PLI1 V4.1A

PL/I-Compiler

Ausgabe Mai 1991

Kritik… Anregungen… Korrekturen…

Die Redaktion ist interessiert an Ihren Kommentaren zu diesem Handbuch. Ihre Rückmeldungen helfen uns, die Dokumentation zu optimieren und auf Ihre Wünsche und Bedürfnisse abzustimmen.

Sie können uns Ihre Kommentare per E-Mail an manuals@ts.fujitsu.com senden.

Zertifizierte Dokumentation nach DIN EN ISO 9001:2000

Um eine gleichbleibend hohe Qualität und Anwenderfreundlichkeit zu gewährleisten, wurde diese Dokumentation nach den Vorgaben eines Qualitätsmanagementsystems erstellt, welches die Forderungen der DIN EN ISO 9001:2000 erfüllt.

cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Copyright und Handelsmarken

Copyright © Fujitsu Technology Solutions GmbH 2009.

Alle Rechte vorbehalten.
Liefermöglichkeiten und technische Änderungen vorbehalten.

Alle verwendeten Hard- und Softwarenamen sind Handelsnamen und/oder Warenzeichen der jeweiligen Hersteller.

Inhalt

1 Einleitung

1.1 Zielgruppe und Konzept des Handbuchs

Das vorliegende Benutzerhandbuch wendet sich an Programmierer, die mit dem Softwareprodukt PLI1 arbeiten. Während die Sprachbeschreibung zu diesem PL/I-Compiler weitgehendst frei von Randbedingungen des Betriebssystems ist, werden im Benutzerhandbuch die Einbettung und der Betrieb des Compilers im BS2000 beschrieben. Allgemeine Kenntnisse über dieses Betriebssystem werden vorausgesetzt. Erläuterungen zu Systemeigenschaften werden nur dort gegeben, wo deren Kenntnis für den PL/I-Anwender nützlich ist.

Voraussetzung ist die Kenntnis der Programmiersprache PL/I, wie sie im Manual

PLI1 (BS2000)
PL/I-Compiler
Beschreibung

beschrieben ist.

Das Benutzerhandbuch ist für Anwendungen sowohl im Dialog als auch im Stapelbetrieb geeignet. Es deckt den Betrieb des Compilers PL/I (D-Level) nicht ab.

Beispiele geben im Kapitel 2 eine erste Übersicht. Der Ablauf einiger einfacher Anwendungsfälle ist anhand der Rechnerprotokolle ausführlich kommentiert.

Kapitel 3 erklärt ausführlich alle Steuermöglichkeiten für den Compiler PLI1.

Dazu gehören:

- Auswahl verschiedener Protokollformen
- Bereitstellung des Quellprogramms
- Ablage und Eigenschaften der erzeugten Bindemoduln
- Hinweise auf die Verwaltung von Bindemoduln in Bibliotheken.

Kapitel 4 beschreibt den Bindevorgang für Standardanwendungen. Will man Anwenderprogramme segmentieren (OVERLAY) oder sonstige spezielle Manipulationen während des Bindens vornehmen, so findet man die dafür notwendigen Angaben im Manual der

Dienstprogramme [3] bzw. bei Einsatz einer BS2000 Version ab V 8.0 im Manual Binder und Lader [12].

Einige Wirkungen der ablauffähigen Anwenderprogramme können von außen gesteuert werden. Das betrifft vor allem

- die Aktivierung von Testhilfen
- die Übergabe von Parametern
- die Kopplung von Systemdateien
- die Auswahl von Standardlisten

Diese Möglichkeiten werden im Kapitel 5 vorgestellt.

Die Verwendung von Dateien wird im Kapitel 6 ausführlich dargestellt. Nach einem Überblick über die vom BS2000 gebotene Dateischnittstelle werden unter verschiedenen Gesichtspunkten die Randbedingungen beschrieben, die im Zusammenhang mit PL/I-Programmen von Bedeutung sind.

Im Kapitel 7 ist die Schnittstelle von PL/I-Prozeduren in maschinennaher Form dargestellt. Daraus lassen sich dann auch die Schnittstellen für ASSEMBLER-Programme ableiten, die sich wie PL/I-Prozeduren verhalten. Ferner sind weitere Schnittstellen beschrieben, die über spezielle Steuerungen eingestellt werden können. Des weiteren sind auch Schnittstellen zu Prozeduren angegeben die in anderen Programmiersprachen geschrieben sind (siehe auch Kapitel 13).

Im Kapitel 8 ist angegeben, wie PL/I-Programme in ihrer Leistung verbessert (optimiert) werden können.

Im Kapitel 9 sind Testhilfen beschrieben, die im Rahmen des PLI1-Systems gesetzt werden können.

Im Kapitel 10 wird die interne Darstellung von Daten und Listen angegeben, soweit sie für den Benutzer beim Testen von Interesse sein kann.

Im Kapitel 11 sind einige Dienstleistungen beschrieben, die der Benutzer in PL/I-Programmen durch einen Unterprogramm-Aufruf (CALL) bzw. durch einen Funktions-Aufruf in Anspruch nehmen kann.

Im Kapitel 12 ist erläutert, was zu tun ist, wenn ein Programm von mehreren Benutzern gleichzeitig verwendet werden soll (sharable).

Im Kapitel 13 sind einige Makros beschrieben, die das Schreiben von ASSEMBLER-Prozeduren erleichtert, die sich genau so verhalten sollen, wie PL/I-Prozeduren.

Wichtige Aussagen wurden, wenn möglich, zu Tabellen zusammengefaßt. Sie sind zusätzlich im Tabellenheft zu PLI1 veröffentlicht. In einigen Fällen wurden formale Syntaxnotierungen verwendet, wobei der Formalismus den in der Sprachbeschreibung [1] definierten Regeln entspricht.

1.2 Änderungen gegenüber der Vorgängerversion

Im folgenden sind die wichtigsten Änderungen gegenüber dem Vorgängermanual beschrieben:

- Der PLI1-Compiler kann alternativ mit dem SDF-Kommando START-PLI1-COMPILER aufgerufen werden (Abschnitte 2.1.2 und 3.2).
- PLI1-Programme können alternativ mit dem SDF-Kommando START-PLI1-PROGRAM aufgerufen werden (Abschnitte 2.1.2 und 5.2).
- Beschreibung der Operanden des SDF-Kommandos START-PLI1-COMPILER und Tabelle über die Abbildung der SDF-Operanden auf die COMOPT-Operanden (Abschnitt 3.12).
- Beschreibung der Operanden des SDF-Kommandos START-PLI1-PROGRAM und Tabelle über die Abbildung der SDF-Operanden auf die RUNOPT-Operanden (Abschnitt 5.7).
- Die PLI1-Sprachverknüpfung erlaubt zur Laufzeit auch die Programmkommunikation nach den Konventionen der Inter Language Communication Services (ILCS). Diese Konventionen ermöglichen die beliebige Verknüpfung von in unterschiedlichen Programmiersprachen geschriebenen Programmen (Abschnitt 7.4).

2 Übersicht über die Benutzung von PLI1

Dieses Kapitel zeigt an einfachen Beispielen, wie PL/I-Übersetzungen und PL/I-Programmläufe im BS2000 durchgeführt werden können. Die Beispiele sind eine Anleitung für häufige Anwendungsfälle.
In der Regel gelten die Beispiele für jeden Auftrag (Prozeß); d.h. sowohl für Stapelverarbeitung als auch für Dialogbetrieb. Wenn es Unterschiede zwischen Stapelverarbeitung und Dialog gibt, sind sie beschrieben.

2.1 Allgemeiner Programmablauf

Zum Verständnis der Kommandofolgen in den Beispielen sei hier zunächst kurz der allgemeine Ablauf für Übersetzung und Programmausführung beschrieben.

Der Compiler PLI1 wandelt PL/I-Quellprogramme in Bindemoduln um. Dabei wird eine syntaktische und semantische Prüfung durchgeführt, und es werden diverse Protokolle erstellt. Die Bindemoduln, auch Objektmoduln genannt, werden in der EAM-Datei oder in einer LMS-Bibliothek abgelegt und können mit dem Binder (TSOSLNK) zu Lademoduln gebunden und in einer Datei abgelegt werden. Anschließend kann man das gebundene Programm laden und ausführen. Die wichtigste Aufgabe des Binders ist es, zu den vom Anwender durch Übersetzung erzeugten Bindemoduln die Laufzeitbibliothek hinzuzufügen. Die Laufzeitbibliothek enthält alle vorgefertigten Bausteine, welche die Ein- und Ausgabevorgänge abwickeln, die Bedingungen behandeln, sowie eingebaute Funktionen und einen Programmrahmen, der den korrekten Ablauf des Programms steuert.

Der Benutzer kann den PL/I-Compiler und das Objektprogramm beeinflussen. Mit Steueranweisungen an den PL/I-Übersetzer kann man festlegen, welche Protokolle, Prüfungen, Optimierungen usw. durchgeführt werden sollen und welche Eigenschaften der erzeugte Objektmodul bzw. das spätere Programm erhalten. Alle Steueranweisungen haben Voreinstellungen, die dann wirken, wenn der Benutzer keine Angabe macht. Diese Voreinstellungen sind so ausgelegt, daß sie den allgemeinen Anwendungsfall abdecken. Beispiele für einfachen Programmablauf sind in den folgenden Kapiteln aufgeführt. Der Benutzer hat dabei die Möglichkeit, alternativ die BS2000-Kommandos im sogenannten ISP-Format [2] oder die SDF-Kommandos (System Dialog Facility) [19] zu verwenden.

2.1.1 Allgemeiner Programmablauf mit ISP-Kommandos

Die Kommandos, die benötigt werden um ein Quellprogramm zu übersetzen, zu binden und auszuführen, kann man z.B. durch folgende einfache Kommandoprozedur (mit BS2000-Kommandos im sogenannten ISP-Format) beschreiben:

```
/ PROCEDURE C,(&NAME, &LIST=SOURCE, &COMOPT='COMLIB=NO'), SUBDTA=&       1)
/         ERASE *
/         SYSFILE SYSDTA=(SYSCMD)
/ REMARK ..................................................... TRANSLATE
/         EXEC $PLI1                                                     2)
               *COMOPT FORMAT=PRINTER(64,72),
               *COMOPT SOURCE=&NAME,
               *COMOPT LIST=&LIST,
               *COMOPT &COMOPT,
               *END
/ REMARK ..................................................... LINK       3)
/         EXEC $TSOSLNK
               PROGRAM &NAME, FILENAM=PROG.&NAME, MAP=NO
               INCLUDE *
               END
/ REMARK ..................................................... EXECUTE    4)
/         EXEC PROG.&NAME
/ REMARK ..................................................... END
/         STEP
/         SYSFILE SYSDTA=(PRIMARY)                                       5)
/         ENDP
```

Die einzelnen Kommandos bewirken folgendes:

1) PROCEDURE

   Die Prozedur hat drei Parameter:

   – &NAME für den Namen der zu übersetzenden Quelle

   – &LIST zur Steuerung der Listenausgaben des Compilers.
     Voreinstellung: Quellauflistung (SOURCE)

   – &COMOPT zur Angabe beliebiger anderer Steueranweisungen für den Compiler.
     COMLIB = NO dient als Platzhalter und entspricht einer üblichen Voreinstellung.

   ERASE

   Löschen der EAM-Datei, falls von vorausgegangenen Übersetzungen dort noch Bindemoduln stehen.

   SYSFILE

   Umschalten von SYSDTA auf die Kommandodatei, damit die Steueranweisungen für Compiler und Binder von dort gelesen werden können.

2) Aufruf von PLI1 mit variablen Steueranweisungen.

3) Aufruf des Binders. Protokoll abgeschaltet.

4) Aufruf des Benutzerprogramms.

5) Rücksetzen von SYSDTA

Als Beispiel sei in der Datei EXAMP21 ein Quellprogramm enthalten.

Der Aufruf der Prozedur könnte dann folgendermaßen aussehen:

```
/DO Proc.(EXAMP21,COMOPT='LIST=NOF')
```

Ablaufprotokoll auf SYSOUT

```
/DO PLI1.PROZ,EXAMP21,COMOPT='LIST=NOF'
/ PROCEDURE C,(&NAME, &LIST=SOURCE, &COMOPT='COMLIB=NO'), SUBDTA=&
/         ERASE *
/         SYSFILE SYSDTA=(SYSCMD)
/ REMARK ..................................................... TRANSLATE
/         EXEC $PLI1
%  BLS0500 PROGRAM 'PLI1', VERSION '4.0A' OF '88-10-03' LOADED.

..... THERE WAS NO DIAGNOSTIC MESSAGE
..... OBJECTMODUL 'EXAMP21'  GENERATED AND WRITTEN TO: *EAM
..... OBJECTMODUL 'EXAMP217' GENERATED AND WRITTEN TO: *EAM
END OF SIEMENS PLI1-COMPILER VERSION 4.0A , TIME USED:     1.43 SEC
/ REMARK ..................................................... LINK
/         EXEC $TSOSLNK
%  BLS0500 PROGRAM 'TSOSLNK', VERSION '21.0C40' OF '87-09-28' LOADED.
%  LNK0500 PROG BOUND
%  LNK0503 PROG FILE WRITTEN: PROG.EXAMP21
%  LNK0504 NUMBER PAM PAGES USED:      7
/ REMARK ..................................................... EXECUTE
/         EXEC PROG.EXAMP21


%  BLS0500 PROGRAM 'EXAMP21', VERSION ' ' OF '88-10-06' LOADED.
END OF PROGRAM EXAMP21 , RTS 4.0A-AAA, TIME USED:     0.16 SEC
/ REMARK ..................................................... END
/         STEP
/         SYSFILE SYSDTA=(PRIMARY)
/         ENDP
```

Protokoll der für die Übersetzung wirksamen Steueranweisungen auf SYSLST (*COMOPT LIST = OPTIONS)

```
COMPILER-OPTIONS USED


  STORAGE = (STACK(16,4),AREA(16,16,975))
  LIST    = (NOESD,NOTERMINAL,NOSUMMARY,OPTIONS,SAVLST,NOMAP,
            NEST,IREF,NOXREF,SOURCE,NOINSOURCE,NOAGGREGATE,
            OFFSET,NOASSM,NOOUTTEXT,NOLINECNT)
  FORMAT  = (TERMINAL(0,80),PRINTER(64,72),ENGLISH)
  MESSAGE = NOSYSLST
  SOURCE  = EXAMP21
  MARGINS = (TEXT(2,72),PAD,NOLINID,NOASACNTRL,GAMKEY(0,0),CHAR60,
            NOSAVMAC)
  DIAGNOST= (NOTERMINAL,NOSAVLST,WARNING)
  COMLIB  = NO
  OBJECT  = (ERROR(32767),ABORT(500),OUT)
  OPTIONS = (NOISO,NOMAIN,NOINTERRUPT,NOMACRO,NOXS,NOBITPTR)
  OPTIMIZE= (NOTIME,NOOVERLAP,NOENABLING,NOREORDER)
  DEBUG   = (NOSTMT,NOPROCTRACE,NOLABTRACE,NOCALLTRACE,
            NOGOTOTRACE,NORETURNTRACE,NOBREAKPOINT)
  SYMTEST = MAP
  MODULE  = *
```

Quellprogrammliste auf SYSLST (*COMOPT LIST = SOURCE)

```
1     EXAMP21: PROC OPTIONS(MAIN);
2              DCL SYSPRINT FILE;
3              PUT PAGE LIST('VERY EASY TO HANDLE');
4              END;
```

Protokoll des Binders (TSOSLNK) auf SYSLST

```
PROGRAM EXAMP21, FILENAM=PROG.EXAMP21, MAP=NO
INCLUDE *
END
PROG BOUND
PROGRAM FILE WRITTEN : PROG.EXAMP21
NUMBER PAM PAGES USED:      7
```

Ausgabe des Objekt-Programms auf SYSLST

```
VERY EASY TO HANDLE
```

Eine einfache Prozedur, wie sie in diesem Beispiel gezeigt ist, reicht nicht mehr aus, wenn:

- die Programme anspruchsvoller werden,
- Steuerungen des Objektprogramms erwünscht sind,
- mehrere Dateien bearbeitet werden sollen,
- vorübersetzte Programmteile in einer Bibliothek gespeichert sind,
- Übersetzer und Übersetzer-Systemdateien nicht in der empfohlenen Weise unter $TSOS installiert wurden.

Die Steuerungsmöglichkeiten für den Übersetzer sind ausführlich in Kapitel 3 erläutert. Die Objektsteuerungen werden in Kapitel 5 erklärt. Die Verwendung von Bibliotheken durch den Binder ist in Kapitel 4 beschrieben.

Es ist wichtig zu wissen, daß ein lauffähiges Programm (Lademodul) nur entstehen kann, wenn alle notwendigen Bindemoduln zur Verfügung stehen.

Das sind:

- Bindemoduln, die der Benutzer durch Übersetzung eigener Prozeduren erzeugt hat. Sind diese Übersetzungen nicht im laufenden Prozeß vorangegangen, so müssen die privaten Bindemoduln aus einer der Benutzerbibliotheken verfügbar gemacht werden.
- Bindemoduln, die als Bestandteil des Systems standardmäßig, aber abhängig von den benutzten Sprachelementen angebunden werden (Laufzeitsystem). Für die folgenden Beispiele war das Laufzeitsystem unter $TSOS.TASKLIB verfügbar.

  Liegt das Laufzeitsystem z.B. in der Datei $TSOS.PLI1.MODLIB, so ist bei jedem Bindevorgang folgende Anweisung einzufügen:

  ```
  RESOLVE,$TSOS.PLI1.MODLIB
  ```

Das vom Benutzer mit dem Binder (TSOSLNK) erzeugte und in einer Datei abgelegte Lademodul kann man anschließend beliebig oft ausführen. Durch Steueranweisungen, die dem Programm übergeben werden, läßt es sich beeinflussen.

Falls ein Programm auf Dateien zugreifen will, muß der Benutzer diese vor dem Programmstart bereitstellen. Er muß dabei beachten, daß sich die im PL/I-Programm beschriebenen Dateieigenschaften und Manipulationen mit den möglichen BS2000-Dateieigenschaften vertragen. Die entsprechenden Zusammenhänge werden ausführlich in Kapitel 6 erläutert.

2.1.2 Allgemeiner Programmablauf mit SDF-Kommandos

Neben den Funktionen der BS2000-Kommandosprache im sogenannten ISP-Format steht je ein SDF-Kommando zum Übersetzen eines PL/I-Quellprogramms und zum Starten eines PL/I-Objektprogramms zur Verfügung.

Mit dem SDF-Kommando START-PLI1-COMPILER wird der PLI1-Compiler für eine Übersetzung gestartet, wobei nahezu alle Steuermöglichkeiten, wie sie als Compiler-Optionen angebbar sind, zur Verfügung stehen. Mit dem SDF-Kommando START-PLI1-PROGRAM wird ein vom PLI1-Compiler erzeugtes PL/I-Objektprogramm gestartet, wobei ebenfalls nahezu alle Steuermöglichkeiten, wie sie als Objektprogramm-Optionen angebbar sind, zur Verfügung stehen.

Im folgenden ist eine SDF-Kommandofolge ausgeführt, die den ISP-Kommandos im vorangegangenen Kapitel nahezu entsprechen:

```
/START-PLI1-COMPILER SOURCE = exampel.source                    1)

/EXEC $TSOSLNK                                                  2)
 PROGRAM exampel,FILENAM=exampel.object,MAP=NO
 INCLUDE *
 END

/ START-PLI1-PROGRAM FROM-FILE = exampel.object                 3)
```

Die einzelnen Kommandos haben folgende Wirkung:

1) Aufruf des PLI1-Compilers mit Steueranweisungen zum Übersetzen für eine PL/I-Prozedur, die in der Datei mit dem Namen exampel.source enthalten ist. Voraussetzungen dafür sind: Die PLI1-Prozedur und die PLI1-Syntaxdatei für SDF müssen installiert sein und der PLI1-Compiler muß in der Datei $TSOS.PLI1 abgelegt sein.
2) Aufruf des Binders mit der Steueranweisung, den vom PLI1-Compiler erzeugten Bindemodul mit dem namen exampel aus der EAM-Bibliothek zu binden und den erzeugten Lademodul in die Datei mit dem Namen exampel.object abzulegen. Voraussetzungen dafür sind: Der vom PLI1-Compiler generierte Bindemodul muß in der EAM-Bibliothek enthalten sein. Das PLI1-Laufzeitsystem muß in der Bibliothek $TSOS.TASKLIB enthalten sein, anderenfalls müßte mit dem SYSFILE-Kommando die Tasklib auf die PLI1-Laufzeitbibliothek umgeleitet werden.
3) Aufruf des gebundenen PL/I-Objektprogramms aus der Datei mit dem Namen exampel.object. Voraussetzungen dafür sind: Die PLI1-Prozedur und PLI1-Syntaxdatei für SDF müssen installiert sein.

Folgende Protokolle werden erstellt:

Ablaufprotokoll auf Systemdatei SYSOUT:

```
/START-PLI1-COMPILER SOURCE=EXAMPEL.SOURCE
% BLS0500 PROGRAM 'PLI1', VERSION '4.1A' OF '91-04-23' Loaded. COPYRIGHT...
———— THERE WAS NO DIAGNOSTIC MESSAGE
———— OBJECTMODUL 'EXAMPEL' GENERATED AND WRITTEN TO: *EAM
———— OBJECTMODUL 'EXAMPEL7'GENERATED AND WRITTEN TO: *EAM
END OF SIEMENS PLI1-COMPILER VERSION 4.1A , TIME USED:     2.11 SEC
/EXEC $TSOSLNK
%  BLS0500 PROGRAM 'TSOSLNK', VERSION 'V21.0D12' OF '90-05-10' LOADED.
%  LNK0500 PROGRAM BOUND
%  LNK0503 PROGRAM FILE WRITTEN: EXAMPEL.OBJECT
%  LNK0504 NUMBER PAM PAGES USED:     11
/START PLI1-PROGRAM EXAMPEL.OBJECT
%  BLS0500 PROGRAM 'EXAMPEL', VERSION ' ' OF '91-05-02' LOADED.
END OF PROGRAM EXAMPEL , RTS 4.1A-DDD, TIME USED:     0.80 SEC
```

Übersetzungsprotokoll auf Systemdatei SYSLIST:

```
COMPILER-OPTIONS USED

STORAGE = ( STACK(16,4), AREA(16,16,1775))
LIST    = ( NOESD, NOTERMINAL, NOSUMMARY, OPTIONS, NOSAVLST, NOMAP, NEST,
            IREF, NOXREF, SOURCE, NOINSOURCE, NOAGGREGATE, OFFSET, NOASSM,
            NOOUTTEXT, NOLINECNT )
FORMAT  = ( TERMINAL(0,80), PRINTER(64,132), ENGLISH )
MESSAGE = NOSYSLIST
SOURCE  = EXAMPEL.SOURCE
MARGINS = ( TEXT(2,72), PAD, NOLIND, NOASACNTRL, GAMEKEY(0,0), CHAR60,
            NOSAVMAC )
DIAGNOST= ( NOTERMINAL, NOSAVLST, WARNING )
COMLIB  = NO
OBJECT  = (ERROR(32767), ABORT(500), OUT )
OPTIONS = ( NOISO, NOMAIN, NOINTERRUPT, NOMACRO, XS, NOBITPTR )
OPTIMIZE= ( NOTIME, NOOVERLAP, NOENABLING, NOREORDER )
DEBUG   = ( NOSTMT, NOPROCTRACE, NOLABTRACE, NOCALLTRACE, NOGOTOTRACE,
            NORETURNTRACE, NOBREAKPOINT )
SYMTEST = MAP
MODULE  = *

        1      EXAMPEL: PROC OPTIONS(MAIN);
        2               DCL SYSPRINT FILE;
        3               PUT PAGE LIST('VERY EASY TO HANDLE');
        4               END;
```

Protokoll des Binders

```
PROGRAM EXAMPEL,FILENAM=EXAMPEL.OBJECT,MAP=NO
INCLUDE *
END
PROG BOUND
PROGRAM FILE WRITTEN : EXAMPEL.OBJECT
NUMBER PAM PAGES USED:     11
```

Ausgabe des PL/I-Programms:

```
VERY EASY TO HANDLE
```

Bei komplizierteren Programmabläufen werden weitere Steuerungsmöglichkeiten notwendig. Eine ausführliche Beschreibung der gesamten Steuermöglichkeiten ist für den PLI1-Übersetzer in Kapitel 3, für den Binder in Kapitel 4 und für das PLI1-Objektprogramm in Kapitel 5 erläutert.

2.2 Beispiele

Die folgenden Beispiele demonstrieren übliche Kommandofolgen, wie sie für häufig auftretende Vorgänge im Zusammenhang mit PL/I-Programmen benötigt werden. Allgemeine Voraussetzungen:

- Compiler PLI1 unter $TSOS gespeichert,
- PLI1-Laufzeitsystem unter $TSOS.TASKLIB verfügbar
- PLI1-Meldungstext-Dateien unter $TSOS.PLI1.TEXT.D bzw. E katalogisiert.

2.2.1 Stapelverarbeitung: Lesen, Drucken

Aufgabe:

Das folgende Programm liest von der Datei SYSIN mit GET LIST bei jedem Aufruf Werte für 3 Variable, wobei jeweils 11 Datenelemente benötigt werden. Die interne Prozedur PROCESS symbolisiert die Verarbeitung der eingelesenen Daten, die hier lediglich in einer formalisierten Ausgabe nach SYSPRINT besteht.

Voraussetzung:

Das PL/I-Programm wird als Stapelprozeß ausgeführt. Die Daten für das Programm werden über SYSIN eingelesen und liegen im Anschluß an die Karten des Quellprogramms. Es gibt zwei Möglichkeiten für den Ablauf des Prozesses:

- Der unten abgebildete Kartenstapel wird über einen Lochkartenleser eingelesen und vom BS2000 bearbeitet.
- Die dem unten abgebildeten Kartenstapel entsprechenden Kommandos, Steueranweisungen, Quellen und Daten befinden sich in einer Datei (sie wurden z.B. mit dem DATA-Kommando eingetragen). In diesem Fall wird der Prozeß mit dem ENTER-Kommando aktiviert, z.B.

```
/ENTER Dateiname, TIME=...
```

Aufbau des Lochkartenstapels:

Datenkarten die von Objekt-Programmen eingelesen werden

```
  15   ,  23  , 'AB'  ,  'CD'  'EF'  ,  15 30 , 16  32 , 25 50
 16 24,'AC' 'CE' 'EG',16 31,17 33,26 51
17 25 'AD' 'CF' 'EH' 17 32 18 34 27 52
```

Ablaufprotokoll auf SYSOUT

```
/ REMARK ..................................................... TRANSLATE
/ EXEC $PLI1                                                          1)
%  BLS0500 PROGRAM 'PLI1', VERSION '4.0A' OF '88-10-03' LOADED.
         *COMOPT LIST=(NO,SOURCE)
         *END
```

Quellprogramm (Programmtext siehe unter Quellprotokoll in SYSLST auf der folgenden Seite)

```
..... THERE WAS NO DIAGNOSTIC MESSAGE
..... OBJECTMODUL 'EXAM221'  GENERATED AND WRITTEN TO: *EAM             2)
..... OBJECTMODUL 'EXAM2217' GENERATED AND WRITTEN TO: *EAM
END OF SIEMENS PLI1-COMPILER VERSION 4.0A , TIME USED:     1.31 SEC
/ REMARK ..................................................... LINK
/ EXEC $TSOSLNK                                                       3)
%  BLS0500 PROGRAM 'TSOSLNK', VERSION '21.0C40' OF '87-09-28' LOADED.
         PROGRAM EXAM221, MAP=NO, FILENAM=EXAM221
         INCLUDE *
         END



%  LNK0500 PROG BOUND
%  LNK0503 PROG FILE WRITTEN: EXAM221
%  LNK0504 NUMBER PAM PAGES USED:      9
/ REMARK ..................................................... EXECUTE
/ EXEC EXAM221                                                        4)
%  BLS0500 PROGRAM 'EXAM221', VERSION ' ' OF '88-10-06' LOADED.
```

Eingabe-Daten (siehe vorhergehende Seite)

```
END OF PROGRAM EXAM221 , RTS 4.0A-AAA, TIME USED:     0.17 SEC
/ REMARK ..................................................... END
```

1) Start des PL/I-Compilers mit dem EXEC-Kommando. Die im Stapel übergebene Steueranweisung LIST = (NO, SOURCE) bewirkt, daß nur die Liste des Quellprogramms auf SYSLST ausgegeben wird.
2) Als Objektmoduln wurden erzeugt und im EAM-Bereich abgelegt:
   - Der Code-Modul EXAM221
   - Der Daten-Modul EXAM2217

   Die Bildungsregel für diese Namen findet man im Abschnitt 4.6.
3) Start des Binders.
4) Start des Lademoduls. Der Name des Programms ergibt sich durch die PROGRAM-Anweisung für den Binder.

Quellprogramm-Liste auf SYSLST

```
 1      EXAM221: PROCEDURE OPTIONS(MAIN);
 2
 3               DCL SYSPRINT      FILE INTERNAL;
 4               DCL A             FIXED BIN     DIM(2)    INIT(0,0);
 5               DCL B             CHAR(5)       DIM(3)    INIT((3)(1)'');
 6               DCL 1 C                         DIM(3),
 7                   2 NO          FIXED DEC                 INIT ((3)0),
 8                   2 VALUE       FIXED DEC                 INIT(0,0,0);
 9               DCL SYSIN         FILE STREAM INPUT;
10
11               OPEN FILE(SYSIN);
12               ON ENDFILE(SYSIN) GOTO END;
13               ON CONVERSION  BEGIN;
14   1                          ON CONVERSION SYSTEM;
15   1                          PUT DATA;
16   1                          END;
17
18            E: GET FILE(SYSIN) LIST(A,B,C);
19               CALL PROCESS;
20               GOTO E;
21      PROCESS: PROCEDURE;
22   1           PUT SKIP(1) EDIT (A) (      (2)F(3))
23   1                            (B) (X(2), (3)A(5))
24   1                            (C) (         F(3));
25   1           PUT SKIP(1);
26   1           END;
27
28          END: PUT SKIP(1) LIST('~~~ END EXAMPLE 1');
29               END;
```

```
PROGRAM EXAM221,MAP=NO,FILENAM=EXAM221
INCLUDE *                                    5)
END
PROG BOUND
PROGRAM FILE WRITTEN : EXAM221
NUMBER PAM PAGES USED:      9
```

5) Protokoll der Steuerangaben für den Binder. Die PROGRAM-Anweisung legt fest, daß kein Binderprotokoll zu erzeugen ist. Das Lademodul soll in der Datei EXAM221 hinterlegt werden. Die Anweisung INCLUDE * bezeichnet den Inhalt des EAM-Bereichs als das zu bindende Material. Dort liegen die durch die vorangegangene Übersetzung erzeugten Moduln.

Achtung

Durch INCLUDE * werden alle Moduln aus dem EAM-Bereich verarbeitet; d.h. auch solche, die von anderen Übersetzungen des gleichen Prozesses stammen können. Den EAM-Bereich sollte man deshalb vor dem Beginn von neuen Übersetzungen mit dem Kommando "/ERASE *" löschen.

Druckausgabe aus dem Lademodul nach SYSLST

```
15 23  AB   CD   EF    15 30 16 32 25 50

16 24  AC   CE   EG    16 31 17 33 26 51

17 25  AD   CF   EH    17 32 18 34 27 52

... END EXAMPLE 1
```

Anmerkung

Diese Betriebsform eignet sich nicht für Dialogprozesse. Zwar können dem PL/I-Compiler die Zeilen des Quellprogramms im Dialog übergeben werden, sie sind aber nicht erneut zugreifbar. Bei einer zweiten Übersetzung des gleichen oder eventuell korrigierten Programms müßte man alle Eingaben nochmals machen. Entsprechendes gilt für die Daten. Für Dialogprozesse ist das folgende Beispiel typisch.

2.2.2 Dialogprozeß: Ein-/Ausgabe über SYSFILE-Kommando

Aufgabe:

Identisch mit Beispiel 2.2.1

Voraussetzung:

Das Quellprogramm liegt in der Datei SOURCE222. Die Daten für die PL/I-Datei SYSIN liegen in der Datei INPUT222. Die Ergebnisausgabe über SYSPRINT soll in die Datei OUTPUT222 geleitet werden. Es wird kein Übersetzungsprotokoll gewünscht.

Ablaufprotokoll auf SYSOUT

```
/REMARK ..................................................... TRANSLATE
/EXEC $PLI1
%  BLS0500 PROGRAM 'PLI1', VERSION '4.0A' OF '88-10-03' LOADED.
          *COMOPT SOURCE=SOURCE222, LIST=NO
          *END

..... THERE WAS NO DIAGNOSTIC MESSAGE
..... OBJECTMODUL 'EXAM222'  GENERATED AND WRITTEN TO: *EAM
..... OBJECTMODUL 'EXAM2227' GENERATED AND WRITTEN TO: *EAM
END OF SIEMENS PLI1-COMPILER VERSION 4.0A , TIME USED:     1.76 SEC
/REMARK ....................................................... LINK
/EXEC $TSOSLNK
%  BLS0500 PROGRAM 'TSOSLNK', VERSION '21.0C40' OF '87-09-28' LOADED.
          PROGRAM EXAM222, MAP=NO, FILENAM=PROG.EXAM222
          INCLUDE *
          END
%  LNK0500 PROG BOUND
%  LNK0503 PROG FILE WRITTEN: PROG.EXAM222
%  LNK0504 NUMBER PAM PAGES USED:      9
/REMARK ..................................................... FILES
/SYSFILE SYSLST=OUTPUT222                                              1)
/SYSFILE SYSDTA=INPUT222
/REMARK .................................................... EXECUTE
/EXEC PROG.EXAM222
%  BLS0500 PROGRAM 'EXAM222', VERSION ' ' OF '88-10-06' LOADED.
END OF PROGRAM EXAM222 , RTS 4.0A-AAA, TIME USED:     0.17 SEC
/REMARK ...................................................... END
```

Als Ergebnis des PRINT-Kommandos erhält man die gleichen Ausgaben wie bei Beispiel 2.2.1.

Erklärung des Ablaufes:

1) Im PL/I-Programm erfolgt die Ausgabe nach SYSPRINT.
   SYSPRINT ist der Systemdatei SYSLST zugeordnet. Da das Ergebnis aber nicht im Ablaufprotokoll, sondern in einer Datei erscheinen soll, ordnet man über das SYSFILE-Kommando die Systemdatei SYSLST der Datei OUTPUT222 zu.

Im Programm wird über SYSIN gelesen. SYSIN ist per Voreinstellung mit SYSDTA identisch, deshalb wird SYSDTA per SYSFILE-Kommando auf INPUT222 umgeschaltet.

Anmerkung zur Lagerung von Quellprogrammen und Daten

Die Aufbewahrung von Quellprogrammen und Eingabedaten in Dateien gestattet die Anwendung der Aufbereitungsprogramme EDT und EDOR zur eventuell nötigen Korrektur des Programms bzw. der Daten.

2.2.3 Stapelverarbeitung: Ein-/Ausgabe über LINK

Aufgabe:

Aus einer Datei STOCK223 sollen strukturierte Datensätze fester Länge gelesen werden. Die Daten sind nach den Werten einer Menge (MG) zu überprüfen. Alle Datensätze, für die MG ≤ 1100 ist, sollen neu strukturiert in die Datei LIST223 geschrieben werden. Diese Datei ist anschließend zu drucken.

Voraussetzung:

Stapelprozeß; die Eingabedatei liegt unter dem Namen STOCK223 vor, muß jedoch mit dem LINK-Namen CARD verkettet werden. Die Ausgabedatei muß eingerichtet werden. Das Quellprogramm und die Steueranweisungen sind im Eingabestapel enthalten. Es sollen alle bei der Übersetzung anfallenden Listen gedruckt werden.

Eingabe-Daten in der Datei STOCK223

Index Datensatz

```
10000000 11111SOAP            10001033000
20000000 12134PERFUME         01120200637
30000000 14667DISH-CLOTH      00510100123
50000000 15678PAPERBAGS       12450500021
60000000 74567NATRON          03060500173
70000000 90000POWDER          01340501073
80000000 91000PAPER           12450100714
```

Ergebnis des Laufs in der Ausgabe-Datei LIST223

```
11111 SOAP           1000 10 33000
12134 PERFUME        0112 02 00637
14667 DISH-CLOTH     0051 01 00123
74567 NATRON         0306 05 00173
90000 POWDER         0134 05 01073
```

SYSOUT-Protokoll

```
/REMARK ..................................................... TRANSLATE
/EXEC $PLI1
%  BLS0500 PROGRAM 'PLI1', VERSION '4.0A' OF '88-10-03' LOADED.
          *COMOPT LIST=ALL, MARGINS=T(2,60),
          *END
```

Quellzeilen (hier fortgelassen; siehe Übersetzungs-Protokoll)

```
..... THERE WAS NO DIAGNOSTIC MESSAGE
..... OBJECTMODUL 'EXAM223'  GENERATED AND WRITTEN TO: *EAM
..... OBJECTMODUL 'EXAM2237' GENERATED AND WRITTEN TO: *EAM
END OF SIEMENS PLI1-COMPILER VERSION 4.0A , TIME USED:     1.81 SEC
/REMARK ..................................................... LINK
/EXEC $TSOSLNK
%  BLS0500 PROGRAM 'TSOSLNK', VERSION '21.0C40' OF '87-09-28' LOADED.
          PROGRAM EXAM223, MAP=NO, FILENAM=EXAM223
          INCLUDE *
          END
%  LNK0500 PROG BOUND
%  LNK0503 PROG FILE WRITTEN: EXAM223
%  LNK0504 NUMBER PAM PAGES USED:      7
/REMARK ..................................................... FILES
/FSTAT STOCK223,ALL                                                  1)
0000003 :B:$STSPL1.STOCK223
  FCBTYPE = SAM   VSNTYPE = PUB       LASTPG = 0000001   2ND ALLO= 00006
  SHARE   = NO    ACCESS  = WRITE
  ACCESS# = 001   CRDATE  = 85-08-22  EXDATE = 85-08-22  LADATE  = 85-08-22
  RDPASS  = NONE  WRPASS  = NONE      EXPASS = NONE
  VERSION = 001   BACKUP# = 000       LARGE  = NO        BACKUP  = A
  DESTROY = NO    AUDIT   = NONE
  BLKTYPE = STD   BLKSIZE = 002048    RECFORM= (V,N)     RECSIZE = 00000
  VSN/DEV/EXT =   PUBB02/D3475/001
  EXTCNT  =   1
:B:  PUBLIC:    1 FILE.  RES=        3, FREE=        2, REL=        0 PAGES
/FILE STOCK223,LINK=CARD                                             2)
/FILE LIST223,LINK=LIST, FCBTYPE=SAM, RECFORM=F, RECSIZE=39, BLKSIZE=STD
/REMARK ..................................................... EXECUTE
/SETSW ON=(1)                                                        3)
/EXEC EXAM223
%  BLS0500 PROGRAM 'EXAM223', VERSION ' ' OF '88-10-06' LOADED.
          *RUNOPT LIST=(OPTIONS,SUMMARY), FORMAT=PRINTER(,120)
          *END
END OF PROGRAM EXAM223 , RTS 4.0A-AAA, TIME USED:     0.29 SEC
/REMARK ..................................................... END
```

Erklärung des Ablaufs von Beispiel 2.2.3

1) Das FSTAT-Kommando wurde gegeben, um zu zeigen, daß die Datei die gewünschten Parameter hat.

2) Vereinbarung des LINK-Namens für die Datei STOCK223.
Einrichten der Datei LIST223 mit den erforderlichen Kenndaten.

3) Das Objektprogramm wurde beim Start mit Steueranweisungen versorgt. Deshalb muß Prozeßschalter 1 vor dem Aufruf gesetzt sein.

Beim Übersetzen wirksame Steueranweisungen auf SYSLST (*COMOPT LIST = OPTIONS)

```
COMPILER-OPTIONS USED


  STORAGE = ( STACK(16,4), AREA(16,16,975))
  LIST    = ( ESD, NOTERMINAL, SUMMARY, OPTIONS, NOSAVLST, MAP, NEST, IREF,
            FULLXREF, EXPAND, NOINSOURCE, AGGREGATE, OFFSET, ASSM,
            OUTTEXT, NOLINECNT )
  FORMAT  = ( TERMINAL(0,80), PRINTER(64,132), ENGLISH )
  MESSAGE = NOSYSLST
  SOURCE  = EXAM223
  MARGINS = ( TEXT(2,60), PAD, NOLINID, NOASACNTRL, GAMKEY(0,0), CHAR60,
            NOSAVMAC )
  DIAGNOST= ( NOTERMINAL, NOSAVLST, WARNING )
  COMLIB  = NO
  OBJECT  = ( ERROR(32767), ABORT(500), OUT )
  OPTIONS = ( NOISO, NOMAIN, NOINTERRUPT, NOMACRO, NOXS, NOBITPTR)
  OPTIMIZE= ( NOTIME, NOOVERLAP, NOENABLING, NOREORDER )
  DEBUG   = ( NOSTMT, NOPROCTRACE, NOLABTRACE, NOCALLTRACE, NOGOTOTRACE,
            NORETURNTRACE, NOBREAKPOINT )
  SYMTEST = MAP
  MODULE  = *
```

Quellprogrammliste auf SYSLST (*COMOPT LIST = EXPAND)

```
 1      EXAM223: PROCEDURE OPTIONS(MAIN);
 2
 3               DCL CARD           FILE RECORD INPUT,
 4                   LIST           FILE RECORD OUTPUT;
 5               DCL 1 IN,
 6                     2 NO         CHAR(5),
 7                     2 BZ         CHAR(15),
 8                     2 MG         CHAR(4),
 9                     2 ME         CHAR(2),
10                     2 PR         CHAR(5);
11               DCL 1 OUT,
12                     2 NO         CHAR(5),
13                     2 FILL1      CHAR(2)   INIT (’  ’),
14                     2 BZ         CHAR(15),
15                     2 FILL2      CHAR(2)   INIT (’  ’),
16                     2 AMG        CHAR(4),
17                     2 FILL3      CHAR(2)   INIT (’  ’),
18                     2 AME        CHAR(2),
19                     2 FILL4      CHAR(2)   INIT (’  ’),
20                     2 APR        CHAR(5);
21
22               OPEN FILE(CARD), FILE(LIST);
23               ON ENDFILE(CARD) GOTO END;
24
25           A1: READ FILE(CARD) INTO(IN);
26               IF IN.MG <= ’1100’ THEN DO;
27   1              OUT.NO  = IN.NO;
```

```
28   1              OUT.BZ  = IN.BZ;
29   1              AMG     = IN.MG;
30   1              OUT.AME = ME;
31   1              APR     = IN.PR;
32   1              WRITE FILE(LIST) FROM(OUT);
33   1              END;
34              GOTO A1;
35
36              END: END;
```

Liste der externen Namen im Code-Modul auf SYSLST (*COMOPT LIST = ESD)

```
LIST OF EXTERNAL NAMES IN THE CODE MODULE

  NAME     TYPE    MA   ADDR   L/ESID

EXAM223   CSECT   04  000000  00041C
EXAM2237  EXTERN  00  000000  404040
P$3#20##  EXTERN  00  000000  404040
P$RECIO#  EXTERN  00  000000  404040
P$OPNEX#  EXTERN  00  000000  404040
CARD      COMMON  00  000000  000200
LIST      COMMON  00  000000  000200
P$START#  ENTRY   00  000078  000001
```

Objektcode-Liste auf SYSLST (Ausschnitt) (*COMOPT LIST = ASSM)

```
          000260 05 EF               BALR  14,15
26        000262 D2 03 D04C D050     MVC   76(4,13),80(13)
          000268 D5 03 D25C B13C     CLC   604(4,13),316(11)       MG,
          00026E 47 20 A2C2          BC    2,706(0,10)             C.1
27        000272 D2 04 D267 D248     MVC   615(5,13),584(13)       NO, NO
28        000278 D2 0E D26E D24D     MVC   622(15,13),589(13)      BZ, BZ
29        00027E D2 03 D27F D25C     MVC   639(4,13),604(13)       AMG, MG
30        000284 D2 01 D285 D260     MVC   645(2,13),608(13)       AME, ME
31        00028A D2 04 D289 D262     MVC   649(5,13),610(13)       APR, PR
32        000290 58 60 B038          L     6,56(0,11)              LIST
          000294 50 60 D0A0          ST    6,160(0,13)
```

Liste der externen Namen im STATIC-Modul auf SYSLST (*COMOPT LIST = ESD)

```
LIST OF EXTERNAL NAMES IN THE STATIC MODULE

  NAME     TYPE    MA   ADDR   L/ESID

EXAM2237  CSECT   00  000000  000008
EXAM223   EXTERN  00  000000  404040
CARD      COMMON  00  000000  000200
LIST      COMMON  00  000000  000200
```

Speicherbelegungsliste auf SYSLST (*COMOPT LIST = MAP)

```
                M A P - L I S T

SOURCE-REF.     TYPE            ADDR   OFFSET    NAME


          0     ROOTBLOCK

          1     ENTRY CONST        0             EXAM223



          1     EXT PROCEDURE                    EXAM223

          3     ESD #              3             CARD
          4     ESD #              4             LIST
          5     AUTO             248             IN
                  MEMBER                  0       NO      IN IN
                  MEMBER                  5       BZ      IN IN
                  MEMBER                 14       MG      IN IN
                  MEMBER                 18       ME      IN IN
                  MEMBER                 1A       PR      IN IN
         11     AUTO             267             OUT
                  MEMBER                  0       NO      IN OUT
                  MEMBER                  5       FILL1     IN OUT
                  MEMBER                  7       BZ      IN OUT
                  MEMBER                 16       FILL2     IN OUT
                  MEMBER                 18       AMG     IN OUT
                  MEMBER                 1C       FILL3     IN OUT
                  MEMBER                 1E       AME     IN OUT
                  MEMBER                 20       FILL4     IN OUT
                  MEMBER                 22       APR     IN OUT
         25     LABEL CONST      236             A1
         36     LABEL CONST      2C6             END



          23    ON UNIT                          ENDFILE



 * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
```

Liste der Struktur-Längen auf SYSLST (*COMOPT LIST = AGGREGATE)

```
                                S T R U C T U R E   L E N G T H   T A B L E
                                                      OFFSET          ELEMENT LENGTH          TOTAL LENGTH
SOURCE-REF  LEVEL IDENTIFIER                DIMENSION   DEC    HEXDEC     DEC    HEXDEC      DEC    HEXDEC

     5      1 IN                                          0         0                             31        1F
              2 NO                                        0         0         5         5
              2 BZ                                        5         5        15         F
              2 MG                                       20        14         4         4
              2 ME                                       24        18         2         2
              2 PR                                       26        1A         5         5

    11      1 OUT                                         0         0                             39        27
              2 NO                                        0         0         5         5
              2 FILL1                                     5         5         2         2
              2 BZ                                        7         7        15         F
              2 FILL2                                    22        16         2         2
              2 AMG                                      24        18         4         4
              2 FILL3                                    28        1C         2         2
              2 AME                                      30        1E         2         2
              2 FILL4                                    32        20         2         2
              2 APR                                      34        22         5         5


********** END OF STRUCTURE LENGTH TABLE **********************************************************
```

Querverweis-Liste auf SYSLST (*COMOPT LIST = FULLXREF)

```
                C R O S S - R E F E R E N C E - T A B L E     -   REFERENCED IDENTIFIERS   -
IDENTIFIER      DIMENSION DATATYPE                              STORAGE            REFERENCES


AME                       CHAR (2)                  UNAL      MEM-2 (OUT)         DCL 18 30
AMG                       CHAR (4)                  UNAL      MEM-2 (OUT)         DCL 16 29
APR                       CHAR (5)                  UNAL      MEM-2 (OUT)         DCL 20 31
A1                        LABEL                               CONSTANT            DCL 25 34
BZ                        CHAR (15)                 UNAL      MEM-2 (OUT)         DCL 14 28
BZ                        CHAR (15)                 UNAL      MEM-2 (IN)          DCL 7 28
CARD                      FILE INPUT RECORD              EXT CONSTANT            DCL 3 22 23 25
END                       LABEL                               CONSTANT            DCL 36 23
ENDFILE                   CONDITION                                               +++ 23
EXAM223                   ENTRY                          EXT CONSTANT            DCL 1
FILL1                     CHAR (2)                  UNAL      MEM-2 (OUT)         DCL INIT 13 11
FILL2                     CHAR (2)                  UNAL      MEM-2 (OUT)         DCL INIT 15 11
FILL3                     CHAR (2)                  UNAL      MEM-2 (OUT)         DCL INIT 17 11
FILL4                     CHAR (2)                  UNAL      MEM-2 (OUT)         DCL INIT 19 11
IN                        STRUCTURE                 UNAL      AUTOMATIC           DCL 5 25
LIST                      FILE OUTPUT RECORD             EXT CONSTANT            DCL 4 22 32
ME                        CHAR (2)                  UNAL      MEM-2 (IN)          DCL 9 30
MG                        CHAR (4)                  UNAL      MEM-2 (IN)          DCL 8 26 29
NO                        CHAR (5)                  UNAL      MEM-2 (IN)          DCL 6 27
NO                        CHAR (5)                  UNAL      MEM-2 (OUT)         DCL 12 27
OUT                       STRUCTURE                 UNAL      AUTOMATIC           DCL 11 32
PR                        CHAR (5)                  UNAL      MEM-2 (IN)          DCL 10 31
..... THERE ARE  N O  IDENTIFIERS  N O T  REFERENCED .....................................

********** END OF CROSS-REFERENCE-TABLE ***************************************************
```

Programm-Statistik des Übersetzers auf SYSLST (*COMOPT LIST = SUMMARY)

```
SUMMARY OF VIRTUAL MAIN STORAGE REQUIREMENTS
PROCEDURE STACK:    20 PAGES; SYSTEM CALLS:     2 REQM,     0 RELM
STANDARD AREA:      21 PAGES; SYSTEM CALLS:     2 REQM
ASSIGNED MEMORY:   237 PAGES CLASS 6   AND:    66 PAGES CLASS 5
```

Binde-Protokoll auf SYSLST

```
PROGRAM EXAM223, MAP=NO, FILENAM=EXAM223
INCLUDE *
END
PROG BOUND
PROGRAM FILE WRITTEN : EXAM223
NUMBER PAM PAGES USED:      7
```

Bei der Programmausführung wirksame Steueranweisungen auf SYSLST (*RUNOPT LIST = SUMMARY)

```
RUNTIME-OPTIONS USED


 STORAGE = ( STACK(16,4), AREA(16,16,1071))
 LIST    = ( NOTERMINAL, SUMMARY, OPTIONS )
 FORMAT  = ( TERMINAL(0,80), PRINTER(64,120), ENGLISH )
 ARGUMENT= ''
 MESSAGE = NOSYSLST
 SYSFILE = ( SYSDTA(SYSIN), SYSLST(SYSPRINT), SYSOUT(SYSOUT),
           LINESIZE(0,0,0), PAGESIZE(0,0), DISPLAY(SYSDTA))
 DUMP    = ( COND, NOSNAP, NOAREA, NOSTACK, NORANGE )
 TRACE   = ( TERMINAL, NOLABTRACE, NOPROCTRACE, NORETURNTRACE,
           NOGOTOTRACE, NOCALLTRACE, NOIFTRACE )
 TABULATOR = (1,11,21,31,41,51,61,71,81,91,101,111,121)
 CONTROL = NOALIGN
 ACTIVE  = YES
```

Programm-Statistik über die Programmausführung auf SYSLST (*RUNOPT LIST = SUMMARY)

```
SUMMARY OF VIRTUAL MAIN STORAGE REQUIREMENTS
PROCEDURE STACK:    20 PAGES; SYSTEM CALLS:     2 REQM,     0 RELM
STANDARD AREA:       1 PAGES; SYSTEM CALLS:     1 REQM
ASSIGNED MEMORY:   101 PAGES CLASS 6   AND:    57 PAGES CLASS 5
```

2.2.4 Verwendung von Bibliotheken

Aufgabe:

Es soll der Umgang mit einer Benutzerbibliothek demonstriert werden.
Zunächst werden die Prozeduren AVG (Mittelwertbestimmung) und AVGDEV (mittlere Abweichung vom Mittelwert) übersetzt und in die Bibliothek PLIBIB eingetragen.
Anschließend wird die Hauptprozedur übersetzt, welche die oben genannten Funktionen benötigt. Beim Binden werden die Funktionen aus der Bibliothek übernommen. Die Hauptprozedur liest Daten ein und bestimmt davon den Mittelwert und das Mittel der Abweichungen vom Mittelwert.

