  SBEH
 5068    5088
 5070  END
 5072    5078
 5074  AUFID
 5076    5120
 5078  AUFREI
 5080  HEN
 5082    5282
 5084  IDEND
 5086    5128
 5088  ENDEBE
 5090  HANDLU
 5092    5158
 5094  HP
 5096       0
 5098  ABLIEF
 5100  ERN
 5102    5312
 5104  AN
 5106    5178
 5108  GANZE
 5110    5400
```


5112 ZAHL
5114 5412
5116 ZLANF
5118 5458
5120 AUFZL
5122 5202
5124 ZLEND
5126 5388
5128 SZEIN
5130 5432
5132 SONDER
5134 ZEICHE
5136 5380
5138 FULL
5140 5324
5142 ZEI
5144 5344
5146 IF
5148 5266
5150 OR
5152 0
5154 H15
5156 5186
5158 THEN
5160 5194
5162 ELSF
5164 5222
5166 HAF
5168 5198
5170 LE
5172 0
5174 HB9
5176 5416
5178 AND
5180 5336
5182 GE
5184 5258
5186 HB0
5188 0
5190 HC0
5192 0
5194 ELSE
5196 0
5198 FI
5200 0
5202 ZL
5204 5288
5206 ZLIND
5208 0
5210 EWORT
5212 5234
5214 FVAL
5216 5464
5218 H0
5220 5262
5222 TK1
5224 0
5226 ET
5228 5364
5230 HF
5232 5332
5234 RD
5236 5246
5238 TLI
5240 5278
5242 TKAL
5244 0
5246 BB
5248 0
5250 LK
5252 0
5254 LKIND
5256 0
5258 IDANF1
5260 5440
5262 IWORT
5264 0
5266 IDZ
5268 0
5270 LEFT
5272 5454
5274 VEL
5276 0
5278 NE
5280 0
5282 IDEINT
5284 RAG
5286 5292
5288 IDVEKT
5290 5340
5292 IDVLA
5294 0
5296 H3F
5298 0
5300 IDLIST
5302 5352
5304 IDIND
5306 5384
5308 BEGIN
5310 5368
5312 HASH
5314 5320
5316 H
5318 5376
5320 E7F
5322 0
5324 RAQ
5326 5348
5328 HW
5330 0
5332 OF
5334 0
5336 WSZ
5338 0
5340 PTR
5342 0
5344 WHILE
5346 5436
5348 DO
5350 5356
5352 LVAL
5354 0
5356 RH
5358 0
5360 TK
5362 0
5364 NOT
5366 0
5368 RETURN
5370 0
5372 DW
5374 0
5376 TK3
5378 0
5380 HRA
5382 0
5384 H21
5386 5394
5388 ENDEEI
5390 NGABE
5392 0
5394 SZDRUC
5396 K
5398 0
5400 ZAHLDR
5402 UCK
5404 0
5406 IDDRUC
5408 K
5410 0
5412 IX
5414 0
5416 LISTE
5418 0
5420 I
5422 0
5424 FOR
5426 0
5428 FROM
5430 0
5432 BY
5434 5444
5436 TO
5438 0
5440 HAFAF
5442 5450
5444 TEILLI
5446 STEN
5448 0
5450 VON
5452 0
5454 K
5456 0
5458 EINGAB
5460 E
5462 0
5464 FINIS

Protokoll der Teillisten der Identifierliste (die Zahl am Anfang der Zeile gibt jeweils an, zu welchem Halbwort aus IDVEKT die Teilliste gehört, dann erscheinen nebeneinander die in der Teilliste enthaltenen Identifier):

```
 TEILLISTEN VON IDLIST
     0  DUMMY
     1  HC0
     2  L8  H0  FINIS
     3  HF  NOT
     5  ZEILE  ZEICHEN  ZEI  RAQ
     6  LESEPROGRAMM  LESEN  EINSTELLEN  AUFID  AUFREIHEN  AUFZL  EIN
GABE
     7  IOC  IOC0  IOC3  IOC6  IOC8  IOC9  IOC10  IOC12  IOC13  IOC14
  IDANF  HP  THEN  HB0  IDANF1
     8  L7
     9  NAECHSTES
    10  IDDRUCK
    11  ID  IDENTIFIER  IDEND  IDEINTRAG
    12  FROM
    13  GE  WSZ
    14  INDEX  L6
    15  GZ  IDIND  LVAL  RH
    16  HB9
    18  IDLIST
    19  ZLANF  IX
    20  L5
    21  SHORT  LK
    23  ZLEND  ZL
    24  VEL  K
    25  RQ  LKIND
    26  L4
    27  SPEC  ZLIND  IDVEKT  IDVLA  PTR