Voraussetzung:

Die drei Prozeduren AVGDEV224, AVG224 und EXMPL4 befinden sich in den Dateien AVGDEV224, AVG224 und SOURCE224.
Sie werden jeweils mit den Angaben übersetzt:

```
*COMOPT SOURCE = Datei, LIST = (NO, SOURCE)
```

Die Eingabedaten werden über SYSIN eingelesen und folgen im Stapel direkt dem Kommando /EXEC EXAM224. Die Ergebnisse werden nach SYSLST ausgegeben.

Erklärung des Ablaufes von Beispiel 2.2.4 (zu folgender Seite)

1) Löschen EAM-Bereich, falls noch Bindemoduln von vorangegangenen Übersetzungen vorhanden sind.

2) Kopieren der Bindemoduln der vorangegangenen Übersetzungen aus dem EAM-Bereich in eine Bibliothek. Verwendet wird das Dienstprogramm LMR (siehe Abschnitt 3.6.4). Steuerung des Dienstprogramms siehe [11].

3) Löschen des Inhalts vom EAM-Bereich. Damit wird ein Zustand simuliert, als ob die Prozeduren AVG und AVGDEV in einem vorangegangenen Prozeß übersetzt und in die Bibliothek eingetragen worden seien.

Protokoll auf SYSOUT

```
/ERASE *                                                                    1)
/REMARK ..................................... TRANSLATE FOR LIBRARY
/EXEC $PLI1
%  BLS0500 PROGRAM 'PLI1', VERSION '4.0A' OF '88-10-03' LOADED.
          *COMOPT SOURCE=AVGDEV224,
          *COMOPT LIST=(NO,SOURCE)
          *END

..... THERE WAS NO DIAGNOSTIC MESSAGE
..... OBJECTMODUL 'AVGDEV' GENERATED AND WRITTEN TO: *EAM
END OF SIEMENS PLI1-COMPILER VERSION 4.0A , TIME USED:     1.39 SEC
/EXEC $PLI1
%  BLS0500 PROGRAM 'PLI1', VERSION '4.0A' OF '88-10-03' LOADED.
          *COMOPT SOURCE=AVG224,
          *COMOPT LIST=(NO,SOURCE)
          *END

..... THERE WAS NO DIAGNOSTIC MESSAGE
..... OBJECTMODUL 'AVG' GENERATED AND WRITTEN TO: *EAM
END OF SIEMENS PLI1-COMPILER VERSION '4.0A' , TIME USED:     2.74 SEC
/REMARK ..................................... INCLUDE IN LIBRARY
/EXEC LMR                                                                   2)
%  BLS0500 PROGRAM LMR.266, VERSION 266 OF 83-03-11 LOADED.
          MODLIB MODLIB224
          COPYALL SOURCE=*
          END
LMR (BS2000) VERSION V26.6506 NORMAL END
/STEP
/ERASE *                                                                    3)
/REMARK ..................................................... TRANSLATE
/EXEC $PLI1
%  BLS0500 PROGRAM 'PLI1', VERSION '4.0A' OF '88-10-03' LOADED.
          *COMOPT SOURCE=SOURCE224,
          *COMOPT LIST=(NO,SOURCE)
          *END
..... THERE WAS NO DIAGNOSTIC MESSAGE
..... OBJECTMODUL 'EXAM224'  GENERATED AND WRITTEN TO: *EAM
..... OBJECTMODUL 'EXAM2247' GENERATED AND WRITTEN TO: *EAM
END OF SIEMENS PLI1-COMPILER VERSION 4.0A , TIME USED:     2.91 SEC
/REMARK ..................................................... LINK
/EXEC $TSOSLNK
%  BLS0500 PROGRAM 'TSOSLNK', VERSION '21.0C40' OF '87-09-28' LOADED.
          PROGRAM EXEM224, MAP=NO, FILENAM=EXAM224
          INCLUDE *
          RESOLVE, MODLIB224
          END
%  LNK0500 PROG BOUND
%  LNK0503 PROG FILE WRITTEN: EXAM224
%  LNK0504 NUMBER PAM PAGES USED:      8
/REMARK ..................................................... EXECUTE
/EXEC EXAM224
%  BLS0500 PROGRAM 'EXAM224', VERSION ' ' OF '88-10-06' LOADED.
8 3 5 6 4 2 7 1 9
END OF PROGRAM EXAM224 , RTS 4.0A-AAA, TIME USED:     0.16 SEC
/REMARK ..................................................... END
```

Quellprozedur AVGDEV (Funktion) auf SYSLST (*COMOPT LIST = SOURCE)

```
 1       AVGDEV: PROCEDURE (N, AVERAGE, X) RETURNS (FLOAT BIN);
 2
 3               DCL X                          DIM(*);
 4               DCL (AVERAGE,SUM)              FLOAT BIN;
 5               DCL (N,K)                      FIXED BIN;
 6
 7               SUM = 0;
 8               DO K = 1 TO N;
 9   1              SUM = SUM + ABS(AVERAGE - X(K));
10   1              END;
11              RETURN (SUM/N);
12              END;
```

Quellprozedur AVG (Funktion) auf SYSLST (*COMOPT LIST = SOURCE)

```
 1      AVG: PROCEDURE (N, X) RETURNS (FLOAT BIN);
 2
 3           DCL X                     DIM(*);
 4           DCL SUM                   FLOAT BIN;
 5           DCL (N, K)                FIXED BIN;
 6
 7           SUM = 0;
 8           DO K = 1 TO N;
 9   1          SUM = SUM + X(K);
10   1          END;
11           RETURN (SUM/N);
12           END;
```

Module in Bibliothek eintragen (Protokoll LMR) auf SYSLST

```
PROGRAM  LMR/26:65 STARTED

MODLIB MODLIB224

COPYALL SOURCE = *

END
```

Quellprozedur EXAM224 (Haupt-Prozedur) auf SYSLST (*COMOPT LIST=SOURCE)

```
 1      EXAM224: PROCEDURE OPTIONS(MAIN);
 2
 3               DCL X                   DIM(1000);
 4               DCL (N,K)               FIXED BIN;
 5               DCL (AVERAGE,AVDEV)     FLOAT BIN;
 6               DCL (AVG, AVGDEV)       EXTERNAL ENTRY RETURNS(FLOAT BIN);
 7               DCL (SYSIN,SYSPRINT)    FILE INTERNAL;
 8
 9               ON ENDFILE(SYSIN) GOTO END;
10          S10: GET LIST(N);
11               PUT SKIP LIST(N);
12               DO K = 1 TO N;
13   1              GET LIST(X(K));
14   1              PUT SKIP LIST(K, X(K));
15   1              END;
16               AVERAGE = AVG(N,X);
17               AVDEV = AVGDEV(N,AVERAGE,X);
18               PUT PAGE LIST('     AVERAGE', 'AVERAGE DEVIATION');
19               PUT SKIP LIST(AVERAGE, AVDEV);
20               GO TO S10;
21
22          END: PUT SKIP LIST('~END EXAMPLE 224');
23               END;
```

Binde-Protokoll auf SYSLST

```
PROGRAM EXAM224,MAP=NO,FILENAM=EXAM224          ⎧ Absättigen der in der
INCLUDE *                                       ⎪ Hauptprozedur verwen-
RESOLVE, MODLIB224 ─────────────────────────────⎨ deten Bezüge auf AVG
END                                             ⎪ und AVGDEV durch Verweis
PROG BOUND                                      ⎩ auf die Bibliothek.
PROGRAM FILE WRITTEN : EXAM224
NUMBER PAM PAGES USED:      8
```

Ergebnis des Programmlaufes auf SYSLST

```
       8
       1  3.00000E+00
       2  5.00000E+00
       3  6.00000E+00
       4  4.00000E+00
       5  2.00000E+00
       6  7.00000E+00
       7  1.00000E+00
       8  9.00000E+00
----.----.----.----.----.----.----.----.----.
    AVERAGE        AVERAGE DEVIATION
4.625000E+00        2.125000E+00                    neue Seite
---END EXAMPLE 224
```

2.2.5 Verwendung von Testhilfen

Aufgabe:

Es soll die Verwendung der Testhilfe TRACE gezeigt werden. Das Beispiel verwendet 2 interne Prozeduren. UP1 listet alle Elemente einer als Parameter übergebenen 2-dimensionalen Matrix auf. UP2 hat die gleiche Wirkung für 1-dimensionale Matrizen. Zu Beginn des Programms werden die Elemente der Matrix A mit den Werten 1 bis 9 belegt. Auf die Elemente der Matrix B werden die Werte 10 bis 19 zugewiesen.
Die Prozeduren bewirken den Ausdruck der Matrizen bzw. Teile davon.

Voraussetzung:

Das Programm benötigt keine Eingaben und druckt nach SYSPRINT. Die Programmquelle wird über SYSDTA eingelesen. Um den Programmablauf zu erkennen, schaltet man Prozedur-Trace ein.

Ablaufprotokoll auf SYSOUT

```
/ERASE *
/REMARK ..................................................... TRANSLATE
/EXEC $PLI1
%  BLS0500 PROGRAM 'PLI1', VERSION '4.0A' OF '88-10-03' LOADED.
          *COMOPT DEBUG=(PROCTRACE,LABTRACE),
          *COMOPT MARGINS=T(1,50)
          *END
```

Quellzeilen (hier fortgelassen; siehe Übersetzungs-Protokoll)

```
..... THERE WAS NO DIAGNOSTIC MESSAGE
..... OBJECTMODUL 'EXAM225'  GENERATED AND WRITTEN TO: *EAM
..... OBJECTMODUL 'EXAM2257' GENERATED AND WRITTEN TO: *EAM
END OF SIEMENS PLI1-COMPILER VERSION 4.0A , TIME USED:     1.70 SEC
/REMARK ..................................................... LINK
/EXEC $TSOSLNK
%  BLS0500 PROGRAM 'TSOSLNK', VERSION '21.0C40' OF '87-09-28' LOADED.
          PROGRAM EXAM225, FILENAM=EXAM225
          INCLUDE *
          END
%  LNK0500 PROG BOUND
%  LNK0503 PROG FILE WRITTEN: EXAM225
%  LNK0504 NUMBER PAM PAGES USED:      7
/REMARK ..................................................... EXECUTE
/SETSW ON=(1)
/EXEC EXAM225
%  BLS0500 PROGRAM 'EXAM225', VERSION ' ' OF '88-10-06' LOADED.
          *RUNOPT TRACE=PROCTRACE,
          *END
END OF PROGRAM EXAM225 , RTS 4.0A-AAA, TIME USED:     0.16 SEC
/REMARK ..................................................... END
```

Quellprogramm-Liste (*COMOPT LIST = SOURCE)

```
 1      EXAM225: PROC OPTIONS(MAIN);
 2
 3               DCL A              DIM(3,3) BIN FIXED,
 4                   B              DIM(3,3) BIN FIXED,
 5                   C              DIM(3)   BIN FIXED;
 6               DCL SYSPRINT       FILE INTERNAL;
 7               DCL (K,I,J)        FIXED BIN;
 8
 9               K = 0;
10               DO I = 1 TO 3;
11   1              DO J = 1 TO 3;
12   2                 K      = K+1;
13   2                 A(I,J) = K;
14   2                 B(I,J) = K+9;
15   2                 END;
16   1              END;
17                CALL UP1(A);
18                C = B(1,*);
19                PUT SKIP(3) LIST(C);
20                CALL UP2(C);
21                CALL UP2(B(1,*));
22
23           UP1: PROC(A1);
24   1            DCL A1(*,*) BIN FIXED;
25   1            PUT SKIP(1) LIST(A1);
26   1            END;
27
28           UP2: PROC(B1);
29   1            DCL B1 DIM(*) BIN FIXED;
30   1            PUT SKIP(1) LIST(B1);
31   1            END;
32
33           END;
```

Für den Übersetzer wirksame Steueranweisungen (*COMOPT LIST = OPTIONS)

```
STORAGE = ( STACK(16,4), AREA(16,16,975))
LIST    = ( NOESD, NOTERMINAL, NOSUMMARY, OPTIONS, NOSAVLST, NOMAP, NEST,
          IREF, NOXREF, SOURCE, NOINSOURCE, NOAGGREGATE, OFFSET,
          NOASSM, NOOUTTEXT, NOLINECNT )
FORMAT  = ( TERMINAL(0,80), PRINTER(64,132), ENGLISH )
MESSAGE = NOSYSLST
SOURCE  = EXAM225
MARGINS = ( TEXT(1,50), PAD, NOLINID, NOASACNTRL, GAMKEY(0,0), CHAR60 )
DIAGNOST= ( NOTERMINAL, NOSAVLST, WARNING )
COMLIB  = NO
OBJECT  = ( ERROR(32767), ABORT(500), OUT )
OPTIONS = ( NOISO, NOMAIN, NOINTERRUPT, NOMACRO )
OPTIMIZE= ( NOTIME, NOOVERLAP, NOENABLING, NOREORDER, NOXS, NOBITPTR)
DEBUG   = ( NOSTMT, PROCTRACE, LABTRACE, NOCALLTRACE, NOGOTOTRACE,
          NORETURNTRACE, NOBREAKPOINT )
SYMTEST = MAP
MODULE  = *
```

Zuordnungsliste (*COMOPT LIST = OFFSET)

```
                                LISTING OF THE OFFSETS OF THE CODE MODULE

 SOURCE-REF.          ADDR | SOURCE-REF.             ADDR | SOURCE REF.              ADDR | SOURCE-REF.

          1         000000 |               6       000098 |               9        0000EE |               10
         11         0000FC |              12       000104 |              13        000110 |               14
         15         00014E |              16       000166 |              17        00017E |               18
         19         000194 |              20       000216 |              21        00022C |               33
         23         000240 |              25       0002A2 |              26        0003FE |               28
         30         000476 |              31       000554
END ADDRESS:          000650
```

Binde-Protokoll

```
PROGRAM EXAM225, FILENAM=EXAM225
INCLUDE *
END
PROG BOUND
PROGRAM FILE WRITTEN : EXAM225
NUMBER PAM PAGES USED:       7
```

Binde-Protokoll

```
                                       BS2000  LINK EDITOR ... PROGRAM MAP
*******************************************************************************************************************
**                                                                                                              **
**                                                                                                              **
**                    PROGRAM: EXAM225                                                 SEGMENT: %ROOT           **
**                    FILE: EXAM225                                                                             **
**                                                                                                              **
**                                                                                                              **
*******************************************************************************************************************
                      DEC                           HEX       DEC
                     _____                         _________________

 SEGMENT NUMBER:       1       LOAD ADDR.:        000000         0
 NO. OF MODULES:       3       SEGMENT LENGTH:    0035B4     13.748
 NO. OF ENTRY PTS.:  124

     MODNAME / DATE         LOAD           MODULE              NO.OF     BIND   / MODULE : CONTAINING OML
     COMNAME / COMMON      ADDRESS         LENGTH      TRAITS  ENTRYS    METHOD / COMMON : CONTAINING SEGMENT
                        HEX        DEC   HEX        DEC
     ________________   _____________    _____________   ______   ______    _____________________________________

     EXAM225 /------    000000       0   000684  1.668   RO            2     EXPLICIT/EAM OBJMOD FILE
     EXAM2257/------    001000   4.096   000208    520                 1     EXPLICIT/EAM OBJMOD FILE
     ITP#AOD#/880816    002000   8.192   0015B4  5.556   RO          121     AUTOLINK/TASKLIB
```

Ablaufprotokoll mit Ablaufverfolgung (trace) und Ausgabedaten
(*COMOPT DEBUG = (PROCTRACE, LABTRACE)
(*RUNOPT TRACE = PROCTRACE)

```
*P: EXAM225  LEVEL:  0    R1=0003C170   R2=FFFEFFF8   R3=00000001   R4=00000004   R14=00000082   R15=6E000000 1)

*P: UP1      LEVEL:  1    R1=0003F32C   R2=FFFEF624   R3=00000001   R4=00000004   R14=6E000184   R15=0000026C 2)

*P: EXAM225  LEVEL:  0    R1=0003D170   R2=FFFEFFF8   R3=00000001   R4=00000004   R14=00000082   R15=6E000000

*P: UP2      LEVEL:  1    R1=0004032C   R2=000005F0   R3=00000001   R4=00000004   R14=6E00018E   R15=00000240

          1          2          3          4          5          6          7          8          9                3)
         10         11         12

*P: UP2      LEVEL:  0    R1=000401A4   R2=00000628   R3=000401A8   R4=0400800F   R14=4E00022C   R15=00000414 4)

         10         11         12                                                                               5)

*P: UP2      LEVEL:  1    R1=0004033E   R2=00000628   R3=000401A8   R4=00000004   R14=6E00023C   R15=00000240 6)

         10         11         12
```

1) TRACE-Ausgabe beim Ansprung der Prozedur EXPL5 vom System aus. Ausgaben beim Prozedur-Trace:

   – Die Kennzeichnung P
   – Der Name der aufgerufenen Prozedur
   – Die Tiefe der Prozedurverschachtelung
   – Die Register R1-R4 und R14-R15. Die Register enthalten:

     R1-R4: Angaben über die ersten 4 Aktualparameter. Die Angaben können Verweise auf die Parameter sein oder der Wert selbst. Genauere Angaben siehe Kapitel 7.
     R14: Adresse der Absprungstelle (Rückkehradresse).
     R15: Adresse der Eingangsstelle der Prozedur.

2) Aufruf der Prozedur UP1. Ausgaben wie bei 1.

3) Ergebnis der Ausgabe in Zeile 21 des Quellprogramms

4) Prozedur-Trace wie 1.; hervorgerufen durch Aufruf von UP2 aus Zeile 26 des Quellprogramms.

5) Ergebnisausgabe von Zeile 30 aus Prozedur UP2.

6) Prozedur-Trace wie 1.; hervorgerufen durch Aufruf von UP2 aus Zeile 33.

7) Ergebnisausgabe von Zeile 30 aus Prozedur UP2.

2.2.6 Dateiorganisation CONSECUTIVE, INDEXED und REGIONAL (1)

Aufgabe:

Es soll die Verwendung unterschiedlicher Organisationsformen und Zugriffsmethoden von Dateien gezeigt werden. Das Programm arbeitet mit den Organisationsformen CONSECUTIVE, INDEXED und REGIONAL(1) auf Dateien mit dem LINK-Namen SEQ, KEY und DIR. Es schreibt zunächst in sequentieller Folge in jede dieser Dateien 6 Datensätze der folgenden Form:

```
ZEILE: n mit 1 ≤ n ≤ 6
```

Anschließend verändert das Programm diese Datensätze.

Voraussetzung:

Alle Dateien haben feste Satzlängen von 27 Zeichen. Die Datei SEQ wird auf die Zugriffsgröße SAM abgebildet. Für KEY wird ISAM und für DIR wird die Zugriffsmethode PAM verwendet. Man beachte in dem Beispiel, welche Attribute jeweils bei den OPEN-Anweisungen verwendet werden. Den Inhalt der Datei DIR kann man nur mit dem Dienstprogramm DPAGE sinnvoll sichtbar machen.

Ablaufprotokoll auf SYSOUT

```
/                   SYSFILE SYSDTA = (SYSCMD)
/ERASE *
/REMARK ..................................................... TRANSLATE
/EXEC $PLI1
%  BLS0500 PROGRAM 'PLI1', VERSION '4.0A' OF '88-10-03' LOADED.
          *COMOPT LIST = (NO,SOURCE)
          *END
```

Quellprozedur (hier fortgelassen; siehe Quellprotokoll weiter unten)

```
..... THERE WAS NO DIAGNOSTIC MESSAGE
..... OBJECTMODUL 'EXAM226'  GENERATED AND WRITTEN TO: *EAM
..... OBJECTMODUL 'EXAM2267' GENERATED AND WRITTEN TO: *EAM
END OF SIEMENS PLI1-COMPILER VERSION 4.0A , TIME USED:     2.01 SEC
/REMARK ..................................................... LINK
/EXEC $TSOSLNK
%  BLS0500 PROGRAM 'TSOSLNK', VERSION '21.0C40' OF '87-09-28' LOADED.
         PROGRAM EXAM226, MAP=NO, FILENAM=EXAM226
         INCLUDE *
         END
%  LNK0500 PROG BOUND
%  LNK0503 PROG FILE WRITTEN: EXAM226
%  LNK0504 NUMBER PAM PAGES USED:      8
/REMARK ..................................................... FILES
/FILE CONSE226,LINK=SEQ,FCBTYPE=SAM,RECFORM=F,RECSIZE=27,BLKSIZE=STD
/FILE INDEX226,LINK=KEY,FCBTYPE=ISAM,BLKSIZE=STD,SPACE=(3,3)
/FILE REG1226, LINK=DIR,FCBTYPE=PAM, BLKSIZE=STD
/REMARK ..................................................... EXECUTE
/EXEC EXAM226
%  BLS0500 PROGRAM 'EXAM226', VERSION ' ' OF '88-10-06' LOADED.
END OF PROGRAM EXAM226 , RTS 4.0A-AAA, TIME USED:     0.99 SEC
/REMARK ..................................................... END
/PRINT CONSE226, ERASE
%  SCP0810 SPOOLOUT OF :B:$STSPL1.CONSE226 ACCEPTED: TSN: 5708, PNAME: PL1
/PRINT INDEX226, ERASE
%  SCP0810 SPOOLOUT OF :B:$STSPL1.INDEX226 ACCEPTED: TSN: 5709, PNAME: PL1
 DPAGE       VER=V21.0A10    CR DATE=840531
ENTER DPAGE-COMMAND
      OPEN REG1226
FILE 'REG1226' OPENED.
      PRINT 2,1-216
      HALT
```

Quellprotokoll auf SYSLST (*COMOPT LIST = SOURCE)

```
 1  EXAM226: PROCEDURE OPTIONS(MAIN);
 2
 3           DCL SEQ        FILE;
 4           DCL KEY        FILE  ENV(INDEXED,     F(27), KEYLOC(1),
 5                                                        KEYLENGTH(6));
 6           DCL DIR        FILE  ENV(REGIONAL(1), F(27));
 7           DCL SYSPRINT   FILE INTERNAL;
 8
 9           DCL KEY3       FIXED BIN(4)  INIT(3);
10           DCL KEY5       FIXED BIN(4)  INIT(5);
11           DCL I          FIXED BIN(4);
12           DCL Z          POINTER;
13           DCL SOURCE     CHAR(27) INIT(' ');
14           DCL RECORD     CHAR(27) BASED(Z);
15           DCL CHANGED    CHAR(27) INIT(' ');
16
17           OPEN FILE(SEQ) RECORD OUTPUT SEQL;
18           OPEN FILE(KEY) RECORD OUTPUT SEQL KEYED;
19           OPEN FILE(DIR) RECORD OUTPUT DIRECT KEYED;
20           DO I=1 TO 6;
21              PUT STRING(SOURCE) EDIT ('LINE: ',I) (X(7),A(6),F(2));
22              WRITE FILE(SEQ) FROM(SOURCE);
23              WRITE FILE(KEY) FROM(SOURCE) KEYFROM(I);         Dateien
24              WRITE FILE(DIR) FROM(SOURCE) KEYFROM(I);         füllen
25              END;
26           CLOSE FILE(KEY);
27           CLOSE FILE(DIR);
28           CLOSE FILE(SEQ);
29
30           OPEN FILE(SEQ) RECORD UPDATE SEQL;
31           READ FILE(SEQ) IGNORE(2);
32           READ FILE(SEQ) INTO(SOURCE);                         CONSECUTIVE-
33                CHANGED = SUBSTR(SOURCE,1,16) || ' - CHANGED'; Datei
34           REWRITE FILE(SEQ) FROM(CHANGED);
35           READ FILE(SEQ) SET(Z);
36                SUBSTR(Z->RECORD,17) = ' - CHANGED';
37           REWRITE FILE(SEQ);
38           CLOSE FILE(SEQ);
39
40           OPEN FILE(KEY) RECORD UPDATE SEQL;
41           READ FILE(KEY) IGNORE(2);
42           READ FILE(KEY) INTO(SOURCE);
43                CHANGED =  SUBSTR(SOURCE,1,16) || ' - CHANGED';
44           REWRITE FILE(KEY) FROM(CHANGED);                     INDEXED-
45           CLOSE FILE(KEY);                                     Datei
46
47           OPEN FILE(KEY) RECORD UPDATE DIRECT KEYED;
48                CHANGED = (6)' ' || ' - CHANGED';
49           REWRITE FILE(KEY) FROM(CHANGED) KEY(KEY5);
50           CLOSE FILE(KEY);
51
52           OPEN FILE(DIR) RECORD UPDATE DIRECT KEYED;
53                CHANGED = ' - CHANGED';                         REGIONAL(1)-
```

```
54            REWRITE FILE(DIR) FROM(CHANGED) KEY(KEY3);          Datei
55            CLOSE FILE(DIR);
56
57            OPEN FILE(DIR) RECORD SEQL UPDATE;
58            READ FILE(DIR) IGNORE(4);
59            READ FILE(DIR) SET(Z);
60                 SUBSTR(Z->RECORD,17) = ' - CHANGED';
61            REWRITE FILE(DIR);
62            CLOSE FILE(DIR);
63            END;
```

Ergebnis des Laufs in der Datei CONSE226

```
      LINE:  1
      LINE:  2
      LINE:  3  - CHANGED
      LINE:  4  - CHANGED
      LINE:  5
      LINE:  6
```

Ergebnis des Laufs in der Datei INDEX226

```
1   LINE:  1
2   LINE:  2
3   LINE:  3  - CHANGED
4   LINE:  4
5   - CHANGED
6   LINE:  6
```

Ergebnis des Laufs in der Datei REG1226 ausgegeben mit $DPAGE (8 mal 27 Bytes)

```
LOGICAL PAGE = 1

KEY — (0001)  (000) BB5AEAB0 01000001 00000000 00000000                                        .!.............

(0001)  (000) 001B0000 00000003 4040D9C5 C7F1F2F2  (010) F6404040 40404040 40404040 40404040   ........  REG12
(0033)  (020) 00000000 00000000 00000000 00000000  (030) 00000000 00000000 00000000 00000000   ................
         NEXT 62 LINES ARE IDENTICAL TO ABOVE LINE.

LOGICAL PAGE = 2

KEY — (0001)  (000) BB5AEAB0 01000002 00000000 00000000                                        .!.............

(0001)  (000) FF000000 00000000 00000000 00000000  (010) 00000000 00000000 00000040 40404040   -...............
(0033)  (020) 4040D3C9 D5C57A40 40F14040 40404040  (030) 40404040 40404040 40404040 40D3C9D5     LINE:  1
(0065)  (040) C57A4040 F2404040 40404040 40404040  (050) 40406040 C3C8C1D5 C7C5C440 40404040   E:  2
(0097)  (060) 40404040 40404040 40404040 40404040  (070) 404040D3 C9D5C57A 4040F440 406040C3
(0129)  (080) C8C1D5C7 C5C44040 40404040 4040D3C9  (090) D5C57A40 40F54040 40404040 40404040   HANGED        L
(0161)  (0A0) 40404040 40404040 40D3C9D5 C57A4040  (0B0) F6404040 40404040 40404040 40FF0000            LINE:
(0193)  (0C0) 00000000 00000000 00000000 00000000  (0D0) 00000000 00000000 FF000000 00000000   ................
```

Die Scheinsätze 0 und 7 erkennt man an ’FF’B4.

3 Übersetzen eines PL/I-Quellprogramms

3.1 Funktionen des Compilers PLI1

Der Compiler PLI1 prüft PL/I-Quellprogramme auf die Einhaltung der syntaktischen und semantischen Regeln und übersetzt sie in Bindemoduln. Die Steuerung für den Übersetzungsvorgang, d.h. die Lokalisierung des Quellprogramms, die Auswahl von auszugebenden Listen, etc. übernehmen Steueranweisungen, die der Übersetzer vor der Bearbeitung des Quellprogramms einliest.

Der Benutzer muß beachten, daß der Compiler PLI1 das PARAM-Kommando nicht auswertet.

Der Compiler PLI1 wird mit dem EXECUTE-Kommando gestartet. Er liest zunächst die Steueranweisungen von SYSDTA ein. Anschließend liest er aus SYSDTA oder aus einer in den Steueranweisungen spezifizierten Datei das Quellprogramm ein. In das Quellprogramm kann man vom Compiler INCLUDE-Texte aus Dateien, GAM-Dateien oder Makrobibliotheken einfügen lassen. Es können mehrere Bibliotheken bzw. GAM-Dateien angegeben werden, wobei sich eine Suchreihenfolge für die einzufügenden Texte festlegen läßt.

Das Quellprogramm kann nach SYSLST - auf Wunsch mit eingefügten INCLUDE-Texten - protokolliert werden. Findet der PL/I-Compiler im Quellprogramm Fehler, so werden sie nach SYSLST gemeldet. Je nach Fehlerart wird bei der Meldung zwischen Warnung, leichtem Fehler und schwerem Fehler unterschieden. Beim Auftreten eines schweren Fehlers wird die Übersetzung vor der Code-Generierung beendet. Auf Wunsch kann man die Übersetzung bereits bei einem leichten Fehler, einer Warnung oder in jedem Fall abbrechen lassen.

Zusätzlich zur Liste des Quellprogramms können folgende Listen ausgegeben werden:

- wirksame Steueranweisungen
- Speicherbelegungsliste (Adreßbuch)
- Referenzliste
- OFFSET-Liste
- INCLUDE-Referenzen
- Objektcode-Protokoll im Assemblerformat
- Statistik über Speicherbelegung

Man kann die Listen und/oder Meldungen zusätzlich in eine spezielle Datei ausgeben lassen (Angabe SAVLST bei den Steueranweisungen LIST bzw. DIAGNOST).
Der erzeugte Objektcode wird in die Bindemoduldatei im EAM-Bereich eingetragen, wenn kein vorzeitiger Abbruch der Übersetzung erfolgte.
Der Übersetzer beendet seinen Lauf mit der Ausgabe einer Endemeldung nach SYSOUT.
Wird der Lauf des Übersetzers mit Fehler beendet, so werden die nachfolgenden Kommandos bis zum STEP- oder LOGOFF-Kommando ignoriert.

Der PLI1-Übersetzer ist z.Zt. noch nicht XS-fähig:
Er kann also nicht im Adressenraum oberhalb von 16 MB-Bytes ablaufen.

3.2 Aufruf des Compilers PLI1

3.2.1 Aufruf des Compilers mit ISP-Kommando

Der Compiler PLI1 wird mit dem EXECUTE-Kommando aufgerufen, d.h. gestartet.

Form des Aufrufes:

```
/{EXECUTE}
 {       }$PLI1[,MONJV=jvname]
 {EXEC   }
```

Hinter dem Wort $PLI1 können noch weitere Parameter angegeben werden, wie sie in der Beschreibung Kommandosprache [2] erläutert sind. Ihre Verwendung ist nicht notwendig.

3.2.2 Aufruf des Compilers mit SDF-Kommando

Der PLI1-Compiler kann alternativ mit dem SDF-Kommando

```
/START-PLI1-COMPILER
```

mit zusätzlichen Operanden gestartet werden. Die Operanden und ihre Bedeutung sind in diesem Kapitel im Abschnitt 3.12 beschrieben.

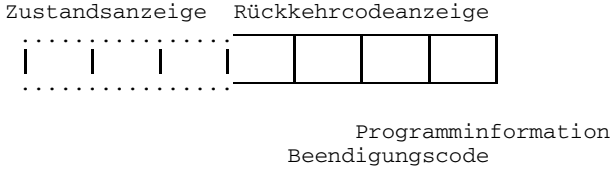
Die Eingabe von SDF-Kommandos und deren Operanden im ungeführten und geführten Dialog ist ausführlich im Manual "Einführung in die Dialog-Schnittstelle SDF" beschrieben.

3.2.3 Überwachung durch Monitorjobvariable

Zur Überwachung des Compilerlaufs kann durch die Angabe 'MONJV=jname' im EXECUTE-Kommando eine Monitorjobvariable (überwachende Jobvariable) spezifiziert werden. Die Zustandsanzeige (Zeichen 1 bis 3) der Jobvariablen wird vom System gesetzt; sie kann folgenden Wert haben:

```
'$R␣' = running
'$T␣' = normal terminated
'$A␣' = abnormal terminated.
```

Die Rückkehrcodeanzeige (Zeichen 4 bis 7) wird beim Ende des Compilerlaufs besetzt und ist wie folgt aufgebaut:



Die Programminformation wird mit dem Wert des höchsten aufgetretenen Fehlergewichts im Format PIC'999' besetzt, dabei sind den Fehlergewichten folgende Werte zugeordnet:

0 Es gab keine Diagnosemeldung
1 Es traten Informationsmeldungen auf
2 Es traten Warnungen auf
3 Es traten Fehler auf
4 Es traten schwere Fehler auf
5 Es trat ein fataler Fehler auf
6 Es trat ein fataler Compilerfehler auf

Der Beendigungscode wird abhängig von der Beendigungsart und vom höchsten aufgetretenen Fehlergewicht besetzt. Dabei kann die Beendigungsart durch Angabe der Steueranweisung *COMOPT OBJECT beeinflußt werden.

Es gibt folgende Zuordnung:

Beendigungsart \ höchstes Fehlergewicht	0	1	2	3	4	5	6	
normal ($T)	0	1	1	1	-	-	-	Beendigungscode
abnormal ($A)	-	-	2[1)]	2[1)]	2	2	3	Beendigungscode

1) abhängig von der Steueranweisung *COMOPT OBJECT

Diese Leistung setzt das Softwareprodukt JV voraus.

3.2.4 Meldungstextdatei

Für die Ausgabe sämtlicher Meldungen benötigt der Compiler die Dateien

```
$TSOS.PLI1.TEXT.D    für Deutsch bzw.
$TSOS.PLI1.TEXT.E    für Englisch
```

Falls diese Dateien unter anderem Namen oder anderer Benutzerkennung vorliegen, muß eine der Dateien durch

```
/FILE Datei,LINK=TEXTLINK
```

verfügbar gemacht werden. Die Steuerung der Sprache (deutsch oder englisch) ist für diese Texte dann bedeutungslos.

3.2.5 Beispiele

Nachfolgend zeigen einige Beispiele des Compiler-Aufrufes unterschiedliche Varianten der Compilersteuerung. Die verwendeten Steueranweisungen sind davon abhängig, ob man im Stapel- oder Dialogbetrieb arbeitet, bzw. wo sich das Quellprogramm und die Steueranweisungen befinden.

Die Beispiele betreffen nur die Umgebung des Compiler-Aufrufes. Häufig verwendet man im BS2000 Prozeduren für die Tätigkeitsfolge Übersetzen - Binden - Ausführen. Ein solches Beispiel findet man im Kapitel 2.
Als Standard für Übersetzungen im Stapelbetrieb gelten die Beispiele 1 bis 3. Es ist ohne Bedeutung, ob die Kommandos über Kartenstapel eingegeben werden oder in einer Datei liegen, die mit dem ENTER-Kommando aktiviert wird.

Beispiel 1

Steueranweisung und Quellprogramm liegen im Kartenstapel. Es soll nur bis einschließlich Semantiklauf übersetzt werden (keine Erzeugung eines Bindemoduls), und es sollen nur Fehler gemeldet werden (Warnungen werden unterdrückt).

```
   .
   .
   .
/EXEC $PLI1
* COMOPT DIAGNOST=E,OBJECT=SE
* END
  Zeilen des Quellprogramms
   .
   .
   .
```

Beispiel 2

Das Quellprogramm liegt in der Datei QUELLE vor. Es werden keine Steueranweisungen benötigt.

```
   .
   .
   .
/SYSFILE SYSDTA=QUELLE
/EXEC $PLI1
/SYSFILE SYSDTA=(SYSCMD)
   .
   .
   .
```

Anmerkung

Da die erste Zeile in SYSDTA (Datei QUELLE) weder mit *COMOPT noch mit *END beginnt, erkennt der Compiler, daß keine Steueranweisungen angegeben sind.

Beispiel 3

Das Quellprogramm enthält INCLUDE-Anweisungen der Form %INCLUDE MAKRO1. Die INCLUDE-Texte liegen in den Bibliotheken BIBA, BIBB und BIBC. Die INCLUDE-Texte sollen in der Liste des Quellprogramms erscheinen.

```
   .
   .
   .
/EXEC $PLI1
* COMOPT COMLIB=(LIBA,LIBB,LIBC)
* COMOPT LIST=EX
* END
  Quellprogramm
   .
   .
   .
```

Anmerkung

Der bei %INCLUDE angegebene Name wird als Elementname interpretiert. BIBA, BIBB, BIBC bezeichnen GAM- (siehe Abschnitt 3.4 und [4]) oder MLU- Dateien (siehe Abschnitt 3.4 und [3]), die in der angegebenen Reihenfolge auf das Element durchsucht werden.
Beispiel 4 bis 7 gelten vor allem für Dialog-Betrieb.

Beispiel 4

Das Quellprogramm liegt in einer Datei A vor. Es werden Steueranweisungen benötigt.

```
   .
   .
   .
/EXEC $PLI1
* COMOPT SOURCE=A,MARGINS=(T(1,72)) ┐ Eingabe
                                    ├ nach
* END                               ┘ Aufforderung
   .
   .
   .
```

Beispiel 5

Das Quellprogramm liegt in der Datei QUELLE, die Steueranweisungen liegen in der Datei STEUER.

```
a)
   .
   .
/SYSFILE SYSDTA=STEUER
/EXEC $PLI1
/SYSFILE SYSDTA=(SYSCMD)
   .
   .
   .

b) .
   .
   .
/SYSFILE SYSDTA=STEUER
/EXEC $PLI1
-Break-Point (Eingabeaufforderung)
/SYSFILE SYSDTA=QUELLE
/RESUME
   .
   .
   .
```

Datei STEUER:

```
*COMOPT SOURCE = QUELLE

*END
```

Datei STEUER:

```
*COMOPT OBJECT = E (5)

*END/
```

Beispiel 6

Das Quellprogramm und die Steueranweisungen werden über die Datenstation eingegeben.

```
   .
   .
   .
/EXEC $PLI1
*COMOPT LIST=MAP,OPTIONS=MAIN                          ⎫
*END                                                   ⎪
BEISPIEL:  PROC;                                       ⎪
           DCL SYSPRINT FILE;                          ⎬ nach Eingabeaufforderung
           PUT SKIP (2) LIST ('--BEISPIEL 6--');       ⎪
           END;                                        ⎪
   .                                                   ⎭
   .
   .
```

Beispiel 7

Das Quellprogramm liegt in der Datei QUELLE, die Steueranweisungen werden an der Datenstation eingegeben.

```
   .
   .
   .
/EXEC $PLI1
*COMOPT LIST=FULLXREF                    ┐
                                         } nach Eingabeaufforderung
*END/                                    ┘

-Break-Point (Eingabeaufforderung)

/SYSFILE SYSDTA=QUELLE                   ┐
/RESUME                                  │
   .                                     } alternativ zu COMOPT SOURCE = QUELLE
   .                                     │
   .                                     ┘
```

3.3 Steuerung des Compilers PLI1

Der Compiler PLI1 benötigt zur Auswahl seiner Funktionen Angaben, die ihm in Form von Steueranweisungen übergeben werden müssen. Um Schreibarbeit zu sparen, sind Voreinstellungen gegeben, die in der nachstehenden Reihenfolge wirksam werden:

1. Voreinstellung von PLI1
   Zuerst werden die Voreinstellungen ausgewertet, die bei jeder Steueranweisung angegeben sind.
2. Voreinstellung des Rechenzentrums
   Ist eine Datei mit dem Namen $TSOS.PLI1.OPTIONS vorhanden, so werden die in ihr enthaltenen Steueranweisungen ausgewertet. Sie überschreiben ggf. die bisher ermittelten Werte.
3. Voreinstellung des Benutzers
   Ist unter dem aktuellen Benutzer-Kennzeichen eine Datei mit dem Namen $Benutzer.PLI1.OPTIONS vorhanden, so werden die in ihr enthaltenen Steueranweisungen ausgewertet. Sie überschreiben ggf. die bisher ermittelten Werte.
4. Einstellung beim Start
   Weitere Steueranweisungen können beim Start des Compilers bzw. beim Start des PL/I-Programms angegeben werden. Sie werden nach dem Start von der Systemdatei SYSDTA eingelesen und ausgewertet. Sie überschreiben die bisher ermittelten Werte.

Dieses Konzept für die Steueranweisungen gilt sowohl beim Compiler (*COMOPT) als auch beim PL/I-Programm (*RUNOPT). Für die Dateien $TSOS.PLI1.OPTIONS und PLI1.OPTIONS gilt folgendes:

- SAM- oder ISAM-Datei mit variabler Satzlänge
- Schlüssellänge bei ISAM-Daten 8 Zeichen
- max. werden 72 Zeichen eines Datensatzes ausgewertet
- COMOPT und RUNOPT dürfen gemischt auftreten
- Das Zeichen * am Anfang kann entfallen
- Die Angabe END kann entfallen

Ansonsten gelten für die Steueranweisungen die gleichen Regeln, wie sie nachfolgend beschrieben sind.

3.3.1 Allgemeine Regeln für Steueranweisungen

Alle Steueranweisungen werden nach dem Start des Übersetzers von SYSDTA gelesen. Sie haben folgendes Format

```
 /EXEC $PLI1
```

```
[[␣...][*]COMOPT{Schlüsselwort={Spezifikation          }},...],...
                {              {(Spezifikationen,...)}}
```

```
 [␣...][*]END[/]
```

Dabei gelten folgende Regeln:

1. Es können mehrere Zeilen mit Steueranweisungen gegeben werden. Die Steueranweisung *END zeigt das Ende der Steueranweisungen an.
2. Für das Format der Steueranweisungen gilt:
   - Die Steueranweisungen dürfen eingerückt werden, d.h. die Zeile kann mit Leerzeichen anfangen.
   - Der * kann entfallen; eine Bedeutungsänderung ist damit nicht verbunden.
   - Mehrere Steueranweisungen werden durch Komma getrennt.
   - Steueranweisungen können in der nächsten Zeile fortgesetzt werden; es muß in der Folgezeile jedoch auch *COMOPT vorangehen. Die in den Zeilen hinter *COMOPT stehenden Angaben werden als eine zusammenhängende Folge interpretiert, das bedeutet:
     Das Zeilenende kann nicht ein Komma ersetzen.
   - Steueranweisungen bestehen aus einem Schlüsselwort, gefolgt von Gleichheitszeichen und einer oder mehreren Spezifikationen.

   Beispiel

   ```
   *COMOPT SOURCE = FILE1, LIST = (SOURCE, XREF),
   *COMOPT DBG = ALL
   *END
   ```

3. Können bei einer Steueranweisung mehr als eine Spezifikation angegeben werden, so sind diese als Liste in Klammern einzuschließen.

Außerdem gilt:

- Eine Steueranweisung kann folgende Formen annehmen:

  ```
  Schlüsselwort = {leer | ? | STD | Spezifikation | Liste}
  ```

  Bedeutung

  leer Der aktuelle Wert wird nicht verändert,

  ? Im Dialogbetrieb erfolgt eine Anfrage an der Datenstation des Benutzers, im Stapelbetrieb hat das Fragezeichen die gleiche Bedeutung wie leer.

  STD Verwendung der unter Voreinstellung angegebenen Spezifikationen (Standard-Einstellung).

- Schlüsselworte und Spezifikationen mit Schlüsselwortcharakter können abgekürzt werden. Die Zeichen, aus denen die Abkürzung besteht, sind bei den Erläuterungen der einzelnen Steueranweisungen unterstrichen.
- Spezifikationen dürfen immer auch als Liste der Form
  (Spezifikation 1, Spezifikation 2, ...)
  auftreten. Ergeben sich dabei Widersprüche, so gilt die letzte Angabe.
- Werden bei der Abarbeitung einer *COMOPT-Zeile Fehler gefunden, gelten weiterhin die vorher ausgewerteten Angaben.
- Die Gültigkeitsdauer der Steueranweisungen bezieht sich nur auf den speziellen Übersetzungslauf, für den sie angegeben sind.
- Wird eine Steueranweisung mehrmals angegeben, so gilt der zuletzt angegebene Wert.

4. Sollen die vom System voreingestellten Spezifikationen benutzt werden, so sind die *COMOPT-Zeilen überflüssig. Die *END-Angabe sollte gemacht werden.

   Anmerkung

   Im Normalfall arbeitet der Compiler zwar auch ohne *END-Angabe fehlerfrei, jedoch sind Fälle denkbar, bei denen es zu Fehlinterpretationen der über SYSDTA eingelesenen Zeichen kommt.

5. Die Form *END/ führt dazu, daß im Anschluß an die Bearbeitung der Steueranweisungen ein Break-Point gesetzt wird, d.h. das Programm (der Compiler) wird unterbrochen, bleibt aber geladen.
   Der Benutzer kann dann z.B. ein SYSFILE-Kommando geben, um SYSDTA umzusteuern. Ein anschließendes RESUME-Kommando beendet die Unterbrechung. Die Verwendung von *END/ in Batch-Prozessen ist analog. Das Programm wird unterbrochen und das nächste Kommando ausgeführt. Durch ein RESUME-Kommando kann an der Unterbrechungsstelle fortgesetzt werden, falls nicht zwischenzeitlich ein anderes Programm geladen wurde.

3.3.2 Fehlerbehandlung bei der Auswertung der Steueranweisungen

Die Feststellung eines Fehlers bei der Syntax- oder Semantikprüfung der Steueranweisungen wird registriert. Erst am Ende der Verarbeitung (bei *END) wird der Fehler gemeldet und mit der fehlerhaften Zeichenfolge nach SYSOUT ausgegeben.

Ein fehlender Stern am Anfang wird nicht gemeldet, die Steuerzeile wird ausgewertet. Eine fehlende *END-Angabe wird nur gemeldet, falls schon eine gültige COMOPT-Angabe verarbeitet worden ist.

Nach Ausgabe einer Fehlermeldung folgt im Dialog auf der Datenstation die fehlerhafte Steueranweisung und die Aufforderung "ENTER CORRECT OPTION OR '␣' OR '@':". Es sind folgende Eingaben möglich:

- Steueranweisungen
  Die fehlerhafte Steueranweisung kann korrigiert und zurückgesendet werden. Außerdem können zusätzliche Steueranweisungen eingegeben werden. Die Eingabe erfolgt ohne *COMOPT.

  Achtung

  Wird bei der Korrektureingabe ein Fehler gemacht, so wird dieser Fall auch gemeldet, jedoch gehen zusätzliche Steueranweisungen, die hinter der falschen Steueranweisung stehen, verloren.

- @ (Kommerzielles "a"-Zeichen)
  Die Bearbeitung der Steueranweisungen wird beendet. Es ist sichergestellt, daß sich im Compiler die Tabelle der gültigen Steueranweisungen in definiertem Zustand befindet.

- Leerzeichen (␣)
  Die fehlerhafte Steueranweisung wird überlesen. Es wird der nächste Fehler ausgegeben.

Nach Ausgabe aller Fehlermeldungen erfolgt im Dialogbetrieb die Anfrage:

```
***CONTINUE (Y/N).
```

Bei der Antwort: Y wird die Übersetzung fortgesetzt. Im Stapel-Betrieb kann man die Fortsetzung der Übersetzung durch gesetzten Prozeß-Wahlschalter 0 erreichen. Wird N eingegeben oder ist Wahlschalter 0 nicht gesetzt, erfolgt Abbruch des Übersetzungsvorganges.

Diese Regeln gelten sinngemäß auch für die Fehlerbehandlung in Objektprogrammen.

3.3.3 Übersicht über die Steueranweisungen des Compilers PLI1

Bild 3-1 zeigt einen Überblick über alle beim Übersetzungsvorgang anwendbaren Steueranweisungen.

In den Abschnitten 3.4 und folgende sind - zu Gruppen zusammengefaßt - die Steueranweisungen einzeln beschrieben.

Die meisten Steueranweisungen und Spezifikationen kann man abkürzen. Bei einigen Spezifikationen kann man eine verneinende Form (z.B. TERMINAL und NOTERMINAL) angeben. Die unter Wirkung stehende Kurzbeschreibung bezieht sich auf die positive Angabe.

Steuer-anweisung	Abk.	Bedeutung	Spezifikation	Abk.	Wirkung	Voreinstellung
COMLIB	CML	Bibliotheks-Angabe	NO (Bibname,...)	N	Keine Bibliothek Angabe einer Bibliothek oder einer Bibliotheks-hierarchie	NO
DEBUG	DBG	Testhilfen Angabe	ALL	ALL	≙(ST,P,L,C,G,R)	
			NO	NO	≙(NST,NP,NL,NC,NG,NR,NBK)	NO
			STMT	ST	Nummer Quellzeile einarbeiten	
			NOSTMT	NST		NST
			PROCTRACE	P	Es werden Befehle eingearbeitet für PROCEDURE-Eingänge	
			NOPROCTRACE	NP		NP
			LABTRACE	L		
			NOLABTRACE	NL		NL
			CALLTRACE	C		
			NOCALLTRACE	NC	Marken (Label)	NC
			GOTOTRACE	G	CALL-Aufrufe	
			NOGOTOTRACE	NG	GOTO-Sprünge	NG
			RETURNTRACE	R	RETURN-Rückkehr	
			NORETURNTRACE	NR		NR
			BREAKPOINT ({[i-]z[.a]},.)	BK	Bei Include-Text i, Zeile z, Anweisung a wird ein Kontrollpunkt eingearbeitet	
			NOBREAKPOINT	NBK		NBK

Bild 3-1 (Teil 1) Steueranweisungen für den Compiler PLI1

Steuer-anweisung	Abk.	Bedeutung	Spezifikation	Abk.	Wirkung	Voreinstellung
DIAGNOST	DIAG	Ausgabe der Diagnosemeldungen	INFORMATION WARNING ERROR SEVERE	I W E S	Meldungen ab angegebener Schwere aufwärts nach SYSLST	W
			TERMINAL NOTERMINAL	T NT	obige Meldungen werden im Dialog zusätzlich auf der Datenstation ausgegeben	NT
			SAVLST NOSAVLST	SV NSV	Ausgabe obiger Meldungen zusätzlich in eine Datei	NSV
FORMAT	FM	Steuerung der Zeilenzahl pro Seite, der Zeilenlänge, Sprache für Meldungen	PRINTER ([m_1][,m_2])	P	Steuerung der Ausgaben auf SYSLST und im Stapelbetrieb auch auf SYSOUT. m_1=Anz. Zeilen/Seite. m_2=Zeilenlänge.	P(64,132)
			TERMINAL (|n_1,||,n_2|)	T	Steuerung der Ausgaben auf Datenstation: n_1 = Anzahl Zeilen, n_2 = Anzahl Zeichen.	T (O,k)[1]
			ENGLISH DEUTSCH	E D	Alle Ausgaben des Compilers erfolgen in englischer oder deutscher Sprache	E

1) k steht für die physikalische Zeilenlänge des aktuellen Ausgabegerätes

Bild 3-1 (Teil 2) Steueranweisungen für den Compiler PLI1

Steuer-anweisung	Abk.	Bedeutung	Spezifikation	Abk.	Wirkung	Voreinstellung
LIST	LST	Listen-steuerung	ALL	ALL	entspricht (EX,NOT,NLC, N,M,FX,I,E,A, NOF,OP,SM)	
			NO	NO	entspricht (NSC,NOT,NLC, NN,NM,NX,NI, NE,NOF,NA,NOP, NSM)	
			SOURCE	SC	Quellprogramm-liste mit %INCLUDE-Sta-tements	SC
			NOSOURCE	NSC		
			EXPAND	EX	Quellprogramm-liste mit ein-gesetzten INCLUDE-Texten	SC
			OUTTEXT [(c)]	OT	Quelltext, Vor-/Nachspann ausgeben und ggf. Rahmen-zeichen c	
			NOOUTTEXT[(c)]	NOT		NOT
			NOLINECNT	NLC	Index ohne führende Nullen oder Lfd.-Nr.	NLC
			LINECNT (EDOR)	LC(ER)	Index gem. EDOR	
			LINECNT (EDT)	LC(ET)	Index gem. EDT	
			LINECNT (PRINT)	LC(PR)	Index gem. PRINT	
			INSOURCE	ISC	Eingabe des Vorüberset-zers proto-kollieren	
			NOINSOURCE	NISC		NISC
			NEST	N	Kennzeichnung von Verschach-telungen	N
			NONEST	NN		
			MAP	M	Ausgabe von Speicherbe-legungslisten	
			NOMAP	NM		NM

Bild 3-1 (Teil 3) Steueranweisungen für den Compiler PLI1

Steuer-anweisung	Abk.	Bedeutung	Spezifikation	Abk.	Wirkung	Voreinstellung
LIST (Fortsetzung)			SHRTXREF	SX	Referenzen und Attribute von Bezeichnern (nur referierte)	NX
			FULLXREF	FX	Referenzen und Attribute von Bezeichnern (alle)	
			NOXREF	NX		
			AGGREGATE NOAGGREGATE	AGG NAGG	Ausgabe der Aggregat-Liste	NAGG
			IREF NOIREF	I NI	INCLUDE-Referenzen	I
			ESD NOESD	E NE	Liste der externen Namen	NE
			OFFSET [({a,e},...)] NOOFFSET	OF NOF	Zuordnungsliste Quellanweisung zu hexadezimalen Programmadressen im Bereich a,e	OF
			ASSM [({a,e},...)] NOASSM	A NA	Ausgabe einer Objektcodeliste im Bereich a,e	NA
			OPTIONS NOOPTIONS	OP NOP	Ausgabe der wirksam gewordenen Steueranweisungen	OP
			SUMMARY NOSUMMARY	SM NSM	Ausgabe einer Programmstatistik	NSM
			SAVLST NOSAVLST	SV NSV	Kopie aller Listings in eine Datei	NSV
			TERMINAL NOTERMINAL	T NT	Kopie aller Listings auf der Datenstation	NT

Bild 3-1 (Teil 4) Steueranweisungen für den Compiler PLI1

Steuer-anweisung	Abk.	Bedeutung	Spezifikation	Abk.	Wirkung	Voreinstellung
MARGINS	MAR	Formaler Aufbau von Quellprogramm und INCLUDE-Texten	CHAR 48 CHAR 60	C48 C60	48- bzw. 60-Zeichensatz wird verwendet	C60
			TEXT (t_1,t_2)	T	Quellprogrammtext beginnt auf Spalte t_1 und endet auf Spalte t_2	T (2,72)
			LINID(l_1,l_2) NOLINID	L NL	Zeilenidentifikation beginnt auf Spalte l_1 und endet auf Spalte l_2	NL
			PAD NOPAD	P NP	Quellzeile mit Leerzeichen auffüllen	P
			ASACNTRL (a) NOASACNTRL	A NA	Vorschubsteuerzeichen für Quellprogrammprotokoll steht in Spalte a	NA
			GAMKEY(n,c)	G	Umwandlung eines Elementnamens zur Bildung eines Gruppenmerkmals (n: Länge, c: Auffüllzeichen)	G (0,0)
			SAVMAC NOSAVMAC	SM NSM	Quellprogramm ist eine SAVMAC-Datei	NSM
MESSAGE	MSG	Steuerung von Meldungen	SYSLST NOSYSLST	S NS	Meldungen im Dialog auch auf SYSLST	NS
MODULE	MOD	Ausgabeziel der Bindemodule	* Bibliothek (*[(Version)]) Bibliothek (Element[(Version)])	*	temp. *EAM-Datei in LMS-Bibliothek *: Modulname wird Elementname	*

Bild 3-1 (Teil 5) Steueranweisungen für den Compiler PLI1

Steuer-anweisung	Abk.	Bedeutung	Spezifikation	Abk.	Wirkung	Voreinstellung
OBJECT	OBJ	Erzeugung des Objektmoduls			Übersetzung wird durchgeführt bis einschließl.:	
			SYNTAX	SY	Syntaxlauf (SY)	O
			SEMATIC	SE	Sematiklauf (SE)	
			CODE	C	Codegenerierung (C)	
			OUT	O	oder Ausgabe in Bindemoduldatei	
			WARNING (n)	W	Abbruch der	
			ERROR(n)	E	Übersetzung, wenn n Fehler der entsprechenden Schwere aufgetreten sind	E(32767)
			ABORT(n)	A	Abbruch der Übersetzung bei n Fehlern und Warnungen	A(500)
			MACRO	MAC	Beenden nach Vorübersetzerlauf	
OPTIMIZE	OPT	Optimierung	NO	NO	entspricht (NE,NOL,NT, NR)	NO
			ALL	ALL	entspricht (E,OL,T,R)	
			ENABLING	E	Bedingungen OFL,UFL,ZDIV voreingestellt	
			NOENABLING	NE	Bedingungen CONV,FOFL,OFL, UFL,ZDIV voreingestellt	NE
			OVERLAP	OL	Überlappung bei Zuweisung möglich	NOL
			NOOVERLAP	NOL		
			TIME	T	Laufzeit optimieren	NT
			NOTIME	NT		
			REORDER	R	Reihenfolge der Anweisungen optimieren	NR
			NOREORDER	NR		

Bild 3-1 (Teil 6) Steueranweisungen für den Compiler PLI1

Steuer-anweisung	Abk.	Bedeutung	Spezifikation	Abk.	Wirkung	Voreinstellung
OPTIONS	OPN	Klassifizierung des Programms	MAIN NOMAIN	M NM	Programm ist Hauptprogramm	NM
			ISO	I	Quellprogramm entspricht PL/I-Standard	
			NOISO	N	Quellprogramm entspricht dem Industriestandard	N
			MACRO NOMACRO	MAC NMAC	Vorübersetzer aufrufen	NMAC
			INTERRUPT NOINTERRUPT	INTR NINTR	Unterbrechungsstellen für ATTENTION-Condition einarbeiten	NINTR
			XS NOXS	X NX	XS-fähige Objekt-Module erzeugen	NX
			BITPTR NOBITPTR	B NB	Absolut-Bitzeiger erzeugen	NB
SOURCE	SRC	Angabe des Eingabemediums für Quellprogramme	*	*	SYSDTA, anschließend an die Steueranweisungen	
			Dateiname		LINK- oder FILE-Name der Datei mit Quellprogramm	*
			Bibliothek (Element)		Element aus GAM-, LMS- oder MLU-Bibliothek	
			Bibliothek (Element [(Version)])		Element aus LMS-Bibliothek	

Bild 3-1 (Teil 7) Steueranweisungen für den Compiler PLI1

Steuer-anweisung	Abk.	Bedeutung	Spezifikation	Abk.	Wirkung	Voreinstellung
STORAGE	STR	Möglichkeit, die Speicherverwaltung beim Übersetzerlauf zu beeinflussen und Optimierungen zu erreichen	AREA ([q_1][,[q_2][,[q_3]]])	A	Für den Standardbereich wird zunächst ein Speicherbereich von q_1 Seiten bereitgestellt. Notwendige Erweiterungen erfolgen in Schritten von q_2 Seiten bis zu einer Maximalgröße von q_3 Seiten. Danach wird ggf. Fehler 30 gemeldet. (Seite = 4KB)	A(16,16,oo)
			STACK ([s_1][,s_2])	S	Der Stapel-Speicher wird in Segementen zu s_1 Seiten belegt. Für Fehler und Endebehandlung im Fall von Speichermangel wird eine Reserve von s_2 angelegt. (Seite = 4KB)	S(16,4)
SYMTEST	SMT	Testhilfe AID	ALL	A	Testhilfe-Inform. erzeugen	M
			NO	N	bzw. nicht erzeugen	
			MAP	M	nur ESD Information "Übersetzungseinheit" erzeugen	

Bild 3-1 (Teil 8) Steueranweisungen für den Compiler PLI1

3.3.4 Compiler-Steuerung durch das Quellprogramm

Die OPTIONS-Angaben in der PROCEDURE-Anweisung erlauben es, daß einige Compilersteuerungen im Quellprogramm angegeben werden können. Die Angabe erfolgt in der Form:

Bezeichner: PROCEDURE OPTIONS (Angabe,...)

Folgende Angaben sind möglich:

Angabe	Wirkung	Beschreibung
MAIN NOMAIN	Diese Prozedur wird als Hauptprozedur übersetzt.	Abschnitt 3.6.3
ISO NOISO	Übersetzung gemäß Norm	Abschnitt 3.6.3
OVERLAP NOOVERLAP REORDER NOREORDER ENABLING NOENABLING	Optimierung	Anschnitt 3.6.4
REENTRANT	Keine; alle PL/I-Prozeduren sind "reentrant"	
XS NOXS	Es sollen XS-fähige Objekte erzeugt werden.	Abschnitt 3.6.3
BITPTR NOBITPTR	Es sollen Bit-Zeiger generiert werden (nur relevant bei OPTIONS (XS))	Abschnitt 3.6.3

Soweit diese Angaben gemacht werden, haben sie Vorrang vor den gleichen Angaben bei *COMOPT OPTIONS = bzw. *COMOPT OPTIMIZE =.

Mit dem OPTIONS-Attribut bei der Vereinbarung von externen ENTRY's steuert man, wie der Compiler den Aufruf von Prozeduren des Benutzerprogramms generieren soll.

Form:

DCL Bezeichner ENTRY [(Parameter-Liste)] OPTIONS (Angabe[,]...);

Beim OPTIONS-Attribut sind folgende Angaben möglich:

Angabe	Wirkung	Beschreibung
{ASSEMBLER ASM}	Der Compiler generiert den Aufruf der damit vereinbarten Prozedur nach Konventionen, die dem Industriestandard entsprechend kompatibel sind.	Abschnitt 7.1.2.3 1)
[INTER]	INTER hat keine zusätzliche Bedeutung.	
PLI1	Sicherung der Fehlerbehandlung bei ASSEMBLER-Programmen nach PLI1-Konventionen.	Abschnitt 7.2.1
COBOL [INTER]	Der Eingang bezieht sich auf ein COBOL-Programm. INTER hat keine zusätzliche Bedeutung.	Abschnitt 7.3
FORTRAN	Der Eingang bezieht sich auf ein FORTRAN-Programm. Unterbrechungen werden von PL/I behandelt.	Abschnitt 7.3
FORTRAN INTER	Der Eingang bezieht sich auf ein FORTRAN-Programm. Unterbrechungen, die nicht von FORTRAN behandelt werden, werden von PL/I behandelt.	Abschnitt 7.3
ILCS	Der Compiler generiert einen Aufruf der vereinbarten Prozedur nach ILCS-Konventionen.	Abschnitt 7.4
LIBRARY	Spezielle Ergänzung für den Eingangsnamen.	Abschnitt 7.1.6
VARIABLE	Der Compiler generiert für den Aufruf der damit vereinbarten Prozedur eine eine ausführliche Parameterliste, die sämtliche verfügbare Informationen über die Aktualparameter und deren Anzahl enthält.	Abschnitt 7.1.2.2 1)
WXTRN	Der Eingang (weak external) wird nicht automatisch vom Binder bearbeitet, sondern, nur wenn explizit das Anbinden gefordert wird oder der Eingang an anderer Stelle ohne WXTRN vereinbart wird. Es liegt in der Verantwortung des Benutzers, daß ein nicht angebundener Eingang nicht angesprungen wird.	[3]TSOSLNK [12] ab BS2000 V 8.0

1) Diese Angabe wird benötigt, wenn die gerufene Prozedur mit unterschiedlicher Parameteranzahl versorgt werden kann (z.B. Assemblerprozeduren). Die genaue Schnittstelle ist im Kapitel 7 beschrieben.

3.3.5 Speicherbedarf des Übersetzers (STORAGE)

Mit der Steueranweisung STORAGE kann man die Belegung des virtuellen Benutzer-Adreßraumes durch den Compiler beeinflussen. Festgelegt werden können Anfangsgröße und Verlängerungen des virtuellen Speichers für Standard-Area und Stapelspeicher (Stack).

Betriebssystemausbau:

Als virtueller Benutzer-Adreßraum ist 1 MB theoretisch ausreichend. Wegen der Beschränkung der Programmgröße auf dann ca. 200 - 1000 Anweisungen, sollte jedoch ein Adreßraum von 2 - 4 MB angestrebt werden (bei 2 MB Speichergröße ca. 1000 - 3000 Anweisungen, 3 MB ca. 2500 - 7000 Anweisungen, 4 MB ca. 4000 - 10000 Anweisungen).

Die Speicherstatistik (LIST = SUMMARY) gibt Hinweise für die optimale Einstellung der Größen von Standard-Area und Stapelspeicher.

```
STORAGE={Spezifikation       }
        {                    }
        {(Spezifikation,...) }
```

Voreinstellung:

```
STORAGE=(AREA (16,16,oo),STACK (16,4))
```

Spezifikationsmöglichkeiten:

AREA ([q_1][,[q_2][,[q_3]]])

Für den Aufbau der Arbeitsdatei während der Übersetzung wird zunächst ein Speicherbereich von q_1 virtuellen Seiten zu je 4KB bereitgestellt. Notwendige Erweiterungen folgen in Schritten von q_2 Seiten bis zu einer Maximalgröße von q_3 Seiten. Ist q_3 nicht angegeben, wird der maximal mögliche Speicher zur Verfügung gestellt. Bei Überschreitung wird Fehler Nr. 30 gemeldet.

Je nach Größe des zulässigen Benutzer-Adreßraums ist für die Standard-Area eine Speicherforderung bis zu maximal 1500 (6 MB) Seiten sinnvoll.

STACK ([s_1][,s_2])

Der Speicher für den Stapelspeicherbedarf des Übesetzers wird in Segmenten zu je s_1 virtuellen Seiten zu je 4KB belegt. Für Fehler- und Endebehandlung bei Speichermangel wird eine Reserve von s_2 Seiten angelegt. Der Maximalwert hängt ab von der Größe des zulässigen Benutzer-Adreßraumes. Im allgemeinen kommt man mit dem Voreinstellungswert 16 aus.

3.4 Steuerung der Quellprogramm-Eingabe

Der PL/I-Compiler sieht verschiedene Möglichkeiten für die Eingabe des Quellprogramms vor. Die Quelle wird entweder über die Systemdatei SYSDTA oder aus einer katalogisierten Datei eingelesen. Das Quellprogramm kann in Form einer SAM- oder ISAM-Datei, aber auch als Element einer makroorganisierten Bibliothek oder als Gruppendatei vorliegen. Die Steuerung des Einlesevorgangs erfolgt über das Kommando SYSFILE und über die Steueranweisung SOURCE. Mit SOURCE wird festgelegt, ob das Quellprogramm einer Datei oder Bibliothek zu entnehmen ist, oder ob aus der Systemdatei SYSDTA gelesen werden soll. Wird aus SYSDTA gelesen, so bedeutet das in der Regel Eingabe über Kartenstapel (Stapelbetrieb) oder von einer Datenstation (Dialog). Außerdem kann mittels SYSFILE-Kommando eine Benutzerdatei die Rolle von SYSDTA übernehmen. Das Einlesen über SYSDTA ist jedoch nicht möglich, wenn das Quellprogramm als Element einer Bibliothek oder als Gruppendatei vorliegt.

Bibliotheken können mit dem Dienstprogramm MLU oder LMS erstellt und verwaltet werden (siehe Dienstprogramme [3] und LMS [13]).

Hinweis

Das Dienstprogramm MLU orientiert sich beim Verarbeiten von Quelltexten an Spalte 1 und 2 zur Erkennung von Steueranweisungen. Steht dort ein Leerzeichen mit folgendem Buchstaben, so wird diese Zeile als Steuerinformation gedeutet. Es wird daher empfohlen, PL/I-Programme entweder ab Spalte 1 oder erst ab Spalte 3 niederzuschreiben, wenn sie für die Hinterlegung in MLU-Bibliotheken vorgesehen sind.

Unter einer Gruppendatei versteht man die Menge der Sätze einer ISAM-Datei, deren Schlüssel mit einer bestimmten Zeichenfolge, dem Namen (Gruppenindex) der Datei beginnen (siehe EDOR-Beschreibung [4]).

Ein weiteres Mittel zur Steuerung der Quellprogramm-Eingabe ist die Übersetzeranweisung %INCLUDE. Sie ist im Quellprogramm anzugeben und spezifiziert einen Text, der dann einer Bibliothek oder Datei entnommen und anstelle der %INCLUDE-Anweisung in das Quellprogramm eingefügt wird. Die Bibliothek wird mit der Steueranweisung COMLIB ausgewählt.

Die bisher genannten Möglichkeiten dienen lediglich der Festlegung der Quelle für den Einlesevorgang. Feinere Steuerungen wie Auswahl des Codes, der Zeilennumerierung und der Stapelauswahl, sowie die Festlegung des zu übersetzenden Elements einer Gruppendatei erfolgen mit der Steueranweisung MARGINS.

Bei der Eingabe des Quellprogramms über eine Datenstation kann die Anforderung weiterer Zeilen durch die Eingabe von

```
*END
```

beendet werden. Diese Anweisung ist nur wirksam, wenn vorher das Ende des Quellprogramms erkannt wurde. Sie muß innerhalb der MARGINS-Angabe liegen.

Eine andere Möglichkeit besteht darin, folgende Eingaben zu machen

```
BREAK-Funktion
/EOF
/R
```

Damit wird ebenfalls die Quelleingabe beendet.

3.4.1 SYSFILE-Kommando (SYSDTA-Umsteuerung)

Mit dem SYSFILE-Kommando kann der Benutzer die Zuordnung der Systemdatei SYSDTA verändern.

Allgemeine Form:

```
                    ⎡ (PRIMARY)              ⎤
                    ⎢ (SYSCMD)               ⎥
                    ⎢ Dateiname              ⎥
/SYSFILE SYSDTA = { ⎢                        }
                    ⎢ Bibliothek (Element)   ⎥
                    ⎢ (CARD)                 ⎥
                    ⎣ (Gerät)                ⎦
```

SYSDTA kann z.B. mit dem Kommando

```
/SYSFILE SYSDTA = Dateiname
```

einer katalogisierten Datei oder mit dem Kommando

```
/SYSFILE SYSDTA = (CARD)
```

dem Kartenleser zugewiesen werden. Näheres entnehme man der Beschreibung der Kommandosprache [2] und dem Abschnitt 6.2.1.

Der Compiler erwartet zunächst in der Systemdatei SYSDTA seine Steueranweisungen. In Abhängigkeit von der Steueranweisung SOURCE wird anschließend eventuell auch das Quellprogramm über SYSDTA eingelesen.

Wurde vor der Übersetzung SYSDTA umgesteuert, ist es in der Regel notwendig, nach der Übersetzung die Zuordnung von SYSDTA zurückzusetzen. Die Rücksetzung erfolgt durch das Kommando

```
/SYSFILE SYSDTA = (SYSCMD) oder
/SYSFILE SYSDTA = (PRIMARY)
```

Dabei beachte man die unterschiedliche Wirkung. Mit SYSCMD wird stets auf den Datenstrom der aktuellen Einspooldatei (Stapelverarbeitung), ENTER-Datei oder DO-Prozedur zurückgeschaltet. Die Wirkung von PRIMARY ist bei Stapelverarbeitung identisch, im Dialog wird damit jedoch stets auf die aktuelle Datenstation umgeschaltet. Das gilt auch dann, wenn das SYSFILE-Kommando innerhalb einer DO-Prozedur steht.

Beispiel 1

Einlesen der Steueranweisungen (und des Quellprogramms) vom Kartenleser

```
/SYSFILE SYSDTA = (CARD)
/EXEC $PLI1
/SYSFILE SYSDTA = (SYSCMD)
```

Beispiel 2

Einlesen der Steueranweisungen (und des Quellprogramms) von der Datei CARD

```
/SYSFILE SYSDTA = CARD
/EXEC $PLI1
/SYSFILE SYSDTA = (SYSCMD)
```

Zu beachten ist, daß im Beispiel 1 mit dem in Klammern eingeschlossenen Begriff CARD der Kartenleser, im Beispiel 2 aber eine Datei mit dem Namen CARD angesprochen wird.

3.4.2 Festlegung der Quelldatei (SOURCE)

Die Steueranweisung SOURCE legt fest, aus welcher Datei oder Bibliothek das Quellprogramm gelesen werden soll.

```
SOURCE={ *                                  }
       { Dateiname                          }
       { Bibliothek (Element [(Version)]) }
```

Voreinstellung: SOURCE = *

*
Im Anschluß an die Bearbeitung der Steueranweisungen wird von der Systemdatei SYSDTA die weitere Eingabe erwartet.

Dateiname
Wenn das Quellprogramm in einer SAM- oder ISAM-Datei gespeichert ist, wird damit der Dateiname oder der Dateikettungsname (LINK-Name) angegeben.

Bibliothek (Element [(Version)])
Ist das Quellprogramm als Element einer Bibliothek abgelegt, so ist der Bibliotheksname, der Name des Quellprogrammes in Klammern und optional eine Versionsangabe ebenfalls in Klammern anzugeben. Die Bibliothek muß nach den Konventionen von LMS, bzw. MLU, bzw. GAM (siehe EDOR) aufgebaut sein.

Namen von LMS-Bibliotheks-Elementen können max. 64 Zeichen lang sein. Sind sie länger, werden sie auf 64 Zeichen gekürzt.
Namen von MLU-Bibliotheks-Elementen können max. 8 Zeichen lang sein. Sind sie länger, so werden sie auf die ersten 5 und die letzten 3 Zeichen gekürzt.
Namen von GAM-Datei-Elementen können max. so lang werden, wie bei der Spezifikation GAMKEY der Steueranweisung MARGINS angegeben wurde. Sind sie länger, so werden sie nach der Kürzungsregel gekürzt, wie sie unter der Spezifikation GAMKEY angegeben ist (siehe Abschnitt 3.4.4).

Die Version wird nur bei LMS-Bibliotheken ausgewertet. Sie kann max. 24 Zeichen enthalten. Ist keine Versionsangabe gemacht, so wird bei LMS-Bibliotheken das Element mit der höchsten Versionsbezeichnung angesprochen.

Dateiname und Bibname werden zuerst als Dateikettungsnamen interpretiert. Führt das nicht zum Erfolg, werden sie als Dateinamen angesehen. Diese Aussage gilt ebenso für die Identifizierung von Bibliotheken, die in %INCLUDE-Anweisungen angesprochen werden.

Anwendungshinweis

Durch die Steueranweisung SOURCE = QUELLE innerhalb einer DO-Prozedur sei z.B. für mehrere Übersetzungen der Name der Quelldatei fest zugeordnet. Man kann nun beliebige Quellprogramme damit übersetzen, wenn man vor jedem Aufruf der DO-Prozedur ein FILE-Kommando mit dem Namen der aktuellen Quelldatei gibt. In diesem Fall:

```
/FILE PROGR1,LINK=QUELLE
```

Beispiel 1

Einlesen eines Quellprogramms aus einer katalogisierten Datei A.

```
   .
   .
/SYSFILE SYSDTA=(SYSCMD)
/EXEC $PLI1
*COMOPT SOURCE=A
*END
/Nächstes Kommando
   .
   .
   .
```

Beispiel 2

Einlesen eines Quellprogramms aus einer Bibliothek BIB1. Name des Quellprogramms sei PROGR5

```
   .
   .
/SYSFILE SYSDTA=(SYSCMD)
/EXEC $PLI1
*COMOPT SOURCE=BIB1(PROGR5)
*END
/Nächstes Kommando
   .
   .
   .
```

Beispiel 3

Einlesen eines Quellprogramms aus einer Datei GAMBIB. Es soll die Satzgruppe übersetzt werden, für die die Sätze mit dem Schlüssel 00003 beginnen. Der für die Datei mit dem FILE-Kommando vereinbarte Schlüssel (KEY) habe eine Länge von 8 Zeichen, von denen die ersten 5 Zeichen als Gruppenmerkmal (hier 00003) interpretiert werden. Damit werden z.B. alle Sätze mit Schlüsseln im Bereich von 00003000 bis 00003999 übersetzt.

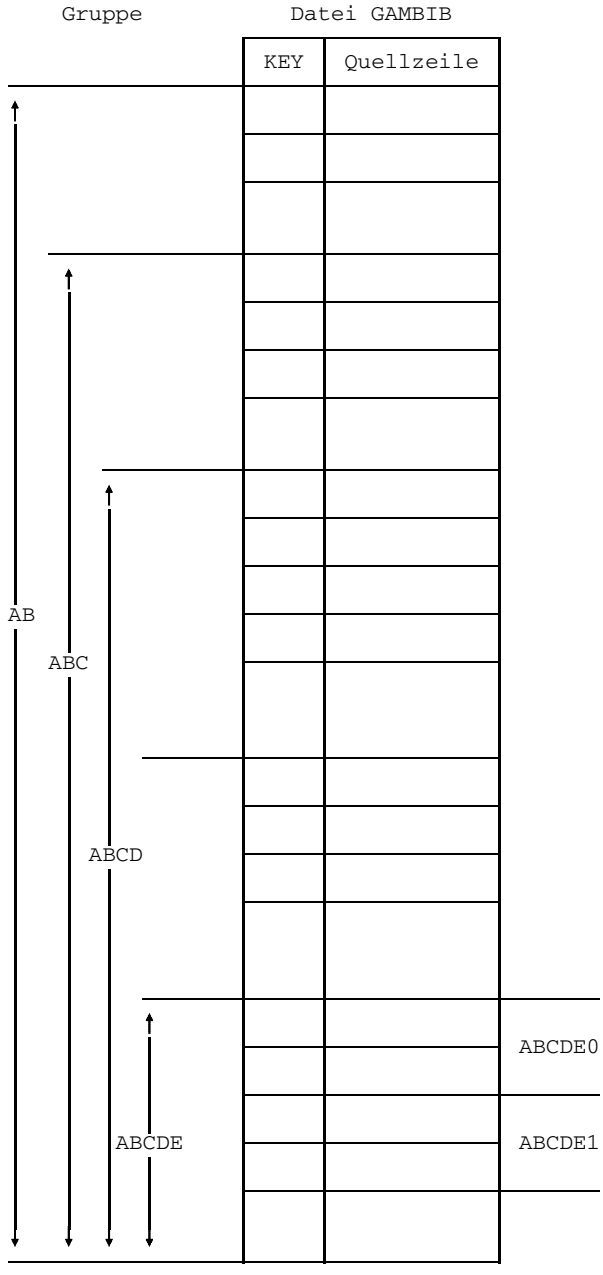
```
   .
   .
/SYSFILE SYSDTA=(SYSCMD)
/EXEC $PLI1
*COMOPT SOURCE=GAMBIB(00003),MARGINS=GAMKEY(5,0)
*END
/Nächstes Kommando
   .
   .
   .
```

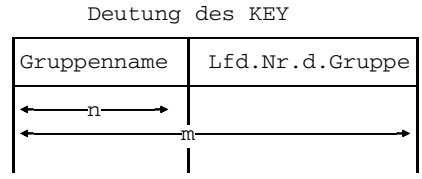
Beispiel 4

Einlesen eines Quellprogramms aus einer Datei GAMBIB. Der LINK-Name sei GRUP. Alle Sätze mit dem Gruppennamen ABCDE0 (siehe Bild) sollen übersetzt werden.

```
   .
   .
/FILE GAMBIB,LINK=GRUP
/SYSFILE SYSDTA=(SYSCMD)
/EXEC $PLI1
*COMOPT SOURCE=GRUP(ABCDE),MARGINS=GAMKEY (6,0)
*END
/Nächstes Kommando
   .
   .
```

Beispiele für die Datei GAMBIB





n: 1.Angabe aus Spezifikation GAMKEY

m: KEYLEN

KEYLEN: Angabe im FILE-Kommando

Macht der Benutzer wie im Beispiel 4 die Angaben SOURCE = GRUP(ABCDE) und MARGINS = GAMKEY (6,0), so werden für die Übersetzung alle Sätze mit dem Gruppennamen ABCDE0 genommen, weil der 5-stellige Elementname mit dem Zeichen "0" zum 6-stelligen Gruppennamen verlängert wird.

Bei MARGINS = GAMKEY (2,0) würden die Sätze mit dem Gruppennamen AB, also alle Sätze der Beispieldatei zur Übersetzung herangezogen.
Bei MARGINS = GAMKEY (4,0) würde aus dem Elementnamen der Gruppenname ABCE gebildet. Diese Gruppe ist in obiger Beispieldatei nicht enthalten. Die Regeln für die Bildung des Gruppennamens sind im Abschnitt 3.4.4 erläutert.

Beispiel 5

Die Steueranweisungen für die Übersetzung eines Quellprogramms stehen in einer Datei OPT. Die Quelldatei soll erst nach Abarbeitung der Steueranweisungen festgelegt werden. Die Quelle stehe in der Datei PROG.

Inhalt der Datei OPT:

```
*COMOPT SOURCE = *
 weitere Steueranweisungen
 *END/
```

Kommandofolge an der Datenstation:

```
   .
   .
   .
/SYSFILE SYSDTA=OPT
/EXEC $PLI1

Es erfolgt ein Break

/SYSFILE SYSDTA=PROG
/RESUME
   .
   .
   .
```

3.4.3 Festlegung INCLUDE-Bibliothek (COMLIB)

Die Steueranweisung COMLIB wird benötigt, wenn in einem PL/I-Quellprogramm %INCLUDE-Anweisungen ohne Bibliotheks-Angaben auftreten; d.h. die Anweisung hat die Form %INCLUDE Name;.
Es müssen durch COMLIB die Namen der Bibliotheken angegeben werden, in denen die bei den %INCLUDE-Anweisungen angegebenen Namen (Elemente) stehen. Die Bibliotheken werden in der Reihenfolge durchsucht, in der sie bei COMLIB angegeben wurden.

```
COMLIB = { NO              }
         { (Bibname,...)   }
```

Voreinstellung: COMLIB=NO

NO — keine Bibliothek; der bei %INCLUDE angegebene Name ist der Name einer SAM- oder ISAM-Datei

Bibname — Der bei %INCLUDE angegebene Name wird als Elementname interpretiert.
Bibname bezeichnet GAM-, LMS- oder MLU-Dateien, die in der angegebenen Reihenfolge auf das Element durchsucht werden. Bibname wird jeweils zunächst als Dateikettungsname interpretiert.
Wird dieser LINK-Name nicht gefunden, so wird eine Datei gleichen Namens gesucht. Es können maximal 8 Bibnamen angegeben werden.

Anmerkung

Aus der Liste der INCLUDE-Referenzen (Steueranweisung LIST = IREF) erfährt der Benutzer, in welcher Bibliothek der jeweilige INCLUDE-Text gefunden wurde.

3.4.4 Quellzeilenformat (MARGINS)

Die Steueranweisung MARGINS beschreibt den formalen Aufbau der Zeilen von Quellprogrammen und INCLUDE-Texten.

```
MARGINS={Spezifikation        }
        {                     }
        {(Spezifikation,...)  }
```

Voreinstellung: MARGINS = (C60,T(2,72),NL,NA,G(0,0),P)

Mögliche Spezifikationen:

CHAR60
CHAR48
: Quellprogramm und INCLUDE-Texte verwenden nur den 60er Zeichensatz, bzw. zusätzlich die Ersatz-Symbole des 48er Zeichensatzes. Die Unterschiede zwischen den Zeichensätzen sind im Abschnitt 3.2 der PL/I-Sprachbeschreibung [1] erläutert.

TEXT(t_1, t_2)
: Der Quellprogrammtext beginnt auf Spalte t_1 und endet auf Spalte t_2, wobei t von $1 \le t_1 \le t_2 \le 255$ sein muß. Diese Angabe hat ebenso Wirkung auf INCLUDE-Texte. Liegt der Bereich t_1 - t_2 außerhalb der Textbereiche, so wird die Zeile ignoriert.

LINID(l_1,l_2)
: LINID bezeichnet in der Quelldatei enthaltene Zeilenidentifikationen. Das Feld für die Zeilenidentifikation beginnt auf Spalte l_1 und endet auf Spalte l_2, wobei auch hier $l \le l_1 \le l_2 \le 255$ sein muß.
Von den Zeichen des Feldes werden max. 8 Stellen (von rechts nach links) ausgewertet. Im Quellprotokoll erscheinen diese Zeichen unverändert. Für Referenzen u. ä. wird intern nur der numerische Anteil (von rechts nach links) ermittelt und gespeichert (im Extremfall 0).
Die Angabe LINID bezieht sich nur auf die durch SOURCE bezeichnete Quelle. Sie hat keine Gültigkeit für die INCLUDE-Texte.

NOLINID
: Es gibt keine Zeilenidentifikation in den Quellzeilen. Es wird der ISAM-Key bzw. eine fortlaufende Numerierung für das Protokoll verwendet.

PAD
NOPAD
: Quellzeilen werden mit Leerzeichen aufgefüllt, so daß sie die durch TEXT (a, e) angegebene Länge haben (e - a + 1).

ASACNTRL (a)
: In Spalte a des Datensatzes befindet sich ein ASA-Vorschubsteuerzeichen, das bei der Protokollierung des Quellprogramms berücksichtigt werden soll (1 bis 255).

NOASACNTRL

Es gibt keine ASA-Vorschubsteuerzeichen in den Datensätzen des Quellprogramms.

GAMKEY(n,c)

Die Spezifikation beschreibt die Aufbereitung eines Elementnamens aus der Steueranweisung SOURCE und der %INCLUDE-Anweisung zur Bildung des Gruppenmerkmals für GAM-Dateien. In GAM-Dateien werden die ersten n Zeichen (n = 1 bis 15) des Satzschlüssels (ISAM-KEY) als Gruppenmerkmal interpretiert.
Wenn z die Länge des Elementnamens aus der Steueranweisung SOURCE oder aus einer %INCLUDE-Anweisung ist, dann gilt bei

n = z: Der Elementname ist das Gruppenmerkmal

n > z: Der Elementname wird durch das Zeichen 'c' auf die Länge n aufgefüllt.

n < z: Die ersten k und die letzten n-k Zeichen des Elementnamens bilden das Gruppenmerkmal.

Dabei ist k = 1 + FLOOR (n/2).

Beispiele zu GAMKEY findet man im Abschnitt 3.4.2.

Voreinstellung: GAMKEY (0,0)

Dies bedeutet, daß der Wert n aus der Angabe bei COMLIB bzw. SOURCE bzw. %INCLUDE ermittelt wird.

SAVMAC

Die Spezifikation dient zum Lesen und Protokollieren einer durch Vorübersetzung entstandenen SAVMAC-Datei. Damit wird gewährleistet, daß in den Übersetzerlisten sowie Laufzeitmeldungen auf die Original-Quellzeilennummern bzw. Original-Includenummern der Vorübersetzer-Eingabe Bezug genommen wird. Diese Angabe ist nur bei getrenntem Vorübersetzer- und Übersetzerlauf nötig, bei unmittelbarem Aufeinanderfolgen von Vorübersetzen und Übersetzen wird diese Angabe intern eingestellt. Diese Spezifikation entspricht

```
MARGINS = (LINID(4,11), TEXT(17,124)).
LIST = NOOUTTEXT
```

mit überschreibender Wirkung.

NOSAVMAC

Das Quellprogramm entspricht nicht den Konventionen einer SAVMAC-Datei (siehe Abschnitt 3.11).

Die Bereiche, die innerhalb eines Satzes für den Text (t_1-t_2), für die Zeilenidentifikation (l_1-l_2) und für das ASA-Vorschubsteuerzeichen vorgesehen sind, dürfen sich nicht überschneiden.

3.5 Steuerung der Protokollausgaben des Compilers PLI1

Der PL/I-Compiler erzeugt während des Übersetzungsvorgangs eine Reihe von Listen mit Informationen über die Struktur des übersetzten Programms und über den Ablauf des Übersetzungsvorgangs.

Die Ausgaben können klassifiziert werden in:

- Meldungen des Übersetzers (M)
- Listen des Quellprogramms (L)
- Diagnosen über das Quellprogramm (D)

Die bei der Übersetzung erzeugten Ausgaben werden je nach Art der Ausgabe nach SYSLST oder SYSOUT geleitet. Der Benutzer hat über verschiedene Steueranweisungen die Möglichkeit, die Ausgaben zusätzlich auf eine zweite Systemdatei zu leiten. Eine Übersicht gibt die folgende Tabelle.

KLASSE	Gegenstand der Ausgabe des Übersetzers	Ausgabedatei		
		Voreinstellung	Zusatzausgabe auf Datei	gesteuert durch
L	Quell- und INCLUDE-Texte sowie sonstige Ausgaben gemäß Steueranweisung LIST	SYSLST	SYSOUT nur im Dialog-betrieb möglich	LIST = TERMINAL (im Stapelbe-trieb wirkungs-los)
D	Quellbezogene Fehlermeldungen und Warnungen	SYSLST	SYSOUT nur im Dialog-betrieb möglich	DIAGNOST = TERMINAL (im Stapelbe-trieb wirkungs-los)
M	Meldungen des Übersetzers	SYSOUT	SYSLST	MESSAGE = SYSLST

Da SYSLST durch das SYSFILE-Kommando auf eine Benutzerdatei geleitet werden kann, besteht eine weitere Steuerungsmöglichkeit. Form des SYSFILE-Kommandos:

```
                    ⎡(PRIMARY)              ⎤
                    ⎢Dateiname              ⎥
/SYSFILE SYSLST=    ⎢                       ⎥
                    ⎢(Dateiname, EXTEND)    ⎥
                    ⎣*DUMMY                 ⎦
```

Einzelheiten entnehme man dem Handbuch Kommandosprache [2] und dem Abschnitt 6.2.1.

3.5.1 Listenauswahl (LIST)

Die Steueranweisung LIST dient der Auswahl von Listen, die im Zusammenhang mit der Übersetzung eines PL/I-Programms entstehen und vom Compiler nach SYSLST ausgegeben werden. Die gleiche Steueranweisung kann auch zur Laufzeit des PL/I-Programms verwendet werden, jedoch sind dann nur die Angaben zu OPTIONS, SUMMARY und TERMINAL von Bedeutung.

```
LIST={Spezifikation          }
     {(Spezifikation,...)    }
```

Voreinstellung: LIST = (SC, NOT, NLC, N, NM, NX, I, NE, OF, NA, OP, NSM, NSV, NT, NISC)

ALL entspricht:
(EX, NOT, NLC, N, M, FX, I, E, NOF, A, OP, SM)

NO entspricht:
(NSC, NOT, NLC, NN, NM, NX, NI, NE, NOF, NA, NOP, NSM)

NO und ALL sind mit anderen Werten kombinierbar. Es gilt dann die Widerspruchsregelung (Gültigkeit der letzten entsprechenden Angabe).

Folgende Spezifikationen können verwendet werden:

EXPAND Quellprogrammliste mit eingesetzten INCLUDE-Texten.

Wird zusätzlich OUTTEXT angegeben, so werden sowohl die %INCLUDE-Anweisung, als auch der INCLUDE-Text protokolliert.

SOURCE
NOSOURCE Ausgabe der Quellprogrammliste mit %INCLUDE-Anweisung

(ohne eingefügte INCLUDE-Texte).

OUTTEXT [(c)]

Es wird nicht nur der durch die Steueranweisung MARGINS = TEXT (a, e) für die Übersetzung ausgewählte Quelltext der Zeile, sondern die gesamte Zeile protokolliert. Es werden außerdem stets alle %-Anweisungen protokolliert.

Für c kann ein einzelnes Zeichen angegeben werden. Dieses Zeichen wird dann zu beiden Seiten des Quelltextes im Protokoll eingefügt und bildet somit einen seitlichen Rahmen für den Quelltext. Bei variabel langen Zeilen kann es sein, daß die rechten Rahmenzeichen nicht untereinander stehen, wenn MARGINS = NOPAD angegeben ist.

NOOUTTEXT [(c)]

Es wird nur der durch MARGINS = TEXT(a, e) ausgegebene Quelltext protokolliert und ggf. der Rahmen gemäß der Angabe c.

NOLINECNT

LINECNT

Diese Angabe bestimmt, in welcher Form in den Listen die Referenz auf eine Quellzeile ausgegeben wird.

Die Quellreferenz hat folgende Form:

(EDOR | EDT | PRINT)

"[Include-] Zeile [:Anweisung]" (siehe Abschnitt 3.8). Die Zeilenangabe wird wie folgt beeinflußt:

NOLINECNT	ohne führende Nullen
EDOR	ohne abschließende Nullen
EDT	ohne abschließende Nullen und ohne führende Nullen und mit Punkt zwischen 4. und 5. Stelle
PRINT	unverändert, wie beim PRINT-Kommando

INSOURCE

Liste der Eingabe für den Vorübersetzer

NOINSOURCE

NEST
NONEST
Kennzeichnung von Verschachtelungen in der Liste des Quellprogramms (PROCEDURE, BEGIN, DO, SELECT, END), Kennzeichnung von Kommentar-Fortsetzungszeilen. Kennzeichnung von zeilenüberschreitenden Zeichenfolge-Konstanten.

MAP
NOMAP
Ausgabe einer Speicherbelegungsliste

SHRTXREF

Ausgabe einer Referenzliste mit Angabe der Bezeichner (nur referierte Bezeichner)

FULLXREF

Referenzliste und Attribute (alle Bezeichner)

NOXREF
Keine Referenzen- und Attributliste

AGGREGATE
NOAGGREGATE
Ausgabe einer Strukturlängenliste (Aggregat-Liste)

IREF
NOIREF
INCLUDE-Referenzliste. Sie gibt eine Zuordnung zwischen den in der Programmliste verwendeten INCLUDE-Nummern und den zugehörigen INCLUDE-Namen (Datei, Bibliothek, Element).

ESD
NOESD
Ausgabe der Liste aller Bezeichner mit dem Attribut EXTERNAL

OFFSET[({a,e},...)]
NOOFFSET

Ausgabe einer Zuordnungsliste zwischen Quellanweisung und hexadezimalem Offset (prozedurrelative Adresse) für die gesamte Übersetzungseinheit bzw. für die durch a, e angegebenen Zeilenbereiche. Ist a = e, so kann e entfallen, jedoch nicht das zugehörige Komma. Die Offsetliste dient der Lokalisierung von Fehlern im Quellprogramm, da normalerweise Fehlermeldungen mit Bezug auf die sedezimale Adresse ausgegeben werden. Man kann auf die Offsetliste verzichten, wenn DEBUG = STMT (siehe Abschnitt 3.6.2) angegeben wurde.

ASSM[({a,e},...)]
NOASSM

Ausgabe einer Liste des vom Übersetzer erzeugten Maschinencodes in Assembler-ähnlicher Form für die gesamte Übersetzungseinheit bzw. für die durch a,e angegebenen Zeilenbereiche. Im letzteren Fall wird die Ausgabe in die durch OFFSET (siehe oben) erzeugte Liste integriert.

OPTIONS
NOOPTIONS

Liste der für diese Übersetzung wirksamen Steueranweisungen

SUMMARY
NOSUMMARY

Ausgabe einer Statistik, die z.B. enthält:

- maximale Anforderung von Stapelspeicher (Stack)
- maximale Belegung des virtuellen Adreßraumes insgesamt
- Anzahl der Speichernachforderungen und Speicherfreigaben

Diese Statistik gibt Hinweise für die Optimierung (siehe Kapitel 8).

SAVLST
NOSAVLST

Kopieren aller oben genannten Listen in eine Datei mit impliziter Namensregelung.
Zunächst wird eine Datei mit dem LINK-Namen SAVLINK bzw. bei Mißerfolg der Dateiname SAVLINK im Katalog des Benutzers gesucht.
Ist keine solche Datei vorhanden, wird eine Datei mit dem Namen SAVLST.PLI1.tsn im Katalog des Benutzers eingerichtet; bei ordnungsgemäßem Syntaxlauf wird diese Datei in SAVLST.PLI1.Modulname umbenannt. Die Datei wird bei einem späteren Übersetzungslauf überschrieben. Diese Regelung gilt auch für die Spezifikation SAVLST der Steueranweisung DIAGNOST.

TERMINAL
NOTERMINAL

Im Dialog zusätzliche Ausgabe aller obengenannten Listen nach SYSOUT (Datenstation). Im Stapelbetrieb wirkungslos.

3.5.2 Auswahl von Diagnosen (DIAGNOST)

Die Steueranweisung DIAGNOST beeinflußt die Ausgabe von Diagnosen über das Quellprogramm. Das sind Fehlermeldungen oder Warnungen die sich auf das übersetzte Programm einschließlich der INCLUDE-Texte beziehen.

```
DIAGNOST={Spezifikation        }
         {                     }
         {(Spezifikation,...)  }
```

Voreinstellung: DIAGNOST = (W, NT, NSV)

Folgende Spezifikationen können angegeben werden:

INFORMATION
WARNING
: Wenn Sie I,W,E oder S angeben, werden Diagnosetexte ab der angegebenen Schwere nach SYSLST ausgegeben.

ERROR
SEVERE
: Die Informationsmeldungen 500 bis 504 werden nur dann ausgegeben, wenn durch *COMOPT OPTMIZE= irgendeine Optimierung verlangt wird.

TERMINAL
NOTERMINAL
: Wenn Sie T oder NT angeben, werden die Diagnosetexte im Dialog zusätzlich auf der Datenstation ausgegeben.
Im Stapelbetrieb wirkungslos.

SAVLST
NOSAVLST
: Wenn Sie SV oder NSV angeben, werden die obigen Meldungen zusätzlich in eine Datei ausgegeben. Siehe hierzu die Anmerkungen zu SAVLST der Steueranweisung LIST.

3.5.3 Zusatzausgabe von Meldungen (MESSAGE)

Die Steueranweisung MESSAGE legt fest, ob Meldungen des PL/I-Compilers zusätzlich nach SYSLST ausgegeben werden sollen.

```
MESSAGE={SYSLST   }
        {         }
        {NOSYSLST }
```

Voreinstellung: MSG = NS

SYSLST
: Alle Meldungen des Compilers werden zusätzlich nach SYSLST ausgegeben.

NOSYSLST
: Meldungen des Compilers werden nur auf SYSOUT ausgegeben.

3.5.4 Ausgabeform (FORMAT)

Die Steueranweisung FORMAT regelt für alle Ausgaben des Compilers auf Drucker, der Datenstation oder in die SAVLST-Datei die Zeilenzahl pro Seite und die Zeilenlänge. Ferner kann angegeben werden, in welcher Sprache (englisch und deutsch) diese Ausgaben erfolgen sollen. Diese Steueranweisung ist gleichbedeutend für PL/I-Übersetzungen und für PL/I-Anwenderprogramme (*RUNOPT FORMAT = ...) anwendbar.

```
FORMAT={Spezifikation        }
       {(Spezifikation,...)  }
```

Voreinstellung: FORMAT = (P(64, 132), T(O,k),E)

k = physikalische Zeilenlänge des jeweiligen Gerätes laut TMODE-Makro.

Mögliche Spezifikationen:

PRINTER([m_1][,m_2])

Diese Steuerung wirkt auf alle Ausgaben, die auf SYSLST und die bei Stapel-Prozessen gegebenfalls zusätzlich auf SYSOUT geleitet werden. Nach der Ausgabe von m_1 Zeilen wird auf eine neue Seite vorgeschoben.
Bei Erreichen einer Zeilenlänge von m_2 Zeichen wird die Zeile aufgebrochen, d.h., es wird eine neue Zeile begonnen.

m_1 = 0 bis 255
m_2 = 1 bis 255

Falls m_1 = 0 ist, wird kein Seitenvorschub ausgeführt; d.h. der Papierfalz wird überdruckt.

TERMINAL([n_1][,n_2])

Die Steuerung wirkt auf alle Ausgaben für die Datenstation. Beim Erreichen von n_1 Zeilen erscheint auf der Datenstation die Meldung:

WEITER (J/N/NDT/NL) bzw.
CONTINUE (Y/N/NDT/NL)

Der Benutzer kann dann z.B. auf dem Bildschirm eine geeignete Cursorposition einstellen und eine der folgenden Eingaben machen:

J oder Y:	Ausgabe fortsetzen
NL	Eine Angabe TERMINAL in der Steueranweisung LIST wird ab sofort wirkungslos. Es handelt sich hier nur um die Listen, die zusätzlich auf der Datenstation ausgegeben werden.

NDT Eine Angabe TERMINAL in der Steueranweisung TRACE bzw. DIAGNOST wird ab sofort wirkungslos. Es wird die Ausgabe von Testhilfen und Diagnosemeldungen auf der Datenstation abgeschaltet.

N wie NL + NDT

Beim Erreichen von n_2 Zeichen pro Zeile wird aufgebrochen.
Falls $n_1 = 0$ (Voreinstellung) ist, erscheint die Anfrage auf der Datenstation nicht.

ENGLISH
DEUTSCH

Sprachauswahl für alle Ausgaben des Compilers (in englischer bzw. deutscher Sprache).
Die Dateien, in denen die Fehlertexte stehen, müssen dem Compiler zur Verfügung stehen(siehe Abschnitt 3.2).

3.6 Steuerung der Objektmodul-Erzeugung

Diese Gruppe von Steueranweisungen bestimmt, ob ein Bindemodul erzeugt wird, wo er abgelegt wird, welche Testhilfen eingebaut werden, in welcher Form Meldungen zur Objektzeit ausgegeben werden usw.

3.6.1 Schalter für Compilerphasen (OBJECT)

Mit der Steueranweisung OBJECT kann man bestimmen, ob ein Bindemodul erzeugt werden soll und wieviele Fehler gegebenenfalls vom Compiler akzeptiert werden dürfen.

```
OBJECT={Spezifikation          }
       {(Spezifikation,...)    }
```

Voreinstellung: OBJECT = (O, E (oo), A(500))

Mögliche Spezifikationen:

MACRO	Es wird nur die Vorübersetzung durchgeführt.
SYNTAX SEMANTIC CODE OUT	Wenn die Übersetzung nicht vorher abgebrochen wird (s.u.), soll die Übersetzung durchgeführt werden bis einschließlich: Syntaxlauf (SY) Sematiklauf (SE) Codegenerierung (C) Codeausgabe in Bindemoduldatei (O)
WARNING[(n)] ERROR[(n)]	Die Übersetzung wird spätestens vor der Codeausgabe abgebrochen, wenn bis dahin die angegebene Anzahl (n) Fehler vom angegebenen oder größeren Gewicht aufgetreten ist. Fehlt die Angabe n, so wird n = 1 angenommen.
ABORT (n)	Die Übersetzung wird sofort abgebrochen, wenn n Warnungen oder Fehler aufgetreten sind. $1 \leq n \leq 32767$

3.6.2 Testhilfen im Objektmodul (DEBUG)

Die Steueranweisung DEBUG wird benötigt, um den Compiler anzuweisen, Testhilfen in den Bindemodul einzufügen.

```
DEBUG={Spezifikation       }
      {(Spezifikation,...)}
```

Voreinstellung: DEBUG = NO

Es gibt folgende Spezifikationen:

ALL — entspricht DEBUG = (ST, P, L, C, G, R)

NO — entspricht DEBUG = (NST, NP, NL, NC, NG, NR, NBK)

STMT
NOSTMT — Erzeugung einer Statement-Tabelle im Objektmodul. Sie ist Voraussetzung dafür, daß sich Meldungen des PL/I-Programms auf die SOURCE-Liste beziehen können.

Es werden Befehle eingearbeitet für die Ablaufverfolgung bei:

PROCTRACE
NOPROCTRACE — PROCEDURE-Eingängen

LABTRACE
NOLABTRACE — Marken (Label)

CALLTRACE
NOCALLTRACE — CALL-Anweisungen

GOTOTRACE
NOGOTOTRACE — GOTO-Anweisungen

RETURNTRACE
NORETURNTRACE — RETURN-Anweisungen

BREAKPOINT({[i-]z[.a]},...)
NOBREAKPOINT

An den angegebenen Stellen werden Kontrollpunkte eingearbeitet.
i: Lfd. Nummer bei INCLUDE-Texten
z: Nummer der Quellzeile
a: Nummer der Anweisung (keine Angabe = 1)

3.6.3 Prozedurstatus (OPTIONS)

Mit der Steueranweisung OPTIONS legt der Benutzer fest, ob die übersetzte Prozedur zur Hauptprozedur erklärt werden soll. Ferner bestimmt diese Steueranweisung, ob das Quellprogramm nach den Regeln des Industriestandards oder nach den Regeln des PL/I-Standards übersetzt wird.

```
OPTIONS={Spezifikation          }
        {                       }
        {(Spezifikation,...)    }
```

Voreinstellung: OPTIONS = (NOMAIN,NOISO,NOINTERRUPT, NOMACRO,NOXS,NOBITPTR)

Spezifikationen können sein:

MAIN NOMAIN	Die zu übersetzende Prozedur ist Hauptprozedur. Genau eine Prozedur eines aus eventuell mehreren Prozeduren bestehenden Programms muß Hauptprozedur sein. Steht in der PROCEDURE-Anweisung des Programms die Angabe OPTIONS (MAIN), so wird die Prozedur in jedem Fall als Hauptprozedur übersetzt, unabhängig von der Angabe in dieser Steueranweisung.
ISO NOISO	Das Quellprogramm wird nach den Regeln der PL/I-Norm (ISO) bzw. nach den Regeln entsprechend dem Industriestandard übersetzt.
INTERRUPT NOINTERRUPT	In das Programm sollen Unterbrechungsstellen für die ATTENTION-Bedingung eingearbeitet werden (siehe Kapitel 5).
MACRO NOMACRO	Es wird ein Vorübersetzerlauf durchgeführt (siehe Abschnitt 3.11 bzw. [1] PL/I-Beschreibung Kapitel 13).
XS	Es sollen XS-fähige Objekte generiert werden (siehe Abschnitt 4.7).
NOXS	Es sollen nicht XS-fähige Objekte generiert werden (siehe Abschnitt 4.7).
BITPTR NOBITPTR	Es sollen Bit-Zeiger generiert werden (nur relevant bei OPTIONS = XS).
REENTRANT:	Diese Angabe wird vom Übersetzer akzeptiert, ist aber bedeutungslos, da beim Übersetzer PLI1 alle Prozeduren diese Eigenschaften erhalten.