    28  MATRIX  ZAHL  ZAHLDRUCK
    29  HW  DO  TO  TEILLISTEN  VON
    30  END  ENDEBEHANDLU  SZEIN  ENDEEINGABE  SZDRUCK
    31  SEGMENT  RECORD  OR  IDZ
    32  L3  I
    33  CO  FOR
    34  REF  HAF  TK1  IWORT  HAFAF
    35  BEGIN  H21
    36  PROCSFB  PROC  PROGRAMMS  H  E7F  TK3
    38  DES  L2
    39  FVAL  RD  OF
    41  SONDERZEICHE  BY
    44  L1  H15
    45  DW
    46  ZUSTAND  ISBEGIN  ZUST  GANZE  AND  LISTE
    47  AN  HASH  RETURN
    48  ANF  ANFANGSBEH  H3F
    49  IS  TKAL  NE
    50  ZIND  EWORT
    51  SVAL  TLI  BB
    53  TK
    55  FULL  HRA
    56  ELSF  ELSE
    57  RA  LEER  LE  FI
    58  LEFT
    59  ET
    60  CALLSFB  CALL
    62  ABLIEFERN
    63  START  STARTADRESSE  IF  WHILE
```

L I T E R A T U R

[1] G. Goos, K. Lagally, G. Sapper
PS440, Eine niedere Programmiersprache
(TU-Bericht 7002)

[2] AEG-TELEFUNKEN
TR440, Große Befehlsliste

[3] AEG-TELEFUNKEN
Unterlagensammlung, TAS-Handbuch,
Programmierung von Operatoren

[4] AEG-TELEFUNKEN
Statusdokumentation TR440, Teil IV

ANHANG 3

ZENTRALCODE - TABELLE

ÜBERSICHTSTABELLE **ZC1**

2^7 2^0 — 1. Sedezimale | 2. Sedezimale

0000	000L	00L0	00LL	0L00	0L0L	0LL0	0LLL	L000	L00L	L0L0	L0LL	LL00	LL0L	LLL0	LLLL
0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0 NUL	16 NL6	32 SUB	48 BEL	64	80	96 ″	112 %	128 ∧	144 +	160 (	176 0	192 A	208 Q	224 a	240 q
1 NU1	17 NL5	33 EM	49 DC1	65	81	97 ′	113 §	129 ∨	145 −	161)	177 1	193 B	209 R	225 b	241 r
2 NU2	18 NL4	34 CAN	50 DC2	66	82	98 ⁄	114 #	130 ¬	146	162 [	178 2	194 C	210 S	226 c	242 s
3 NU3	19 NL3	35	51 DC3	67	83	99 ∖	115 $	131	147 /	163]	179 3	195 D	211 T	227 d	243 t
4 NU4	20 NL2	36	52 DC4	68	84	100	116 ¢	132	148	164	180 4	196 E	212 U	228 e	244 u
5 NU5	21 NL	37 TE	53 FL	69	85	101 ^	117	133	149	165	181 5	197 F	213 V	229 f	245 v
6 SOH	22 CR	38	54	70	86	102 °	118 @	134 ↑	150	166 ‹	182 6	198 G	214 W	230 g	246 w
7 STX	23 NF	39	55	71	87	103 ~	119 &	135	151 =	167 ›	183 7	199 H	215 X	231 h	247 x
8 ETX	24 VT	40	56	72	88	104	120 *	136	152 ‡	168 } (MZ)	184 8	200 I	216 Y	232 i	248 y
9 EOT	25 VT3	41	57	73	89	105	121	137	153	169 .	185 9	201 J	217 Z	233 j	249 z
10 ENQ	26 VT4	42	58	74	90	106	122	138	154	170 ,	186	202 K	218 Ä	234 k	250 ä
11 ACK	27 VT5	43 HT	59	75	91	107 \	123	139 |	155 <	171 :	187	203 L	219 Ö	235 l	251 ö
12 DLE	28 VT6	44 BS	60 IS4	76	92	108 ‚	124 ¤	140	156 >	172 ;	188	204 M	220 Ü	236 m	252 ü
13 NAK	29 VT7	45 ESC	61 IS3	77	93	109 ’	125	141 10	157 ⩽	173 !	189	205 N	221	237 n	253 ß
14 SYN	30 VT8	46 SO	62 IS2	78	94	110 _	126	142	158 ⩾	174 ?	190	206 O	222	238 o	254
15 ETB	31 NL7	47 SI	63 IS1	79	95	111 –	127 π	143	159	175 SP	191 { (PZ)	207 P	223	239 p	255 DEL

0	0000
1	000L
2	00L0
3	00LL
4	0L00
5	0L0L
6	0LL0
7	0LLL
8	L000
9	L00L
A	L0L0
B	L0LL
C	LL00
D	LL0L
E	LLL0
F	LLLL

Stand: März 1972

SACHWORTREGISTER