Einige dieser Angaben können auch bei der Anweisung PROCEDURE unter OPTIONS (siehe Abschnitt 3.3.4) angegeben werden. Diese haben dann Vorrang vor den durch diese Steueranweisungen gegebenen Werten.

3.6.4 Optimierung (OPTIMIZE)

Die Steueranweisung OPTIMIZE legt fest, welche Optimierungen auf den Objektmodul angewendet werden sollen.

```
OPTIMIZE=⎧Spezifikation       ⎫
         ⎩(Spezifikation,...) ⎭
```

Voreinstellung: OPTIMIZE = (NE,NOL,NT,NR)

Es gibt folgende Spezifikationen:

NO entspricht OPTIMIZE = (NE,NOL,NT,NR)

ALL entspricht OPTIMIZE = (E,OL,T,R)

ENABLING Als Voreinstellung für die berechenbaren Bedingungen sind nur OFL, UFL, ZDIV wirksam.

NOENABLING
Als Voreinstellung für die berechenbaren Bedingungen gilt die System-Vorgabe.

OVERLAP Für die Überlappung wird eine Optimierung durchgeführt: Steht auf der rechten Seite einer Zuweisung eine skalare Zeichenfolge-Variable und kann der Übersetzer nicht feststellen, ob sich Quell- und Ziel-Variable überlappen, so nimmt er an, daß sie sich nicht überlappen und erzeugt einen günstigen Code. In allen diesen Fällen wird eine Warnung ausgegeben.

NOOVERLAP
Für die Überlappung wird keine Optimierung durchgeführt. Dort, wo nicht festgestellt werden kann, ob eine Überlappung von Quell- und Ziel-Variable vorliegt, wird der ungünstige Fall angenommen und ein sicherer Code erzeugt.

TIME Es wird eine Zeitoptimierung durchgeführt.

NOTIME Es wird keine Zeitoptimierung durchgeführt.

REORDER Diese Angabe hat die gleiche Wirkung, als wenn in der ersten PROCEDURE-Anweisung einer externen Prozedur die Angabe REORDER gemacht wird. Eine explizite Angabe in der PROCEDURE-Anweisung überschreibt die Angabe REORDER.

NOREORDER

Diese Angabe entspricht der Angabe ORDER in der PROCEDURE-Anweisung. Ansonsten gelten sinngemäß die obigen Angaben. Bei PL/I ist ORDER die System-Vorgabe.

Die Optimierung gemäß ENABLING läßt sich auch dadurch erreichen, daß die entsprechenden Bedingungen im Quellprogramm explizit unwirksam gesetzt werden (NO...).

Einige dieser Angaben können auch bei der Anweisung PROCEDURE unter OPTIONS (siehe Abschnitt 3.3.4) angegeben werden. Diese haben dann Vorrang vor den durch diese Steueranweisungen gegebenen Werten.

3.6.5 Testhilfe AID (SYMTEST)

Durch diese Steueranweisungen kann angegeben werden, daß der Übersetzer Information ergänzen soll, die das Arbeiten mit der Testhilfe AID ermöglicht.

```
SYMTEST={Spezifikation       }
        {                    }
        {(Spezifikation,...) }
```

Voreinstellung: SYMTEST = MAP

Es gibt folgende Spezifikationen:

ALL	Es wird Information für das Testsystem AID erzeugt.
NO	Es wird keine Information für das Testsystem AID erzeugt.
MAP	Wie NO, zusätzlich werden ESD Informationen vom Typ "Übersetzungseinheit" erzeugt.

Eine ausführliche Beschreibung der Handhabung von AID finden Sie in [18] AID Testen von PLI1-Programmen.

3.6.6 Bindemodul-Ablage (MODULE)

Mit der Steueranweisung "*COMOPT MODULE=Ziel," kann angegeben werden, wohin die erzeugten Bindemodule abgelegt werden sollen. Fehlt diese Angabe, so werden die Bindemodule in die temporäre *EAM-Datei eingetragen. Werden zwei Bindemodule erzeugt (Code-Modul und Static-Modul) so gelten alle Aussagen für beide Bindemodule. Weitere Angaben können dem Abschnitt 3.7 entnommen werden. Voraussetzung für die Ablage der Bindemodule in eine Bibliothek ist, daß die Zugriffsmethode PLAM in das Betriebssystem integriert ist.

```
                 ⎧ *                                          }
                 ⎪                                            ⎫
MODULE=          ⎪                  ⎧*      ⎫                 ⎪
                 ⎨ Bibliothek [(    ⎨       ⎬ [(Version)])] ⎬
                 ⎪                  ⎩Element⎭                 ⎪
                 ⎩                                            ⎭
```

Voreinstellung: MODULE = *

Bibliothek
: Der Name einer Datei, die eine Bibliothek nach den Konventionen vom LMS [13] enthält. Besteht diese Datei nicht, so wird eine entsprechende Bibliothek eingerichtet. Die Angabe einer Dateigenerationsgruppe ist nicht möglich. Der Bibliothekname kann auch ein LINK-Name sein. Gibt es den angegebenen Namen sowohl als LINK- als auch als Datei-Name, so wird der LINK-Name verwendet.

Element
*
: Der Name, den der Bindemodul in der Bibliothek erhalten soll.
Wird '*' angegeben, so wird der erste Eingangs-Name der übersetzten Externen Prozedur übernommen.

  Die Elementnamen können maximal aus 7 Zeichen bestehen: Längere Namen werden auf die ersten 4 und letzten 3 Zeichen gekürzt. Ein Unterstrich (_) wird durch das Zeichen Dollar ($) ersetzt. Für Elementnamen sind folgende Zeichen erlaubt:

  - Buchstaben A bis Z
  - Ziffern 0 bis 9; jedoch nicht als erstes Zeichen
  - Sonderbuchstaben
    - $ Dollar
    - # Nummerzeichen
    - @ kaufmännisches "a"-Zeichen
  - Sonderzeichen
    - \- Bindestrich
    - _ Unterstrich
    - . Punkt

  Sonderzeichen dürfen nicht erstes oder letztes Zeichen sein. Der Bindestrich darf nicht rechts von einem Sonderbuchstaben oder einem Sonderzeichen stehen. Es dürfen zwei gleiche Sonderzeichen nicht hintereinander stehen.

Wird der Eingangsname der PL/I-Prozedur übernommen, so gelten die Regeln für PL/I-Namen.

Entstehen beim Übersetzen zwei Bindemodule, so gilt für den Code-Modul der nachstehend erläuterte Versionsname. Für den Static-Modul wird der Elementname aus dem Namen der Code-Module abgeleitet, indem der Elementname auf 7 Zeichen mit dem Zeichen '@' aufgefüllt und eine Ziffer angehängt wird, die die ursprüngliche Länge des Namens (1 bis 7) angibt.

Version

Versionsname, den das Element enthält. Wird keine Angabe gemacht, so erhält das Element die höchste Version, die in einer LMS-Bibliothek möglich ist (X'FF'):
Dieser wird nicht protokolliert.

Der Versionsname kann aus maximal 24 Zeichen bestehen. Erlaubt sind:

- Buchstaben A bis Z
- Ziffern 0 bis 9
- Sonderzeichen - Bindestrich
  . Punkt

  Sonderzeichen dürfen nicht erstes oder letztes Zeichen sein. Zwei gleiche Sonderzeichen dürfen nicht hintereinander stehen. Ein Bindestrich darf nicht nach einem Punkt stehen.

Ist in der angegebenen Bibliothek bereits ein Element vorhanden, das den gleichen Element- und Versionsnamen hat, so wird es überschrieben. Bei jedem Eintrag eines Elements wird seine Variante (siehe LMS [13]) um 1 hochgezählt. Der Typ des Elements (siehe LMS [13]) ist stets 'R'.

In der *EAM-Datei wird das Element stets unter dem ersten Namen der übersetzten externen Prozedur eingetragen. Werden zwei Elemente gleichen Namens eingetragen ist die Folgewirkung undefiniert.

Es wird nach SYSOUT protokolliert, welcher Modul ausgegeben und wohin er ausgegeben wird.

3.7 Verwalten von Bindemoduln

Bei der Übersetzung einer externen Prozedur entstehen im allgemeinen zwei Bindemodule: der Code-Modul und der Static-Modul. Der Code-Modul wird in jedem Fall erzeugt, der Static-Modul wird nur unter bestimmten Bedingungen erzeugt. Im Kapitel 12 ist erläutert, in welchem Zusammenhang ein Static-Modul entsteht, wann er unerwünscht ist und wie er vermieden werden kann.

Wohin die Bindemodule abgelegt werden, kann durch die Steueranweisung

*COMOPT MODULE = Ziel

angegeben werden (siehe Abschnitt 3.6.6). Dabei stehen zwei Möglichkeiten zur Verfügung, die in den nachfolgenden Abschnitten weiter erläutert sind.

- Temporäre *EAM-Datei
  Diese Datei besteht nur für die Dauer eines Prozesses (Dialog- oder Stapelbetrieb). Nach Abschluß des Prozesses gehen die in ihr abgelegten Bindemodule verloren. Sie müssen ggf. vorher weiterverarbeitet werden (z.B. Binden mit dem Binde-Programm $TSOSLNK) oder in einer Bibliothek abgelegt werden (z.B. durch die Programme LMS, LMR).
- Bibliothek (LMS)
  Dies ist stets eine Bibliothek nach den Konventionen von LMS [13]. Hierbei ist Voraussetzung, daß die Zugriffsmethode PLAM im Betriebssystem integriert ist.

Bild 3-2 Datenfluß für Bindemodule

Achtung

Werden zwei Bindemodule erzeugt, so gehören diese stets zusammen und müssen stets gemeinsam gebunden werden.

3.7.1 Bindemodule nach *EAM-Datei

Wird zur Ablage der Bindemodule keine Angabe gemacht oder wird die Angabe

```
*COMOPT MODULE = *
```

gemacht, so werden die Bindemodule in die temporäre *EAM-Datei eingetragen und können von dort weiter verarbeitet werden.

Die Namen der Bindemodule werden aus dem ersten Eingangsnamen der übersetzten externen Prozedur gebildet (erster Name der ersten PROCEDURE-Anweisung). Weitere Einzelheiten zur Bildung der Namen sind im Abschnitt 4.6 zu finden.

Der Inhalt der *EAM-Datei geht bei Ende des Prozesses verloren; eine Weiterverarbeitung ist also vorher erforderlich. Dies wird im allgemeinen folgendes sein:

– Binden der Module zu einem Programm

  Im einfachsten Fall werden die Bindemodule durch das Bindeprogramm $TSOSLNK [3] bzw. ab BS2000 V 8.0 [12] zu einem Programm gebunden. Dabei werden durch die Steueranweisungen

INCLUDE *	alle Bindemodule
INCLUDE a,*	der Bindemodul a
INCLUDE (a,...),	die Bindemodule a,...

  der *EAM-Datei in den Bindevorgang mit einbezogen (siehe Kapitel 4). Nachstehend ist ein Beispiel gezeigt.

```
/ REMARK " ..................................... UEBERSETZEN"
/         EXEC $PLI1
               *COMOPT SOURCE=BEISPIEL,
               *COMOPT MODULE=*,
               END
/ REMARK " ........................................... BINDEN"
/         EXEC $TSOSLNK
               PROGRAM BEISPIEL,FILENAM=OBJEKT,MAP=N
               INCLUDE *
               END
/ REMARK " ........................................... STARTEN"
          EXEC OBJEKT
```

  Die externe Prozedur in der Datei BEISPIEL wird übersetzt und die erzeugten Bindemodule in die *EAM-Datei eingetragen. Der Binder bindet alle Module der *EAM-Datei zum Lademodul BEISPIEL, der in die Datei OBJEKT abgelegt wird.

– Eintragen der Module in eine Bibliothek

  Will man die Bindemodule für die spätere Verwendung erhalten, so muß man sie mit dem Programm $LMR [3] oder mit dem Programm $LMS [13] in eine Bibliothek eintragen. Nachstehend ist dazu ein Beispiel gegeben.

```
/ REMARK " ..................................... UEBERSETZEN"
/        EXEC $PLI1
              *COMOPT SOURCE=BEISPIEL,
              *COMOPT MODULE=*
              END
/ REMARK " ...................................... EINTRAGEN"
/        EXEC $LMR
              MODLIB  BIBLIOTHEK
              COPYALL SOURCE=*
              END
/ REMARK " ........................................... BINDEN"
/        EXEC $TSOSLNK
              PROGRAM BEISPIEL,FILENAM=OBJEKT,MAP=N
              INCLUDE  *,BIBLIOTHEK
              END
/ REMARK " ........................................... STARTEN"
/        EXEC OBJEKT
```

Die externe Prozedur in der Datei BEISPIEL wird übersetzt und die erzeugten Bindemodule in die *EAM-Datei eingetragen. Danach werden alle Bindemodule der *EAM-Datei in die LMR-Bibliothek BIBLIOTHEK eingetragen. Der Binder bindet alle Bindemodule, die sich in der Bibliothek BIBLIOTHEK befinden zu einem Lademodul.

3.7.2 Bindemodul nach LMS-Bibliothek

Sollen Bindemodule erhalten bleiben, so ist es zweckmäßig, sie vom Übersetzer gleich in eine LMS-Bibliothek eintragen zu lassen. Dies kann durch die Steueranweisung

```
*COMOPT MODULE = Bibliothek (*)
*COMOPT MODULE = Bibliothek (Element)
```

erreicht werden. Voraussetzung ist, daß die Zugriffsmethode PLAM im Betriebssystem integriert ist. Die vollständige Beschreibung ist im Abschnitt 3.6.6 zu finden.

Die Namen, unter denen die Bindemodule abgelegt werden - die Elementnamen - ergeben sich entweder aus dem ersten Eingangsnamen der übersetzten Externen Prozedur, d.i. der erste Name der ersten PROCEDURE-Anweisungen (wenn * angegeben) oder aus dem angegebenen Element-Namen. Weitere Einzelheiten zur Bildung der Elementnamen sind im Abschnitt 3.6.6 zu finden.

Nachstehend ist ein Beispiel gezeigt.

```
/ REMARK " ..................................... UEBERSETZEN"
/         EXEC $PLI1
               *COMOPT SOURCE=BIBLIOTHEK(BEISPIEL),
               *COMOPT MODULE=BIBLIOTHEK(*),
               *END
/ REMARK " ........................................... BINDEN"
/         EXEC $TSOSLNK
               PROGRAM BEISPIEL,FILENAM=OBJEKT,MAP=N
               INCLUDE   (BEISIEL,BEISIEL7),BIBLIOTHEK
               RESOLVE                     ,BIBLIOTHEK
               END
/ REMARK " ........................................... STARTEN"
/         EXEC OBJEKT
```

Die externe Prozedur im Element BEISPIEL (Typ S) der LMS-Bibliothek BIBLIOTHEK wird übersetzt und die Bindemodule in der gleichen Bibliothek abgelegt (Typ R). Die Bindemodule BEISPIEL und BEISPIEL 7 aus der Bibliothek und ggf. weitere Bindemodule (RESOLVE) aus der Bibliothek werden zu einem Lademodul BEISPIEL gebunden, der in der Datei OBJEKT abgelegt wird.

3.7.3 Inhaltsverzeichnis einer Bibliothek

Welche Elemente (Bindemodule) sich in einer LMS-Bibliothek befinden, kann über das Programm $LMS [13] ermittelt werden. Nachstehend ist dazu ein Beispiel gegeben.

```
/ EXEC $LMS
       LIB     BIBLIOTHEK
       PRT     (LST)                        siehe Hinweis
       TOCR     *
       END
```

Hinweis

```
PRT (LST)   nach SYSLST im Stapelbetrieb voreingestellt
PRT (CON)   nach SYSOUT             }
PRT (BOTH)  nach SYSLST und SYSOUT  } nur im Dialogbetrieb
```

3.8 Listen des Übersetzers (LIST =)

Alle Listen des Übersetzers werden nach SYSLST ausgegeben. Jede Seite enthält eine Kopfzeile mit folgendem Inhalt:

- Name des Übersetzers mit Versions-Nummer
- Name der Datei, die die Quell-Prozedur enthält (wenn SOURCE = Datei) oder "SYSDTA" (bei SOURCE = *)
- Datum und Uhrzeit der Übersetzung
- Lfd. Nummer der Seite.

Welche der Listen ausgegeben werden sollen, wird durch die Steueranweisungen

*COMOPT LIST = Ausgabe

bestimmt. Einzelheiten dazu sind im Kapitel 3 zu finden.

Einige der nachfolgend erläuterten Listen enthalten eine Referenz auf eine Quellzeile. Diese Angabe bezieht sich immer auf die Zeilenreferenz, die im Quellprotokoll gegeben ist (siehe Abschnitt 3.8.3). Die Zeilenreferenzen sind in allen Listen gleichartig dargestellt und haben folgenden allgemeinen Aufbau

```
[Include-] Zeile [:Anweisung]
```

Zeile: Die Zeilennummer ist stets vorhanden. Sie wird wie folgt gebildet:

- Quelle in ISAM-Datei: Index aus der Datei
- Quelle in SAM-Datei: Laufende Nummer
- COMOPT MARGINS=LINID(p,n): Index mit n Zeichen als Position p (siehe Abschnitt 3.4.4)

Werden Indizes verwendet, die nicht numerisch sind, so wird nur der numerische Teil verwendet. Dies kann zu mehrdeutigen Zeilennummern führen; es wird eine Warnung ausgegeben.

Include: Zeilen, die auf Grund einer %INCLUDE-Anweisung eingefügt werden, wird eine laufende Include-Nummer vorangestellt. Die laufende Include-Nummer ist in der Liste der Include-Nummern (siehe Abschnitt 3.8.5) aufgeführt.

Anweisung: Anweisung gibt die laufende Nummer der Anweisung innerhalb einer Zeile an. Der Wert ":1" wird unterdrückt. In einigen Listen wird die Anweisungs-Nummer nicht ausgegeben.

Die Zeilen-Referenz wird bei einigen Listen spaltengerecht untereinander ausgegeben, in anderen Listen wird sie gepackt, d.h. ohne Leerzeichen ausgegeben.

3.8.1 Optionen (OPTIONS)

In dieser Liste sind alle die Optionen-Werte angegeben, die für den Übersetzungslauf eingestellt sind. Die Ausgabe dieser Liste kann wie folgt beeinflußt werden:

```
*COMOPT LIST=OPTIONS              Steueranweisungen auflisten
           =NOOPTIONS             keine Auflistung
```

```
COMPILER-OPTIONS USED

  STORAGE = (STACK(16,4),AREA(16,16,975))
  LIST    = (NOESD,NOTERMINAL,NOSUMMARY,OPTIONS,SAVLST,NOMAP,
             NEST,IREF,NOXREF,SOURCE,NOINSOURCE,NOAGGREGATE,OFFSET,NOASSM,
             NOOUTTEXT,NOLINECNT)
  FORMAT  = (TERMINAL(0,80),PRINTER(64,72),ENGLISH)
  MESSAGE = NOSYSLST
  SOURCE  = EXAMP21
  MARGINS = (TEXT(2,72),PAD,NOLINID,NOASACNTRL,GAMKEY(0,0),CHAR60,
             NOSAVMAC)
  DIAGNOST= (NOTERMINAL,NOSAVLST,WARNING)
  COMLIB  = NO
  OBJECT  = (ERROR(32767),ABORT(500),OUT)
  OPTIONS = (NOISO,NOMAIN,NOINTERRUPT,NOMACRO,NOXS,NOBITPTR)
  OPTIMIZE= (NOTIME,NOOVERLAP,NOENABLING,NOREORDER)
  DEBUG   = (NOSTMT,NOPROCTRACE,NOLABTRACE,NOCALLTRACE,NOGOTOTRACE,
             NORETURNTRACE,NOBREAKPOINT)
  SYMTEST = MAP
  MODULE  = *
```

Bild 3-3 Optionen-Liste des Übersetzers (Ausschnitt eines Beispiels)

3.8.2 Vorübersetzer (INSOURCE)

Es werden die Texte ausgegeben, die vom Vorübersetzer als Eingabedaten verwendet wurden. Es sind dies:

- eine Liste für den Quelltext gemäß SOURCE =
- je eine Liste für jeden eingefügten Include-Text.

Jede dieser Listen beginnt auf einer neuen Seite. Die Ausgabe der Listen kann wie folgt gesteuert werden:

```
*COMOPT LIST = INSOURCE              Listen ausgeben
             = NOINSOURCE            keine Liste
```

Der Aufbau der Listen ist der gleiche, wie für das Quellprotokoll (siehe Abschnitt 3.8.3), jedoch entfällt die Kennzeichnung für Kommentar-Fortsetzung und Verschachtelungsstufe.

3.8.3 Quellprotokoll (SOURCE, EXPAND)

Das Quellprotokoll enthält den Quelltext, der übersetzt wird, eine Zeilen-Numerierung und ggf. noch weitere Informationen. Die Ausgabe des Quellprotokolls kann wie folgt beeinflußt werden:

```
*COMOPT LIST = EXPAND               Quellprotokoll mit Include-Texten
             = SOURCE               Quellprotokoll ohne Include-Texte
             = NOSOURCE             kein Quellprotokoll

*COMOPT LIST = NEST                 Verschachtelung wird gezeigt
             = NONEST               keine Angabe zur Blockverschachtelung

*COMOPT LIST = NOLINECNT            Zeilenindex ohne führende Nullen.
             = LINECNT (EDOR)       Zeilenindex gemäß EDOR
             = LINECNT (EDT)        Zeilenindex gemäß EDT
             = LINECNT (PRINT)      Zeilenindex gemäß PRINT

*COMOPT MARGINS = TEXT  (a, e)      Quelltext
        MARGINS = LINID (a, e)      Index
        MARGINS = PAD               Quelltext mit Leerzeichen auffüllen

*COMOPT LIST = OUTTEXT (c)          Quelltext-Vor- und Nachspann +
                                    ggf. Rahmung
             = NOOUTTEXT (c)        ggf. Rahmung
```

Das Quellprotokoll kann maximal aus folgenden Spalten bestehen:

Zeilen-Referenz	Include-Nummer Index (bzw. lfd. Nr.)
Kennzeichnung	* (Kommentar-Fortsetzung) Verschachtelungsstufe (oder leer)
Quellzeile	Quelltext-Vorspann (optional) Rahmenzeichen links (optional) Quelltext Rahmenzeichen rechts (optional) Quelltext-Nachspann (optional)

Für die einzelnen Spalten gilt folgendes:

- Zeilen-Referenz
  Die Darstellung der Zeilen-Numerierung ist abhängig von der Steueranweisung LIST = LINECNT und vom Eingabe-Medium bzw. von der Steueranweisung MARGINS = LINID (a, e). Wird LINID angegeben, so gilt für die folgenden Betrachtungen das gleiche wie für ISAM-Dateien.

NOLINECNT	(Voreinstellung):
ISAM-Datei	Include-Nummer Index ohne führende Nullen

sonst	Include-Nummer laufende Nummer

LINECNT:

ISAM-Datei	Include-Nummer Index (gemäß EDOR/EDT/PRINT)
sonst	Include-Nummer laufende Nummer (gemäß EDOR/EDT/PRINT)

Die Zeilen-Referenz hat folgenden Aufbau:

[Include-] Zeile

– Include
  Die Include-Nummer wird nur angegeben, wenn die Quellzeile aus einem Include-Text stammt, andernfalls ist sie leer. Die Include-Nummer ist eine laufende Nummer, die den verwendeten Include-Texten zugeordnet wird (siehe auch Abschnitt 3.8.5).

– Zeile
  Dies ist der Index aus der ISAM-Datei oder eine laufende Nummer bei SAM-Datei bzw. gemäß der Steueranweisung MARGINS = LINID (a,b). Dabei werden jedoch maximal 8 Ziffernstellen verwendet. Andere als Ziffernstellen werden nicht übernommen. Die Darstellung des Index hängt von der Steueranweisung LIST = LINECNT (a) wie folgt ab:

NOLINECNT	ohne führende Nullen
LINECNT (EDOR)	ohne abschließende Nullen
LINECNT (EDT)	Punkt zwischen 4. und 5. Stelle ohne führende und abschließende Nullen
LINECNT (PRINT)	mit allen Nullen (wie bei PRINT-Kommando)

- Kennzeichnung
  Zwischen der Zeilen-Numerierung und den Quellzeilen gibt es zwei Kennzeichnungen, die sich auf die Struktur der Quellzeile beziehen:

*	In dieser Zeile wird ein Kommentar von der Vorzeile fortgesetzt.
Stufe	Eine Zahl zeigt die statische Verschachtelung von Blöcken und DO-Gruppen an. Bei der obersten Stufe (0) wird keine Nummer angegeben. Für die erste Verschachtelungs-Stufe wird 1 angegeben, usw. Die Zeile, die auf die einleitende Anweisung PROCEDURE, BEGIN, DO oder SELECT folgt, erhält eine um 1 erhöhte Stufe. Durch die korrespondierende Anweisung END wird die Stufe um 1 zurückgesetzt, so daß in der Zeile nach der Anweisung END die erniedrigte Stufe steht.

Beide Kennzeichnungen können durch *COMOPT LIST=NONEST unterdrückt werden.

```
          PROCEDURE  OPTIONS(MAIN);

          DCL X         DIM(10,10);
          DCL (I,J);

          B: PROCEDURE;
1            PUT SKIP(2);
1            DO I = 10 TO 10;
2               DO J = 1 TO 10 BY 1;
3                  X(I,J) = I*10 + J;
3                  END;
2               BEGIN;
3                  DO I = 1 TO 10;
4                     PUT SKIP;
4                     DO J= 1 TO 10;
5                        PUT EDIT (X(I,J)) (F(5));
5                        END;
4                     END;
3                  SELECT (I);
4                     WHEN (4) PUT SKIP LIST ('4');
4                     OTHER;
4                     END;
3                  /* EIN KOMMENTAR
3                     UEBER 2 ZEILEN */
3                   END;
2              END;
1           END;
         END;
```

Bild 3-4 Beispiel für das Protokoll einer Quellprozedur mit Verschachtelungsangabe

- Quellzeile
  Aus der Quellzeile, wie sie aus dem Eingabe-Medium zur Verfügung steht, kann durch die Steueranweisung MARGINS = TEXT (a,e) der Quelltext von Spalte a bis Spalte e ausgeblendet werden; nur dieser wird übersetzt. Durch die Steueranweisung OUTTEXT kann erreicht werden, daß auch der Text vor der Spalte a (Vorspann) und der Text nach der Spalte e (Nachspann) mit protokolliert wird. Außerdem kann erreicht werden, daß in jeder Zeile vor und hinter dem Quelltext das Zeichen (c) eingefügt wird, das bei der Steueranweisung OUTTEXT (c) angegeben ist. Die Quellzeile kann aus folgenden Spalten bestehen:

  – Vorspann, Nachspann
    Ein Vorspann entsteht auf Grund der Steueranweisung MARGINS=TEXT(a,e), wenn a größer 1 ist.
    Der Nachspann entsteht, wenn es hinter der Spalte e noch Zeichen gibt. Beide

werden nur dann ausgegeben, wenn die Steueranweisung LIST = OUTTEXT angegeben wurde.

Nach einem Vorübersetzer-Lauf stehen Vor- und Nachspann nicht mehr zur Verfügung und erscheinen dann nicht im Quell-Protokoll.

– Rahmenzeichen
Das Rahmenzeichen c wird vor und hinter den Quelltext gesetzt, wenn es in der Steueranweisung LIST = OUTTEXT (c) bzw. LIST = NOOUTTEXT (c) angegeben wurde. So wird z.B. durch OUTTEXT (") erkennbar, wieviel Leerzeichen am Anfang und Ende des Quelltextes vorhanden sind.

Bei variabler Länge des Quelltextes braucht das rechte Rahmenzeichen nicht immer an der gleichen Stelle zu stehen. Soll es immer an der gleichen Stelle stehen, so kann dies durch die zusätzliche Steueranweisung MARGINS = PAD erreicht werden. So wird z.B. durch MARGINS = PAD, LIST=OUTTEXT (|) erreicht, daß der Quelltext beidseitig in senkrechte Striche eingerahmt wird.

– Quelltext
Dies ist der Text, der die PL/I-Prozedur bildet und übersetzt wird. Leerzeichen am Anfang und am Ende des Textes können durch die Rahmenzeichen (siehe oben) sichtbar gemacht werden.

3.8.4 Externe Namen (ESD)

Diese Liste enthält alle externen Namen der Prozedur, die in der Quelle vereinbart wurden oder die vom Übersetzer generiert wurden. Sie enthält Informationen über die externen (bindefähigen) Namen der Prozedur. Diese dienen zur Verbindung der Prozedur mit anderen Prozeduren, Laufzeitprozeduren und fremden Prozeduren. Die ausgegebenen ESD-Informationen entsprechen den ESD-Sätzen, die beim Übersetzungsvorgang erzeugt werden.
CSECT/COMMON-Sections werden wie folgt angelegt:

```
Typ/Level-1-Größe          Anzahl der vergebenen ESID-Nr.
EXTERNAL VARIABLE                    1
EXTERNAL VARIABLE INIT               2
FILE EXTERNAL CONSTANT               2
ENTRY EXTERNAL CONSTANT              1
```

Die Angabe der Liste kann wie folgt beeinflußt werden:

```
*COMOPT LIST = ESD                  Listen ausgeben
             = NOESD                keine Listen
```

Es wird eine Liste für den Code-Modul und, soweit generiert, eine Liste für den Static-Modul ausgegeben.

```
LISTE DER EXTERNEN NAMEN IM CODE-MODUL

  NAME      TYP     MA   ADR    LNG/ESID
BH$384     CSECT   04  000000  000238
BH$384@6   EXTERN  00  000000  404040
P$3#00##   EXTERN  00  000000  404040
EXTINIT    COMMON  00  000000  000004
EXTSTAT    COMMON  00  000000  000004
EINGANG    VKONST  00  000000  404040
P$START#   ENTRY   00  000074  000001
EINGNG1    ENTRY   00  000004  000001
EINGNG2    ENTRY   00  000124  000001
```

```
LISTE DER EXTERNEN NAMEN IM STATIC-MODUL

  NAME      TYP     MA   ADR    LNG/ESID
BH$384@6   CSECT   00  000000  000008
BH$384     EXTERN  00  000000  404040
EXTINIT    COMMON  00  000000  000008
EXTNIT     CSECT   00  000008  000008
EXTSTAT    COMMON  00  000000  000008
```

Bild 3-5 Liste der externen Namen im Code-Modul und im Static-Modul Beispiel

Die Spalten haben folgende Bedeutung:

- NAME
  Externer Name, der in der Prozedur vereinbart oder der vom Übersetzer generiert wurde. Zu lange Namen sind auf die ersten 4 und die letzten 3 Zeichen bzw. auf die ersten 8 Zeichen gekürzt (siehe Abschnitt 4.6).
- TYP
  Die Art des externen Namens:

CSECT	Programmabschnitt
ENTRY	Verknüpfungsadresse
EXTERN	Externe Verknüpfungsadresse
COMMON	Gemeinsamer Hilfsabschnitt
VKONST	Externe Verknüpfungsadresse
CU	Übersetzungseinheit (compilation unit) nur wenn *COMOPT SYMTEST ≙ NO

- MA

Merkmal:	00	keine Angabe
	04	schreibgeschützte CSECT

- ADR

  Adresse relativ zum Modul. Nur bei Eingängen auf Grund der Anweisung ENTRY ungleich 0.

- LNG/ESID

bei CSECT, COMMON	Länge in Bytes
bei ENTRY	ESID-Nummer der CSECT
bei CU	1. Byte: Anzahl Module gesamt 2. Byte: leer 3. Byte: lfd. Nummer des Moduls
Rest	Leerzeichen (X'404040')

3.8.5 Include-Texte (IREF)

In dieser Liste ist angegeben, welche Include-Texte in der Prozedur aufgerufen und welcher Datei sie entnommen wurden. Die Angabe der Liste kann wie folgt beeinflußt werden:

```
*COMOPT LIST = IREF                  Liste ausgeben
             = NOIREF                Keine Liste
```

```
                I N C L U D E - R E F E R E N C E S

SOURCE-REF.   # LEVEL FILENAME                    MEMBERNAME          VERSION

      5       1   1   $PLI1.SRC.INCL             ALLE$EXT            A01
      6       2   1   $PLI1.SRC.INCL             BLOCK000            002
      7       3   1   $PLI1.SRC.INCL             LABEL000            015
      8       4   1   $PLI1.SRC.INCL             SYMBOL00            B01
      9       5   1   $PLI1.SRC.INCL             REFERNCE            C00
     10       6   1   $PLI1.SRC.INCL             TOKEN000            C00
     11       7   1   $PLI1.SRC.INCL             ARRAY000            003
     12       8   1   $PLI1.SRC.INCL             CROSSNCE            003
     13       9   1   $PLI1.SRC.INCL             NODES000            001
```

Bild 3-6 Liste der Include-Texte (Ausschnitt eines Beispiels)

Die Spalten der Liste haben folgende Bedeutung:

- QUELL-BEZUG (SOURCE-REF.)
  Hier ist die Nummer der Quellzeile und gegebenenfalls die Nummer der Zeile des Include-Textes angegeben, in der die INCLUDE-Anweisung steht.
- #
  Eine laufende Numerierung der Include-Texte in der Reihenfolge, in der sie in die Prozedur eingefügt wurden.
- STUFE (LEVEL)
  Die Verschachtelungsstufe, in der die INCLUDE-Anweisung stand. Eine 1 bedeutet, daß die Anweisung in der Quell-Prozedur steht; alle Werte größer 1 bedeuten, daß die Anweisung in einem Include-Text steht.
- FILENAME
  Name der Datei, der der Include-Text entnommen wurde.
- MEMBERNAME
  Name des Include-Textes, wie er beim Aufruf in der INCLUDE-Anweisung ausgegeben wurde. Bei zu langem Namen werden nur die ersten 4 und die letzten 3 Zeichen angegeben. Kürzere Namen werden mit Nullen auf 7 Zeichen aufgefüllt.
- VERSION
  Versionsbezeichnung des Bibliothekelementes, das den Include-Text enthält. Die Versionsbezeichnung ist max. 3 Zeichen lang.

3.8.6 Querverweis-Liste

Die Querverweisliste enthält für jeden Bezeichner (Namen) der externen Prozedur den vollständigen Attributsatz und eine Auflistung aller Zeilennummern, in denen der Bezeichner explizit oder implizit verwendet wird. Die Liste besteht aus zwei Teilen. Im ersten Fall sind alle Bezeichner aufgelistet, auf die in der Prozedur Bezug genommen wird. Der zweite Teil enthält alle Bezeichner die zwar vereinbart wurden, auf die jedoch nicht Bezug genommen wird. Beide Teile sind alphabetisch nach dem Bezeichner sortiert.
Die Angabe der Liste kann wie folgt beeinflußt werden:

```
*COMOPT LIST = FULLXREF              referierte und nicht referierte
                                     Bezeichner
             = SHRTXREF              nur referierte Bezeichner
             = NOXREF                keine Querverweis-Liste
```

Soweit es für die Angaben in der Liste eine Entsprechung in PL/I gibt, ist möglichst die Schreibweise der Sprache übernommen worden. Darüber hinaus gibt es folgende spezielle Schreibweisen:

@ Dieses Zeichen steht für eine Angabe (Bezug oder Ausdruck), die nicht ermittelt werden konnte.

< > Eine in spitzen Klammern stehende Angabe (Bezug) wurde wegen zu großer Länge rechts gekürzt.

```
                Q U E R V E R W E I S - L I S T E       -   REFERIERTE BEZEICHNER   -

BEZEICHNER      DIMENSION DATENTYP                          SPEICHER            BEZUEGE


$PRINT_CPDCL              BIT (1)                 ALIG      MEM-2 <ALLE_EXT_ST> DCL 1-1400 33150000

$ROOT                     POINTER                 ALIG      MEM-2 <ALLE_EXT_ST> DCL 1-8800 13150000

$SHRTXREF                 BIT (1)                 ALIG      MEM-2 <ALLE_EXT_ST> DCL 1-1100 13220000

ADDR                      BUILTIN                                               DCL 11250000 13040000

ALGOL                     BIT (1)                 UNAL      MEM-8 (SYMBOL)      DCL INIT 4-165000 75490000

ALIGNED                   BIT (1)                 UNAL      MEM-4 (SYMBOL)      DCL INIT 4-77000 641500000

ALLOCATED                 BIT (1)                 UNAL      MEM-3 (SYMBOL)      DCL INIT 4-4000 33240000 33330

ALT_ZEILENNUMMER          CHAR (13)          VAR  UNAL      AUTOMATIC           DCL INIT 51270000 67080000 811

ANFANG                    FIXED BIN (15)          ALIG      AUTOMATIC           DCL INIT 51340000 53210000 624
                                                                                76130000

ANFANG                    FIXED BIN (15)          ALIG      AUTOMATIC           DCL 83070000 83100000 83160000

ANFANGSPOSITION           FIXED BIN (15)          ALIG      PARAMETER           DCL 83060000 83030000 83100000

ANFANGSZEIGER             POINTER                 ALIG      AUTOMATIC           DCL INIT 51100000 53010000 530
                                                                                53160000 53190000 53240000 533
                                                                                53410000 53450000 68060000

AREA                      BIT (1)                 UNAL      MEM-4 (SYMBOL)      DCL INIT 4-61000 63360000

ARITHMETIK_ERMITTELN      ENTRY (                       INT CONSTANT            DCL 71030000 63110000 63150000
                           POINTER ALIG)

ARRAY                     BIT (1)                 UNAL      MEM-3 (LABEL)       DCL INIT 3-500 62330000

ARRAY                     POINTER                 ALIG      MEM-2 (SYMBOL)      DCL INIT 4-21000 62090000 6212

ASSEMBLER                 BIT (1)                 UNAL      MEM-6 (SYMBOL)      DCL INIT 4-161000 75470000
```

Bild 3-7 Liste der Querverweise (Ausschnitt eines Beispiels)

Zusammengehörige Attribute werden in einer Spalte dargestellt. Die einzelnen Spalten der Liste haben folgende Bedeutung:

- BEZEICHNER (IDENTIFIER)
  In dieser Spalte beginnen die Bezeichner. Sie sind stets in voller Länge angegeben.
- DIMENSION
  Hier stehen alle Dimensionen, die für den Bezeichner gelten, auch die gegebenenfalls von einer übergeordneten Struktur ererbten.
- DATENTYP (DATA TYPE)
  In dieser Spalte sind die Attribute angegeben, die den Datentyp bestimmen. Die Attribute VAR und CPLX sind rechtsbündig angeordnet.

  Beim Attribut ENTRY sind auch die Daten-Attribute der Parameter angegeben, sowie

das RETURNS-Attribut mit den zugehörigen Daten-Attributen. Eine Einrückung um 1 Zeichen zeigt die Verschachtelung an.
Eine Dimension-Angabe für die Parameter und den RETURNS-Wert steht in der Spalte DIMENSION.

Beim Attribut PICTURE wird hinter PIC in Klammern die Länge der Variablen im Speicher angegeben (ohne V, F, K). Ferner ist der mit der Maske verbundene Attributsatz (CHAR, FIXED, FLOAT) und ggf. die Genauigkeit angegeben.

Beim Attribut CONDITION wird durch den Zusatz "(USER)" angegeben, daß es sich nicht um eine System-Bedingung, sondern um eine Benutzer-Bedingung handelt.

- Ausrichtung (ohne Überschrift)
  In dieser Spalte steht - soweit relevant - ALIG (für ALIGNED) oder UNAL (für UNALIGNED).
- Scope (ohne Überschrift)
  In dieser Spalte steht - soweit relevant - EXT oder INT.
- SPEICHER (STORAGE)
  In dieser Spalte steht - soweit relevant - die Speicherklasse. Bei BASED und DEFINED wird auch der Bezug angegeben.

  Bei den Gliedern einer Struktur wird hier "MEM-n (haupt)" ausgegeben, wobei n die normalisierte Stufennummer und "haupt" der Bezeichner der Hauptstruktur ist.
- BEZUEGE (REFERENCES)
  In dieser Spalte stehen die Nummern der Zeilen, in denen der Bezeichner explizit oder implizit verwendet wurde. Wird ein Bezeichner in einer Zeile mehrmals verwendet, so wird i.a. die Zeilennummer nur einmal aufgelistet.

  Abhängig davon, wie ein Bezeichner vereinbart wurde, beginnt die Spalte mit

  DCL wenn durch DECLARE-Anweisung oder wenn durch Anweisungs-Vorsatz (Maske, Format, Eingang) oder wenn durch Parameter bei einem Eingang

  `+++` wenn textabhängig (Bereich, Datei, Zeiger, Bedingung, eingebaute Funktion)

  `---` wenn implizit.

  Gibt es eine Initialisierung, so folgt "INIT". Die erste Zeilennummer gibt im allgemeinen die Nummer der Zeile an, in der die Vereinbarung steht.

  Die Darstellung der Zeilen-Referenz entspricht der im Quellprotokoll. Die Include-Nummer steht vor der Zeilennummer und ist hier durch einen Bindestrich von ihr getrennt.

3.8.7 Strukturlängen (AGGREGATE)

In der Liste der Strukturlängen sind alle Strukturen der externen Prozedur aufgelistet, sowie die Länge des Speicherplatzes, den die Strukturen und die in ihnen enthaltenen Unterstrukturen, Matrizen und Elementarglieder benötigen, in dezimaler und sedezimaler (hexadezimaler) Form. Die Ausgabe der Liste kann wie folgt beeinflußt werden:

```
*COMOPT LIST = AGGREGATE             Liste wird ausgegeben
             = NOAGGREGATE           Keine Liste
```

Bei der OFFSET-Angabe und bei der Längenangabe sind folgende Werte mit enthalten:

- bei Elementen mit dem Attribut VARYING die zwei Bytes für die aktuelle Länge.
- bei Matrizen und Strukturen die Füllzeichen, die wegen der Ausrichtung und der internen Darstellung erforderlich sind.

```
 8     P: PROCEDURE;
 9  1
10  1     DCL 1 STRUKTUR1          ALIGNED,
11  1           2 BIT1         BIT(3),
12  1           2 BIT2         BIT(6),
13  1           2 UNTER             DIM(3),
14  1             3 CHAR       CHAR(3),
15  1             3 VARCHAR    CHAR(5) VAR,
16  1             3 VARBIT     BIT(5)  VAR,
17  1           2 FIXED        FIXED BIN;
18  1
19  1     DCL 1 STRUKTUR2          BASED,
20  1           2 DIM            FIXED BIN,
21  1           2 LAENGE         FIXED BIN,
22  1           2 UNTER1 ,
23  1             3 ZEICHEN      CHAR(3) DIM(5 REFER (DIM)),
24  1             3 BIT          BIT(5 REFER (LAENGE)),
25  1           2 UNTER2,
26  1             3 ZEICHEN      CHAR(P1),
27  1             3 BIT          BIT(P2),
28  1           2 ZEIGER         POINTER;
29  1
30  1     DCL 1 STRUKTUR3,
31  1           2 FEST           FIXED DEC,
32  1           2 UNTER1               DIM(3),
33  1             3 CHARVAR      CHAR(5) VAR,
34  1             3 BITS         BIT(3),
35  1           2 UNTER2               DIM(P1),
36  1             3 CHAR         CHAR(5),
37  1             3 BIT          BIT(3) DIM(3);
38  1
39  1      END;
40
```

Bild 3-8 Programmbeispiel zu Bild 3-9

```
                                  S T R U K T U R  -  L A E N G E N
                                                 OFFSET          ELEMENT-LAENGE      GESAMT-LAENGE
QUELL-BEZUG  STUFE BEZEICHNER          DIMENSION   DEZ    SEDEZ     DEZ    SEDEZ      DEZ    SEDEZ

         10   1 STRUKTUR1                          0       0                          52      34
                2 BIT1                             0       0        0.3     0.3
                2 BIT2                             1       1        0.6     0.6
                2 UNTER                    3       2       2         15       F       48      30
                  3 CHAR                   3       2       2          3       3   INTERLEAVED
                  3 VARCHAR                3       6       6          7       7   INTERLEAVED
                  3 VARBIT                 3      14       E        2.5     2.5   INTERLEAVED
                2 FIXED                            50      32          2       2

         19   1 STRUKTUR2                          0       0                      VARIABLE
                2 DIM                              0       0          2       2
                2 LAENGE                           2       2          2       2
                2 UNTER1                           4       4                      VARIABLE
                  3 ZEICHEN                @       4       4          3       3   VARIABLE
                  3 BIT                              VARIABLE          VARIABLE
                2 UNTER2                           VARIABLE                      VARIABLE
                  3 ZEICHEN                        VARIABLE          VARIABLE
                  3 BIT                            VARIABLE          VARIABLE
                2 ZEIGER                           VARIABLE          4       4

         30   1 STRUKTUR3                          0       0                      VARIABLE
                2 FEST                             0       0          3       3
                2 UNTER1                   3       3       3        7.3     7.3      24      18
                  3 CHARVAR                3       3       3          7       7   INTERLEAVED
                  3 BITS                   3      10       A        0.3     0.3   INTERLEAVED
                2 UNTER2                   @      27      1B        6.1     6.1      VARIABLE
                  3 CHAR                   @      27      1B          5       5   INTERLEAVED
                  3 BIT                  @,3      32      20        0.3      0.   INTERLEAVED
```

Bild 3-9 Liste der Strukturlängen (gemäß Programm in Bild 3-8)

Für die Darstellungen innerhalb der Liste gelten folgende allgemeine Angaben:

- Bytes.Bits
  Die Werte werden in Bytes zu 8 Bits angegeben. Soweit der Wert über die Bytes hinaus noch Bits umfaßt, folgen diese der Byte-Angabe getrennt durch einen Punkt.
- VARIABLE
  Dieses Wort besagt, daß der Wert zur Übersetzungszeit noch nicht ermittelt werden konnte.
- INTERLEAVED
  Dieses Wort besagt, daß diese Elemente mit anderen Elementen verschachtelt sind, so daß sie nicht zusammenhängend im Speicher liegen und es somit auch keine zusammenhängende Gesamtlänge gibt.
- @
  Dieses Zeichen besagt, daß der Wert zur Übersetzungszeit noch nicht ermittelt werden konnte.

Die einzelnen Spalten der Liste haben folgende Bedeutung:

- QUELL-BEZUG (SOURCE-REF.)
  Hier steht die Nummer der Quellzeile, in der die Struktur vereinbart wurde. Handelt es sich um einen Include-Text, so steht links davor die Nummer des Include-Textes (siehe dazu Abschnitt 3.8).
- STUFE BEZEICHNER (LEVEL IDENTIFIER)
  Hier steht die normalisierte Stufennummer und der Bezeichner. Sie sind für die zweite und jede folgende Stufe um je 2 Stellen eingerückt. Zu lange Bezeichner werden rechts abgeschnitten.
- DIMENSION
  Hier steht für jede Dimension die Anzahl der Elemente, jeweils durch ein Komma getrennt. Dabei werden auch die von einer übergeordneten Struktur ererbten mit aufgeführt. Ist die Anzahl der Elemente einer Dimension zur Übersetzungszeit nicht festzustellen, so wird das Zeichen @ ausgegeben.
- OFFSET
  In dieser Spalte ist angegeben, bei welchem Byte und welchem Bit relativ zum Anfang der Haupt-Struktur die Unter-Struktur bzw. das Struktur-Element beginnt. Dabei wird die Zählung bei 0 begonnen. Ist der Wert für das Bit = 0, so wird die Angabe unterdrückt.

  Diese Angabe kann auch so interpretiert werden, daß sie angibt, wieviel Bytes und Bits in der Haupt-Struktur vor diesem Element stehen.
- ELEMENT-LAENGE (ELEMENT LENGTH)
  Diese Spalte enthält folgende Angaben:
  - Wenn keine Dimension vorhanden ist, so ist diese Spalte für Strukturen leer und für die Elementarglieder einer Struktur steht hier die Länge.
  - Wenn eine Dimension vorhanden ist, so steht hier sowohl bei Strukturen als auch bei Elementargliedern einer Struktur die Länge des Matrix-Elementes. Dimensionen in untergeordneten Strukturgliedern sind hierbei berücksichtigt, die eigene Dimension jedoch nicht. Die hier angegebene Länge multipliziert mit der eigenen Dimension ergibt den insgesamt benötigten Speicherbedarf. Dieser steht, soweit er zusammenhängend ist (nicht INTERLEAVED), in der Spalte GESAMT-LAENGE.
- GESAMT-LAENGE (TOTAL LENGTH)
  In dieser Spalte steht für Strukturen und Matrizen die Gesamt-Länge. Falls der Speicherplatz nicht zusammenhängend ist, steht hier das Wort "INTERLEAVED". Solche Variable unterliegen in PL/I bestimmten Einschränkungen. Der Speicherbedarf ist der übergeordneten Struktur zu entnehmen oder mit Hilfe der Spalten DIMENSION und ELEMENT-LAENGE oder mit Hilfe der Spalte OFFSET zu errechnen.

3.8.8 Speicherbelegung (MAP)

In dieser Liste wird für jeden Block der externen Prozedur eine Teilliste ausgegeben. Jede dieser Teillisten besteht aus

- einer Überschrift, die Angaben zum Blocktyp enthält und
- einer Auflistung von Konstanten und Variablen des Blockes und Angaben zur Anordnung im Speicher.

Die Nummer der Quellzeile, in der die Größe vereinbart ist und der vereinbarte Name ergänzen die Liste.
Die Ausgabe der Liste kann wie folgt beeinflußt werden:

```
*COMOPT LIST = MAP                  Liste ausgeben
             = NOMAP                Keine Liste
```

```
                    M A P - L I S T E

  QUELL-BEZUG      TYP           ADRESSE OFFSET  NAME


             0     ROOTBLOCK
             2     ENTRY CONST       0           BH_388


             2     EXT PROCEDURE                 BH_388
             6     ESD #              A          O$P
             9     AUTO              98          STRUKTUR
                     MEMBER                0     BIT1   IN STRUKTUR
                     MEMBER                0(3)  BIT2   IN STRUKTUR
                     MEMBER                2     UNTER  IN STRUKTUR
                     MEMBER                2     CHAR   IN UNTER IN STRUKTUR
                     MEMBER                9     BIT    IN UNTER IN STRUKTUR
                     MEMBER               20     FIXED  IN STRUKTUR
            18     ENTRY CONST      1AC          P
            27     AUTO              80          Z
            26     ESD #              3          SYSPRINT


            18     INT PROCEDURE                 P
            20     STATIC(FILE)      18          DATEI
            21     STATIC             8          STATISCH
            24     AUTO              80          ZEIGER
 1-         60     CONSTANT         10C          TOKEN_NODE
            35     AUTO              90          KOPFZEILE
            38     LABEL CONST      4B4          SCHLEIFE
            41     ENTRY CONST      4D4          SCHREIBEN


            28     ON UNIT                       O.ENDFILE*1


            41     INT PROCEDURE    360          SCHREIBEN( QUICK ) OWNER: P


            44     BEGIN BLOCK      3A8           ( QUICK ) OWNER: P
```

Bild 3-10 Speicherbelegungsliste (Beispiel)

Die Spalten der Liste haben folgende Bedeutung:

- QUELL-BEZUG (SOURCE-REF.)
  In dieser Spalte steht die Nummer der Quellzeile und ggf. die Nummer des Include-Textes, in der der Block bzw. die Variable oder Konstante vereinbart wurde (siehe Abschnitt 3.8).
- TYP (TYPE)
  In dieser Spalte steht als Überschrift der Typ des Blockes oder aber der Datentyp.
  Für den Blocktyp gibt es folgende Angaben:
  - ROOTBLOCK (Dies ist stets der erste Block)
  - EXT PROCEDURE
  - INT PROCEDURE
  - BEGIN BLOCK
  - ON UNIT

  Für den Datentyp gibt es folgende Angaben:
  - LABEL CONST — skalare Marken-Konstanten
  - ENTRY CONST — Eingangs-Konstanten
  - CONSTANT — Marken-Matrizen, interne STATIC-Konstanten, unbenannte Format-Konstanten
  - STATIC — interne STATIC-Variable
  - STATIC (CTL) — Anker für CONTROLLED-Variable
  - STATIC (FILE) — interne Datei-Konstanten
  - AUTO — AUTOMATIC-Variable
  - ESD # — ESD-Nummer für externe Variable und externe Eingänge
  - MEMBER — Glied einer Struktur
- ADRESSE (ADR)
  Die Angabe in dieser Spalte hängt von der Angabe in der Spalte TYP ab. Sie kann folgende Werte enthalten:
  - Bei INT PROCEDURE oder BEGIN BLOCK, die gemäß Spalte NAME vom Typ QUICK sind:
    Anfangsadresse des Aktivierungssatzes relativ zum Anfang des Aktivierungssatzes des Vaters
  - bei LABEL CONST, ENTRY CONST, CONSTANT:
    Anfangsadresse im Konstanten-Teil des Code-Moduls
  - bei STATIC, STATIC (CTL), STATIC (FILE):
    Anfangs-Adresse im Static-Modul
  - bei AUTO: Anfangs-Adresse relativ zum Anfang des Aktivierungssatzes
  - bei ESD #: ESD-Nummer des externen Namens

  In den anderen Fällen ist diese Spalte leer.

- OFFSET
  Diese Spalte ist nur beim Typ MEMBER besetzt und enthält die Byte-Adresse zum Anfang der Haupt-Struktur in sedezimaler Darstellung und die Bit-Adresse, soweit dieser Wert nicht Null ist.
- NAME
  Diese Spalte enthält jeweils die vereinbarten Namen. Bei den Typen BEGIN BLOCK und ON UNIT gibt es keine Eingangs-Namen, so daß hier die Spalte leer bleibt.

  Bei Blöcken, die als "QUICK"-Blöcke den Aktivierungsblock des Vater-Blocks mitbenutzen, steht in der Spalte NAME ein Zusatz, der auf den Vater-Block weist. Ist der Vater-Block vom Typ PROCEDURE, so lautet der Zusatz

  (QUICK) OWNER: Prozedurname

  Ist der Vater-Block vom Typ BEGIN BLOCK, so lautet der Zusatz

  (QUICK) OWNER: BEGIN BLOCK IN ZEILE n

  wobei n die Nummer der Quellzeile ist, in der der Block vereinbart wurde (siehe Spalte ZEILE).

3.8.9 Zuordnungsliste (OFFSET)

Diese Liste enthält Zuordnungen von Zeilennummern zu Adressen relativ zum Anfang der Prozedur. Mit Hilfe dieser Listen kann für eine Adressenangabe in einer Laufzeitmeldung die Zeilennummer der verursachenden Anweisung ermittelt werden.

3.8.10 Assembler-Code (ASSM)

In dieser Liste wird der erzeugte Maschinencode und seine Rückübersetzung in der Form der Assembler-Sprache dargestellt. Für die Interpretation der Liste ist daher die Kenntnis der Assembler-Sprache erforderlich. Die Ausgabe der Liste kann wie folgt beeinflußt werden:

```
*COMOPT LIST = ASSM                       ganze Prozeduren
             = ASSM ({Anfang,Ende};...)   Teile der Prozedur
             = NOASSM                     Keine Liste
```

Dabei ist für den Anfang bzw. das Ende eine Zeilennummer gemäß dem Quell-Protokoll anzugeben.

Die Spalten der Liste haben folgende Bedeutung:

- QUELL-BEZUG (SOURCE-REF.)
  Hier wird in dezimaler Darstellung die Zeilennummer, wie sie sich aus dem Quell-Protokoll ergibt, ggf. die Nummer des Include-Textes und die laufende Nummer der Anweisung innerhalb der Zeile angegeben.
- ADR INTERNCODE
  Hier wird in sedezimaler Darstellung die Adresse des Maschinenbefehls und der Befehl selbst angegeben.
- EXTERNCODE
  In dieser Spalte wird eine Rückübersetzung der Maschinenbefehle in Form des Assembler-Codes angegeben. Die Kommentare beziehen sich auf die Quellprogramm-Namen. Zusätzliche Kommentare werden für optimierte Schleifen ausgegeben.
- KOMMENTARE
  Enthalten Bezüge auf Quellnamen. Für Anweisungen, die durch die Schleifenoptimierung bearbeitet wurden, werden folgende Hinweise ausgegeben:

  'Vorgezogener invarianter Code'
  'Induktionsvariable initialisieren'
  'Inkrement initialisieren'
  'Induktionsvariable inkrementieren'.

Falls die Assembler-Code-Liste nur teilweise angefordert wird (*COMOPT LIST = ASSM (a,e)), wird diese Liste mit der Zuordnungsliste (siehe Abschnitt 3.8.9) kombiniert, d.h. die Teile des Programms, deren Assembler-Code nicht angegeben wird, werden in Form der Zuordnungsliste ausgegeben. Eine eigenständige Zuordnungsliste existiert in diesem Fall nicht mehr.

```
                                  OBJECTCODEPROTOKOLL DES CODE-MODULES
QUELL-BEZUG            ADR  INTERNCODE                  EXTERNCODE

                     000000                   BH$387    START O,READ
                                                        ENTRY P$START
                                                        EXTRN BH$387@6
                                                        EXTRN P$3#00#
                                                        EXTRN P$PPREP
                                                        EXTRN P$PTERM
                                                        EXTRN P$PVL##
                                                           .
                                                           .
                                                           .
          5          000260 05 EF                       BALR  14,15
                     000262 D2 03 D04C D050             MVC   76(4,13),80(13)
                     000268 D5 03 D25C B13C             CLC   604(4,13),316(11)       MG,
                     00026E 47 20 A2C2                  BC    2,706(0,10)             C.1
                     000272 D2 04 D267 D248             MVC   615(5,13),584(13)       NO, NO
                     000278 D2 0E D26E D24D             MVC   622(15,13),589(13)      BZ, BZ
                     00027E D2 03 D27F D25C             MVC   639(4,13),604(13)       AMG, MG
                     000284 D2 01 D285 D260             MVC   645(2,13),608(13)       AME, ME
                     00028A D2 04 D289 D262             MVC   649(5,13),610(13)       APR, PR
                     000290 58 60 B038                  L     6,56(0,11)              LIST
                     000294 50 60 D0A0                  ST    6,160(0,13)
                     000272 D2 04 D267 D248             MVC   615(5,13),584(13)       NO, NO
                     000278 D2 0E D26E D24D             MVC   622(15,13),589(13)      BZ, BZ
                     00027E D2 03 D27F D25C             MVC   639(4,13),604(13)       AMG, MG
                     000284 D2 01 D285 D260             MVC   645(2,13),608(13)       AME, ME
                     00028A D2 04 D289 D262             MVC   649(5,13),610(13)       APR, PR
                     000290 58 60 B038                  L     6,56(0,11)              LIST
                     000294 50 60 D0A0                  ST    6,160(0,13)
                     000278 D2 0E D26E D24D             MVC   622(15,13),589(13)      BZ, BZ
                     00027E D2 03 D27F D25C             MVC   639(4,13),604(13)       AMG, MG
                     000284 D2 01 D285 D260             MVC   645(2,13),608(13)       AME, ME
                     00028A D2 04 D289 D262             MVC   649(5,13),610(13)       APR, PR
                     000290 58 60 B038                  L     6,56(0,11)              LIST
```

Bild 3-11 Liste des Assembler-Code (Ausschnitt eines Beispiels)

3.8.11 Statistik (SUMMARY)

In dieser Liste sind Angaben darüber enthalten, wieviel virtueller Speicher der Übersetzer belegt hat, und wie häufig Speicher nachgefordert wurde. Siehe dazu auch die Steueranweisung STORAGE.

```
STATISTIK DER BELEGUNG DES VIRTUELLEN KERNSPEICHERS
PROZEDUR-STACK:    20 SEITEN; SYSTEM-AUFRUFE:     2 REQM,     0 RELM
STANDARD-AREA:     22 SEITEN; SYSTEM-AUFRUFE:     2 REQM
SPEICHERBEDARF:   219 SEITEN KLASSE 6   UND:    18 SEITEN KLASSE 5
```

Bild 3-12 Statistik der Übersetzung (Beispiel)

Die Angaben in der Statistik haben folgende Bedeutung:

Prozedur-Stack	Maximale Anzahl Speicherseiten (4K Bytes), die für den Prozedur-Stack zugeordnet wurden.
Standard-Area	Maximale Anzahl Speicherseiten (4K Bytes), die in der Standard-Area belegt wurden.
Speicherbedarf	Maximale Anzahl Speicherseiten (4K), die vom gesamten Programm (Prozedur-Stack, Standard-Area, Lademodul) der Klasse 5- und 6-Speicher angefordert wurden.
REQM	Anzahl der Speicheranforderungen beim System.
RELM	Anzahl der Speicherfreigaben beim System.

3.9 Diagnosemeldungen (DIAGNOST)

Diagnosemeldungen sind Informationen über den Ablauf der Übersetzung. Entsprechend der Wichtigkeit werden folgende Gruppen unterschieden:

1. Fatale Fehler (sofortiger Abbruch der Übersetzung)
2. Schwere Fehler (es wird kein Code erzeugt)
3. Fehler (der Übersetzer versucht zu korrigieren)
4. Warnungen (Hinweis auf mögliche Fehler)
5. Informationen (Hinweise für die Optimierung)

In einigen Informations-Meldungen gibt es einen Hinweis, für den es im Anhang A.6 weitere Informationen gibt. Es sind dies

– Informationsmeldung Nr. 500
  Die Nummer der Out-line-Folge ist im Anhang A.6 erläutert.

– Informationsmeldung Nr. 503
  Der Name Out-line-Folge ist im Anhang A.6 erläutert.

– Informationsmeldung Nr. 504
  Der Typ der Out-line-Folge für Konvertierung ist im Anhang A.6 erläutert.

Die Informationsmeldungen 500 bis 504 werden nur dann ausgegeben, wenn durch *COMOPT OPTIMIZE= irgendeine Optimierung verlangt wird.

Welche Art von Diagnosemeldungen beim Übersetzer und Vorübersetzer aufgetreten sind, wird nach SYSOUT protokolliert. Die Diagnosemeldungen selbst werden nach SYSLST ausgegeben.

Die Ausgabe der Diagnosemeldungen kann wie folgt beeinflußt werden:

```
*COMOPT DIAGNOST = SEVERE             Meldungen ab Gruppe 2
                 = ERROR              Meldungen ab Gruppe 3
                 = WARNING            Meldungen ab Gruppe 4
                 = INFORMATION        Meldungen ab Gruppe 5

*COMOPT DIAGNOST = TERMINAL           auch nach Datenstation
                 = NOTERMINAL         nur nach SYSLST

*COMOPT DIAGNOST = SAVLST             zusätzlich nach Datei
                 = NOSAVLST           nicht nach Datei

*COMOPT FORMAT   = DEUTSCH            Meldungen in Deutsch
                 = ENGLISH            Meldungen in Englisch

*COMOPT OPTIMIZE =                    Einfluß auf Informationsmeldungen
                                      500 bis 504 (siehe oben)
```

```
 1    BH_39:                /*BEISPIEL FUER DIAGNOSE-LISTE*/
 2      PROCEDURE;
 3
 4      CALL LESEN;
 5      CALL SCHREIBEN;
 6
 7      DCL ANTON       CHAR(3) VARING;
 8      DCL BERTA       FIXED   BINARI;
 9
10      DCL DATEINAME  FILE STREAM  EXTERNAL;
11      PUT FILE(DATEINAME) SKIP(2);
12
13      BERTA = VERIFY(CHAR(BERTA),'+-0123456789');
14      ANTON = SEARCH(CHAR(ANTON),'0');
15
16      BERTA = VERIFY(CHAR(BERTA,'+-0123456789');
17      ANTON = SEARCH(CHAR('0',ANTON);
18
19      GOTO WEITER;
20      BERTA = 3;
21   WEITER:
22
23      DCL 1 STRUKTUR           ALIGNED,
24            2 BIT1         BIT(3),
25            2 UNTER            DIM(3),
26              3 VARCHAR    CHAR(5) VAR,
27              3 VARBIT     BIT(5) VAR,
28            2 FIXED        FIXED BIN INIT(1);
29
30       PUT LIST(STRUKTUR);
31
32
33       END;
```

Bild 3-13 Prozedur zu den Diagnosemeldungen in Bild 3-14

```
      C O M P I L E R - D I A G N O S E M E L D U N G E N

SCHWERE FEHLER

+++++SCHWERER FEHLER NR.49
RECHTE SEITE DIESER ZUWEISUNG IST KEIN AUSDRUCK

     QUELL-BEZUG      QUELL-BEZUG      QUELL-BEZUG      QUELL-BEZUG      QUELL-BEZUG      QUELL-BEZUG
            16               17

FEHLER

+++++FEHLER NR.7
UNBEKANNTES ATTRIBUT BEI DER DEKLARATION VON '.....'

     QUELL-BEZUG      '.....'                                QUELL-BEZUG      '.....'
              7       'ANTON'                                         8       'BERTA'

+++++FEHLER NR.64
'.....' WURDE GEMAESS KONTEXT ALS 'ENTRY EXT CONSTANT' DEKLARIERT

     QUELL-BEZUG      '.....'                                QUELL-BEZUG      '.....'
              4       'LESEN'                                         5       'SCHREIBEN'

WARNUNGEN

+++++WARNUNG NR.56 IN ZEILE 20
ANWEISUNG KANN WEGEN UNBED. GOTO BZW. RETURN DAVOR NIE DURCHLAUFEN WERDEN

+++++WARNUNG NR.234 IN ZEILE 14
KONVERTIERUNG VON ARITH. NACH STRING

+++++WARNUNG NR.304
EXTERNER NAME '.....' AUF ERSTE 4 UND LETZTE 3 ZEICHEN GEKUERZT

     QUELL-BEZUG      '.....'                                QUELL-BEZUG      '.....'
              10      'DATEINAME'                            ——/——            'SCHREIBEN'
```

Bild 3-14 Diagnosemeldungen zu Prozedur in Bild 3-13

Die Meldungen werden in der Reihenfolge ihrer Wichtigkeit ausgegeben (Gruppe 1 zuerst), wobei jede Gruppe eine eigene Überschrift hat. Innerhalb der Gruppe werden gleiche Meldungen zusammengefaßt und der Meldungs-Text nur einmal ausgegeben. Dem Text folgen die Nummern der Quellzeilen und die laufende Nummer der Anweisung innerhalb der Zeile, die Anlaß für die Meldung war. Daran schließt sich die Ergänzung für den Meldungs-Text an.

Die Meldungs-Texte sind selbsterklärend. Es ist darauf zu achten, daß sich bei ausführbaren Anweisungen die Meldung auf die Zeile bezieht, in der die Anweisung beginnt.

Weiterhin ist darauf zu achten, daß Meldungen auch Folgen anderer Meldungen sein können.

3.10 Aufbau der Datei SAVLST

Durch die Steuerungsangaben für den Übersetzer

```
*COMOPT LIST     = SAVLST
*COMOPT DIAGNOST = SAVLST
```

können Listen und Meldungen des Übersetzers zusätzlich in einer Datei abgelegt werden (siehe Abschnitte 3.5.1 und 3.5.2).

Die Datei muß eine ISAM-Datei mit einer Schlüssellänge von 8 Bytes sein, mit dem Satzformat V. Ist keine Datei vorhanden, so wird eine neue Datei eingerichtet, entsprechend den folgenden Angaben:

```
/FILE SAVLST.PLI1.tsn,LINK=SAVLINK,SPACE=(3,3),FCBTYPE=ISAM,KEYPOS=5,      -
 KEYLEN=8,RECFORM=V,BLKSIZE=STD
```

Am Ende des Laufs wird der Name der Datei umgeändert in

```
SAVLST.PLI1.Modulname
```

Die Sätze der Datei haben folgenden Aufbau:

4 Byte Längenangabe
8 Byte Schlüssel
max. 255 Byte Information.

Die Kopfzeilen am Seitenanfang sind in der Datei enthalten, Vorschubsteuerzeichen sind nicht vorhanden.

3.11 Benutzung des Vorübersetzers

Der Vorübersetzer ist ein eigenes Programm, das in den PL/I-Übersetzer integriert ist. Beim Start des Übersetzers kann durch eine Steueranweisung angegeben werden, daß vor dem Übersetzungslauf ein Vorübersetzerlauf stattfindet.

Der Vorübersetzer erwartet einen Quelltext, der in der Programmiersprache PL/I geschrieben ist. Zwischen den PL/I-Anweisungen befinden sich Anweisungen an den Vorübersetzer. Diese Anweisungen, welche in Bezug auf Syntax und Semantik den PL/I-Anweisungen sehr ähnlich sind, werden vom Vorübersetzer ausgewertet. Das Ergebnis dieser Auswertung ist i.a. eine Textersetzung, die beträchtliche Änderungen an der Quellprozedur bewirken kann. Nach der Arbeit des Vorübersetzers wird die Quellprozedur in eine Zwischen-Datei übertragen, wobei die meisten Vorübersetzer-Anweisungen aus der Quellprozedur entfernt sind. Lediglich Anweisungen, die die Protokollierung des Übersetzungs-Protokolls beeinflussen, sind in der Quellprozedur noch erhalten.

Folgende Steueranweisungen für den PL/I-Übersetzer sind für den Vorübersetzer von Bedeutung:

```
OPTIONS = MACRO                    mit Vorübersetzer-Lauf
        = NOMACRO                  ohne Vorübersetzer-Lauf

LIST    = INSOURCE                 Protokoll der Vorübersetzer-Eingabe
        = NOINSOURCE               kein Protokoll der Vorübersetzer-Eingabe

OBJECT  = MACRO                    Nur Vorübersetzer-Lauf; keine
                                   Übersetzung

COMLIB  = (Bibliothek,...)         Bibliotheken aus denen einzufügende
                                   Texte zu entnehmen sind.

LIST    = IREF                     INCLUDE-Referenzliste
        = NOIREF
```

Bild 3-15 Datenfluß für den Lauf des Vorübersetzers und die Steueranweisung

Der Vorübersetzer liest den Quelltext gemäß der Steueranweisung SOURCE = Quelle. Ist die Steueranweisung LIST = INSOURCE angegeben, so wird ein Eingabe-Protokoll erstellt. Include-Dateien werden anschließend protokolliert.

Einige Anweisungen werden vom Vorübersetzer nach der Interpretierung in der veränderten Quell-Prozedur an den Übersetzer weitergereicht. Es sind dies die Anweisungen, die eine steuernde Wirkung auf das Übersetzungs-Protokoll haben.

Das Ergebnis des Vorübersetzers-Laufs ist eine Quelltext-Datei, die bis auf die erwähnten Ausnahmen frei von Vorübersetzer-Anweisungen ist. Diese Datei dient ausschließlich

ohne weiteres Zutun des Benutzers als Eingabe-Datei für die eigentliche Übersetzung. Die Datei kann auf folgende Art eingerichtet werden:

a) Der Benutzer gibt vor dem Aufruf des Übersetzers folgendes FILE-Kommando

```
/FILE Dateiname,LINK=SAVMAC[,SPACE=...]
```

In diesem Fall schreibt der Vorübersetzer seine Ausgabe in die angegebene Datei.

b) Der Vorübersetzer kreiert eine Ausgabe-Datei mit dem Namen

```
PLI1.SAVMAC.Prozeßfolgenummer
```

wobei die Prozeßfolgenummer des laufenden Prozesses genommen wird.

Der Vorübersetzer schreibt seine Ausgabe in eine Datei mit dem festgelegten Dateikettungsnamen SAVMAC. Ist diesem vom Benutzer keine Datei zugeordnet, so kreiert er selbst eine Datei gemäß dem folgenden Kommando

```
/FILE PLI1.SAVMAC.Prozeßfolgenummer,
      LINK=SAVMAC,
      FCBTYPE=ISAM,BLKSIZE=STD,RECSIZE=136,
      RECFORM=V,KEYPOS=5,KEYLEN=8
```

Diese Datei kann man z.B. mit einem der Kommandos

```
/PRINT PLI1.SAVMAC.Prozeßfolgenummer

/PRINT PLI1.SAVMAC.Prozeßfolgenummer,
       STARTNO=17,ENDNO=124
```

ausgeben.

In beiden Fällen kann der Benutzer die vom Vorübersetzer erzeugte Datei für spätere Übersetzungen, ggf. nach Anbringung von Korrekturen, direkt als Eingabe für den Übersetzer wieder verwenden. In diesen Fällen ist i.a. die Steueranweisung OPTIONS = NOMACRO anzugeben, da bereits alle Vorübersetzer-Anweisungen aufgelöst sind. Die Angabe MARGINS = SAVMAC ist dann zweckmäßig.

Stellt der Vorübersetzer bei der Bearbeitung der Vorübersetzer-Anweisungen Fehler fest, so gibt er entsprechende Meldungen aus, die in jeder Beziehung analog den Meldungen des Übersetzers sind. Dabei kann auch die weitere Übersetzung unterdrückt werden, wenn z.B. schwere Fehler aufgetreten sind und bei der Steueranweisung OBJECT = entsprechende Angaben gemacht sind.

```
/FILE DATEI,LINK=SAVMAC                    Zwischendatei
/EXEC $PLI1                                Start Übersetzer
      *COMOPT SOURCE=QUELLE,               Quelleingabe
      *COMOPT COMLIB=(...),                mit Vorübersetzer
      *COMOPT OPTIONS=MACRO,
      *COMOPT LIST=INSOURCE,               Eingabeprotokoll
      *COMOPT LIST=IREF,                   mit Vorübersetzer
      *COMOPT OBJECT=MACRO,                Übersetzungs-
      *COMOPT LIST=SOURCE,                 Protokoll
```

Bild 3-16 Kommandos und Steueranweisungen die den Vorübersetzer betreffen

Alle Formen der Eingabe für die Quellen sind mit und ohne Vorübersetzer identisch, d.h. der Vorübersetzer akzeptiert dieselben Formen der Eingabe wie der PL/I-Übersetzer.

Für die Anweisung %INCLUDE muß ggf. der einzufügende Text als Datei oder Bibliothek bereitgestellt werden. Eventuell muß durch die Steueranweisung

```
COMLIB = (Bibliothek,...)
```

die Verbindung zu bestimmten Bibliotheken hergestellt werden.

Die Datensätze der Ausgabe-Datei des Vorübersetzers haben folgendes Format:

Position	Inhalt
1 - 3	Nummer des Include-Textes
4 - 11	Original Quellzeilennummer gemäß Eingabe-Datei für den Vorübersetzer: Index aus einer ISAM-Datei oder laufende Nummer bei SAM-Datei bzw. gemäß der Steueranweisung MARGINS = LINID (a,b).
12	Vorschubzeichen aus der Eingabe-Datei, sonst Leerzeichen.
13	Leerzeichen
14 - 15	Stufentiefe der Ersetzung durch den Vorübersetzer bzw. Leerzeichen, falls keine Ersetzung stattfand.
16	Leerzeichen
17 - 24	Quellzeile, wie sie durch MARGINS = TEXT (a,b) ausgeblendet und/oder durch den Vorübersetzer verändert wurde. Wenn der Textbereich mehr als 108 Zeichen enthält, werden zusätzliche Sätze in der Ausgabe-Datei angelegt mit gleicher Originalzeilennummer. Ist der Textbereich kleiner als 108 Zeichen, wird die Ausgabezeile auf 108 Zeichen und Leerzeichen aufgefüllt.

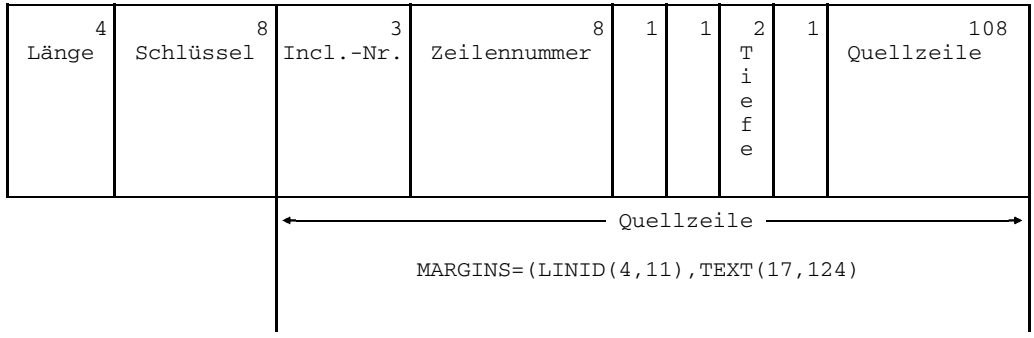


Bild 3-17 Darstellung des Datensatzes und der Quellzeile wie sie vom Vorübersetzer in der Zwischendatei für den Übersetzer abgelegt wird

Dabei ist zu beachten, daß dem Datensatz noch ein Schlüssel mit 8 Bytes und ein Längenfeld mit 4 Bytes vorangeht. Der Schlüssel wird, beginnend bei 00000001 in Schritten von 1 hochgezählt.

3.12 Beschreibung der Operanden des SDF-Kommandos START-PLI1-COMPILER

3.12.1 Übersicht über die Operanden

Name des Operanden	Zweck
SOURCE	Bestimmen der Eingabequelle des Quellprogramms
INCLUDE-LIBRARY	Zuweisen der PLAM-Bibliothek, in der die %INCLUDE-Elemente stehen
SOURCE-PROPERTIES	Beschreiben des Eingabeformats des Quellprogramms
PREPROCESSING	Aufruf des Vorübersetzers
COMPILER-ACTION	Teilweise Durchführung des Übersetzungslaufs und Beeinflussung einiger Eigenschaften der erzeugten Objekte
MODULE-LIBRARY	Bestimmen des Namens und Ausgabezieles der erzeugten Objekte
LISTING	Steuern der Listenausgabe
TEST-SUPPORT	Steuern der Testhilfe
OPTIMIZATION	Steuern der Optimierung
COMPILER-TERMINATION	Bestimmen, wann der Übersetzungslauf abgebrochen werden soll
MONJV	Überwachen der Übersetzung mit Jobvariablen
LANGUAGE	Ausgabe der Compilermeldungen in Englisch oder Deutsch

3.12.2 Einzelbeschreibung der Operanden

SOURCE-Operand

Dieser Operand bestimmt, ob das Quellprogramm von SYSDTA, aus einer katalogisierten Datei oder aus einer PLAM-Bibliothek eingelesen wird.

```
SOURCE = *SYSDTA /
         <full-filename 1..54 without gen-vers> /
         *LIBRARY-ELEMENT(...)

     *LIBRARY-ELEMENT(...)
        |
        | LIBRARY = <full-filename 1..54>
        |
        |,ELEMENT = <full-filename 1..54 without gen-vers>(...)
        |   |
        |   | VERSION = *HIGHEST-EXISTING / <alphanum-name 1..24>
```

SOURCE = *SYSDTA
Voreingestellt ist die Eingabe von der Systemdatei SYSDTA, die im Dialogbetrieb der Datensichtstation zugewiesen ist. Mit dem ASSIGN-SYSDTA-Kommando kann vor dem Aufruf des Compilers SYSDTA auf eine katalogisierte Datei oder ein PLAM-Bibliothekselement umgewiesen werden.

SOURCE = <full-filename 1..54 without gen-vers>
Mit <full-filename> wird eine katalogisierte Datei zugewiesen.

SOURCE = *LIBRARY-ELEMENT(...)
Mit diesem Parameter wird eine PLAM-Bibliothek und ein Element daraus angegeben.

LIBRARY = <full-filename 1..54>
Mit <full-filename> wird eine PLAM-Bibliothek zugewiesen.

ELEMENT = <full-filename 1..54 without gen-vers>(...)
<full-filename> bezeichnet den vollqualifizierten Namen eines Elements aus der zuvor angegebenen PLAM-Bibliothek. Das Element muß vom Typ -S (=source) sein.

VERSION = *HIGHEST-EXISTING
Enthält die Elementangabe keine Versionsbezeichnung, verwendet der Compiler die höchste Version.

VERSION = <alphanum-name 1..24>
Der Compiler verwendet die mit <alphanum-name> angegebene Version.

INCLUDE-LIBRARY-Operand

Dieser Operand bestimmt aus welcher/welchen Bibliotheken %INCLUDE-Elemente eingelesen werden sollen.

```
INCLUDE-LIBRARY = *NONE / list-poss: <full-filename 1..54> /
```

INCLUDE-LIBRARY = *NONE
Es werden keine %INCLUDE-Elemente gelesen.

INCLUDE-LIBRARY = <full-filename 1..54>
<full-filename> ist der Datei- oder Linkname einer oder mehrerer PLAM-Bibliotheken. Die PLAM-Bibliotheken werden in der angegebenen Reihenfolge nach %INCLUDE-Elementen durchsucht. Als letztes wird auch der Systemkatalog ($TSOS) durchsucht. <full-filename> wird zuerst als Linkname, dann als Dateiname interpretiert.

SOURCE-PROPERTIES-Operand

Dieser Operand beschreibt das Eingabeformat des Quellprogramms.

```
SOURCE-PROPERTIES = STD / PARAMETERS(...)

   PARAMETERS(...)
      |  FROM-COLUMN = 2 / <integer 1..256>
      |  TO-COLUMN = 72 / <integer 1..256>
      | ,LANGUAGE-STANDARD = STD / ISO
      | ,SPECIAL-KEYWORDS = NO / YES
```

SOURCE-PROPERTIES = STD
Es werden die voreingestellten Operandenwerte der nachfolgenden PARAMETERS-Struktur übernommen.

SOURCE-PROPERTIES = PARAMETERS(...)

FROM-COLUMN = 2 / <integer 1..256>
Bestimmt die Anfangsspalte des Quelltextes. Voreingestellt ist Spalte 2.

TO-COLUMN = 72 / <integer 1..256>
Bestimmt die letzte Spalte des Quelltextes. Voreingestellt ist Spalte 72.

LANGUAGE-STANDARD = STD / ISO
Der Quelltext entspricht dem Industriestandard (STD) bzw. dem PL/I-Standard (ISO).

SPECIAL-KEYWORDS = NO / YES
YES: Ersatz-Schlüsselwörter wie z.B. B, CAT, LE werden als reservierte Namen behandelt.

PREPROCESSING-Operand

Dieser Operand bestimmt, ob der Vorübersetzer aufgerufen werden soll.

```
PREPROCESSING = NONE / PARAMETERS(...)

      PARAMETERS(...)
        |
        |  OUTPUT = *NONE / *STD
```

PREPROCESSING = NONE
Der Vorübersetzer wird nicht aufgerufen.

PREPROCESSING = PARAMETERS(...)
Der Vorübersetzer wird aufgerufen.

OUTPUT = *NONE
Das Ergebnis der Vorübersetzung wird nicht ausgegeben.

OUTPUT = *STD
Der Vorübersetzer schreibt das Ergebnis in die SAVMAC-Datei.

COMPILER-ACTION-Operand

Dieser Operand legt fest, ob und wie ein Objekt erzeugt werden soll.

```
COMPILER-ACTION = SYNTAX-CHECK / MODULE-GENERATION(...)

      MODULE-GENERATION(...)
        |
        |  MAIN-PROGRAM = NO / YES
        |
        | ,EXTENDED-SYSTEM = NONE / PARAMETERS(...)
        |
        |       PARAMETERS(...)
        |         |  BIT-POINTER = NO / YES
```

COMPILER-ACTION = SYNTAX-CHECK
Der Compilerlauf wird nach der Syntaxanalyse beendet.

COMPILER-ACTION = MODULE-GENERATION(...)
Es wird ein vollständiger Übersetzungslauf durchgeführt. Einige Eigenschaften der zu erzeugenden Objekte lassen sich mit den Parametern der MODULE-GENERATION-Struktur beeinflussen.

MAIN-PROGRAM = NO / YES
YES: Das Objekt ist ein Hauptprogramm.

EXTENDED-SYSTEM = NONE
Das Objekt ist nicht XS-fähig.

EXTENDED-SYSTEM = PARAMETERS(BIT-POINTER = NO / YES)
Das Objekt ist XS-fähig.
Bei BIT-POINTERS=YES sind alle Zeiger Bitzeiger.

MODULE-LIBRARY-Operand

Dieser Operand steuert, in welcher Bibliothek und unter welchem Namen das erzeugte Objekt abgelegt werden soll.

```
MODULE-LIBRARY = *OMF / <full-filename 1..54 without gen-vers>(...)

  <full-filename 1..54 without gen-vers>(...)
     |
     |  ELEMENT = *STD(...) / <full-filename 1..54 without gen-vers>(...)
     |
     |     *STD(...)
     |       |
     |       |  VERSION = *UPPER-LIMIT / <alphanum-name 1..24>
     |
     |     <full-filename 1..54 without gen-vers>(...)
     |       |
     |       |  VERSION = *UPPER-LIMIT / <alphanum-name 1..24>
```

MODULE-LIBRARY = *OMF
Das Objekt wird in die temporäre EAM-Datei geschrieben.

MODULE-LIBRARY = <full-filename 1..54 without gen-vers>(...)
Name der PLAM-Bibliothek, in die das Objekt geschrieben werden soll.

ELEMENT = *STD(...)
Der Elementname wird aus dem ersten Eingangsnamen der externen Prozedur gebildet.

VERSION = *UPPER-LIMIT / <alphanum-name 1..24>
Gleichnamige Moduln können durch Versionsbezeichnungen unterschieden werden. Fehlt die Versionsangabe, steht für den Bindelauf das Element mit der höchsten Version (*UPPER-LIMIT) zur Verfügung.

ELEMENT = <full-filename 1..54 without gen-vers>(...)
Mit <full-filename> kann ein Elementname zugeteilt werden.

VERSION = *UPPER-LIMIT / <alphanum-name 1..24>
Siehe oben

Wird kein Elementname angegeben, erhält das Objekt den ersten Eingangsnamen der externen Prozedur.

LISTING-Operand

```
LISTING = STD / NONE / PARAMETERS(...)

    PARAMETERS(...)
        OPTIONS = NO / YES
       ,PREPROCESSING-OUTPUT = NO / YES
       ,SOURCE = NONE / PARAMETERS(...)
           PARAMETERS(...)
              INCLUDE-EXPANSION = NO / YES
             ,PAGE-FRAME = NONE / PARAMETERS(...)
                   PARAMETERS(...)
                      LINE-DELIMITER = ' ' / <c-string 1..1>
             ,LINE-NUMBER-LAYOUT = STD / EDT / EDOR
       ,DATA-ALLOCATION-MAP = NO / YES
       ,CROSS-REFERENCE = NONE / REFERENCED / FULL
       ,INCLUDE-REFERENCE = YES / NO
       ,STRUCTURE-LAYOUT = NO / YES
       ,EXTERNAL-DICTIONARY = NO / YES
       ,STATEMENT-ADDRESS = YES / NO
       ,ASSEMBLER-CODE = NO / YES
       ,SUMMARY = NO / YES
       ,ADDITIONAL-OUTPUT = *NONE / *TERMINAL
```

LISTING = STD
Es werden die voreingestellten Operandenwerte der PARAMETERS-Struktur übernommen.

LISTING = NONE
Es werden keine Listen erzeugt.

LISTING = PARAMETERS(...)
Mit den folgenden Parametern wird bestimmt, welche Listen erzeugt werden sollen. Der Wert NO bewirkt, daß die jeweilige Liste nicht erzeugt wird.

OPTIONS=NO / YES
YES: Ausgabe der wirksamen Steueranweisungen.

PREPROCESSING-OUTPUT = NO / YES
YES: Ausgabe der Vorübersetzer-Liste.

SOURCE = NONE
Es wird keine Quellprogrammliste ausgegeben.

SOURCE = PARAMETERS(...)
Ausgabe der Quellprogrammliste.

INCLUDE-EXPANSION = NO / YES
YES: Die Quellprogrammliste enthält die eingesetzten INCLUDE-Texte.

PAGE-FRAME = NONE / PARAMETERS(LINE-DELIMITER = ' ' / <c-string 1..1>
PARAMETERS: Ausgabe des Quellprogramms mit Vor- und Nachspann sowie ggf. mit Rahmenzeichen <c-string>.

LINE-NUMBER-LAYOUT = STD / EDT / EDOR
Format der Zeilennummer.
STD: unverändert, wie beim PRINT-Kommando
EDT: ohne führende und abschließende Nullen, mit Punkt zwischen 4. und 5. Stelle
EDOR: ohne abschließende Nullen

DATA-ALLOCATION-MAP = NO / YES
YES: Ausgabe der Speicherbelegungsliste.

CROSS-REFERENCE = NONE / REFERENCED / FULL
Ausgabe der Referenzen und Attribute von Bezeichnern.

INCLUDE-REFERENCE = YES / NO
YES: Ausgabe der INCLUDE-Referencen.

STRUCTURE-LAYOUT = NO / YES
YES: Ausgabe der Aggregat-Liste.

EXTERNAL-DICTIONARY = NO / YES
YES: Ausgabe der Liste der externen Namen.

STATEMENT-ADDRESS = YES / NO
YES: Ausgabe der Zuordnungsliste: Anweisung zur sedezimalen Adresse des generierten Codes.

ASSEMBLER-CODE = NO / YES

YES: Ausgabe der Objektcodeliste.

SUMMARY = NO / YES
YES: Ausgabe der Programmstatistik.

ADDITIONAL-OUTPUT = *NONE / *TERMINAL
*TERMINAL: Die Ausgabe der Listen erfolgt zusätzlich auf der Datensichtstation.

TEST-SUPPORT-Operand

Dieser Operand wählt Optionen zur Testunterstützung aus.

```
TEST-SUPPORT = NONE / PARAMETERS(...)

   PARAMETERS(...)
      |  STATEMENT-TABLE = NO / YES
      |
      | ,TOOL-SUPPORT = NONE / AID
      |
      | ,TRACE-SUPPORT(TRC) = NONE / ALL / PARAMETERS(...)
      |
      |       PARAMETERS(...)
      |          |  PROCEDURE-ENTRY = YES / NO
      |          |
      |          | ,PROCEDURE-EXIT = YES / NO
      |          |
      |          | ,PROCEDURE-CALL = YES / NO
      |          |
      |          | ,LABELLED-STATEMENT = YES / NO
      |          |
      |          | ,GOTO-STATEMENT = YES / NO
```

TEST-SUPPORT = NONE
Keine Testunterstützung.

TEST-SUPPORT = PARAMETERS(...)
Mit den folgenden Parametern wird bestimmt, welche Testunterstützungen erzeugt werden sollen. Der Wert NO bewirkt, daß die jeweilige Testunterstützung nicht erzeugt wird.

STATEMENT-TABLE = NO / YES
YES: Bereitstellen der Quellzeilennummern.

TOOL-SUPPORT = NONE / AID
AID: Erzeugen von LSD-Informationen für die Testhilfe AID.

TRACE-SUPPORT = NONE / ALL / PARAMETERS(...)
Erzeugen von Informationen für Trace.

PROCEDURE-ENTRY = YES / NO
YES: Prozedurtrace

PROCEDURE-ENTRY = YES / NO
YES: Trace bei Verlassen einer Prozedur

PROCEDURE-CALL = YES / NO
YES: Trace bei Aufruf einer Prozedur

LABELLED-STATEMENT = YES / NO
YES: Trace an den Anweisungsmarken

GOTO-STATEMENT = YES / NO
YES: Trace bei Sprüngen

OPTIMIZATION-Operand

Dieser Operand wählt die Optimierungsart.

```
OPTIMIZATION = NONE / PARAMETERS(...)

    PARAMETERS(...)
      |  SUPPRESS-COND-CHECK = NO / YES
      | ,REORDER-EXPRESSION = NO / YES
      | ,IGNORE-STRINOVERLAP = NO / YES
```

OPTIMIZATION = NONE
Die Optimierung wird ausgeschaltet.

OPTIMIZATION = PARAMETERS(...)

SUPPRESS-COND-CHECK = NO / YES
YES: Die Generierung von Prüfcode wird unterdrückt.

REORDER-EXPRESSION = NO / YES
YES: Die Reihenfolge der Anweisungen darf verändert werden.

IGNORE-STRINGOVERLAP = NO / YES
YES: Zuweisungen werden nicht auf Überlappung überprüft.

COMPILER-TERMINATION-Operand

Dieser Operand steuert den Abbruch der Übersetzung.

```
COMPILER-TERMINATION = STD / PARAMETERS(...)

    PARAMETERS(...)
      |  CPU-LIMIT = JOB-REST / <integer 1..32767>
      | ,MAX-ERROR-NUMBER = 500 / <integer 1..32767>
```

COMPILER-TERMINATION = STD
Es werden die voreingestellten Operandenwerte der PARAMETERS-Struktur übernommen.

COMPILER-TERMINATION = PARAMETERS(...)
Auswahl der Kriterien für einen Ubersetzungsabbruch

CPU-LIMIT = JOB-REST / <integer 1..32767>
Maximale Übersetzungszeit in Sekunden

MAX-ERROR-NUMBER = 500 / <integer 1..32767>
Fehleranzahl, bei der die Übersetzung abgebrochen wird

MONJV-Operand

Dieser Operand legt fest, ob der Compilerlauf durch eine Jobvariable überwacht werden soll.

```
MONJV = *NONE / <full-filename 1..54 without gen>
```

MONJV = *NONE
Der Compilerlauf wird nicht durch eine Jobvariable überwacht.

MONJV = <fill-filename 1..54 without gen>
Name der Jobvaribalen zur Überwachung des Compilerlaufs.

LANGUAGE-Operand

Dieser Operand legt die Sprache fest, in der die Compilermeldungen ausgegeben werden.

```
LANGUAGE = ENGLISH / DEUTSCH
```

LANGUAGE = ENGLISH / DEUTSCH
Die Compilermeldungen können in Englisch (Voreinstellung) oder in Deutsch ausgegeben werden.

3.12.3 Abbildung der SDF-Operanden auf die COMOPT-Operanden

SDF-Operand	**COMOPT-Operand**
SOURCE	SOURCE
INCLUDE-LIBRARY	COMPLIB
SOURCE-PROPERTIES	
FROM-COLUMN, TO-COLUMN	MARGINS=TEXT
LANGUAGE-STARNDARD=ISO	OPTIONS=ISO
SPECIAL-KEYWORDS=YES	MARGINS=CHAR48
PREPROCESSING=YES	OPTIONS=MACRO
OUTPUT=*STD	MARGINS=SAVMAC
COMPILER-ACTION=	OBJECT=
SYNTAX-CHECK	CODE
MODULE-GENERATION	OUT
MAIN-PROGRAM=YES	OPTIONS=MAIN
EXTENDED-SYSTEM=YES	OPTIONS=XS
BIT-POINTER=YES	OPTIONS=BITPTR
MODULE-LIBRARY	MODULE
LISTING	LIST=
OPTIONS=YES	OPTIONS
PREPROCESSING-INPUT=YES	INSOURCE
SOURCE=YES	SOURCE
INCLUDE-EXPANSION=YES	EXPAND
PAGE-FRAME=YES	OUTTEXT
LINE-NUMBER-LAYOUT	LINECNT
DATA-ALLOCATION=YES	MAP
CROSS-REFERENCE= REFERENCED FULL	 SHRTXREF FULLXREF
STRUCTURE-LAYOUT=YES	AGGREGATE

SDF-Operand	**COMOPT-Operand**
LISTING (Fortsetzung)	
EXTERNAL-DICTIONARY=YES	ESD
STATEMENT-ADDRESS=YES	OFFSET
ASSEMBLER-CODE=YES	ASSM
SUMMARY=YES	SUMMARY
ADDITIONAL-OUTPUT=*TERMINAL	TERMINAL
TEST-SUPPORT	DEBUG, SYMTEST
STATEMENT-TABLE=YES	DEBUG=STMT
TOOL-SUPPORT=AID	SYMTEST=ALL
TRACE-SUPPORT	
PROCEDURE-ENTRY=YES	DEBUG=PROCTRACE
PROCEDURE-EXIT=YES	DEBUG=RETURNTRACE
PROCEDURE-CALL=YES	DEBUG=CALLTRACE
LABELLED-STATEMENT=YES	DEBUG=LABTRACE
GOTO-STATEMENT=YES	DEBUG=GOTOTRACE
OTIMIZATION	OPTIMIZE=
SUPPRESS-COND-CHECK=NO	ENABLING
REORDER-EXPRESSION=YES	REORDER
IGNORE-STRINGOVERLAP=YES	OVERLAP
COMPILER-TERMINATION	
CPU-LIMIT=	1)
MAX-ERROR-NUMBER=	OBJECT=ABORT()
MONJV=	1)
LANGUAGE=ENGLISH	FORMAT=ENGLISH

1) Wird im EXECUTE-Kommando eingetragen

4 Binden und Laden eines PL/I-Programms

4.1 Allgemeines

Ein durch Übersetzung entstandener PL/I-Bindemodul muß durch das Anbinden anderer Moduln zu einem lauffähigen Lademodul gemacht werden. Diese anderen Bindemoduln werden aus der PL/I-Laufzeitbibliothek entnommen oder sie sind aus separat übersetzten (PL/I)-Prozeduren entstanden. Die Laufzeitbibliothek enthält die Menge aller vorgefertigten Bausteine eines Objektprogrammes wie Programmrahmen, Ein-/Ausgabesystem, eingebaute Funktionen, Bedingungs- und Fehlerbehandlung usw. Diese Bausteine werden in der vorliegenden Beschreibung auch als Laufzeitsystem bezeichnet. Es stehen zwei Laufzeitsysteme zur Verfügung (siehe Abschnitt 4.5).
Durch den Bindevorgang werden alle die Bindemoduln zusammengefügt, welche beim Aufruf des Binders genannt werden und zusätzlich solche, die explizit oder implizit von den genannten Moduln aus angesprochen werden. Explizit angesprochen werden Objektmoduln, für die im jeweils betrachteten Modul Vereinbarungen (DCL...ENTRY EXTERNAL;) vorkommen, sofern die Vereinbarung nicht zusätzlich die Angabe OPTIONS (WXTRN) enthält. Dagegen werden Moduln der Laufzeitbibliothek, wie sie für die Konvertierung, die Ein- und Ausgabe usw. benötigt werden, implizit angesprochen. Das Herstellen der Adreßbezüge zwischen den EXTERNAL-Größen der Moduln sowie die Fixierung der relativen Adressen in den Bindemoduln, sind weitere wichtige Funktionen, die ebenfalls vom Binder durchgeführt werden.
Die Lademoduln werden im BS2000 vom Binder (TSOSLNK) in eine katalogisierte Datei geschrieben. Diese Lademoduln können aus der katalogisierten Datei beliebig oft aufgerufen werden.

Während der Entwicklungs- und Testphase kann statt des Binders TSOSLNK der dynamische Bindelader (DLL) angewendet werden (nur geringe Übersetzungszeit, Unterprogramme nur aus einer Bibliothek, "sharable" programmiert). Das Bindeladen erfolgt mit der Anweisung /EXEC* bzw. /LOAD*.

4.2 Steuerung des Binders (TSOSLNK)

Der Binder setzt Bindemoduln zu Lademoduln zusammen. Zunächst werden alle durch die INCLUDE-Anweisung genannten Bindemoduln zusammengefügt. Enthalten sie Verweise auf weitere Prozeduren oder externe Namen (externe Referenzen), so müssen diese Prozeduren ebenfalls hinzugebunden werden. Die Suche nach Prozeduren mit den Definitionen der externen Namen geschieht in drei Stufen. Zuerst wird versucht, die Definitionen in den bei INCLUDE angegebenen Bindemoduln zu finden. Dann wird versucht, die noch nicht zugeordneten externen Referenzen aus eigens dafür angegebenen Bindemodulbibliotheken zu befriedigen (RESOLVE-Anweisung). Für die noch immer nicht aufgelösten Referenzen wird in der Bibliothek [$TSOS.]TASKLIB nach passenden Definitionen gesucht. Findet sich auch hier keine Definition, so bleibt die Referenz unaufgelöst und es wird eine Fehlermeldung ausgegeben. Nicht angebunden werden solche Prozeduren, auf die nur sogenannte bedingte Externbezüge (OPTIONS(WXTRN)) (siehe Abschnitt 3.3.4) vorhanden sind.

Weitere Funktionen des Binders sind:

- die Änderung von externen Namen,
- die Zusammenfassung und Speicherplatzbestimmung für EXTERNAL-Größen (COM),
- das Erzeugen von Überlagerungsstrukturen,
- die Festlegung des Programm-Startpunkts.

Vom Binder werden auf Wunsch Protokoll-Listen erzeugt.

Die Steuerung des Binders (TSOSLNK) geschieht über eigene Steueranweisungen. Es werden an dieser Stelle die wichtigsten Steueranweisungen mit ihren Parametern kurz vorgestellt. Genauere Informationen über die Funktion des Binders und die Steueranweisung sind im Manual Dienstprogramme [3] bzw. bei Einsatz einer BS2000 Version ab V 8.0 im Manual Binder und Lader [12] zu finden.

4.2.1 Aufruf des Binders

Der Binder wird durch das Kommando

```
/{EXECUTE}
 {      }$TSOSLNK
 {EXEC   }
```

aufgerufen.

Der Binder erwartet als Eingabe Steueranweisungen und Bindemoduln, die er beide über SYSDTA einliest. Durch eine spezielle Anweisung (INCLUDE-Anweisung) kann der Binder veranlaßt werden, Bindemoduln von der temporären EAM-Datei des laufenden Prozesses (referiert durch das Symbol *) oder aus einer Bindemodulbibliothek zu lesen und dem Lademodul beizufügen.

Dabei kann der Benutzer wählen:

- Von der EAM-Bindemoduldatei die ganze Datei oder bestimmte Moduln
- Von Bindemodulbibliotheken nur die spezifischen Moduln

Der Binder schreibt den erzeugten Lademodul in eine katalogisierte Datei. Diese Datei wird bei einem Binderlauf von vorne neu beschrieben, wenn sie schon besteht. Falls der Benutzer vor dem Binderaufruf kein FILE-Kommando für diese Datei gibt, werden die Zuweisungen von Speicherplatz und die Katalogisierung vom Binder durchgeführt.

Wenn das PL/I-Programm statisch gebunden wird bzw. wenn beim dynamischen Binden ein großer vorgebundener Modul entsteht, ist es vorteilhaft, mit einem FILE-Kommando den Speicherplatz vor dem Binden festzulegen. Man vermeidet dadurch eine unnötige Zerstückelung des Lademoduls und verkürzt die Ladezeiten. Eventuell zuviel angeforderten Speicher kann man anschließend durch ein FILE-Kommando mit negativer SPACE-Angabe wieder freigeben.

Fehlermeldungen des Binders werden auf SYSOUT und SYSLST ausgegeben.

Beispiel

```
.
.
/SYSFILE SYSDTA=PROG
/EXEC $PLI1
/SYSFILE SYSDTA=(SYSCMD)
/FILE PL1.PROGR.003,SPACE=(90,6)
/EXEC $TSOSLNK
PROGRAM P003,FILENAM=PL1.PROG.003
INCLUDE *
END
/FILE PL1.PROG.003,SPACE=-100
.
.
```

Anmerkung

SYSDTA wurde für den Übersetzungsvorgang der Quellprogrammdatei PROG zugeordnet und muß vor Aufruf des Bindens wieder zurückgesetzt werden.

4.2.2 Anweisungen für den Binder

PROGRAM-Anweisung

Mit der PROGRAM-Anweisung wird der Name des Lademoduls festgelegt. Diese Anweisung muß unbedingt angegeben werden, braucht aber nicht die erste Anweisung zu sein. Werden mehrere PROGRAM-Anweisungen gegeben, so gilt die zuletzt angegebene.

```
PROG[RAM] Programmname[,FILENAM=Dateiname]
```

Programmname — Name des gebundenen Programms (maximal 8 Zeichen)

Dateiname — Name der katalogisierten Datei, in die das gebundene Programm abgelegt werden soll. Wird dieser Parameter nicht angegeben, so gilt Programmname auch als Dateiname (Name des Lademoduls).

Von den sonstigen Parametern der PROGRAM-Anweisung können in Sonderfällen noch folgende verwendet werden:

LET = Y[ES] — Das Programm wird auch dann gebunden, wenn nach dem Durchsuchen der Bibliotheken noch offene externe Referenzen bestehen. Es sollte in diesem Fall jedoch sichergestellt sein, daß diese Referenzen beim Programmablauf nicht benötigt werden, da sich das Programm andernfalls undefiniert verhält.

PL1 = Y[ES] — Diese Angabe verhindert Fehlermeldungen des Binders für den Fall, daß Variable mit dem Attribut EXTERNAL in mehreren Bindemoduln einen Initialwert erhalten.

Achtung

Falls dieser Parameter angegeben wird, dürfen keine mit IQ beginnenden externen Namen verwendet werden (siehe Beschreibung des Binders [3] bzw. bei Einsatz einer BS2000 Version ab V 8.0 im Manual Binder und Lader [12]).

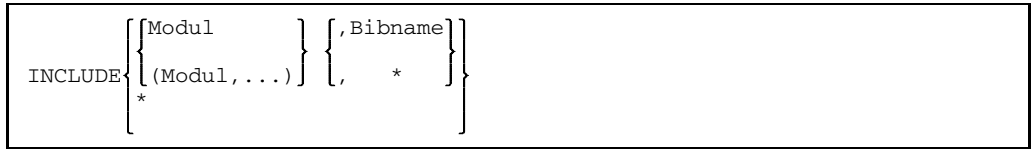
START = P$START

Nur nötig, falls als erster Modul ein Nicht-PL/I-Modul eingebunden wird.

Die nicht genannten PROGRAM-Parameter haben die übliche Wirkung oder sind für PL/I-Programme ohne Bedeutung.

INCLUDE-Anweisung

Die INCLUDE-Anweisung holt einen oder mehrere Bindemoduln aus der spezifizierten Bibliothek und fügt sie dem Lademodul bei. Diese Anweisung muß angegeben werden.



Modul	Name des Bindemoduls, der dem Lademodul beigefügt werden soll
Bibname	Name einer Bindemodulbibliothek, aus der die Moduln zu entnehmen sind
*	OMF-Datei des laufenden Prozesses

Achtung

Die Anweisung INCLUDE * bewirkt, daß alle in der OMF-Datei vorhandenen Bindemoduln zusammengebunden werden, auch wenn sie aus verschiedenen aufeinanderfolgenden Übersetzungen ggf. unterschiedlicher Übersetzer stammen. Es ist deshalb angebracht vor einer Übersetzung /ERASE auszuführen.

RESOLVE-Anweisung

Die RESOLVE-Anweisung gibt eine Bindemodulbibliothek an, in der der Binder versuchen soll, externe Referenzen aufzulösen. Die Bindemoduln aus der angegebenen Bibliothek, welche die externen Referenzen befriedigen, werden dann ebenfalls dem Lademodul beigefügt. Diese Anweisung ist nur bei Bedarf anzugeben.

```
          ⎧Externreferenz          ⎫
RESOLVE[  ⎨                        ⎬],Bibliotheksname
          ⎩(Externreferenz,...)    ⎭
```

Externreferenz

Name einer externen Referenz, die aufgelöst werden soll. Werden keine Referenzen spezifiziert, so versucht der Binder, alle Definitionen bisher unbefriedigter Referenzen in der angegebenen Bibliothek zu finden.

Bibliotheksname

Der Name der Bibliothek, in der der Binder nach den passenden Definitionen suchen soll.

Hinweis

Sind mehrere RESOLVE-Anweisungen angegeben, so werden diese von hinten nach vorn abgearbeitet. Es werden also die offenen Externbezüge zunächst in der zuletzt angegebenen Bibliothek gesucht, dann in der vorletzten usw. Bleiben nach der Abarbeitung aller RESOLVE-Anweisungen noch Externbezüge offen, wird zum Schluß in der Datei [$TSOS.]TASKLIB gesucht.

Anmerkung

Werden alle Bibliotheksmoduln von PLI1 in der Bibliothek $TSOS.TASKLIB gehalten, so sind keine diesbezüglichen RESOLVE-Angaben des Anwenders notwendig. Richtet der Systemverwalter für PLI1 jedoch eine eigene Bibliothek ein, um z.B. den Katalog der Systembibliothek klein zu halten, so ist deren Name dem Binder über eine RESOLVE-Anweisung ohne explizite Modulnamen bekannt zu machen. Der Bibliotheksname ist in solchen Fällen beim Systemverwalter zu erfragen.

END-Anweisung

Die END-Anweisung schließt die Anweisungen an den Binder ab und muß angegeben werden.

```
END
```

4.3 Beispiel für Binder

Es werden 3 PL/I-Prozeduren übersetzt und anschließend gebunden. Sie enthalten Externbezüge auf die Prozeduren FUNKT1 und FUNKT2, welche in der Bibliothek FUNKLIB stehen. Beispiele für das Eintragen von Prozeduren in Bibliotheken findet man in den Abschnitten 2.2.4 und 3.6.4.

```
/SYSFILE SYSDTA=(SYSCMD)                    ┐
/EXEC $PLI1                                 │
*COMOPT SOURCE=PL1A,OPTIONS=MAIN            │
*END                                        │
/EXEC $PLI1                                 │
*COMOPT SOURCE=PL1B                         ┘ 1)
*END                                        │
/EXEC $PLI1                                 │
*COMOPT SOURCE=PL1C                         │
*END                                        ┘

/EXEC $TSOSLNK
PROGRAM PRO05,FILENAM=PL1.PROGR.05      2)
INCLUDE *                               3)
RESOLVE,FUNKLIB                         4)
END
/EXEC PL1.PROGR.05                      5)
   .
   .
   .
```

1) Die erzeugten Bindemoduln - sie mögen ebenfalls PL1A, PL1B und PL1C heißen - werden in die EAM-Datei geschrieben. Eventuell entsteht je übersetzter Prozedur zusätzlich ein sogenannter Static-Modul, dessen Name aus dem Prozedurnamen abgeleitet wird (siehe Abschnitt 4.6). Die Haupt-Prozedur muß an der ersten Stelle stehen.

2) Angabe des Lademodulnamens und der katalogisierten Datei, in welche der Lademodul geschrieben werden soll.

3) Anweisung, alle Moduln aus der EAM-Datei zusammenzubinden.

4) Die externen Referenzen FUNKT1 und FUNKT2 sollen mit Definitionen in Bindemoduln aus der Datei FUNKLIB aufgelöst werden.

5) Ausführen des übersetzten und gebundenen Programms. PL1.PROGR.05 ist die katalogisierte Datei, in die der Lademodul PRO05 geschrieben wurde.

4.4 Laden

Der Lader lädt den vom Binder erzeugten Lademodul.

Genaue Informationen über den Lader findet man im Manual Dienstprogramme [3] bzw. bei Einsatz einer BS2000 Version ab V 8.0 im Manual Binder und Lader [12]. Über die Steuerung des geladenen und gestarteten Programms durch RUNOPT gibt Kapitel 5 Auskunft.

Der Aufruf des Laders erfolgt durch das Kommando

```
/LOAD Dateiname[,TIME=Zahl]
```

Dateiname — Name der katalogisierten Datei, die den Lademodul enthält. Es ist jener Name anzugeben, welcher in der PROGRAM-Steueranweisung für den Binder verwendet wurde.

Zahl — Die maximale CPU-Zeit in Sekunden, die dieses Programm laufen soll. Wenn dieser Parameter nicht angegeben ist, wird die als Systemstandard festgelegte Zeit genommen.

Will man ein PL/I-Programm mit DTH-Anweisungen (AT-Kommando) beeinflussen, so kann die Kommandofolge:

```
/LOAD     Dateiname
/AT...
/RESUME
```

zum Laden und Ausführen des Programms verwendet werden. Die DTH-Anweisungen können sich dabei nur auf virtuelle Adressen beziehen, die man aus der Offsetliste bzw. dem Objektcode-Protokoll (LIST = ASSM beim Übersetzen) und dem Binderprotokoll errechnen kann, und die beim Binden bzw. beim Eintragen in die Bibliothek das Kennzeichen TRAITS READONLY = N erhalten haben. Einzelheiten siehe Abschnitt 9.8.

4.5 Laufzeitsystem

Für PL/I-Programme stehen zwei Laufzeitsysteme zur Verfügung, die im Programmlauf identisch sind, die sich aber in Bezug auf Binden, Laden und in Bezug auf Speicherbedarf und Rechenzeit unterschiedlich verhalten.

Der wesentliche Unterschied liegt darin, daß im einen Fall der größte Teil der Bindemoduln zu zwei Großmoduln vorgebunden sind und diese erst beim Lauf des Programms dynamisch angebunden werden. Dies bewirkt, daß der Bindevorgang wesentlich weniger Zeit benötigt und der Lademodul kleiner wird. Diese Zusammenhänge sind im Bild 4-1 in einer Übersicht skizziert.

Jedes der beiden Laufzeitsysteme ist in sich abgeschlossen. Beide verwenden die gleichen elementaren Bindemoduln. Welches der beiden Laufzeitsysteme verwendet wird, hängt davon ab, ob auf Grund der Binderstrategie, wie sie in den vorangehenden Abschnitten beschrieben ist, zuerst der Verbindungsmodul ITP#AOS# oder ITP#AOD# gefunden wird.

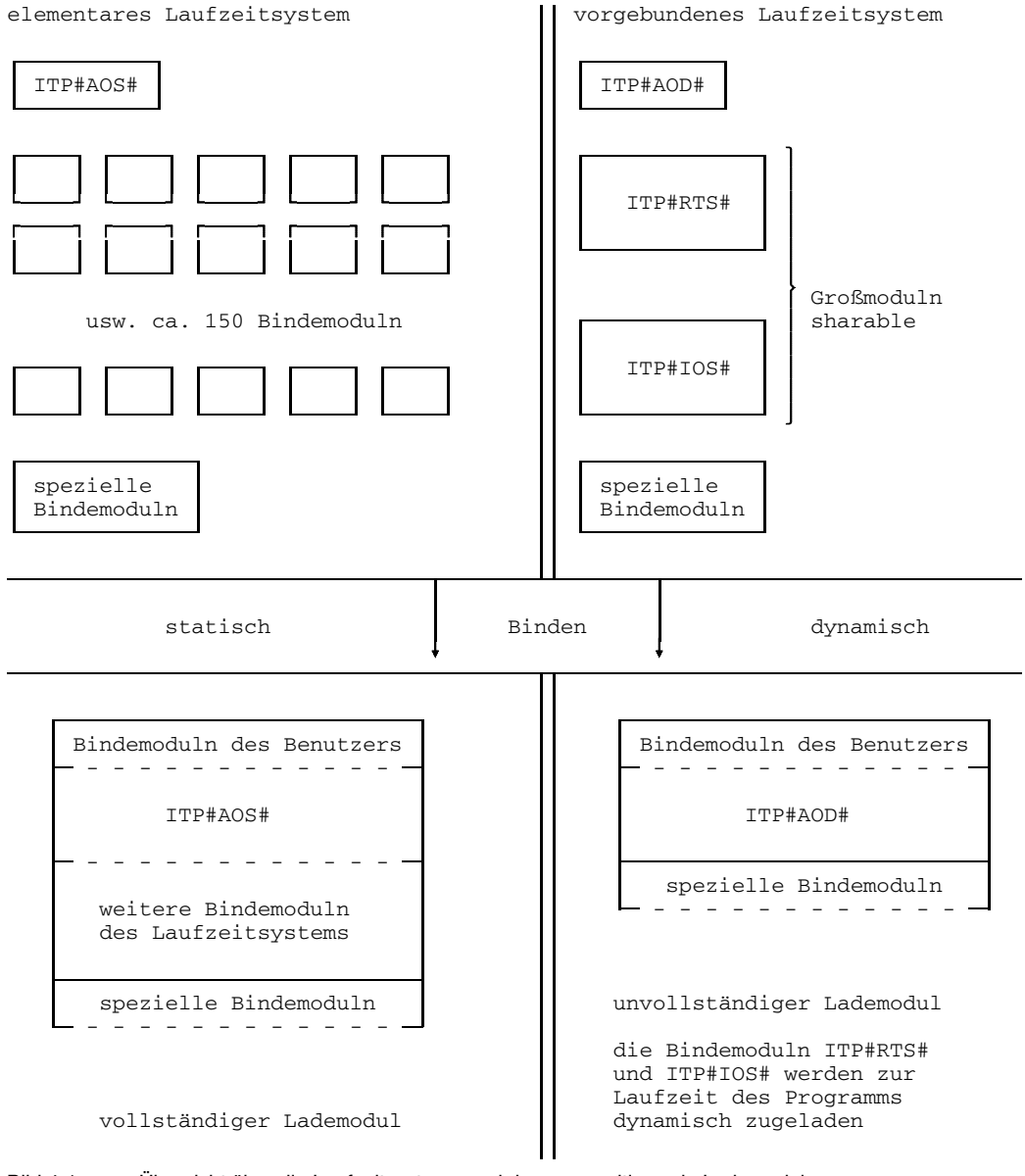


Bild 4-1 Übersicht über die Laufzeitsysteme und daraus resultierende Lademoduln

4.5.1 Elementares Laufzeitsystem

Beim elementaren Laufzeitsystem sind alle elementaren Bindemoduln einzeln vorhanden. Folgende Bindemoduln sind vorhanden:

- Verbindungsmodul ITP#AOS#
- etwa 150 weitere elementare Bindemoduln
- spezielle Bindemoduln
  (Diese sind hier lediglich deswegen getrennt aufgeführt, weil sie beim vorgebundenen Laufzeitsystem besonders behandelt werden.)

Beim Bindevorgang werden auf Grund des Verbindungsmoduls ITP#AOS# die vom Objektprogramm benötigten Bindemoduln angebunden. So entsteht ein vollständiger Lademodul, der alle vom Quellprogramm geforderten Leistungen enthält. Für den Ladevorgang und für den Lauf wird der Lademodul stets als Einheit betrachtet.

Für jedes Quellprogramm ist ein vollständiger Binde- und Ladevorgang erforderlich. Beim Binden wird ein vollständiges Bindeprotokoll erzeugt.

4.5.2 Vorgebundenes Laufzeitsystem

Beim vorgebundenen Laufzeitsystem sind die meisten der elementaren Bindemoduln zu zwei Großmoduln vorgebunden. Damit umfaßt das Laufzeitsystem folgende Bindemoduln:

- Verbindungsmodul ITP#AOD#
- Großmoduln ITP#RTS# und ITP#IOS#
  Diese Moduln können in die "Share-Tabelle" eingetragen werden.
- Spezielle Bindemoduln
  Hierzu gehören Bindemoduln für den Aufruf des Sortierprogramms, usw. Diese Moduln werden nicht vorgebunden, sondern bleiben elementare Bindemoduln.

Alle Moduln mit Ausnahme von ITP2SRT# und den Moduln für Sprachtransfer sind "reentrant" programmiert.
Beim Bindevorgang werden auf Grund des Verbindungsmoduls ITP#AOD# die Bindemoduln, die aus der Übersetzung der PL/I-Prozeduren hervorgegangen sind, mit dem Modul ITP#AOD# verbunden. Ferner werden die speziellen Bindemoduln angebunden, soweit sie erforderlich sind. Die Summe dieser Bindemoduln bilden einen unvollständigen Lademodul. Da die beiden Großmoduln noch nicht angebunden sind, ist der Umfang dieses Lademoduls gering; er benötigt weniger Speicherplatz als ein vollständiger Lademodul. Der Bindevorgang umfaßt nur wenige Bindemoduln und benötigt daher nur sehr wenig Zeit.

Die für den Lauf des Programms erforderlichen Großmoduln werden erst nach dem Start des Programms dynamisch zugeladen. Da die Moduln bereits vorgebunden sind, ist hierfür nur ein geringer Zeitaufwand erforderlich.

Die Großmoduln sind "reentrant" programmiert und können daher vom Systemverwalter in die "Share-Tabelle" des Betriebssystems eingetragen werden. In diesem Fall wird die für das dynamische Zuladen erforderliche Zeit weiter reduziert. Durch die gemeinsame Benutzung der Großmoduln durch mehrere PL/I-Programme kann auch der Speicherbedarf günstiger sein.
Da beim Bindevorgang nur wenige Module zu binden sind, gibt es auch nur ein eingeschränktes Bindeprotokoll. Wird für die Fehlerursache ein vollständiges Bindeprotokoll erforderlich, so muß mit dem elementaren Laufzeitsystem gearbeitet werden.

4.5.3 Lagerung der Laufzeitsysteme

Welches der Laufzeitsysteme an der Anlage zur Verfügung steht, wird durch den Systemverwalter festgelegt und kann vom Benutzer nicht direkt gesteuert werden. In vielen Fällen kann es von Vorteil sein, wenn beide Laufzeitsysteme gleichzeitig zur Verfügung stehen.

Zweckmäßigerweise wird der Systemverwalter das bevorzugte Laufzeitsystem (bzw. das einzige Laufzeitsystem) in die Systemdatei TASKLIB lagern. Hierbei ist für die Steuerung des Bindevorgangs vom Benutzer der geringste Aufwand erforderlich.

Das zweite Laufzeitsystem kann in eine Bibliothek gelegt werden. Bei der Steuerung des Bindevorgangs sind dann die entsprechenden Angaben (RESOLVE) zu machen.

4.6 Namenskonventionen für PL/I-Bindemoduln

In den Steueranweisungen des Binders können Modulnamen angegeben werden. In den Protokollen des Binders werden die Namen der Moduln und die Namen anderer externer Programmgrößen aufgelistet. Diese Namen ergeben sich direkt aus den entsprechenden Bezeichnern der gebundenen PL/I-Prozeduren, wobei jedoch wegen Beschränkungen im Binder und anderen Programmiersprachen folgende Regeln zu beachten sind.

- Bei Eingangs-Konstanten, die mit OPTIONS (ASSEMBLER oder COBOL) vereinbart sind, werden stets die ersten 8 Zeichen verwendet.
- Von allen anderen Bezeichnern mit OPTIONS (PLI1 oder FORTRAN oder VARIABLE) und dem Attribut EXTERNAL werden im erzeugten Bindecode höchstens 7 Zeichen abgelegt, bei zu langen Bezeichnern sind das die ersten 4 und die letzten 3 Zeichen.
- Hat ein Bezeichner des PL/I-Programms zusätzlich zu EXTERNAL das Attribut ENTRY (Prozedurname, Eingang), so werden außerdem alle Zeichen "_" durch das Zeichen "$" ersetzt.

Die Übersetzung einer PL/I-Prozedur erzeugt in der Regel zwei Bindemoduln, nämlich einen Code- und Datenmodul. Der Datenmodul wird immer dann erzeugt, wenn die PL/I-Prozedur Variable mit dem Attribut STATIC enthält. Diese Moduln werden nach folgenden Regeln benannt:

- Der Codemodul erhält den eventuell auf 7 Zeichen gekürzten und mit $ gemäß obiger Regel modifizierten Namen des ersten (primären) Bezeichners mit dem Attribut ENTRY. Das ist der erste Name des äußersten Prozedur-Blockes einer einzeln übersetzten Prozedur.
- Der Name des Datenmoduls wird aus dem Namen des Codemoduls gewonnen, wozu dieser immer auf 8 Zeichen ergänzt wird. Hierfür wird der Name zunächst mit dem Zeichen "@" auf 7 Stellen aufgefüllt, wenn er kürzer war. Das 8. Zeichen ist stets eine der Ziffern "1" bis "7", die angibt, wie lang der ursprüngliche Name war.

Man beachte, daß nur in den Modulnamen und Eingängen die Ersetzung von "_" durch "$" erfolgt, nicht für sonstige Bezeichner mit dem Attribut EXTERNAL. Für letztere gilt nur die Verkürzungsregel.

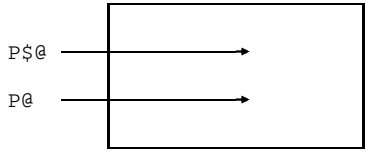
Beispiel

Das Programm lautet:

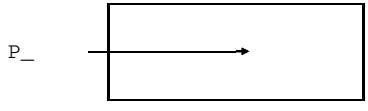
```
P_@ : PROCEDURE;
P@  : ENTRY;
DCL P_STATIC EXTERNAL INIT (0);
END;
```

Es liefert folgende Moduln:

Name Codemodul: P$@



Name Datenmodul: P$@@@@@3



Für die Haupt-Prozedur (MAIN) wird der externe Name P$START# erzeugt. Er bezeichnet die Stelle, an der das PL/I-Programm gestartet wird.

4.7 Erweiteter Adreßraum (XS)

Bei Rechnern deren Adreßraum größer als 16 M-Bytes ist (bis 2 G-Bytes) sprechen wir von XS-Systemen. Soll der größere Adreßraum von einem PL/I-Programm ausgenutzt werden, so müssen vom Übersetzer Module erzeugt werden, die nicht nur Adressen von 24 Bits verwenden, sondern Adressen von 31 Bits. Solche Module können vom PLI1-Übersetzer der Version 4.0A erzeugt werden. Danach gibt es folgende Modul-Typen:

alte Module	die von PLI1-Übersetzern bis zur Version 4.0A erzeugt wurden
NXS-Module	die vom PLI1-Übersetzern ab der Version 4.0A erzeugt werden, wenn *COMOPT OPTIONS=NXS angegeben wurde (Voreinstellung).
XS-Module	die vom PLI1-Übersetzer ab Version 4.0A erzeugt werden, wenn *COMOPT OPTIONS=XS.

Für das Binden gilt nun:

- alte Module und NXS-Module sind miteinander verträglich und können gemischt gebunden werden. Das Ergebnis ist entweder ein NXS-Lademodul oder ein NXS-Großmodul, der wiederum wie ein NXS-Modul weiter gebunden werden kann.
- XS-Module sind mit NXS-Modulen und alten Modulen nicht verträglich: Es dürfen also XS-Module nur mit XS-Modulen gebunden werden. Das Ergebnis ist entweder ein XS-Lademodul oder ein XS-Großmodul, der wiederum wie ein XS-Modul weiter gebunden werden kann.

Diese Regeln gelten sowohl für statisches als auch für dynamisches Binden.

Die Module des PLI1-Laufzeitsystems sind sowohl mit XS- als auch mit NXS-Modulen verträglich.

Beim Bindevorgang wird durch den Parameter LOADPT die Ladeadresse des Programmes und damit seine Lage im Speicherbereich bestimmt. Hier gilt folgendes:

- NXS-Lademodule können in jedem Fall im Adressenraum bis 16 M-Byte geladen werden: sei es nun in einem System mit erweiterten Adressenraum oder in einem System ohne erweiterten Adressenraum.
- XS-Lademodule können in beiden Systemen unbeschränkt laufen, d.h. im System mit oder ohne erweiterten Adressenraum.

Für die Einhaltung der Regeln ist der Benutzer selbst verantwortlich. Beim Binden können bestimmte Prüfungen gefordert werden. Siehe hierzu den Operanden

```
ARMODE-CHECK
```

sowie den Abschnitt "XS-Unterstützung" im Manual [12] Binder und Lader ab Version 21.0B (BS2000 Version 9.0).

5 Ausführung des PL/I-Programms

5.1 Allgemeines

Nachdem der vom Binder erzeugte Lademodul des PLI1-Anwenderprogramms in einer Datei abgelegt wurde, kann er mit dem EXECUTE-Kommando geladen und ausgeführt werden. Ähnlich wie beim PL/I-Compiler können dem gestarteten Ladeprogramm Steueranweisungen übergeben werden.

Bevor ein PL/I-Programm ausgeführt wird, müssen folgende Zuordnungen getroffen werden:

- Die im Programm benötigten Dateien müssen über FILE- oder CHANGE-Kommandos eingerichtet bzw. bei bestehenden Dateien dem Programm zugeordnet werden. Alle dafür notwendigen Kenntnisse werden im Kapitel 6 vermittelt.
- Die Steueranweisungen (*RUNOPT) müssen bereitgestellt werden, falls man das Programm über diese beeinflussen möchte. In diesem Fall ist zusätzlich der Prozeßschalter 1 zu setzen.
- Die Meldungstextdateien müssen wie bei der Übersetzung zur Verfügung stehen (siehe Abschnitt 3.2).

Nach dem Start des Programms werden vom Laufzeitsystem die Steueranweisungen von SYSDTA eingelesen. Bei Angabe der Steueranweisung LIST = OPTIONS werden die wirksamen Steueranweisungen nach SYSLST ausgegeben. Daneben erscheinen in der Systemdatei SYSOUT alle Meldungen, die während des Objektlaufs vom Laufzeitsystem erzeugt werden.

Mit Beendigung des Programmlaufs wird die Meldung

```
END OF PROGRAM Name, RTS v.www-ari, TIME USED: xxxxx.xx SEC
```

nach SYSOUT ausgegeben.

Die Angaben haben folgende Bedeutung:

Name	Name des Lademoduls aus der PROGRAM-Anweisung des Binders. Er wird nicht ausgegeben, wenn das Programm mit EXEC * oder EXEC (modul) gestartet wurde.
x	Verbrauchte Zeit in Sekunden
v.w	Versions-Nummer des Laufzeitsystems
a	Änderungs-Kennziffer des Moduls ITP#AOS# des elementaren Laufzeitsystems bzw. ITP#AOD# des vorgebundenen Laufzeitsystems.
r	Änderungs-Kennziffer des Moduls ITP#RTS# des vorgebundenen Laufzeitsystems bzw. 0 beim elementaren Laufzeitsystem.
i	Änderungs-Kennziffer des Moduls ITP#IOS# des vorgebundenen Laufzeitsystems bzw. 0 beim elementaren Laufzeitsystem.

Siehe dazu auch den Abschnitt 4.5. Beim vorgebundenen Laufzeitsystem wird geprüft, ob die Versions-Nummern der Module zueinander passen. Ist dies nicht der Fall, so wird eine Fehlermeldung gegeben und der Lauf abgebrochen.

Wird der Lauf mit Fehler beendet, so werden die nachfolgenden Kommandos bis zum STEP- oder LOGOFF-Kommando ignoriert.

5.2 Programmausführung

5.2.1 Ausführung mit ISP-Kommando

Das als Lademodul in einer Datei vorliegende PL/I-Programm wird mit dem EXECUTE-Kommando gestartet. Das Programm wird über den in der PROGRAM-Anweisung des Binders festgelegten Dateinamen referiert.

Form des Aufrufes:

```
/{EXECUTE}
 {       }Dateiname [,TIME=t][,MONJV=jvname]
 {EXEC   }
```

Das Betriebssystem läßt beim EXEC-Kommando weitere Parameter zu, die jedoch z.Zt. für PL/I-Programme wirkungslos sind oder zu Fehlern führen können. Die Angabe des TIME-Parameters ist sinnvoll, um unnötigen Zeitverbrauch des Programms beim Auftreten von Programmfehlern zu verhindern.

5.2.2 Ausführung mit SDF-Kommando

Ein als Lademodul in einer Datei vorliegendes PL/I-Programm kann alternativ mit dem SDF-Kommando

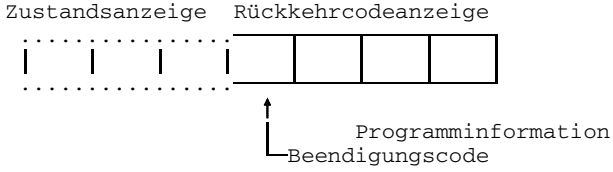
```
/START-PLI1-PROGRAM
```

mit zusätzlichen Operanden gestartet werden. Die Operanden und ihre Bedeutung sind in diesem Kapitel in Abschnitt 5.7 beschrieben.

Die Eingabe von SDF-Kommandos und deren Operanden im ungeführten und geführten Dialog ist ausführlich im Manual "Einführung in die Dialog-Schnittstelle SDF" beschrieben.

5.2.3 Überwachung durch Monitorjobvariable

Zur Überwachung des Objektlaufs kann durch die Angabe 'MONJV=jvname' im EXECUTE-Kommando eine Monitorjobvariable (überwachende Jobvariable) spezifiziert werden. Die Zustandsanzeige (Zeichen 1 bis 3) der Jobvariablen wird vom System gesetzt (Werte '$R␣','$T␣','$A␣'). Die Rückkehrcodeanzeige (Zeichen 4 bis 7) wird beim Ende des Objektlaufs besetzt und ist wie folgt aufgebaut:



Der Beendigungscode wird wie folgt besetzt:

'0' Normales Programmende
'2' Fehlerende (z.B. Error-Condition aufgetreten)
'3' Fataler Fehler im PLI1-Laufzeisystem

Die Programminformation wird mit '000' besetzt. Durch Aufruf der Bibliotheks-Prozedur PLIRETC (siehe Kapitel 11 Dienstleistungen) kann die Programminformation auf andere Werte (zwischen '000' und '999') gesetzt werden.

Diese Leistung setzt das Softwareprodukt JV voraus.

5.2.4 Beispiele

Abhängig davon, ob Steueranweisungen und ob Dateien für das PL/I-Programm benötigt werden, gibt es verschiedene Alternativen, von denen in den nachfolgenden Beispielen einige gezeigt werden:

Beispiel 1

Das PL/I-Programm enthält keine Ein-/Ausgabedateien; Ausgaben erfolgen nach SYSLST. Der Hauptprozedur (MAIN) des Programms soll ein Datum als Parameter übergeben werden.

Folgende Kommandos und Steueranweisungen sind einzugeben:

```
  .
  .
  .
/SETSW ON=1
/EXEC PROG1
*RUNOPT ARGUMENT='24.12.78'
*END
  .
  .
  .
```

Beispiel 2

Das Programm benötigt eine Eingabedatei mit dem PL/I-Namen (TITLE) PLIEIN und eine Ausgabedatei mit dem PL/I-Namen PLIAUS. Darüber hinaus werden keine Steuerungen vorgesehen. Die Kommandos haben folgendes Aussehen:

```
  .
  .
  .
/FILE EINGABE,LINK=PLIEIN
/FILE AUSGABE,LINK=PLIAUS,FCBTYPE=SAM,
/RECSIZE=160,BLKSIZE=STD,RECFORM=V
/EXEC PROG3
  .
  .
  .
```

Beispiel 3

Die Steueranweisungen für den Objektlauf stehen in der Datei OPT.

Die Kommandos haben folgenden Aufbau:

```
  .
  .
  .
/SYSFILE SYSDTA=OPT
/SETSW ON=1
/EXEC PROG2
/SYSFILE SYSDTA=(PRIMARY)
/RESUME
    Daten
  .
  .
  .
```

Datei OPT:

```
*RUNOPT LIST = OP,
*RUNOPT MESSAGE = S,
*RUNOPT DUMP = ST
*END/
```

Mit der Steueranweisung *END/ wird ein Breakpoint gesetzt. Vor dem RESUME-Kommando wird das Kommando

```
/SYSFILE SYSDTA=(PRIMARY)
```

gegeben, um anschließend Daten von SYSDTA einlesen zu können.

Beispiel 4

In diesem Beispiel erfolgt die Ausgabe über die PL/I-Datei PLIAUS, die Eingabedaten für die Datei PLIEIN sollen der Systemdatei SYSDTA entnommen werden.

Die Kommandos haben folgenden Aufbau:

```
  .
  .
  .
/SETSW ON=(1)
/FILE AUSGABE,LINK=PLIAUS...
/EXEC PROG4
*RUNOPT SYSFILE=SYSDTA(PLIEIN)
*END
   Daten
  .
  .
  .
```

Beispiel 5

Das Programm enthält PLIEIN und PLIAUS als Eingabe- bzw. Ausgabe-Datei. Zusätzlich sollen Daten für die PL/I-Datei mit dem Title A über die Systemdatei SYSDTA eingelesen werden und Ausgabedaten der PL/I-Datei mit dem Title B sollen in die Systemdatei SYSLST ausgegeben werden. Wegen dieser Forderung muß mit der Steueranweisung SYSFILE eine neue Zuordnung für die Systemdateien getroffen werden.
Alle verwendeten Steueranweisungen sollen bei Programmbeginn aufgelistet werden.

Die Kommandos haben folgenden Aufbau:

```
   .
   .
   .
/SETSW ON=(1)
/FILE EINGABE,LINK=PLIEIN...
/FILE AUSGABE,LINK=PLIAUS...
/EXEC PROG5
*RUNOPT SYSFILE=(SYSDTA(A),SYSLST(B)),
*RUNOPT LIST=OPTIONS
*END
   Daten
   .
   .
   .
```

Anmerkung

Die TITLE-Angaben SYSPRINT und SYSIN dürfen in diesem Programm nicht vorkommen.

5.3 Steuerung des PL/I-Programms

Ähnlich wie für den Compiler gibt es auch für das Ladeprogramm eine Steuermöglichkeit. Es sind dies die Steueranweisungen ARGUMENT, DUMP, FORMAT, LIST, MESSAGE, STORAGE, SYSFILE und TRACE.

5.3.1 Allgemeine Regeln für Steueranweisungen

Die Regeln über die Bereitstellung und den Aufbau der Steueranweisungen des Anwenderprogramms entsprechen denen des PL/I-Compilers. Man findet diese Regeln unter Abschnitt 3.3.1. Unterschiede bestehen lediglich in dem einleitenden Schlüsselwort - hier *RUNOPT - und einer nur teilweise gleichen Menge von Steueranweisungen.
Das Format der Steueranweisungen für das Anwenderprogramm hat folgendes Aussehen:

```
*RUNOPT Steueranweisung...
```

Die Eingaben werden wie beim Compiler durch *END oder *END/ abgeschlossen. Alle weiteren Regeln sind dem oben genannten Abschnitt zu entnehmen.
Wie beim Compiler werden die Steueranweisungen über SYSDTA eingelesen. Voraussetzung dafür ist, daß zuvor mit dem Kommando SETSW der Prozeßwahlschalter 1 gesetzt wurde.

5.3.2 Fehlerbehandlung bei der Auswertung der Steueranweisungen

Die Fehlerbehandlung während der Abarbeitung der Steueranweisungen für das Anwenderprogramm ist identisch mit der im Compiler.
Daher gelten die unter Abschnitt 3.3.2 stehenden Aussagen.

5.3.3 Steuerung der Protokollausgabe

Während des Programmlaufes werden Meldungen und Listen je nach Art der Ausgabe nach SYSLST oder SYSOUT ausgegeben. Der Benutzer kann über verschiedene Steueranweisungen die Ausgaben zusätzlich auf eine zweite Systemdatei zu leiten. Auf diese Weise steuerbar sind nur die Ausgaben des Laufzeitsystems, nicht jedoch durch PUT-, DISPLAY- oder WRITE-Anweisungen programmierte Ausgaben.

Gegenstand der Ausgabe im Laufzeitsystem	Ausgabedatei		
		Zusatzausgabe	
	Voreinstellung	auf Datei	gesteuert durch
- Liste der aktiven Steueranweisungen - Programmstatistik	SYSLST	SYSOUT	LIST=TERMINAL (im Stapelbetrieb wirkungslos)
- Procedure- und Sprung-Überwachung (TRACE)	SYSLST	SYSOUT	TRACE = TERMINAL (im Stapelbetrieb wirkungslos)
- Condition-Meldung - Rückverfolger (SNAP) - Programm-Unterbrechung - Fehler im Laufzeitsystem - Endemeldung - Prozedur ERROUT	SYSOUT	SYSLST	MESSAGE = SYSLST

Da SYSLST durch das SYSFILE-Kommando auf eine Benutzerdatei geleitet werden kann, besteht eine weitere Steuerungsmöglichkeit (siehe Abschnitte 3.3.4 und 6.2.1).

5.3.4 Übersicht über Steueranweisungen für PL/I-Programme

Die folgenden Abschnitte geben einen Überblick über die Steueranweisungen des Programms. Die meisten Steueranweisungen können abgekürzt werden. Bei den Einzelbeschreibungen sind die relevanten Buchstaben unterstrichen. Bei einigen Parametern kann eine verneinende Form (z.B. TERMINAL und NOTERMINAL) angegeben werden. Die in der Spalte Wirkung stehende Kurzbeschreibung bezieht sich auf die positive Parameterangabe.

Steueranweisung f. Objektprogramm	Abk.	Bedeutung	Spezifikation	Abk.	Wirkung	Voreinstellung
ACTIVE	ACT	Kontrollpunkte aktivieren	YES NO	YES NO	Aktivieren nicht aktivieren	YES
ARGUMENT	ARG	Übergabe einer Zeichenfolge an die MAIN-Prozedur des Programms auf Parameterposition.	'Die Zeichenfolge kann bis zu 253 Stellen lang sein und wird in Hochkommata eingeschlossen'.		Die Interpretation der Zeichenfolge bleibt dem Benutzer überlassen.	'' (leere Zeichenfolge)
CONTROL	CTL	Steuerung	STD NO ALL ALIGN NOALIGN	STD NO ALL AL NAL	}entspricht NOALIGN entspricht ALIGN bei Ausrichtungsfehler Befehlssimulation bei Ausrichtungsfehler ERROR mit ONCODE=8098	 NAL

Steueranweisung f. Objektprogramm	Abk.	Bedeutung	Spezifikation	Abk.	Wirkung	Voreinstellung
DUMP	DMP	Ausgabe von Dumps und Rückverfolgung	ALL	ALL	entspricht (ST,A,SN, R (X'0',X'7FFFFF'))	NO
			NO	NO	entspricht (NST,NA, NR,NSN)	
			STACK NOSTACK	ST NST	Dump des gesamten Stapel-Speichers	
			AREA NOAREA	A NA	Dump der gesamten Standard-Area	
			RANGE(a,e) NORANGE	R NR	Dump eines Bereichs von a bis e. Adresse in Sedezimalform.	
			SNAP NOSNAP	SN NSN	Ausgabe der Prozedurschachtelung bei Programmende.	
			COND UNCOND	C UC	Am Programmende alle obigen Angaben nur im Fehlerfall bzw. immer.	C
FORMAT	FM	Steuerung der Zeilenzahl pro Seite, der Zeilenlänge, Sprache für Meldungen	PRINTER ([m_1][,m_2])	P	Steuerung der Ausgaben aus SYSLST und im Stapelbetrieb auch auf SYSOUT.m_1= Anz.Zeilen/Seite. m_2=Zeilenlänge.	P(64, 132)
			TERMINAL ([n_1][,n_2])	T	Steuerung der Ausgaben auf Datenstation: n_1 = Anzahl Zeilen n_2 = Anzahl Zeichen.	T(O,k) 1)
			ENGLISH DEUTSCH	E D	Alle Ausgaben des Laufzeitsystems erfolgen in englischer oder deutscher Sprache.	E

1) k steht für die physikalische Zeilenlänge des aktuellen Ausgabegerätes

Steueranweisung f. Objektprogramm	Abk.	Bedeutung	Spezifikation	Abk.	Wirkung	Voreinstellung
LIST	LST	Steuerung der Listen (es werden hier nur die Parameter beschrieben, die zur Objektzeit von Bedeutung sind)	ALL NO OPTIONS NOOPTIONS SUMMARY NOSUMMARY TERMINAL NOTERMINAL	ALL NO OP NOP SM NSM T NT	entspricht (OP,SM) entspricht (NOP,NSM) Liste der wirksam gewordenen Steueranweisungen Ausgabe einer Programmstatistik Kopie aller Listen des Laufzeitsystems auf der Datenstation	 NOP NSM NT
MESSAGE	MSG	Steuerung Meldungen des Laufzeitsystems	SYSLST NOSYSLST	S NS	Meldungen zusätzlich auf SYSLST Meldungen nur auf SYSOUT	NS
STORAGE	STR	Möglichkeit, die Speicherverwaltung des Laufzeitsystems zu beeinflussen und Optimierungen zu erreichen.	AREA ([q_1][,[q_2][,[q_3]]])	A	Für den Standardbereich wird zunächst ein Speicherbereich von q_1 Seiten bereitgestellt. Notwendige Erweiterungen erfolgen in Schritten von q_2 Seiten bis zu einer Maximalgröße von q_3 Seiten. Danach wird ggf. STORAGE-Bedingung gesetzt. (Seite = 4KB)	A(16, 16,oo)
			STACK ([s_1][,s_2])	S	Der Stapel-Speicher wird in Segmenten zu s_1 Seiten belegt. Für Fehler und Endebehandlung im Fall von Speichermangel wird eine Reserve von s_2 angelegt. (Seite = 4KB)	S(16,4)

Steueranweisung f. Objektprogramm	Abk.	Bedeutung	Spezifikation	Abk.	Wirkung	Voreinstellung
SYSFILE	SFL	Zuordnung der PL/I-TITLE zu den BS2000 Systemdateien. Ferner Änderungen der Voreinstellung für PAGESIZE und LINESIZE	SYSLST (title)	SL	Die Datei mit dem PL/I-TITLE 'title' wird auf die Systemdatei SYSDTA, SYSOUT bzw. SYSLST abgebildet.	SL (SYSPRINT)
			SYSOUT (title)	SO		SO (SYSOUT)
			SYSDTA (title)	SD		SD (SYSIN)
			LINESIZE (l_1,l_2,l_3)	LS	l_1,l_2,l_3 überschreiben, falls sie nicht 0 sind, die Voreinstellungen von LINESIZE für SYSLST, SYSOUT und SYSDTA	LS(0,0)
			PAGESIZE(p_1,p_2)	PS	p_1 und p_2 überschreiben, falls sie nicht 0 sind, die Voreinstellungen von SIZE für SYSLST und SYSOUT	PS(0,0)
		sowie Steuerung für die DISPLAY-Anweisung	DISPLAY(SYSOUT) dia = SYSOUT	DP(SO)	Im Dialogbetrieb wird die Ausgabe für die Anweisung DISPLAY auf die Dialogstation geleitet, die Eingabe für DISPLAY REPLY von der Dialogstation erwartet.	DP(SO)
			DISPLAY(SYSCON) dia = SYSCON	DP(SC)	Im Dialogbetrieb wird die Ausgabe für die DISPLAY Anweisung auf die Operateur-Konsole gesteuert, die Eingabe für DISPLAY Reply von der Operateur-Konsole entgegengenommen.	

Steueranweisung f. Objektprogramm	Abk.	Bedeutung	Spezifikation	Abk.	Wirkung	Voreinstellung
SYSFILE (Fortsetzung)			DISPLAY(SYSDTA) dia = SYSDTA	DP(SD)	Im Dialogbetrieb wird die Ausgabe für die DISPLAY-Anweisung auf die Dialogstation eingestellt, die Eingabe über DISPLAY REPLY aus der Systemdatei SYSDTA gelesen.	
			DISPLAY(SYSOUT) bat = SYSOUT	DP(SO)	Im Stapelbetrieb wird die Ausgabe für DISPLAY in die Systemdatei SYSOUT geleitet, die Eingabe für DISPLAY REPLY von der Systemdatei SYSDTA angenommen.	
			DISPLAY(SYSCON) bat = SYSCON	DP(SC)	Im Stapelbetrieb wird die Ausgabe für die DISPLAY-Anweisung auf die Operateur-Konsole gesteuert, die Eingabe für DISPLAY REPLY von der Operateur-Konsole entgegengenommen.	
			DISPLAY(SYSDTA) bat = SYSDTA	DP(SD)	Im Stapelbetrieb wird die Ausgabe für DISPLAY in die Systemdatei SYSOUT gebracht. Die Eingabe für DISPLAY REPLY wird von der Systemdatei SYSDTA angenommen.	

Steueranweisung f. Objektprogramm	Abk.	Bedeutung	Spezifikation	Abk.	Wirkung	Voreinstellung
TABULATOR	TAB	Tabulatoren setzen	(n,...)		n werden Tabulatorpositionen	(1,11, 21,31, 41,usw.)
TRACE	TRC	Ein- und ausschalten der TRACE-Angaben	ALL	ALL	≙(P,L,C,G,R)	NO
			NO	NO	≙(NP,NL,NC,NG,NR)	
			PROCTRACE	P	Ein- bzw. ausschalten der Ablaufverfolgung für PROCEDURE-Eingänge	NP
			NOPROCTRACE	NP		
			LABTRACE	L	Marken (Label)	NL
			NOLABTRACE	NL		
			CALLTRACE	C	CALL-Aufrufe	NC
			NOCALLTRACE	NC		
			GOTOTRACE	G	GOTO-Sprünge	NG
			NOGOTOTRACE	NG		
			RETURNTRACE	R	RETURN-Rückkehr	NR
			NORETURNTRACE	NR		
			TERMINAL	T	zusätzliche Ausgabe der Traces auf der Datenstation	T
			NOTERMINAL	NT		

5.4 Einzelbeschreibung der Steueranweisungen für PL/I-Programme

Bei den Steueranweisungen für das Anwenderprogramm unterscheidet man zwei Gruppen.

- Die reinen Objektzeit-Steueranweisungen ARGUMENT, CONTROL, DUMP, SYSFILE, TABULATOR, TRACE und STORAGE. Diese Steueranweisungen sind nur zur Laufzeit des Anwenderprogramms verwendbar.
- Die für Übersetzer und Anwenderprogramm im gleichen Sinne gültigen Steueranweisungen FORMAT, LIST und MESSAGE. Sie können in beiden Fällen für den gleichen Zweck verwendet werden, wobei jedoch die zulässigen Spezifikationen teilweise unterschiedlich sind.
  Wird ein Quellprogramm übersetzt und im gleichen Prozeß anschließend ausgeführt, so müssen selbstverständlich auch die gemeinsamen Steueranweisungen separat für den Übersetzungslauf und für die Ausführung des Programms angegeben werden.
  Nachfolgend werden die für das Benutzerprogramm gültigen Steueranweisungen beschrieben.

5.4.1 Parameterübergabe (ARGUMENT)

Die Steueranweisung ARGUMENT bietet die Möglichkeit, eine Zeichenfolge von bis zu 253 Zeichen anzugeben, die unverändert (einschließlich Leerstellen) der MAIN-Prozedur des Programms auf Parameterposition übergeben wird. In der MAIN-Prozedur ist der Parameter mit:

```
DCL Name CHAR(n) VARYING;
```

zu vereinbaren, wobei n ≤ 253 sein muß.

Die Interpretation der übergebenen Zeichenfolge bleibt dem Benutzerprogramm überlassen. Die Zeichenfolge ist in Apostrophe einzuschließen. Apostrophe, die Bestandteil der Zeichenfolge sein sollen, müssen durch zwei Apostrophe dargestellt werden. Die Angabe mehrerer einzelner Zeichenfolgen ist nicht sinnvoll, da nur die letzte wirksam wird.

ARGUMENT = 'Zeichenfolge'

Voreinstellung:

ARGUMENT = '' (leere Zeichenfolge)

5.4.2 Dump-Steuerung (DUMP)

Die Steueranweisung DUMP legt fest, ob am Programmende Dumps und Rückverfolgungsliste (SNAP) auszugeben sind.

```
DUMP = { Spezifikation          }
       { (Spezifikation,...)    }
```

Voreinstellung: DUMP = (NO, C)

Als Spezifikation können verwendet werden:

NO ALL	entspricht: (NST,NA,NR,NSN) entspricht: (ST,A,R(X'0',X'7FFFFF'),SN)
STACK NOSTACK	Dump des gesamten Stapel-Speichers.
AREA NOAREA	Dump der gesamten Standard-Area.
RANGE (a,e) NORANGE	Dump eines Bereichs von der Anfangsadresse a bis zur Endadresse e. Die Adressen sind in Sedezimalform (X'...') anzugeben.
SNAP NOSNAP	Ausgabe der aktuellen Prozedurverschachtelung am Programmende.
COND UNCOND	Alle Informationen, die aufgrund der Steueranweisung ST, A, R und SN gewünscht sind, werden am Programmende ausgegeben, und zwar: – nur im Fehlerfall (C) – immer (UC)

5.4.3 Ausgabeformen (FORMAT)

Die Steueranweisung FORMAT bestimmt die Zeilenzahl pro Seite und die jeweilige Zeilenlänge für Ausgaben auf dem Drucker und auf der Datenstation. Daneben kann angegeben werden, in welcher Sprache (Deutsch oder Englisch) die Meldungen des Laufzeitsystems ausgegeben werden sollen.

Die einzelnen Spezifikationen der Steueranweisung FORMAT sind unter Abschnitt 3.5.4 beschrieben.

Alle Spezifikationen der Steueranweisung FORMAT wirken nicht auf Ausgaben des PL/I-Programms, die durch PUT-, WRITE- oder DISPLAY-Anweisungen zustande kommen.

5.4.4 Listenauswahl (LIST)

Durch die Steueranweisung LIST wird festgelegt, welche Listen des Laufzeitsystems ausgegeben und wohin sie ausgegeben werden sollen. Diese Steuerung wirkt nicht auf Ausgaben, welche aufgrund von PUT-, WRITE- oder DISPLAY-Anweisungen erfolgen.

```
LIST = { Spezifikation          }
       {                        }
       { (Spezifikation,...)    }
```

Voreinstellung: LIST = (NOP, NSM, NT)

Mögliche Spezifikationen sind:

ALL	entspricht: (OP,SM)
NO	entspricht: (NOP,NSM)
OPTIONS NOOPTIONS	Auflistung aller für das Programm wirksamen Steueranweisungen auf SYSLST.
SUMMARY NOSUMMARY	Ausgabe einer Programmstatistik auf SYSLST, die enthält: – maximale Belegung des Stapelspeichers (Stack) durch AUTOMATIC-Variable usw. – maximale Belegung des allgemeinen Speichers für Bereiche (Standard-Area) aufgrund von ALLOCATE-Anweisungen
TERMINAL NOTERMINAL	Im Dialog zusätzliche Ausgabe obiger Listen auf Datenstation (SYSOUT), im Stapelbetrieb wirkungslos.

5.4.5 Zusatzausgabe von Meldungen (MESSAGE)

Durch diese Steueranweisung kann festgelegt werden, ob Meldungen des Laufzeitsystems zusätzlich auf SYSLST ausgegeben werden sollen. Ausgaben des Anwenderprogramms mittels PUT-, WRITE- oder DISPLAY-Anweisung werden durch MESSAGE nicht beeinflußt, jedoch Ausgaben der Prozedur ERROUT.

```
MESSAGE = { SYSLST   }
          {          }
          { NOSYSLST }
```

Voreinstellung: MSG = NS

SYSLST: Die Meldungen des Laufzeitsystems werden im Dialogbetrieb zusätzlich auf SYSLST ausgegeben.

NOSYSLST:
Die Meldungen werden nur auf SYSOUT ausgegeben.

5.4.6 Speicherbedarf (STORAGE)

Mit der Steueranweisung STORAGE kann man die Speicherverwaltung des Programms beeinflussen. Festgelegt werden können Anfangsgröße und Verlängerungen des virtuellen Speichers für Standard-Area und Stapelspeicher (Stack).
In der Standard-Area werden CONTROLLED- und BASED-Variable sowie Ein-/Ausgabebereiche angeordnet. Der Belegungszustand wird daher vom Programm durch die Anweisungen ALLOCATE/FREE und OPEN/CLOSE bestimmt. Im Stapelspeicher werden AUTOMATIC-Variable, Hilfsvariable und sichergestellte Registerzustände angeordnet. Sein Belegungszustand wird vor allem durch die Anzahl der aktiven Blöcke und die in diesen Blöcken vereinbarten Variablen bestimmt. Die Speicherstatistik (LIST=SUMMARY) gibt Hinweise für die optimale Einstellung der Größen von Standard-Area und Stapelspeicher.

```
STORAGE = { Spezifikation          }
          {                        }
          { (Spezifikation,...)    }
```

Voreinstellung: STORAGE=(AREA(16,16,oo),STACK(16,4))

Spezifikationsmöglichkeiten:

AREA([q_1][,[q_2][,[q_3]]]):

Für die Standard-Area wird zunächst ein Speicherbereich von q_1 virtuellen Seiten zu je 4KB bereitgestellt. Notwendige Erweiterungen folgen in Schritten von q_2 Seiten bis zu einer Maximalgröße von q_3 Seiten. Danach wird ggf. STORAGE-Bedingung gesetzt.

Je nach Größe des zulässigen Benutzer-Adreßraums ist für die Standard-Area eine Speicherforderung bis zu maximal 1500 (6 MB) Seiten sinnvoll.

STACK([s_1][,s_2]):

Der Speicher für den Stapel wird in Segmenten zu je s_1 virtuellen Seiten zu je 4KB belegt. Für Fehler- und Endbehandlung bei Speichermangel wird eine Reserve von s_2 Seiten angelegt.
Der Maximalwert hängt ab von der Größe des zulässigen Benutzer-Adressenraums.

5.4.7 Zuordnung der Systemdateien (SYSFILE)

Durch die Steueranweisung SYSFILE wird festgelegt, welche PL/I-TITLE auf die BS2000-Systemdateien SYSDTA, SYSLST und SYSOUT führen sollen. Ferner können die Voreinstellungen für PAGESIZE und LINESIZE für diese Dateien verändert werden.
Außerdem wird festgelegt, welche Systemdateien für die DISPLAY-Anweisung zugeordnet werden sollen.

```
SYSFILE = ⎧Spezifikation       ⎫
          ⎩(Spezifikation,...) ⎭
```

Voreinstellung:

```
SYSFILE = (SYSDTA(SYSIN),SYSLST(SYSPRINT),SYSOUT(SYSOUT),
          PAGESIZE(undef,undef),LINESIZE(undef,undef,undef),DISPLAY(SYSOUT))
```

Werte können sein:

```
SYSDTA(title-1)  Die Dateien mit dem PL/I-TITLE title-1, title-2
SYSLST(title-2)  und title-3 werden auf die Systemdateien SYSDTA,
SYSOUT(title-3)  SYSLST bzw. SYSOUT abgebildet.
```

PAGESIZE(p_1,p_2)

falls Werte für p_1 oder p_2 angegeben sind, werden die Voreinstellungen des Programms (z.B. Einstellung durch OPEN-Anweisung) für PAGESIZE bei SYSLST bzw. SYSOUT überschreiben. p_1 ist SYSLST, p_2 ist SYSOUT zugeordnet. Die Angabe "0" für p_1 oder p_2 wirkt wie "undef".

LINESIZE(l_1,l_2,l_3)

wenn Werte für l_1, l_2 oder l_3 angegeben sind, werden die Voreinstellungen des Programms (z.B. Einstellung durch OPEN-Anweisung) für LINESIZE bei SYSLST, SYSOUT oder SYSDTA überschrieben. l_1 ist SYSLST, l_2 ist SYSOUT und l_3 ist SYSDTA zugeordnet. Die Angabe "0" für l_1, l_2 oder l_3 wirkt wie "undef".

dia = SYSOUT

Im Dialogbetrieb wird die Augabe für die Anweisung DISPLAY auf die Dialogstation geleitet, die Eingabe für DISPLAY REPLY von der Dialogstation erwartet.

dia = SYSSON

Im Dialogbetrieb wird die Ausgabe für die DISPLAY-Anweisung auf die Operateur-Konsole gesteuert, die Eingabe für DISPLAY REPLY von der Operateur-Konsole entgegengenommen.

dia = SYSDTA

Im Dialogbetrieb wird die Ausgabe für die DISPLAY-Anweisung auf die Dialogstation eingestellt, die Eingabe über DISPLAY REPLY aus der Systemdatei SYSDTA gelesen.

bat = SYSOUT

Im Stapelbetrieb wird die Ausgabe für DISPLAY in die Systemdatei SYSOUT geleitet, die Eingabe für DISPLAY REPLY von der Systemdatei SYSDTA angenommen.

bat = SYSCON

Im Stapelbetrieb wird die Ausgabe für die DISPLAY-Anweisung auf die Operateur-Konsole gesteuert, die Eingabe für DISPLAY-REPLY von der Operateurkonsole entgegengenommen.

bat = SYSDTA

Im Stapelbetrieb wird die Ausgabe für DISPLAY in die Systemdatei SYSOUT gebracht. Die Eingabe für DISPLAY REPLY wird von der Systemdatei SYSDTA angenommen.

Fehlt die Angabe bat, so wird die Angabe dia für die Angabe bat übernommen.

5.4.8 Trace-Steuerung (TRACE)

Die Steueranweisung bewirkt das Ein- bzw. Ausschalten der im Programm vorgesehenen TRACE-Ausgaben (siehe hierzu auch die Steueranweisung DEBUG im Abschnitt 3.6.2).

```
TRACE = { Spezifikation       }
        {                     }
        { (Spezifikation,...) }
```

Voreinstellung: TRACE = (NO, T)

ALL entspricht: (P,L,C,G,R)

NO entspricht: (NP,NL,NC,NG,NR)

PROCTRACE NOPROCTRACE	Ein- bzw. Ausschalten der Ablaufverfolgung für: PROCEDURE-Eingänge Marken (Label) CALL-Aufrufe GOTO-Sprünge RETURN-Rückkehr
LABTRACE NOLABTRACE	
CALLTRACE NOCALLTRACE	
GOTOTRACE NOGOTOTRACE	
RETURNTRACE NORETURNTRACE	

TERMINAL
NOTERMINAL

Im Dialog zusätzliche Ausgabe der Überwachung auf der Datenstation (SYSOUT). Im Stapelbetrieb wirkungslos.

5.4.9 Tabulatoren setzen (TABULATOR)

Bei der Ausgabe von Datenelementen durch die Anweisung PUT LIST wird jedes Element auf der nachfolgenden Tabulatorposition begonnen. Durch die Steueranweisung TABULATOR können die Tabulatorpositionen verändert werden.

TABULATOR = (n,...)

Die Werte von n müssen ganze Zahlen ≥ 1 sein. Jede folgende Zahl muß größer als die vorhergehende sein.

Voreinstellung: TABULATOR = (1, 11, 21, 31, 41, usw.).

5.4.10 Steuerung (CONTROL)

Mit dieser Steueranweisung kann festgelegt werden, wie bei Ausrichtungsfehlern verfahren werden soll.

```
CONTROL = { Spezifikation          }
          {                        }
          { (Spezifikation,...)    }
```

Voreinstellung: CONTROL = NOALIGN

Folgende Spezifikationen sind möglich:

STD entspricht NOALIGN

NO entspricht NOALIGN

ALL entspricht ALIGN

ALIGN Tritt im Objektprogramm ein Ausrichtungsfehler auf, so wird die notwendige Ausrichtung simuliert und das Programm normal fortgesetzt.

Achtung
Die Simulation ist sehr rechenzeitaufwendig und sollte nur zur Erleichterung des Programmtests eingesetzt werden.

NOALIGN Treten im Objektprogramm Ausrichtungsfehler auf, so wird die Bedingung ERROR gesetzt. Der zugehörige ONCODE-Wert ist 8098.

5.5 Kontrollereignisse (ACTIVE)

Durch die Steueranweisung werden Kontrollpunkte, die durch *COMOPT DEBUG = BREAKPOINT (x,...) eingearbeitet werden für den Lauf des Programms aktiviert und damit wirksam.

```
ACTIVE = { YES }
         {     }
         { NO  }
```

Voreinstellung: ACT = YES

5.6 Programmunterbrechung

Der Lauf eines PL/I-Programms, der an der Dialogstation gestartet wurde, kann vom Benutzer unterbrochen werden. Dazu wird an der Dialogstation die ESCAPE- bzw. BREAK-Funktion ausgelöst (bei den Sichtgeräten 8160 und 8161 z.B. durch die Taste K2). Das Programm bleibt dann geladen und auf der Dialogstation wird ein Schrägstrich ausgegeben, der anzeigt, daß Kommandos eingegeben werden können. Von speziellem Interesse sind hier folgende Kommandos:

- /RESUME
  Der Programmlauf wird an der Unterbrechungsstelle fortgesetzt.
- /INTR Text
  Der "Text" wird an das PL/I-Programm übergeben.
  Wird "/INTR * STOP" angegeben, so führt dies zum sofortigen Ende des Programmlaufs.
  Wurde bei der Übersetzung nicht explizit eine Unterbrechungsbehandlung eingearbeitet, so werden andere Texte ignoriert und der Programmlauf an der Unterbrechungsstelle fortgesetzt.
- /TCHNG
  Es können logische Eigenschaften der Datenstation geändert werden.
- /EOF
  Dateneingabe an der Datenstation beenden.

Weitere Einzelheiten siehe der Beschreibung der Kommandos [2].

Wurde der Programmlauf in einer DO-Prozedur gestartet, so führt diese Unterbrechung zur Anfrage, ob die Prozedur abgebrochen werden soll. Wird sie abgebrochen, so bleibt das Programm geladen und es kann wie oben verfahren werden. Es ist jedoch zu beachten, daß die restlichen Kommandos der DO-Prozedur nicht ausgeführt werden.

Zusätzlich zu dem oben beschriebenen Unterbrechungs-Mechanismus ist es möglich, das PL/I-Programm an bestimmten Stellen zu unterbrechen, um den durch das Kommando "/INTR Text" übergebenen Text im PL/I-Programm auszuwerten. Innerhalb des PL/I-Programms kann mit Hilfe der Bedingung ATTENTION entschieden werden, welche Aktion auf Grund des Kommandos INTR (interrupt) und des ggf. mitgelieferten Textes durchgeführt wird.

Um das Kommando INTR auswerten zu können, muß die externe PL/I-Prozedur mit

```
*COMOPT OPTIONS = INTERRUPT
```

übersetzt werden. Es werden dann in diese externen Prozeduren Unterbrechungsstellen eingefügt, an denen abgefragt wird, ob ein unerledigtes Kommando INTR vorhanden ist. Dies geschieht an folgenden Stellen:

- Hinter jeder Marke

- Vor jeder Anweisung END
- Vor jeder Anweisung RETURN

Wird an einer der Unterbrechungsstellen festgestellt, daß ein INTR-Kommando ansteht, so wird die Bedingung ATTENTION gesetzt. Außerdem wird der beim Kommando INTR angegebene Text an die ON-Variable ONINTR übergeben. Er steht über die eingebaute Funktion ONINTR dem Benutzer zur Anwendung zur Verfügung. Wird kein Text angegeben, so ergibt der Aufruf von ONINTR die leere Folge. Die Syntax des Textes ist beim Kommando INTR [2] beschrieben.

Die vollständige Beschreibung der Bedingung ATTENTION und der eingebauten Funktion ONINTR ist in der PL/I-Sprachbeschreibung [1] zu finden. Die Steuerung des Übersetzers durch *COMOPT OPTIONS=INTERRUPT ist im Abschnitt 3.6 erläutert.

Im Bild 5-1 ist eine Übersicht gegeben über den Programmablauf bei der Unterbrechung. Ein Beispiel für die ON-Einheit ist bei der Beschreibung der Bedingung ATTENTION zu finden.

Bild 5-1 Unterbrechung des Programmlaufs durch eine BREAK-Funktion an der Datenstation und Datenstation und Eingabe des Kommandos INTR

Wird der Programmlauf wegen der Eingabe /INTR STOP abgebrochen, so wird

auf SYSOUT folgendes protokolliert:

- Eine Unterbrechungs-Meldung
- Registerstände
- Dump der Befehlsumgebung

Es wird die Bedingung FINISH (nicht ERROR oder ATTENTION) gesetzt.

Eine Meldung zur Bedingung ATTENTION kann durch den Aufruf CALL ERROUT ausgegeben werden; durch die System-Einheit wird keine Meldung ausgegeben.

5.7 Beschreibung der Operanden des SDF-Kommandos START-PLI1-PROGRAM

5.7.1 Übersicht über die Operanden

Name des Operanden	Zweck
FROM-FILE	Bestimmen des Namens und der Eingabequelle des Objektmoduls bzw. Lademoduls
CPU-LIMIT	Maximale Programmablaufzeit in CPU-Sekunden
MONJV	Überwachen des Programmablaufs mit Jobvariablen
START-PARAMETERS	Parametereingabe für das MAIN-Programm
LANGUAGE	Ausgabe der Programmmeldungen in Englisch oder Deutsch
ASSIGN-SYSLST ASSIGN-SYSOUT ASSIGN-SYSDTA	Zuweisen von PL/I-TITLE
LISTING	Steuern der Listenausgabe des Laufzeitsystems
HEAP-ADMINISTRATION	Steuern der Speicherverwaltung der Standardarea
STACK-ADMINISTRATION	Steuern der Speicherverwaltung des Stapelspeichers
TABULATOR-POSITION	Tabulatorpositionen für die Ausgabe mit PUT LIST

5.7.2 Einzelbeschreibung der Operanden

FROM-FILE-Operand

Dieser Operand weißt den Namen und die Eingabequelle des Objekts zu, das zum Ablauf gebracht werden soll.

```
FROM-FILE = <full-filename 1..54 without gen> / *MODULE(...) /
            *PHASE(...)

      *MODULE(...)
        |
        |  LIBRARY = *OMF / <full-filename 1..54 without gen>
        |
        | ,ELEMENT = *ALL / <full-filename 1..54 without gen-vers>

      *PHASE(..)
        |
        |  LIBRARY = <full-filename 1..54 without gen>
        |
        | ,ELEMENT = <full-filename 1..54 without gen-vers>(...)
        |     |
        |     |  VERSION = *HIGHEST-EXISTING / <alphanum-name 1..24>
```

FROM-FILE = <full-filename 1..54 without gen>
<full-filename> ist der Name der katalogisierten Datei, die den mit TSOSLNK erzeugten Lademodul enthält.

FROM-FILE = *MODULE(...)
Mit den Parametern der *MODULE-Struktur wird ein vom Compiler erzeugter Objektmodul bzw. ein mit dem TSOSLNK vorgebundener Großmodul zugewiesen. Diese Moduln sind in PLAM-Bibliotheken als Elemente vom Typ R abgespeichert.

FROM-FILE = *PHASE(...)
Mit den Parametern der *PHASE-Struktur wird ein mit dem TSOSLNK erzeugter Lademodul zugewiesen, der als Element vom Typ C in einer PLAM-Bibliothek abgespeichert ist.

Siehe auch START-PROGRAM-Kommando in den Manualen "Benutzerkommandos (SDF-Format)" und "Binder-Lader-Starter".

CPU-LIMIT-Operand

Dieser Operand spezifiziert die maximale CPU-Zeit in Sekunden, die die Ausführung des Programms dauern kann.

```
CPU-LIMIT = JOB-REST / <integer 1..32767>
```

MONJV-Operand

Dieser Operand spezifiziert den Namen einer Jobvariablen, die den Programmablauf überwacht.

```
MONJV = *NONE / <full-filename 1..54 without gen>
```

START-PARAMETERS-Operand

Mit diesem Operanden kann eine Zeichenfolge als Parameter an das MAIN-Programm übergeben werden.

```
START-PARAMETERS = ' ' / <c-string 1..253>
```

LANGUAGE-Operand

Dieser Operand legt die Sprache fest, in der die Meldungen des Moduls/Programms ausgegeben werden.

```
LANGUAGE = ENGLISH / DEUTSCH
```

ASSIGN-SYSLST-Operand

Dieser Operand ordnet PL/I-TITLE der Systemdatei SYSLST zu.

```
ASSIGN-SYSLST = STD / PARAMETERS(...)

     PARAMETERS(...)
       |  TO-TITLE = SYSPRINT / <alphanum-name 1..8>
       | ,MAX-LINE-SIZE = 132 / <integer 1..256>
       | ,LINES-PER-PAGE = 60 / <integer 1..256>
```

ASSIGN-SYSOUT-Operand

Dieser Operand weist PL/I-TITLE der Systemdatei SYSOUT zu.

```
ASSIGN-SYSOUT = STD / PARAMETERS(...)

     PARAMETERS(...)
       |  TO-TITLE = SYSOUT / <alphanum-name 1..8>
       | ,MAX-LINE-SIZE = 120 / <integer 1..256>
       | ,LINES-PER-PAGE = 60 / <integer 1..256>
```

ASSIGN-SYSDTA-Operand

Dieser Operand weist PL/I-TITLE der Systmedatei SYSDTA zu.

```
ASSIGN-SYSDTA = STD / PARAMETERS(...)

     PARAMETERS(...)
       |  TO-TITLE = SYSIN / <alphanum-name 1..8>
       | ,MAX-LINE-SIZE = 120 / <integer 1..256>
```

TEST-SUPPORT-Operand

Dieser Operand steuert die Testhilfe.

```
TEST-SUPPORT = NONE / PARAMETERS(...)

     PARAMETERS(...)
       |
       |  TOOL-SUPPORT = NONE / AID
       |
       | ,TEST-POINT-INTERRUPT = NO / YES
       |
       | ,DUMP(DMP) = NONE / ON-ERROR(...) / ON-TERMINATION(...)
       |
       |       ON-ERROR(...)
       |         |  PROCEDURE-NEST = YES / NO
       |         | ,HEAP-STORAGE = NO / YES
       |         | ,STACK-STORAGE = NO / YES
       |
       |       ON-TERMINATION(...)
       |         |  PROCEDURE-NEST = YES / NO
       |         | ,HEAP-STORAGE = NO / YES
       |         | ,STACK-STORAGE = NO / YES
       |
       | ,TRACE(TRC) = NONE / PARAMETERS(...)
       |
       |       PARAMETERS(...)
       |         |  PROCEDURE-ENTRY = YES / NO
       |         | ,PROCEDURE-EXIT = YES / NO
       |         | ,PROCEDURE-CALL = YES / NO
       |         | ,LABELLED-STATEMENT = YES / NO
       |         | ,GOTO-STATEMENT = YES / NO
       |         | ,ADDITIONAL-OUTPUT = *NONE / TERMINAL
```

TEST-SUPPORT = NONE
Keine Testhilfeunterstützung

TEST-SUPPORT = PARAMETERS

TOOL-SUPPORT = NONE / AID
Das Programm wir ohne (NONE) bzw. mit (AID) LSD-Sätzen geladen.

TEST-POINT-INTERRUPT = NO / YES
Aktivieren von Testpunkten

DUMP = NONE / ON-ERROR(...) / ON-TERMINATION(...)
Steuern der DUMP-Ausgabe

DUMP = NONE
Keine Dumpausgabe

DUMP = ON-ERROR(...)
Dumpausgabe nur bei abnormaler Programmbeendigung

PROCEDURE-NEST = YES / NO
Ausgabe der Prozedurverschachtelungen bei Programmende

HEAP-STORAGE = NO / YES
Dump der gesamten Standardarea

STACK-STORAGE = NO / YES
Dump des gesamten Stapelspeichers

DUMP = ON-TERMINATION(...)
Dumpausgabe bei normaler und abnormaler Programmbeendigung.
Zu den Parametern der ON-TERMINATION-Struktur siehe ON-ERROR-Struktur.

LISTING-Operand

Dieser Operand steuert die Listenausgabe des Laufzeitsystems.

```
LISTING(LST) = NONE / PARAMETERS(...)

     PARAMETERS(...)
       |  OPTIONS = NO / YES
       | ,SUMMARY = NO / YES
       | ,ADDITIONAL-OUTPUT = *NONE / *TERMINAL
```

LISTING = NONE
Keine Listenausgabe

LISTING = PARAMETERS(...)

OPTIONS = NO / YES
YES: Ausgabe der wirksamen Steueranweisungen.

SUMMARY = NO / YES
YES: Ausgabe der Programmstatistik

ADDITIONAL-OUTPUT = *NONE / *TERMINAL
*TERMINAL: Die Ausgabe der Listen erfolgt zusätzlich auf der Datensichtstation.

HEAP-ADMINISTRATION-Operand

Dieser Operand steuert die Speicherverwaltung der Standardarea und ermöglicht eine Optimierung.

```
HEAP-ADMINISTRATION = STD / PARAMETERS(...)

     PARAMETERS(...)
       |
       |  PRIMARY-ALLOCATION = 16 / <integer 1..524288>
       |
       | ,SECONDARY-ALLOCATION = 16 / <integer 1..524288>
       |
       | ,MAXIMAL-SIZE = 512 / <integer 1..524288>
```

HEAP-ADMINISTRATION = STD
Es gelten die Standarwerte der folgenden PARAMETERS-Struktur.

HEAP-ADMINISTRATION = PARAMETRS(...)

PRIMARY-ALLOCATION = 16 / <integer 1..524288>
Anfangsgröße

SECONDARY-ALLOCATION = 16 / <integer 1..524288>
Verlängerung

MAXIMAL-SIZE = 512 / <integer 1..524288>
Maximale Größe

STACK-ADMINISTRATION-Operand

Dieser Operand steuert die Speicherverwaltung des Stapelspeichers und ermöglicht eine Optimierung.

```
STACK-ADMINISTRATION = STD / PARAMETERS(...)

     PARAMETERS(...)
       |
       |  PRIMARY-ALLOCATION = 16 / <integer 1..524288>
       |
       | ,SECONDARY-ALLOCATION = 4 / <integer 1..524288>
```

STACK-ADMINISTRATION = STD
Es gelten die Standardwerte der folgenden PARAMETERS-Struktur.

STACK-ADMINISTRATION = PARAMETERS(...)

PRIMARY-ALLOCATION = 16 / <integer 1..524288>
Anfangsgröße

SECONDARY-ALLOCATION = 4 / <integer 1..524288>
Verlängerung

TABULATOR-POSITION-Operand

Dieser Operand legt die Tabulatorpositionen für die Ausgabe von Datenelementen durch die Anweisung PUT LIST fest. Voreingestellt sind Tabulatoren mit der Schrittweite 10 (1, 10, 21, ...)

```
TABULATOR-POSITION = EVERY-TEN / list-poss: <integer 1..256>
```

5.7.3 Abbildung der SDF-Operanden auf die RUNOPT-Operanden

SDF-Operand	**RUNOPT-Operand**
FROM-FILE	1)
CPU-LIMIT	1)
MONJV	1)
START-PARAMETERS	ARGUMENT
LANGUAGE	FORMAT
ASSSIGN-SYSLST=STD	SYSFILE=SYSLST(SYSPRINT)
ASSIGN-SYSOUT=STD	SYSFILE=SYSLST(SYSOUT)
ASSIGN-SYSDTA=STD	SYSFILE=SYSLST(SYSIN)
TEST-SUPPORT	
TOOL-SUPPORT	1)
TEST-POINT-INTERRUPT	ACTIVE
DUMP PROCEDURE-NEST=YES HEAP-STORAGE=YES STACK-STORAGE=YES	DUMP= SNAP AREA STACK
TRACE PROCEDURE-ENTRY=YES PROCEDURE-EXIT=YES PROCEDURE-CALL=YES LABELLED-STATEMENT=YES GOTO-STATEMENT=YES ADDITIONAL-OUTPUT=*TERMINAL	TRACE= PROCTRACE RETURNTRACE CALLTRACE LABTRACE GOTO-TRACE TERMINAL
LISTING	LISTING=
SUMMARY=YES	SUMMARY
OPTIONS=YES	OPTIONS
ADDITIONAL-OUTPUT=*TERMINAL	TERMINAL
HEAP-ADMINISTRATION	STORAGE=AREA()
STACK-ADMINISTRATION	STORAGE=STACK()
TABULATOR-POSITION=()	TABULATOR=()

1) Wird im EXECUTE-Kommando eingetragen

6 Dateizugriff durch PL/I-Programme

6.1 Allgemeines

In diesem Abschnitt werden die für PL/I-Programme wichtigen Eigenschaften und Bearbeitungsmöglichkeiten von BS2000-Dateien beschrieben und der Zusammenhang zu den entsprechenden PL/I-Sprachelementen erläutert. Ein PL/I-Programmierer muß beachten, daß er einzelne Sprachelemente von PL/I nur verwenden kann, wenn die angesprochene BS2000-Datei gewisse Eigenschaften besitzt, insbesondere muß der die Datei bildende Datenbestand dem Programm entsprechend strukturiert/organisiert sein.

Im nachfolgenden Abschnitt 6.2 werden die im BS2000 verfügbaren Dateien und deren Eigenschaften kurz erläutert. Abschnitt 6.3 gibt eine Übersicht über die logischen Dateien von PL/I-Programmen mit ihren Organisationsformen und der Umgebungsbeschreibung (ENVIRONMENT). Im Abschnitt 6.4 werden Probleme der Zuordnung von PL/I-Dateien zu BS2000-Dateien behandelt. Die Abschnitte 6.5 und 6.6 beschreiben wichtige Zusammenhänge bei der strom- und satzorientierten Ein- und Ausgabe in Abhängigkeit von Organisationsformen des BS2000 und geben Hinweise für die Angaben im FILE-Kommando.

6.2 Dateien des BS2000

In diesem Abschnitt werden die Dateiarten des BS2000 kurz beschrieben, soweit sie für PL/I-Programme von Bedeutung sind. Man unterscheidet hier System- und Benutzerdateien.

6.2.1 Systemdateien

Das System verwendet für bestimmte Funktionen Systemdateien. Für PL/I-Programme und sinngemäß auch für den Compiler sind von Bedeutung:

- SYSCMD
  Dieser Datei entnimmt das Betriebsystem alle Kommandos.
  Im Stapelbetrieb ist das standardmäßig die Einspooldatei (Lochkarteneingabe) oder eine Datei, die mit dem ENTER-Kommando aktiviert wurde. Im Dialog ist es standardmäßig die Datenstation. SYSCMD kann durch das DO-Kommando vorübergehend auf eine beliebige Datei umgeschaltet werden.
- SYSDTA
  Die Systemdatei SYSDTA kann nur als Eingabedatei verwendet werden. Standardmäßig werden im Stapelbetrieb über SYSDTA die Datensätze geliefert, welche dem EXEC-Kommando unmittelbar folgen. Dabei ist es gleichbedeutend, ob der Prozeß über einen Einspoolvorgang (Lochkarteneingabe) oder mit dem ENTER-Kommando aktiviert wurde. Begrenzt werden die Datensätze durch das nächste Kommando, wobei die Kommandos BREAK und EOF (siehe [2]) eine spezielle Begrenzung bilden.
  Im Dialogbetrieb ist SYSDTA standardmäßig die Datenstation des Benutzers, wobei in der Regel die Datensätze einzeln (zeilenweise) von dort abgerufen werden.
  Steht das EXEC-Kommando in einer Prozedur-Datei und sollen die anschließenden Datensätze über SYSDTA verarbeitet werden, so muß SYSDTA mit der Systemdatei SYSCMD identisch sein (siehe SYSFILE-Kommando im Abschnitt 3.4.1 und [2]).
- SYSOUT
  Meldungen des Betriebssystems (Quittungen, Fehler usw.) und Meldungen des PL/I-Laufzeitsystems (auch vom Compiler) werden über die Systemdatei SYSOUT ausgegeben. Ferner können Ausgaben der Programme auf SYSOUT geleitet werden. Im Dialogbetrieb wird die Information, die in die Systemdatei SYSOUT geschrieben werden soll, auf der jeweiligen Datenstation ausgegeben. Im Stapelprozeß werden die Ausgaben in eine Spooldatei gespeichert und nach Prozeßende auf dem Schnelldrucker ausgegeben (ausgespult). Die Datei wird anschließend gelöscht.

- SYSLST
  Ausgaben der Programme werden in der Regel nach SYSLST geleitet, wenn im PL/I-Programm der Dateiname SYSPRINT verwendet wird. Alle Listen des Compilers werden standardmäßig ebenfalls in diese Systemdatei geleitet. Der Inhalt dieser Spooldatei wird nach Prozeßende auf dem Drucker ausgegeben und anschließend gelöscht.
- SYSOPT und SYSIPT
  Die Systemdateien SYSOPT und SYSIPT werden von PLI1 nicht unterstützt.

Mit dem Kommando SYSFILE (siehe Abschnitte 3.4.1 und 3.5) können den Systemdateien SYSDTA und SYSLST Benutzerdateien zugeordnet werden. Aus diesen werden dann die entsprechenden Eingaben entnommen oder dorthin in die entsprechenden Ausgaben geschrieben. Bild 6-1 zeigt die oben geschilderten Möglichkeiten.

Systemdatei	Voreinstellung (PRIMARY)		Beeinflussung durch Kommando:
	Stapel	Dialog	
SYSCMD	Einspooldatei	Datenstation	DO;ENDP
SYSDTA			SYSFILE;(ENDP)
SYSLST	Ausspooldatei	Ausspooldatei	SYSFILE
SYSOUT		Datenstation	-

Bild 6-1 Zuordnung von Systemdateien

Ferner ist durch die PLI1-Steueranweisung SYSFILE (siehe Abschnitt 5.4.7) eine Zuordnung zwischen Titel bzw. Dateinamen im Programm und den Systemdateien festgelegt.

Der Benutzer kann in gewissen Grenzen steuern, welche Ausgaben nach SYSLST bzw. SYSOUT geleitet werden sollen. Die Möglichkeiten dazu sind in den Kapiteln 3 und 5 beschrieben (Steueranweisungen MESSAGE und SYSFILE).
SYSLST und SYSOUT sind in einem Stapelprozeß Dateien für die Druckausgabe. In einem Dialogprozeß werden die Daten, die für den Drucker bestimmt sind, in die Systemdatei SYSLST geschrieben, dagegen wird die Ausgabe für die Datenstation in der Systemdatei SYSOUT abgelegt.
Gemeinsam gilt, daß die Systemdateien SYSDTA, SYSOUT und SYSLST nur sequentiell verarbeitbar sind, SYSDTA nur als Eingabedatei und SYSOUT und SYSLST nur als Ausgabedateien benutzt werden können. Systemdateien können in Verbindung mit der strom- oder satzorientierten Ein-/Ausgabe benutzt werden.

6.2.2 Benutzerdateien

6.2.2.1 Allgemeines

Während eine Systemdatei automatisch eingerichtet und gelöscht wird und der Benutzer somit keinen Einfluß auf die Dateieigenschaften hat (es sei denn er verwendet das SYSFILE-Kommando), kann er bei Benutzerdateien die wesentlichen Eigenschaften der Datei selbst bestimmen. Es gibt eine Vielzahl von Dateieigenschaften und Angaben zu einer Datei. Im Zusammenhang mit PL/I-Programmen können folgende Gruppen von Dateieigenschaften bzw. bearbeitungsspezifische Größen unterschieden werden:

- der Dateiname
- der Dateikettungsname
- die Dateiorganisation
- der Satzaufbau
- der Datenträger
- die Zugriffsberechtigung
- die Dateigröße
- sonstige Kenndaten.

Man kann alle Eigenschaften einer Datei mit dem FILE- bzw. CATALOG-Kommando angeben. Mit diesen Kommandos wird eine Datei eingerichtet und katalogisiert oder bei vorhandenen Dateien werden Angaben abgeändert oder ergänzt. Mit dem FILE-Kommando kann man darüber hinaus bearbeitungsspezifische Größen wie Dateikettungsname, Öffnungsmodus usw. festlegen.

Fast alle Eigenschaften/Kenndaten werden erst zum Zeitpunkt des Schließens der Datei (CLOSE) in den Katalog übernommen. Die Datei bleibt so lange erhalten, bis sie durch ein ERASE-Kommando wieder gelöscht wird. Bei bereits existierenden Dateien sind die meisten Eigenschaften schon bestimmt. Eine Veränderung ist dann nur noch eingeschränkt möglich.

Anmerkung

Anstelle der Festlegung durch das FILE-Kommando lassen sich einige Dateieigenschaften im PL/I-Programm mit dem ENVIRONMENT-Attribut festlegen.

6.2.2.2 Dateiname

Die physikalischen Dateien (Datenbestände) werden innerhalb des BS2000 durch einen Dateinamen identifiziert, der bis zu 54 Zeichen lang sein kann. Erstellt der Benutzer eine Datei, so wird vom BS2000 der Dateiname der erstellten Datei mit einer Benutzerkennung (max. 10 Zeichen) als Vorsatz versehen.

Der Dateiname wird als 1. Parameter im FILE-Kommando angegeben. Unter Verwendung der Benutzerkennung eines anderen Anwenders kann unter bestimmten Bedingungen mit dessen Dateien gearbeitet werden.

Die allgemeine Form für einen Dateinamen ist:

```
[$benken.]Name1[[.Name2]...[(⎧Generation⎫)]
                            ⎩Version   ⎭
```

$benken — ist eine Benutzerkennung und muß nur dann angegeben werden, wenn man auf eine Datei eines anderen Anwenders zugreifen will.

Name1 — ist der eigentliche Dateiname, wenn nicht Name2... folgt. Er ist der Name einer Kategorie von Dateien, wenn weitere Teilnamen folgen. Einige BS2000-Kommandos gestatten die Angabe von Datei-Kategorien (z.B. /ERASE); d.h. diese Kommandos verarbeiten bei einem Aufruf mehrere Dateien.
Dateinamen für den PL/I-Compiler (Quellprogramm) und in FILE- bzw. SYSFILE-Kommandos sind immer Namen von Einzel-Dateien; d.h. Kategorien können nicht angegeben werden.

```
(⎧Generation⎫)
 ⎩Version   ⎭
```

Mit diesen Angaben können gleichnamige Dateien weiter unterschieden werden. Generation ist eine absolute oder relative Zahlenangabe. Version wird für Banddateien verwendet und kann ein Name sein. Eine Besonderheit besteht darin, daß die Versionsangabe nicht mit im Kennsatz auf Band aufgezeichnet wird (siehe [7]).

*DUMMY — Symbol für eine Scheindatei mit folgenden Eigenschaften:

- Eingabe:
  Beim ersten Lesen wird ENDFILE signalisiert.
- Ausgabe:
  Die zu schreibenden Datensätze werden nur in den der Datei zugeordneten Puffer übertragen und gelangen nicht auf externe Träger. Da für Scheindateien kein Katalogeintrag vorhanden ist, müssen im FILE-Kommando alle Angaben gemacht werden, wie sie für eine neu einzurichtende Datei notwendig sind.

Beispiele für gültige Dateinamen

```
EINGABE
ERGEBNIS.0001
$MUELLER.UEBERGABE.TEXTE
```

Der letzte Dateiname bezieht sich auf die Datei UEBERGABE.TEXTE, die unter dem Benutzerkennzeichen $MUELLER mit SHARE = YES katalogisiert sein muß, wenn der Zugriff von einem fremden Benutzer aus erfolgen soll, während die ersten beiden Beispiele Dateien im benutzereigenen Katalog benennen.

6.2.2.3 Dateikettungsnamen

Mit Dateikettungsnamen bezeichnet man einen vom Datenbestand unabhängigen Namen, der die Verbindung zwischen der PL/I-Datei und der mit dem FILE-Kommando bezeichneten BS2000-Datei temporär herstellt. Der Dateikettungsname wird im LINK-Parameter des FILE-Kommandos angegeben. Die Verknüpfung einer PL/I-Datei mit einer BS2000-Datei erfolgt im PL/I-Programm über die Angabe TITLE. Der TITLE wird im OPEN-Statement des Programms festgelegt. Beim Fehlen der TITLE-Angabe wird der TITLE aus dem PL/I-Dateinamen gewonnen. Der so gewonnene TITLE muß den Regeln für die LINK-Namen des BS2000 entsprechen. BS2000-Dateien werden von PL/I nur über den Dateikettungsnamen angesprochen. Weitere Angaben hierzu findet man im Abschnitt 6.4.

6.2.2.4 Dateiorganisation (Zugriffsmethode)

Die Dateiorganisation legt fest, wie die Sätze auf dem Datenträger aufgezeichnet werden und nach welchen Verfahren intern auf die einzelnen Sätze zugegriffen wird. Folgende Zugriffsmethoden des BS2000 werden vom PLI1-Ein/Ausgabesystem unterstützt.

```
SAM     ISAM   PAM (bzw. UPAM)
```

Die Zugriffsmethoden werden im FILE-Kommando mit den Angaben im FCBTYPE-Parameter bestimmt, man beachte jedoch die Zusammenhänge zu der durch die im ENVIRONMENT-Attribut festgelegten PL/I-Organisationsform (siehe Abschnitt 6.4.2.).

Die Zugriffsmethode BTAM (Basic Tape Access Method) wird von PLI1 nicht verwendet und daher nachfolgend nicht mehr erwähnt. Zugriff auf Magnetbänder ist von PL/I-Programmen aus mit der Zugriffsmethode SAM möglich. Die Zugriffsmethode EAM (Evanescent Access Method) wird vom PLI1 für Anwenderprogramme nicht unterstützt. Sie wird lediglich vom Übersetzer im Zusammenhang mit der Bindemodulerzeugung verwendet.

- SAM (Sequential-Access-Method)
  Die Zugriffsmethode SAM ermöglicht sequentielles Verarbeiten von Sätzen. SAM-Dateien werden in PL/I mit der Organisationsform CONSECUTIVE bearbeitet.
- ISAM (Index Sequential Access Method)
  Eine ISAM-Datei kann sowohl sequentiell als auch im Dirketzugriff über Schlüssel verarbeitet werden.
  Über den Parameter KEYPOS im FILE-Kommando oder über KEYLOC im ENVIRONMENT-Attribut muß die Position des ersten Bytes des Schlüssels angegeben werden. Diese Position ist bei den Sätzen einer Datei unveränderlich.
  Über den Parameter KEYLEN im FILE-Kommando bzw KEYLENGTH im ENVIRONMENT-Attribut wird die Schlüssellänge in Anzahl Zeichen angegeben. Die Länge ist für alle Sätze einer Datei konstant. ISAM-Dateien werden in PL/I in der Regel mit der Organisationsform INDEXED bearbeitet.

  Die Organisationsform CONSECUTIVE ist auf ISAM-Dateien anwendbar, um Dateien verarbeiten zu können, die auch zur Bearbeitung durch die Dienstprogramme EDT und EDOR oder andere Sprachen (z.B. ALGOL) vorgesehen sind. Hierfür sind feste Schlüssellängen und die Schlüsseldarstellung verabredet (siehe Abschnitt 6.6.1.3).

  Für ISAM-Dateien beachte man den PAD-Parameter im FILE-Kommando, welcher bei der Ersterstellung der Datei eine wirtschaftliche Einteilung des Speicherplatzes steuert, wenn man nachträglich Sätze in bestehende Dateien einfügen will. Ohne PAD-Angabe im FILE-Kommando ist für Dateien die von PLI1 bearbeitet werden, PAD = 0 wirksam.

Anmerkung

Sätze von ISAM-Dateien erhalten vom Datenverwaltungssystem stets 4 zusätzliche Zeichen vorangestellt, wenn sie mit RECFORM = F eingerichtet wurden.

Man berücksichtige dies bei der Betrachtung von Speicherauszügen. Die Dateien sehen dabei aus, als hätten sie ein Satzlängenfeld wie variabel lange Sätze. Für den Benutzer hat diese Tatsache nur für die Berechnung der Dateigröße Bedeutung.

- PAM (Primary Access Method)
  Mit der Zugriffsmethode PAM ist ein Direktzugriff auf physikalischem Blocklevel möglich. Die Blöcke haben feste Länge (2048 Bytes) und werden auch als PAM-Blöcke bezeichnet. PAM-Dateien werden für die PL/I-Organisationsformen REGIONAL(1) und REGIONAL(3) verwendet. PAM-Dateien sind auch mit der Organisationsform CONSECUTIVE verarbeitbar.

6.2.2.5 Satzaufbau

Es gibt drei verschiedene Satzformate für die Sätze in Dateien:

Sätze fester Länge:	F-Format (für alle Zugriffsmethoden)
Sätze variabler Länge:	V-Format (für SAM, ISAM)
Sätze undefinierter Länge:	U-Format (für SAM, PAM)

Eine Datei kann nur aus Sätzen mit gleichem Satzformat bestehen. Das Satzformat wird im FILE-Kommando mit dem Parameter RECFORM oder im ENVIRONMENT-Attribut festgelegt. Die Übertragung von Sätzen zwischen Zentralspeicher und externem Speicher (Datenbestand) erfolgt blockweise. Die Zusammenhänge zwischen Satzlänge und Blockgröße sind nachfolgend erläutert:

- Satzlänge
  Die physikalische Länge eines Satzes im Datenbestand wird im FILE-Kommando mit dem Parameter RECSIZE oder im ENVIRONMENT-Attribut mit der RECSIZE-Angabe festgelegt. Die Satzlänge bei den Ein-/Ausgabeanweisungen im Quellprogramm beziehen sich im allgemeinen auf die Länge des logischen Satzes, während bei der Angabe im FILE-Kommando die zusätzliche Verwaltungsinformation mitgerechnet werden muß. Ein Sonderfall liegt bei INDEXED-Dateien vor, wenn man im ENVIRONMENT-Attribut KEYLOC = 0 verwendet (siehe Abschnitt 6.3.3).

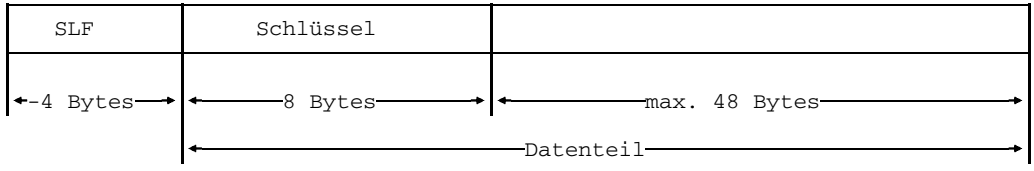
Zur Verwaltungsinformation zählen z.B. das Längenfeld von Folgen mit dem Attribut VARYING, wenn bei ENVIRONMENT SCALARVARYING angegeben ist (siehe Abschnitt 6.3.5), die Satzlängenangabe bei RECFORM = V und das Vorschubsteuerzeichen für PRINT-Dateien.

Die Information über die Länge des einzelnen Satzes wird bei Satzformat V durch ein zusätzliches Informationsfeld am Beginn des Satzes, das Satzlängenfeld (SLF) angegeben. Das 4 Bytes lange Satzlängenfeld enthält in den beiden höherwertigen Bytes die Länge des Satzes (mit Berücksichtigung des Satzlängenfeldes). Die beiden restlichen Bytes sind für die Dateiverwaltung reserviert.

Beispiel eines Satzes mit V-Format in einer ISAM-Datei:

```
/FILE DAT1,FCBTYPE=ISAM,RECSIZE=60,RECFORM=V,KEYPOS=5,KEYLEN=8,
```

Daraus ergibt sich folgender Aufbau:



- BLOCK
  Die Übertragung der Daten zwischen Zentralspeicher (Programm) und externem Speicher und zurück erfolgt nicht satzweise, sondern die Sätze werden zu Blöcken zusammengestellt und blockweise übertragen. Die Größe dieser Blöcke wird mit dem Parameter BLKSIZE im FILE-Kommando festgelegt. Eine Beeinflussung durch das ENVIRONMENT-Attribut ist nicht möglich. Man muß zwischen dem Begriff des physikalischen und des logischen Blocks unterscheiden.

  Ein physikalischer Block ist die Einheit für die Datenübertragung von und zu einem Ein-/Ausgabegerät. Er ist bei allen Direktzugriffsdatentägern 2048 Bytes lang (Halbseite). Man spricht von einem Standardblock bzw. von einem PAM-Block. Dateien auf Magnetbändern werden auch mit Nicht-Standardblöcken verarbeitet, deren Länge beliebig, jedoch maximal 32767 Bytes ist.

  ISAM arbeitet grundsätzlich mit Standardblöcken. SAM kann sowohl Standard- als auch Nicht-Standardblöcke (bei Magnetbanddateien) verarbeiten. Auf Direktzugriffsdatenträgern arbeitet SAM mit Standardblöcken.

  Daneben gibt es den Begriff des logischen Blocks. Ein logischer Block umfaßt gegebenenfalls mehrere PAM-Blöcke; das ermöglicht die Verwendung von Datensätzen über die Grenzen des physikalischen Blocks hinaus. Dazu muß im BLKSIZE-Parameter die logische Blockgröße so festgelegt werden, daß sie größer oder gleich der Satzlängenangabe (RECSIZE) ist. Bei SAM-Dateien im V-Format muß zusätzlich ein 4 Bytes langes Blocklängenfeld berücksichtigt werden. Darüberhinaus wird bei SAM- und ISAM-Dateien auf nicht PAM-Key-behafteten Plattendateien ein 12 Byte langes Kontrollfeld vom Datenverwaltungssystem für jeden logischen Block belegt.

  Bei Sätzen mit U-Format wird je Block nur ein Satz übertragen. Bei Magnetband wird in diesem Fall nur die aktuelle Satzlänge übertragen. Bei Direktzugriffsträgern bilden immer ein oder mehrere Standardblöcke die Transporteinheit. Der Teil des Arbeitsspeichers, in den vom Betriebssystem die Daten geschrieben oder aus dem die Daten gelesen werden, wird als Puffer bezeichnet. Bei Nicht-Standardblöcken sind Puffer und physikalischer Block äquivalent. Sonst entspricht die Pufferlänge dem logischen Block und kann ein Vielfaches der Standardblöcke sein (maximal 16 Standardblöcke).

Bei Sätzen mit dem Satzformat F und V werden von PL/I ein oder mehrere Sätze in den Puffer geschrieben. Ist vor dem Schreiben eines Satzes der noch freie Platz im Puffer größer oder gleich der aktuellen Satzlänge, so wird der Satz eingetragen und der Pufferinhalt noch nicht zum Datenbestand übertragen. Ist nicht mehr genügend Platz im Puffer vorhanden, wird der Pufferinhalt übertragen und der Satz in den "nächsten" Puffer geschrieben. Aufgrund der PAD-Angabe im FILE-Kommando kann bei ISAM-Dateien eventuell vor der vollständigen Füllung des Blockes die Übertragung zum Datenbestand erfolgen.

Bei Verwendung des ISO-Codes für Banddateien (Angabe CODE im FILE-Kommando) stehen die Angaben im Block- und Satzlängenfeld als 4stellige Dezimalzahl (PIC '9999') d.h. ISO-Code-Blöcke im variablen Format können maximal 9999 Bytes lang sein.

An den folgenden Beispielen sollen die Verhältnisse zwischen Satz, Block und Puffer demonstriert werden:

Beispiel 1

Vorausgesetzt wird:

- SAM-Datei
- Satzformat F: Satz mit 100 Bytes
- Standardblöcke
- Puffergröße = 2 Blöcke ≙ 4096 Bytes

Zugehöriges FILE-Kommando:

```
/FILE Dateiname,LINK=Title,FCBTYPE=SAM,RECFORM=F,
/    RECSIZE=100,BLKSIZE=(STD,2)
```

Alternativ:

```
ENVIRONMENT (... F,RECSIZE(100)...) und zusätzlich
/FILE Dateiname,LINK=Title,FCBTYPE=SAM,BLKSIZE=(STD,2)
```

Der Puffer wird mit 40 logischen Sätzen gefüllt; die rechtsstehenden 96 Bytes werden nicht verwendet, dennoch werden zwei komplette Blöcke zu je 2048 Bytes geschrieben. Zu bemerken ist, daß Satz 21 des Puffers in zwei Blöcken enthalten ist; die ersten 48 Bytes befinden sich in Block 1, die verbleibenden 52 Bytes sind in Block 2 enthalten.

Beispiel 2

Vorausgesetzt wird:

- SAM-Datei
- Satzformat F: Satz mit 100 Bytes
- Nicht-Standardblöcke (nur für Band)
- Puffergröße = 2020

Zugehöriges FILE-Kommando:

```
/FILE Dateiname,LINK=Title,FCBTYPE=SAM,RECFORM=F,
/    RECSIZE=100,BLKSIZE=2020,DEVICE=TAPE,VOLUME=...
```

Alternativ:

```
ENVIRONMENT (...F(,100)...) und zusätzlich
/FILE Dateiname,LINK=Title,FCBTYPE=SAM,BLKSIZE=2020,
/    DEVICE=TAPE,VOLUME=...
```

Der Puffer wird 2020 Bytes groß angelegt. Es passen 20 Sätze in den Puffer und es wird ein Block mit 2000 Bytes geschrieben.

Beispiel 3

Vorausgesetzt wird:

- SAM-Datei
- Satzformat F: Satz mit 1500 Bytes
- Standardblöcke
- Puffergröße 1 Block ≙ 2048 Bytes

Zugehöriges FILE-Kommando:

```
/FILE Dateiname,LINK=Title,FCBTYPE=SAM,RECFORM=F,
/    RECSIZE=1500
```

Alternativ:

```
ENVIRONMENT (...F(,1500)...) und zusätzlich
/FILE Dateiname,LINK=Title
```

Diese Wahl wurde ungünstig getroffen, da 548 Bytes verschwendet werden. Eine erstrebenswerte Puffergröße wären 6144 Bytes (3 Blöcke); dann könnte der Puffer 4 Sätze aufnehmen und von den 6144 Bytes würden 6000 benutzt werden. In diesem Falle würden drei Blöcke zu je 2048 Bytes geschrieben.

Beispiel 4

Vorausgesetzt wird:

- SAM-Datei
- Satzformat V mit den folgenden Satzlängen einschließlich SLF:
  100, 500, 300, 600, 200, 400, 700, 50, 100
- Nicht-Standardblöcke (nur für Band), BLKSIZE = 1000

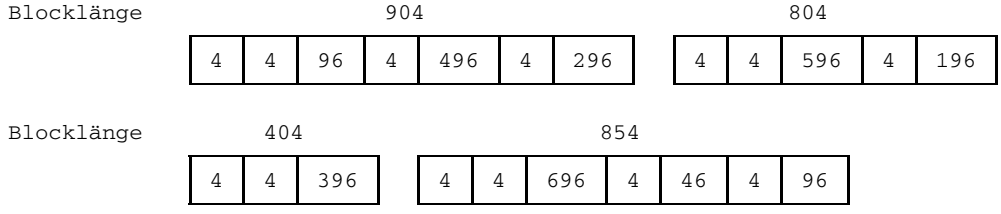
Zugehöriges FILE-Kommando:

```
/FILE Dateiname,LINK=Title,FCBTYPE=SAM,RECFORM=V,
/    RECSIZE=700,BLKSIZE=1OO,DEVICE=TAPE,VOLUME=...
```

Alternativ:

```
ENVIRONMENT (... V RECSIZE(700)...) und zusätzlich
/FILE Dateiname,LINK=Title,FCBTYPE=SAM,BLKSIZE=1000,
/    DEVICE=TAPE,VOLUME=...
```

Es werden folgende Blöcke erstellt:



Anmerkung

Nicht-Standardblöcke mit Sätzen variabler Länge enthalten je Satz 4 Bytes Längenangabe, sowie 4 Bytes pro Block für die Blocklänge.

Beispiel 5

Vorausgesetzt wird:

- SAM-Datei
- Satzformat F: Satz mit 8192 Bytes
- Standardblöcke
- Puffergröße = 8 Blöcke = 16 384 Bytes

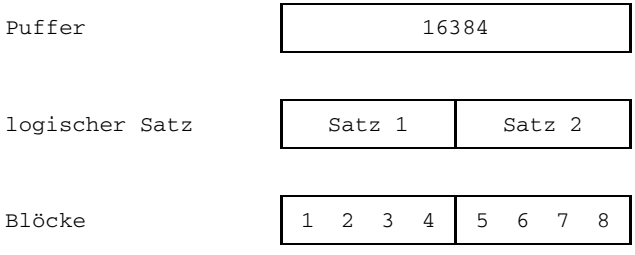
Zugehöriges FILE-Kommando:

```
/FILE Dateiname,FCBTYPE=SAM,RECFORM=F,
/    RECSIZE=8192,BLKSIZE=(STD,8)
```

Alternativ:

```
ENVIRONMENT (...F(8192)...) und zusätzlich
/FILE Dateiname,LINK=Title,FCBTYPE=SAM,BLKSIZE=(STD,8)
```

Jeder Puffer enthält zwei Sätze, jeder Satz benötigt 4 Standardblöcke.



Beispiel 6

Vorausgesetzt wird:

- SAM-Datei
- Satzformat V
- maximale Satzlänge 132
- Standardblöcke

```
Zugehöriges FILE-Kommando:

/FILE Dateiname,FCBTYPE=SAM,RECFORM=V,RECSIZE=132

Alternativ:

ENVIRONMENT (...V(, 132)...) und zusätzlich
/FILE Dateiname,LINK=Title
```

Ein Block ist analog zu Beispiel 4 aufgebaut und kann mindestens (2048-4)/ 132 = 15 Sätze maximaler Länge aufnehmen. Zu beachten ist, daß die Füllung eines Blockes beendet wird, wenn der noch freie Platz die Größe des aktuellen Satzes unterschreitet. Im vorliegenden Fall könnte also ein unbelegter Rest von 131 Bytes vorkommen, wenn am Blockende der nächste zum Schreiben vorgesehene Satz 132 Zeichen belegen würde. Er muß dann in den folgenden Block eingetragen werden.

6.2.2.6 Datenträger

Der externe Träger von Datenbeständen wird durch die Parameter DEVICE und VOLUME im FILE-Kommando festgelegt.

Die Datenträger werden in gemeinschaftliche (PUBLIC) und private Träger unterteilt. Zu den privaten Datenträgern, die nicht jederzeit während des laufenden Prozesses 'on-line' sein müssen, zählen alle Magnetbänder, Plattenstapel, Papier im Drucker und Kartenstapel im Lochkartenleser (Papiergeräte). Der gemeinschaftliche Datenträger befindet sich auf Plattenstapeln, die an dem System ständig angeschlossen sind; er steht vorformatiert zur Aufnahme von Benutzerdateien bereit und ist immer 'on-line'. Die auf ihm gespeicherten Dateien stehen u.U. mehreren Prozessen gleichzeitig zur Verfügung.

Durch die Voreinstellung des Systems wird stets der gemeinschaftliche Träger adressiert. DEVICE und VOLUME muß man angeben, wenn private Träger verwendet werden. Papiergeräte erreicht man gegebenenfalls auch über das SYSFILE-Kommando.

Es gibt eine Kopplung zwischen der Organisationsform einer Datei und der Art des Datenträgers. So sind z.B. Magnetbandgeräte nur mit der Organisationsform CONSECUTIVE verarbeitbar und die Organisationsformen INDEXED, REGIONAL(1) und REGIONAL(3) erfordern als Datenträger ein Gerät mit Direktzugriff. Die Art des Datenträgers ist im PL/I-Programm in der Regel ohne Bedeutung, da die charakteristischen Dateieigenschaften bereits durch andere Angaben festgelegt sind. Die einzige Ausnahme sind dabei Anweisungen, bei denen bestehende Sätze überschrieben werden - dies ist unabhängig von der Organisationsform - nur bei Direktzugriffsmedien möglich.

6.2.2.7 Zugriffsberechtigungen

- Kennwort
  Mit der Angabe eines Kennwortes kann der Benutzer seine Dateien gegen unberechtigte Benutzer schützen. Es gibt Lese- und Schreibkennwörter. Das Kennwort besteht aus einer 1 bis 4 Zeichen langen Konstanten, die in einer gesonderten Tabelle des Systems gespeichert wird. Hat die Datei ein Lese- und Schreibkennwort, genügt die Angabe des Schreibkennwortes, um die Datei auch lesen zu können. Die Kennwörter werden den Dateien im CATALOG-Kommando durch die Parameter RDPASS und WRPASS zugeordnet. Will man auf geschützte Dateien zugreifen, so muß man im laufenden Prozeß das PASSWORD-Kommando verwenden.

- Zugriffschutz
  Mit dem Parameter ACCESS im CATALOG-Kommando kann man festlegen, ob eine Datei nur zum Lesen oder zum Lesen und Schreiben verwendet werden darf. Man kann sich somit gegen unbeabsichtigte Veränderung eines Datenbestandes schützen.
  Mit dem Parameter SHARE im CATALOG-Kommando wird festgelegt, ob auf die Datei nur der Eigentümer oder auch ein Auftrag unter anderer Benutzerkennung zugreifen darf.

- Veränderungssperre
  Soll ein Schreibschutz nur vorübergehend bestehen, so kann man diesen Zeitraum mit dem RETPD-Parameter des FILE- oder CATALOG-Kommandos festlegen. Die Datei kann in dieser Zeit nicht verändert werden.

- Gleichzeitiger Zugriff
  Durch die Angabe SHARUPD im FILE-Kommando kann bestimmt werden, ob eine Datei, die zum Schreiben eröffnet ist, auch von anderen Prozessen zum Lesen oder Schreiben eröffnet werden darf (SHARUPD = YES), oder nur dann mehrere Prozesse die Datei eröffnen können, wenn sie alle nur lesend zugreifen (SHARUPD = NO).

  Haben mehrere Prozesse aufgrund der vorstehend genannten Regeln die gleiche Datei eröffnen können und tritt bei einem Zugriff auf einen Datensatz ein Konflikt mit einem anderen Zugriff auf, so wird der eine Zugriff in eine Warteschlange eingereiht und alle 5 Sekunden der Zugriff erneut versucht. Nach 100 Versuchen wird die Bedingung TRANSMIT gesetzt. Bei einer Rückkehr von der ON-Einheit der TRANSMIT-Bedingung wird mit der auf die EA-Anweisung folgenden Einheit fortgefahren.

  Die Angabe SHARUPD ist nur bei ISAM-Dateien möglich.

6.2.2.8 Dateigröße

Die Größe einer Datei, d.h. der von ihr auf dem Medium belegte Platz, wird durch die Angabe SPACE = (p,s) im FILE-Kommando bestimmt. Ist im FILE-Kommando eine SPACE-Angabe vorhanden, so wird bei jeder Ausführung des FILE-Kommandos

- der Speicherplatz um p PAM-Seiten (zu 2048 Bytes) erweitert und ggf. weitere Seiten hinzugefügt, so daß die Gesamt-Seitenzahl ein Vielfaches von 3 ist. Ist die Datei noch nicht vorhanden, so bedeutet dies eine Erstzuweisung; ist sie bereits vorhanden, so bedeutet dies eine Erweiterung bzw. Verkürzung bei negativem Wert.
- Soweit angegeben, wird der Wert s (Sekundär-Zuweisung) in die Datei-Kenndaten übernommen. Er wird immer dann ausgewertet, wenn im PL/I-Programm beim Schreiben der Datensätze festgestellt wird, daß nicht mehr genügend Speicherplatz vorhanden ist. Es wird dann der Speicherplatz um s PAM-Seiten verlängert und ggf. weitere Seiten hinzugefügt, so daß die Gesamt-Seitenzahl ein Vielfaches von 3 ist. In Verbindung mit REGIONAL ist s bedeutungslos.

Wird durch das FILE-Kommando eine neue Datei im Katalog eingerichtet und ist keine SPACE-Angabe gemacht, so wird normalerweise SPACE = (3,3) verwendet. Dieser Wert kann jedoch bei der System-Generierung anders eingestellt werden.

Die Erweiterung des Speichers von Dateien ist zeitaufwendig. Es wird daher empfohlen, die Werte so zu wählen, daß nicht zu oft erweitert werden muß.

Weitere Einzelheiten sind der Beschreibung des FILE-Kommandos zu entnehmen. Die aktuelle Speicherbelegung in PAM-Seiten und die Sekundär-Zuweisung (Wert s aus der SPACE-Angabe) können durch das Kommando FSTATUS ermittelt werden.

Wird im PL/I-Programm eine Datei eröffnet, so liegt der Speicherplatz für die Datei und der Wert der Sekundär-Zuweisung (d.i. der Wert s aus der Angabe SPACE = (p,s)) bereits in den Datei-Kenndaten vor. Abhängig von dem Attribut, mit dem die Datei eröffnet wird, ergibt sich folgendes:

INPUT	Es findet keine Veränderung der Werte statt.
UPDATE	Beim Ausgeben von Datensätzen wird der Speicherplatz ggf. um die Sekundär-Zuweisung erweitert.
OUTPUT	Beim Eröffnen wird ggf. die Größe des aktuellen Speichers und der Wert der Sekundär-Zuweisung korrigiert (siehe weiter unten). Beim Ausgeben von Datensätzen wird der Speicherplatz ggf. um die Sekundär-Zuweisung erweitert.

Um zu vermeiden, daß wegen einer unzureichenden SPACE-Angabe ein Fehler auftritt, werden beim Eröffnen einer Datei mit OUTPUT die in den Datei-Kenndaten vorhandenen Angaben über die Sekundär-Zuweisung ggf. korrigiert.
Hier gilt folgendes:

SAM CONSECUTIVE

Die Größe des aktuellen Speicherplatzes und der Wert der Sekundär-Zuweisung werden ggf. so erhöht, daß sie ohne Rest sowohl durch 3 als auch durch die Puffergröße n, wie sie durch BLKSIZE = (STD,n) im FILE-Kommando angegeben ist, teilbar ist.

Beim Erweitern einer Datei (/FILE..., OPEN = EXTEND) wird der aktuelle Speicherplatz nur dann im obigen Sinne korrigiert, wenn der nicht belegte Teil Null ist oder nicht den obigen Bedingungen genügt.

ISAM INDEXED
ISAM CONSECUTIVE

Die Größe des aktuellen Speicherplatzes wird ggf. so erhöht, daß sie mindestens n + 1 Seiten beträgt, wobei n die Puffergröße ist, wie sie durch BLKSIZE = (STD,n) im FILE-Kommando angegeben ist, und sie außerdem durch 3 teilbar ist.

PAM CONSECUTIVE

Keine Korrektur der Werte.

PAM REGIONAL Die Größe des aktuellen Speicherplatzes wird ggf. so erhöht, daß sie mindestens r + 1 Seiten beträgt, wobei r die Größe einer Region ist, und sie außerdem durch 3 teilbar ist. Die Größe einer Region r ergibt sich wie folgt:

- bei REGIONAL(1): aus der Datensatzlänge d, wie sie sich aus den Datei-Kenndaten
  /FILE RECSIZE = x bzw. ENV(RECSIZE(x)) ergibt.
- Bei REGIONAL(3): aus der Pufferlänge b, wie sie bei BLKSIZE = b im FILE-Kommando angegeben ist.

6.2.2.9 Sonstige Dateikenndaten

Der PL/I-Anwender sollte noch über folgende Kenndaten einer Benutzerdatei Kenntnis haben:

- Der OPEN-Modus einer Datei wird durch die Datei-Attribute in PL/I bestimmt. Bei der Ausgabe ist ggf. eine zusätzliche Steuerung über die OPEN-Angabe beim FILE-Kommando möglich.
- Bei Dateien, die in PL/I mit SEQUENTIAL eröffnet werden, kann durch OPEN = EXTEND bzw. bei PAM-Dateien durch OPEN = INOUT bestimmt werden, daß bei einer bestehenden Datei die neuen Datensätze hinzugefügt werden. Im anderen Falle werden die Kenndaten der alten Datei gelöscht und die alten Datensätze stehen nicht mehr zur Verfügung; die Kenndaten werden wie für eine neue Datei neu ermittelt.

Über die Wirkung der in Abschnitt 6.2.2 nicht genannten Parameter des FILE- bzw. CAT-Kommandos siehe [2]. Sie haben für PL/I in der Regel keine Bedeutung.

6.2.3 Schnittstellen des PLI1-Ein/Ausgabesystems zum BS2000

Für den Zugriff auf Dateien benutzen die Ein- und Ausgaberoutinen von PLI1 einige der Makros, welche vom Datenverwaltungssystem (DVS) und dem Ablaufteil des Betriebssystems zur Verfügung gestellt werden.
Von den DVS-Makros werden verwendet:

OPEN	zum Eröffnen einer Datei
CLOSE	zum Schließen einer Datei
FCB	zur Definition eines FILE-Kontrol-Blocks
IDFCB	zur Versorgung eines FCB mit symbolischen Namen
FSTAT	zur Information über den Zustand einer Datei
RDTFT	zum Lesen der Task-File-Table

Die Transport- und Zugriffs-Makros der jeweiligen DVS-Zugriffsmethode.

Von den Makros des Ablaufteils werden verwendet:

RDATA	für SYSDTA
WRLST	für SYSLST
WROUT	für SYSOUT
WROUT	für DISPLAY ohne REPLY
WRTRD	für DISPLAY mit REPLY im Datenstationsbetrieb
TYPIO	für DISPLAY mit REPLY im Stapelbetrieb
WRTRD	für TRANSIENT-Datei
WROUT	für TRANSIENT-Datei

6.3 Programm-Dateien und ENVIRONMENT-Attribut

Die Sprache PL/I kennt den Datentyp Datei (FILE), der stellvertretend für programmexterne Datenbestände steht. Die PL/I-Datei wird durch das FILE-Attribut vereinbart. Die Datenbestände sind in der Regel in Sätze gegliedert. PL/I kennt jedoch zwei Möglichkeiten, Daten zwischen Programm und Datenbestand zu übertragen:

- die stromorientierte bzw. zeichenweise Übertragung von Daten (Dateiattribut STREAM). Dabei faßt das Ein-/Ausgabesystem den Datenbestand als kontinuierlichen Strom von Zeichen auf und stellt dem Programm die Daten elementweise zur Verfügung bzw. stellt die vom Programm gelieferten Ausgaben zu einem Strom zusammen (siehe Abschnitt 6.5).
- die satzorientierte Übertragung von Daten (Dateiattribut RECORD). Das Ein-/Ausgabesystem stellt dem Benutzer die Daten satzweise zur Verfügung, so wie die Gliederung im Datenbestand ist, bzw. gliedert den Datenbestand so in Sätze, wie das Programm sie anliefert (siehe Abschnitt 6.6).
  Die zeichenorientierte und satzweise Übertragung dürfen für die gleiche Datei in einem OPEN-CLOSE-Zyklus nicht gemischt auftreten.

Dateien werden im PL/I-Programm über ihren Dateinamen angesprochen. Der Dateiname hat nach PL/I-Regeln eine Maximallänge von 31 Zeichen. Dieser Dateiname muß nicht mit dem Namen des Datenbestandes (BS2000-Dateiname) übereinstimmen. Der Kontakt wird beim Eröffnen einer Datei über die TITLE-Angabe hergestellt (siehe Abschnitt 6.4.1).

Bei der Vereinbarung einer Datei im Programm können mit dem ENVIRONMENT-Attribut Angaben über die Struktur dieser Datei gemacht werden. Ein Teil dieser Angaben entspricht analogen Angaben im FILE-Kommando. Falls an beiden Stellen Angaben zum selben Parameter gemacht werden, wird stets die ENVIRONMENT-Angabe verwendet.

Format des ENVIRONMENT-Attributes:

```
                          [⌐␣¬          ]
ENVIRONMENT (Angabe       [{ } Angabe ] ...)
                          [⌊,⌋          ]
```

Die möglichen Angaben sind Bild 6-2 zu entnehmen. Einzelheiten findet man in den folgenden Abschnitten.

Angabe	bei	Bedeutung	
CONSECUTIVE INDEXED REGIONAL(1) REGIONAL(3)	DCL	Organisationsform	
F F (b) 2) F ([b],r) 2)	DCL	feste Satzlänge	entspricht im FILE-Kommando: r: RECSIZE = r c: KEYPOS = p (für c ≠ 0) k: KEYLEN = k b: BLKSIZE = b F: RECFORM = F V: RECFORM = V U: RECFORM = U
V V (b) 2) V ([b],r) 2)	DCL	variable Satzlänge	
U U (b) 2) U ([b]),r) 1) 2)	DCL	undefinierte Satzlänge	
RECSIZE (r)	DCL	Satzlänge	
KEYLOC (c)	DCL	Schlüsselanfang	
KEYLENGTH (k)	DCL	Schlüssellänge	
BLKSIZE (b) 2)	DCL	Blocklänge	
CTLASA CTLMACH	DCL	Vorschubsteuerzeichen	CTLASA: RECFORM = (x,A) CTLMACH: RECFORM = (x,M)
SCALARVARYING	DCL	Steuerung der Ein-/Ausgabe bei skalaren VARYING-Variablen	
LEAVE	CLOSE	Magnetband nicht zurückspulen	
UNLOAD	CLOSE	Magnetband zurückspulen und entladen	
LIMCT (n)	DCL	REGIONAL (3): n + 1 Regionen werden betrachtet	
GENKEY	DCL	Gruppenschlüssel bei INDEXED	
TERMINAL (m,e)	DCL	Gerätemodi für TRANSIENT m: COMP|LINE l: ILCASE	entsprechend Operanden des WRTRD-Makros: MODE = COMP | LINE ILCASE = YES

1) Blocklängenangabe b wird ignoriert

2) Falls r nicht angegeben gilt:
bei F: r = b
bei V: r = b - 4

Bild 6-2 ENVIRONMENT-Angaben

6.3.1 Organisationsformen

PL/I kennt verschiedene Organisationsformen, um Sätze in einem Datenbestand zu organisieren. Es gibt Abhängigkeiten zwischen der PL/I-Organisation und den Zugriffsmethoden des BS2000 (siehe Abschnitt 6.4.2). Die Organisationsform kann nicht im FILE-Kommando angegeben werden.

Mögliche Angaben bei ENVIRONMENT:

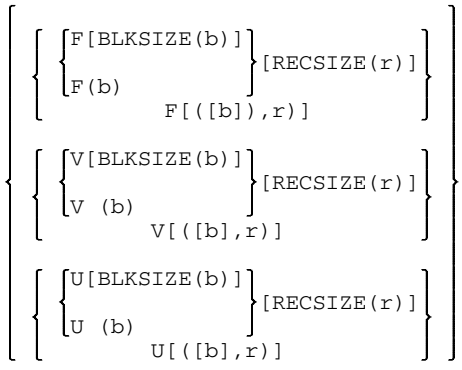
```
[CONSECUTIVE]
{INDEXED     }
{REGIONAL(1) }
[REGIONAL(3) ]
```

- CONSECUTIVE
  Die Sätze sind sequentiell in einer Datei abgelegt und können nur sequentiell verarbeitet werden. CONSECUTIVE-organisierte Datenbestände können strom- oder satzweise verarbeitet werden.
- INDEXED
  Die Sätze werden mit einem Schlüssel (Index) versehen. (Die Ablage erfolgt gewöhnlich in aufsteigender Reihenfolge der Schlüssel.)
  INDEXED-organisierte Datenbestände können sequentiell oder im Direktzugriff aber nur satzweise verarbeitet werden.
- REGIONAL(1)
  Die Datei besteht aus einzelnen Regionen. Pro Region wird ein Satz abgelegt, wobei der Zugriffsschlüssel der relativen Satznummer innerhalb des Datenbestandes entspricht.
  REGIONAL(1)-organisierte Dateien können sequentiell oder im Direktzugriff aber nur satzweise verarbeitet werden
- REGIONAL(3)
  Bei REGIONAL(3)-organisierten Dateien können pro Region mehrere Sätze abgespeichert werden.
  Zur Adressierung eines Satzes ist ein Schlüssel anzugeben, der die Region bezeichnet und zusätzlich innerhalb dieser Region zur Identifikation des Satzes dient.
  REGIONAL(3)-organisierte Dateien können sequentiell oder im Direktzugriff aber nur satzweise verarbeitet werden.

6.3.2 Satzaufbau

Der Aufbau der Datensätze des Datenbestandes kann sowohl im FILE-Kommando als auch im ENVIRONMENT-Attribut beschrieben werden.

Mögliche Angaben bei ENVIRONMENT:



Wird nur b angegeben, so wird diese Angabe wie folgt erweitert:

F (b) entspricht F (b,r) mit r = b V (b) entspricht V (b,r) mit r = b-4

Entsprechendes gilt für BLKSIZE, wenn RECSIZE nicht angegeben ist.

- F|V|U
  Hiermit wird festgelegt, ob die Satzlänge für alle Sätze des Datenbestandes fest, variabel oder unbestimmt ist. Die Angabe entspricht dem Parameter RECFORM des FILE-Kommandos.
  Einzelheiten siehe Abschnitt 6.2.2.5.
- Angabe r
  Die Angabe r muß eine ganzzahlige positive Konstante sein. Sie entspricht genau dem RECSIZE-Parameter des FILE-Kommandos. Die Angabe r ergibt sich aus der Länge des zu übertragenden Datensatzes unter Berücksichtigung zusätzlicher Informationen wie Satzlängenfeld bei V-Dateien und Schlüssel bei ISAM-Dateien.
  Eine Übersicht gibt Bild 6-3.
  Einzelheiten siehe im folgenden Abschnitt und Abschnitt 6.2.2.5.
- BLKSIZE und Angabe b
  Die Angabe einer Blockgröße bei U im ENVIRONMENT-Attribut wird von PLI1 ignoriert.

	Organisation	Dateityp	Satzform	Schlüsselposition		Satzlänge
ENV	↓		↓	KEYLOC(c)		RECSIZE(r)
FILE		FCBTYPE =	RECFORM =		KEYPOS = p	RECSIZE = r
	CONSECUTIVE	SAM	F V U			Satz 4 + Satz Satz
		ISAM	F	1 1)		Schl. + Satz
			V	5 1)		4 + Schl. + Satz
		PAM	F			Satz
			V			4 + Satz
			U			Satz
	INDEXED	ISAM	F	0	1	Schl. + Satz
				1...r		Satz
			V	0	5	4 + Schl. + Satz
				5...r		4 + Satz
	REGIONAL(1)	PAM	F			Satz
	REGIONAL(3)	PAM	F			Satz
			V			Satz

Satz: Länge des PL/I-Datensatzes in Zeichen
Schl.: Länge des Schlüssels

Angabe nicht möglich bzw. nicht nötig

1) Schlüssellänge nur 4: Schlüssel = FIXED BIN (31,0) (FORTRAN-Schlüssel)
oder 8: Schlüssel = PICTURE '(8)9'

Bild 6-3 Bestimmung der Satzlänge

6.3.3 Schlüsselangaben

Für INDEXED und REGIONAL(3) müssen Festlegungen über den Satzschlüssel getroffen werden.

Mögliche Angaben bei ENVIRONMENT:

```
{KEYLENGTH (k)        KEYLOC (c) }
```

- KEYLENGTH (k)
  Diese Angabe legt die Schlüssellänge in Anzahl Zeichen fest.
  k muß eine positive ganze Zahl größer Null sein. Diese Angabe entspricht dem KEYLEN-Parameter des FILE-Kommandos.
- KEYLOC (c)
  Diese Angabe ist nur für INDEXED-Dateien von Bedeutung und legt die Position des ersten Zeichens vom Schlüssel fest. c muß eine positive ganze Zahl sein. Für Werte größer Null entspricht die Angabe genau dem KEYPOS-Parameter des FILE-Kommandos.
  Für c = 0 speichert PLI1 den Schlüssel vor dem Datensatz. Als Satzlänge muß dann die Länge von Datensatz (Nutzinformation) und Schlüssel angegeben werden. Außerdem muß in Verbindung mit KEYLOC(0) das Satzformat (F oder V) im ENVIRONMENT-Attribut festgelegt werden. Wird diese Angabe nur im FILE-Kommando gemacht, erhält man beim Eröffnen der Datei eine UNDEFINEDFILE-Bedingung.

  Beispiel

  ```
  ENVIRONMENT (INDEXED, F (,88), KEYLOC(0), KEYLENGTH(8))
  ```

  Die zu übertragenden Datensätze müssen 80 Zeichen lang sein.

  Für KEYLOC bzw. KEYPOS $\neq$ 0 wird beim Schreiben in eine Datei der Schlüssel stets auf die bezeichnete Stelle im Satz gespeichert, d.h. eine im Programm an dieser Stelle des Satzes stehende Information ist in der Datei vom Schlüssel überschrieben.

6.3.4 Vorschubsteuerung

Zum Drucken vorgesehene Dateien müssen als erstes Zeichen in jedem Datensatz ein Vorschubsteuerzeichen haben. Diese Steuerzeichen werden bei der stromorientierten Ausgabe für Dateien mit dem Attribut PRINT automatisch erzeugt. Die Codierung der Steuerzeichen kann ausgewählt werden.

- CTLMACH

  ’00’B4 Kein Vorschub
  ’0n’B4 n Zeilen Vorschub nach dem Druck
  ’4n’B4 n Zeilen Vorschub vor dem Druck und eine Zeile nach dem Druck, falls die Ausgabeseite nicht leer ist
  ’81’B4 Vorschub auf erste Zeile neue Seite nach dem Drucken
  ’8n’B4 Vorschub gemäß Lochstreifen-Kanal n nach dem Drucken
  ’C1’B4 Vorschub auf neue Seite vor dem Druck und, falls die Ausgabezeile nicht leer ist, eine Zeile vorschieben nach dem Druck

  Diese Angabe ist nicht erlaubt für Dateien mit TITLE (’SYSPRINT’) oder TITLE (’SYSOUT’) sowie für Dateien, die auf die System-Dateien SYSLST und SYSOUT umgelenkt werden sollen.

Mögliche Angaben Bei ENVIRONMENT:

$$\left\{\begin{array}{l}\texttt{CTLASA}\\ \texttt{CTLMACH}\end{array}\right\}$$

Voreinstellung: CTLMACH bei PRINT-Dateien,
sonst keine (/FILE ..., RECFORM = (x, N).

Diese Angaben entsprechen analogen Angaben im RECFORM-Parameter des FILE-Kommandos.

- CTLASA

  Es gelten folgende Steuerzeichen:

  ’4E’B4 (Pluszeichen) Kein Vorschub
  ’40’B4 (Leerzeichen) Eine Zeile Vorschub vor Druck
  ’F0’B4 (Zeichen 0) Zwei Zeilen Vorschub vor Druck
  ’60’B4 (Minuszeichen) Drei Zeilen Vorschub vor Druck
  ’F1’B4 (Zeichen 1) Vorschub auf 1. Zeile neue Seite

  ’Cn’B4 Vorschub gemäß Lochstreifen-Kanal n vor dem Drucken, und eine Zeile nach dem Druck, falls die Ausgabezeile nicht leer ist.

6.3.5 Steuerung VARYING-Variable

Bei der internen Darstellung von Folgen-Variablen mit dem Attribut VARYING wird dem Speicherplatz, der den Wert aufnimmt, ein Halbwort vorangestellt, das die aktuelle Länge der Folge aufnimmt.
Wird der Inhalt einer skalaren Variablen mit dem Attribut VARYING als Datensatz in eine Datei eingetragen, so sind Entscheidungen erforderlich, ob die Längenangabe mit ausgegeben wird oder nicht. Wird ein so geschriebener Datensatz wieder gelesen, so muß angegeben werden, ob in diesem Datensatz die Längenangabe enthalten ist oder nicht.

Die Steuerung dieses Vorgangs geschieht bei der Vereinbarung der Datei durch die Angabe SCALARVARYING beim Attrubut ENVIRONMENT.
Sie hat die Form

```
DCL a FILE RECORD usw. ENVIRONMENT (SCALARVARYING)
```

Die Angabe SCALARVARYING wird nur für Dateien ausgewertet, die mit dem Attribut RECORD eröffnet werden. Die weiteren Erläuterungen beziehen sich auf skalare Variable mit dem Attribut VARYING, die im Zusammenhang mit Datensätzen verwendet werden. Datei werden folgende Fälle unterschieden:

- Es wird mit READ INTO (a), ein Datensatz gelesen.
- Es wird mit WRITE FROM (a) oder REWRITE FROM (a) ein Datensatz geschrieben.
- Es wird mit READ SET ein Datensatz gelesen.
- Es wird mit LOCATE SET ein Datensatz geschrieben.

In den nachfolgenden Abschnitten bezieht sich die Beschreibung auf die Zeichenfolge-Variable (CHARACTER). Das gleiche gilt sinngemäß auch für die Bitfolge-Variable (BIT).

Des weiteren werden in die Betrachtungen ein ggf. vorhandener Schlüssel und die Längenangabe für die Satzform V (RECFORM = V) nicht mit einbezogen. Sie sind entsprechend der Beschreibung in den maßgeblichen Abschnitten noch zu berücksichtigen.

6.3.5.1 Lesen eines Datensatzes mit READ INTO

Wird ein Datensatz mit READ INTO (a) in die skalare Zielvariable a gelesen und hat diese Variable das Attribut VARYING, so werden zwei Fälle unterschieden:

Ist im ENVIRONMENT-Attribut der Datei SCALARVARYING nicht angegeben, so wird angenommen, daß im Datensatz die VARYING-Längenangabe nicht enthalten ist. Es wird der Datensatz der Zielvariablen zugewiesen und dabei die Längenangabe bei der VARYING-Variablen auf den richtigen Wert gesetzt, also auf die Länge des Datensatzes.

Ist bei der Vereinbarung der Datei SCALARVARYING angegeben, so wird angenommen, daß die ersten beiden Bytes des Datensatzes die Längenangabe enthalten. Es werden dann die ersten beiden Bytes in die Längenangabe der Variablen übertragen und die restlichen Zeichen des Datensatzes im Speicherbereich der Zielvariablen abgelegt. Ist dabei "Längenangabe>Datensatz – 2", so kann dies bei einem späteren Zugriff auf die Variable zu undefinierten Ergebnissen führen.

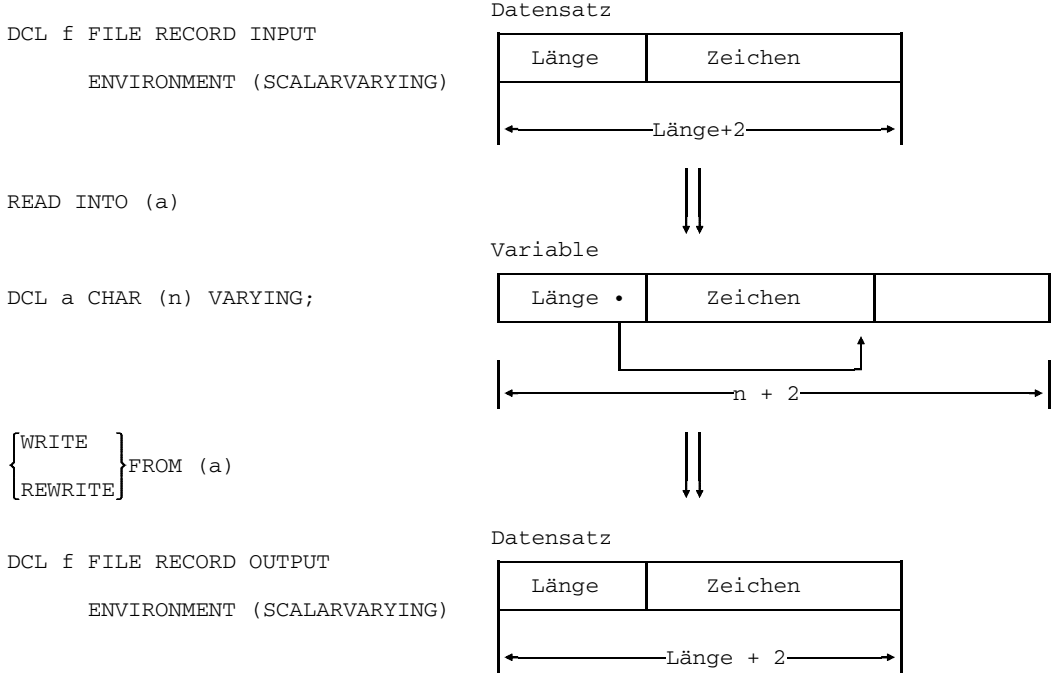


Bild 6-4 Datenübergabe bei einer skalaren Zeichenfolge-Variable variabler Länge mit SCALARVARYING bei INTO bzw. FROM

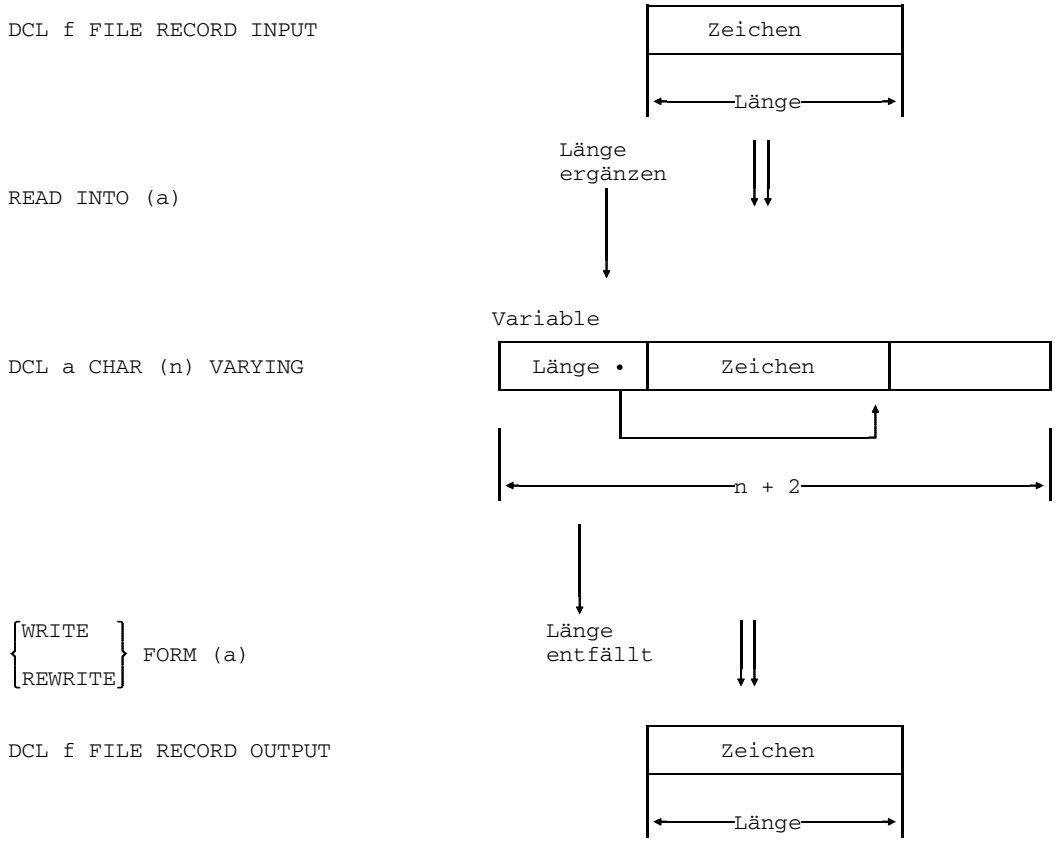


Bild 6-5 Datenübergabe bei einer skalaren Zeichenfolge-Variable variabler Länge ohne SCALARVARYING bei INTO bzw. FROM

6.3.5.2 Schreiben eines Datensatzes mit WRITE FROM

Wird ein Datensatz mit WRITE FROM (a) oder REWRITE FROM (a) aus der skalaren Quellvariablen a in eine Datei geschrieben und hat die Variable das Attribut VARYING so werden zwei Fälle unterschieden:

Ist bei der Vereinbarung der Datei SCALARVARYING nicht angegeben, so wird nur der Wert der Quellvariablen, nicht aber seine Längenangabe als Datensatz ausgegeben. Die Länge des Datensatzes ergibt sich also aus der aktuellen Länge der Quellvariablen.

Ist bei der Vereinbarung der Datei SCALARVARYING angegeben, so wird die Längenangabe und der Wert der Quellvariablen als Datensatz ausgegeben. Die Länge des Datensatzes ergibt sich also aus der aktuellen Länge der Variablen + 2.

6.3.5.3 Lesen eines Datensatzes mit READ SET

Wird ein Datensatz mit READ SET gelesen, so gibt es keine Zielvariable, in die der Datensatz transportiert wird; es wird lediglich ein Zeiger zur Verfügung gestellt, der auf den aktuellen Datensatz im Eingabepuffer weist.

Bei der Ausführung der READ-Anweisung ist nicht bekannt, welche Attribute die Variable hat, mit der auf den Datensatz zugegriffen werden soll.

Die Angabe SCALARVARYING ist für diesen Fall ohne Bedeutung.

Auf den Datensatz kann nur dann mit einer Variablen mit dem Attribut VARYING zugegriffen werden, wenn die ersten beiden Bytes eine Längenangabe enthalten. Wie bei allen Zugriffen mit BASED-Variablen ist der Benutzer selbst verantwortlich, daß im Datensatz die richtige Längenangabe vorhanden ist.

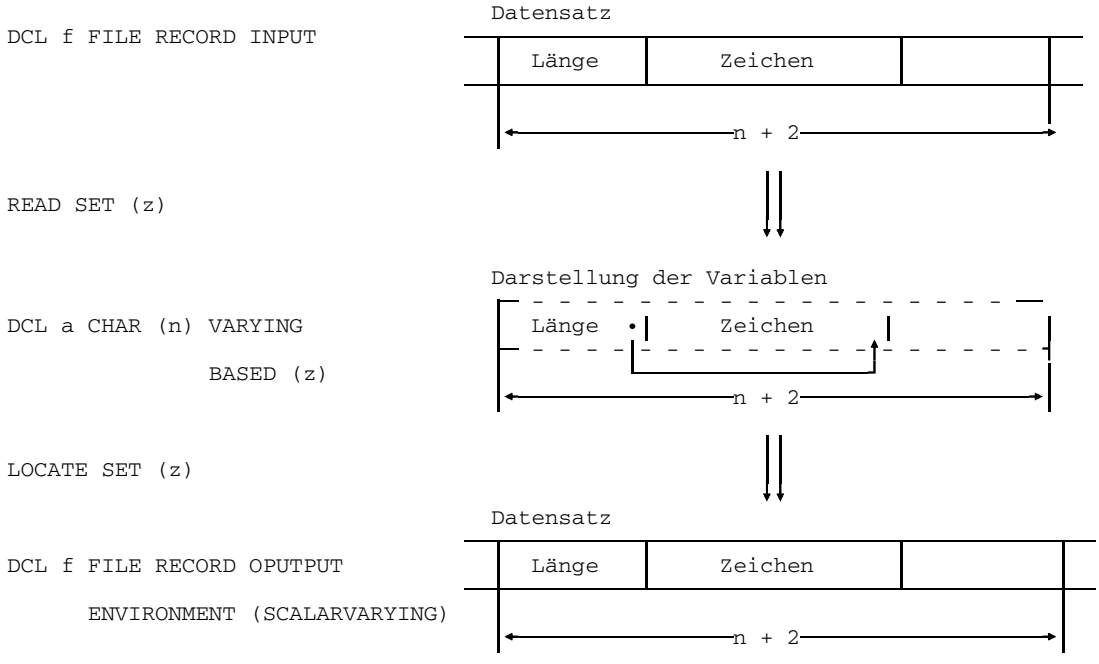


Bild 6-6 Datensatzübergabe bei einer skalaren Zeichenfolge-Variable variabler Länge bei LOCATE SET

6.3.5.4 Schreiben eines Datensatzes mit LOCATE SET

Wird ein Datensatz mit LOCATE SET geschrieben, so wird durch die Anweisung lediglich im Ausgabepuffer Speicherplatz für einen Datensatz reserviert. Dieser Speicherplatz bildet gleichzeitig den Speicherplatz für die Variable. Es wird ein Zeiger gesetzt, der auf den Anfang des Speicherplatzes weist.

Dieser Speicherplatz kann über eine BASED-Variable mit Information gefüllt werden; er wird erst zu einem späteren Zeitpunkt in die Datei übertragen.

Ist für die Datei die Angabe SCALARVARYING gemacht, so wird für eine BASED-Variable mit dem Attribut VARYING außer dem Speicherplatz für den Wert auch noch 2 Bytes für die Längenangabe reserviert, so daß dem Datensatz über eine BASED-Variable mit dem Attrubut VARYING ein Wert zugewiesen werden kann. Unabhängig von der aktuellen Länge der Folge hat der Datensatz immer die maximale Länge + 2 Bytes.

Ist für die Datei die Angabe SCALARVARYING nicht gemacht, so wird bei der Speicherreservierung für Variable mit dem Attribut VARYING kein Speicherplatz für die Längenangabe berücksichtigt. Dies kann bei der Zuweisung eines Wertes über eine BASED-Variable mit dem Attribut VARYING zur Zerstörung anderer Daten führen, ohne daß dies erkannt wird. Diese Betriebsart ist daher nicht erlaubt. Die Verantwortung liegt, wie bei BASED-Variablen allgemein, beim Benutzer.

6.3.6 Gerätesteuerung bei TRANSIENT-Dateien

TRANSIENT-organisierte Dateien bedienen die Dialogstation mit den BS2000-Makros WRTRD und WROUT (siehe BS2000 Makroaufrufe [16]). Folgende Gerätemodi können über die ENVIRONMENT-Angabe TERMINAL gesteuert werden:

```
ENV (TERMINAL (m,e))  mit m = COMP|LINE
                      und e = ILCASE.
```

Diese Angaben entsprechen den gleichnamigen Operanden des WROUT- bzw. WRTRD-Makro-Aufrufes und haben deren Wirkung.

Die Angabe ILCASE bewirkt, daß bei der Eingabe von der Dialogstation auch Kleinbuchstaben übergeben werden (entspricht ILCASE=YES beim WRTRD-Makro).
Die Angabe COMP bedeutet kompatibler Betriebsmodus für Dialogstations-Ein-/Ausgabe (entspricht MODE = COMP bei WROUT-oder WRTRD-Makro).

Die Angabe LINE bedeutet LINE-Modus für Dialogstation-Ein-/Ausgabe (entspricht MODE = LINE bei WROUT-oder WRTRD-Makro). Der LINE-Modus erlaubt, daß die ein- oder auszugebenden Sendungen aus mehr als einer Zeile bestehen können. Damit ist eine Formatierung bei der Ein- und Ausgabe möglich. Folgende Steuerzeichen können in der zu übergebenden Information enthalten sein:

X’15’: Beginn neue Zeile
X’0C’: Beginn neuer (d.h. vorher gelöschter) Bildschirm
X’1D’: Beginn kursiver oder blinkender Bereich
X’1E’: Ende kursiver oder blinkender Bereich
X’0E’: Beginn zweiter Zeichenvorrat von Dialogstation
X’0F’: Ende zweiter Zeichenvorrat von Dialogstation

Mit entsprechender ENVIRONMENT-Angabe bei RECSIZE kann zum Beispiel bei Benutzung der aufgeführten Gerätesteuerzeichen mit einer READ- oder WRITE-Anweisung ein kompletter Bildschirminhalt gelesen oder geschrieben werden.
Die ENVIRONMENT-Angaben TERMINAL sind nur für Dialogstationen 816x und 9750 wirksam.

6.3.7 Sonstige Angaben

Die weiteren Angaben sind in folgenden Abschnitten beschrieben:

LEAVE, UNLOAD	6.7.4
LINCNT	6.6.4.1
GENKEY	6.6.2.1

6.4 Zuordnung von Programm-Dateien zu BS2000-Dateien

Die Zuordnung einer PL/I-Datei zu einer BS2000-Datei erfolgt über den Dateikettungsnamen. BS2000-seitig ist das die LINK-Angabe im FILE-Kommando oder CHANGE-Kommando. PL/I-seitig ist das die TITLE-Spezifikation der OPEN-Anweisung.
Fehlt die TITLE-Spezifikation in der OPEN-Anweisung, so wird der in der Vereinbarung angegebene Dateiname zur Bildung des TITLE verwendet. Für die Systemdateien gelten besondere Regeln, die dem folgenden Abschnitt zu entnehmen sind. Eine Zuordnung von PL/I-Dateien zu BS2000-Dateien ist nur sinnvoll, wenn die Organisationsform und Zugriffsmethode verträglich sind.

6.4.1 Regeln für die Zuordnung der Dateinamen

Die Verknüpfung des PL/I-Programms mit der Datei im BS2000 erfolgt entweder explizit durch die OPEN-Anweisung oder implizit beim OPEN-Vorgang. Beim Fehlen einer TITLE-Angabe oder beim impliziten Eröffnen von Dateien liefert der Dateiname den TITLE. Der TITLE wird bei der Zuordnung der PL/I-Datei zur BS2000-Datei als BS2000-Link-Name interpretiert. Aus diesem Grund muß der TITLE den Regeln für Link-Namen entsprechen. Damit ist eine Zeichenfolge von maximal 8 Zeichen zulässig. Ist der PL/I-TITLE oder der als TITLE verwendete PL/I-Dateiname länger als 8 Zeichen, so werden nur die ersten 8 Zeichen genommen. Im TITLE werden außerdem stets alle Zeichen "_" durch "$" ersetzt.

Anmerkung

Wenn die Vereinbarung der PL/I-Datei mit dem Attribut EXTERNAL erfolgt, wird der Dateiname durch PL/I entsprechend den Konventionen für externe Namen auf 7 Zeichen nach der 4:3 Regel (d.h. die ersten vier und die letzten drei Zeichen des Namens) gekürzt. Man beachte das bei der Betrachtung von Binderprotokollen.

Für die Bildung des TITLE aus dem Dateinamen steht der ursprüngliche (nicht der gekürzte 7 Zeichen lange) Name zur Verfügung. Desgleichen erscheint in der 'ONFILE-Angabe' bei den Pseudo-Variablen und in Fehlermeldungen der ungekürzte Dateiname.

Für die Systemdateien SYSDTA, SYSLST und SYSOUT gelten folgende durch die Steueranweisung SYSFILE (siehe Abschnitt 5.4.7) voreingestellten Zuordnungen:

- Wird im PL/I-Programm als TITLE 'SYSIN' ermittelt, so wird die BS2000-Systemdatei SYSDTA angesprochen.
- Wird im PL/I-Programm als TITLE 'SYSPRINT' ermittelt, so wird die BS2000-Systemdatei SYSLST angesprochen.
- Wird im PL/I-Programm als TITLE 'SYSOUT' ermittelt, so wird die BS2000-Systemdatei SYSOUT angesprochen.

Anmerkung

PL/I ergänzt bei der GET-Anweisung eine fehlende FILE-Angabe stets mit SYSIN, und entsprechend wird bei der PUT-Anweisung die FILE-Angabe mit SYSPRINT ergänzt.

Die oben genannten Zuordnungen zwischen TITLE und Systemdatei können mit der Steueranweisung SYSFILE (siehe Abschnitt 5.4.7) verändert werden. Gibt man beim Start eines PL/I-Programms eine andere Zuordnung vor, so können beliebige PL/I-TITLE mit den Systemdateien verknüpft werden. Man beachte ferner, daß bei Anwendung des SYSFILE-Kommandos die Systemdateien vorübergehend auf Benutzerdateien umgesteuert werden können. Es können auch sonstige Ausgaben (z.B. Fehlermeldungen) nach SYSPRINT bzw. SYSOUT geleitet werden. Man kann diese Ausgaben über die Steuer-Anweisungen MESSAGE, FORMAT und DIAGNOST (siehe Kapitel 3 und 5) beeinflussen.

Bei der Verwendung von Systemdateien ist noch zu beachten, daß nur sequentielle Verarbeitung möglich ist, wobei jedoch satz- oder stromorientierte Betriebsart angewendet werden darf. Satz- und stromorientierte Verarbeitung dürfen für die gleiche Systemdatei in einem OPEN-CLOSE-Zyklus nicht gemischt auftreten.

Bei der Verwendung von Systemdateien ist weiterhin zu berücksichtigen, daß eine exakt chronologische Koordination zwischen dem Ein-/Ausgabesystem und anderen Diensten, die die gleiche Systemdatei ansprechen, nicht immer gewährleistet ist. So kann z.B. eine Trace-Ausgabe auf SYSLST geschrieben werden ohne zu berücksichtigen, daß vom Ein-/Ausgabesystem zeitlich vorher schon eine Ausgabe für SYSLST vorbereitet, aber noch nicht ausgegeben wurde.

Ein weiterer Sonderfall ergibt sich bei der Verwendung der DISPLAY-Anweisung. Die DISPLAY-Anweisung mit REPLY schreibt und liest je nach RUNOPT-Angabe SYSFILE=DISPLAY (siehe Abschnitt 5.4.7) auf die bzw. von der Dialogstation, auf die bzw. von der Operateuer-Konsole oder in die Systemdatei SYSOUT bzw. von der Systemdatei SYSDTA.

Zusammenfassung

Bei der Zuordnung von PL/I-Dateien zu den physikalischen Dateien im BS2000 laufen folgende Schritte ab:

- Ermittlung des TITLE der Datei aus der TITLE-Spezifikaktion der OPEN-Anweisung oder ersatzweise aus dem PL/I-Dateinamen nach obigen Regeln. Merke:

GET ohne FILE oder STRING PUT ohne FILE oder STRING	GET FILE (SYSIN) PUT FILE (SYSPRINT)
OPEN ohne TITLE	OPEN TITLE('Dateiname')

- Prüfung, ob dieser TITLE mit Steueranweisung SYSFILE einer Systemdatei zugeordnet ist. Voreingestellt ist:

Title	Systemdatei
SYSIN	SYSDTA
SYSPRINT	SYSLST
SYSOUT	SYSOUT

  Man beachte, daß die Systemdateien SYSDTA und SYSLST mit dem Kommando SYSFILE auf eine Benutzerdatei geleitet sein können (siehe Abschnitt 3.4.1).

- Ist der TITLE nicht mit Steueranweisung SYSFILE einer Systemdatei zugeordnet, so muß es eine Benutzerdatei mit identischem LINK-Symbol geben.
- Wird der TITLE bei diesen Suchvorgängen nicht identifiziert, erfolgt Signalisierung der UNDEFINEDFILE-Bedingung.

Je nach Ergebnis obiger Entscheidungskette kann die Wirkung eines PL/I-Programms unterschiedlich sein. Wird nämlich der TITLE in der Liste der Steueranweisung SYSFILE gefunden, erfolgt die Bearbeitung der Datei mit Makros des Ablaufteils vom BS2000. Andernfalls werden DVS-Makros verwendet. DVS-Makros beachten die Kenndaten der Benutzerdateien wie Satzlängen, Blockgröße usw. Entscheidet sich PLI1 jedoch für Makros des Ablaufteils, so treten die für Systemdateien gültigen Einstellungen in Kraft (siehe hierzu z.B. Abschnitt 6.5.1). Insbesondere kann es zur Abänderung von Kenndaten wie RECSIZE und BLKSIZE kommen, wenn eine Benutzerdatei über SYSFILE-Kommando wie eine Systemdatei bearbeitet wird.

Will man erreichen, daß PL/I-Dateien mit dem TITLE 'SYSIN', 'SYSPRINT' oder 'SYSOUT' nicht mit den Makros des Ablaufteils sondern mit DVA-Makros bearbeitet werden, so ist die durch die Steueranweisung *RUNOPT SYSFILE festgelegte bzw. voreingestellte Zuordnung aufzuheben.

Z.B. kann man dort beliebige im Programm nicht vorkommende Namen verwenden (*RUNOPT SYSFILE = SYSDTA (DUMMY)).

6.4.2 Regeln für die Zuordnung der Organisationsformen

Die im Ein-/Ausgabesystem der PL/I-Programme benutzten Zugriffsmethoden des BS2000 werden in Abhängigkeit von der im Programm angegebenen Dateiorganisation und der physikalischen Datei angewandt. Bestimmend dafür sind die bei der Dateivereinbarung angegebenen Dateiattribute wie SEQUENTIAL, DIRECT usw. sowie die Angabe für die Organisationsform im ENVIRONMENT-Attribut. Die nachfolgende Tabelle gibt einen Überblick über die zulässigen Kombinationen.

PL/I-Datei Zugriff	Dateiorganisation	BS2000 Zugriffsmethode
SEQUENTIAL (ohne KEYED)	CONSECUTIVE	SAM[1)] PAM ISAM[1)] Systemdateien[1)]
	INDEXED	ISAM
	REGIONAL(1)	PAM
	REGIONAL(3)	PAM
SEQUENTIAL KEYED	INDEXED	ISAM
	REGIONAL(1)	PAM
	REGIONAL(3)	PAM
DIRECT KEYED	INDEXED	ISAM
	REGIONAL(1)	PAM
	REGIONAL(3)	PAM

1) Nur diese Kombination für stromorientierte Verarbeitung anwendbar

Es gibt weitere Regeln, welche die Zulässigkeit bestimmter Formen von Ein-/Ausgabevorgängen für obige Kombinationen festlegen. Diese findet man unter der jeweiligen Dateiorganisation im Abschnitt 6.6.

Folgende Voreinstellungen sind getroffen:

- Wird durch das Attribut ENVIRONMENT keine Organisationsform festgelegt, so wird die Datei CONSECUTIVE bearbeitet.
- Ist die Zugriffsmethode nicht durch das FILE-Kommando festgelegt worden, so legt PL/I folgende Angaben fest:
  - SAM für CONSECUTIVE-Dateien
  - ISAM für INDEXED-Dateien
  - PAM für REGIONAL-Dateien

6.4.3 Zusammenhang zwischen FILE-Kommando und ENVIRONMENT-Angabe

Im Abschnitt 6.3 wurde beschrieben, welche Dateieigenschaften im Programm mit dem ENVIRONMENT-Attribut festgelegt werden können. Einige dieser Eigenschaften können nur so festgelegt werden, für andere ist die Festlegung im Programm mit ENVIRONMENT oder im FILE-Kommando möglich. Eine dritte Gruppe von Eigenschaften kann nur im FILE-Kommando festgelegt werden. Eine Übersicht gibt Bild 6-7.
Des weiteren können kontextabhängige Voreinstellungen wirksam werden.

		ENVIRONMENT-Angabe	FILE-Kommando
Organisation		CONSECUTIVE INDEXED REGIONAL(1) REGIONAL(3)	
Dateityp			FCBTYP = {SAM | ISAM | PAM}
Datensatz	Format	F V U	RECFORM = {F | V | U}
	Länge	RECSIZE (r)	RECSIZE = r
Schlüssel	Position	KEYLOC (c) für c = 0	KEYPOS = 1 | 5 [2]
		KEYLOC (c) für c ≠ 0	KEYPOS = c
	Länge	KEYLENGTH (k)	KEYLEN = k
Blocklänge		BLKSIZE (b) [3]	BLKSIZE = b
Kombinationen		F (b) → F (b,r) [3] V (b) → V (b,r) [3] U (b) → U (b,r) [1]	RECFORM = F|V|U BLKSIZE = b RECSIZE = r
Speicherbedarf			SPACE = s
Vorschubsteuerzeichen		CTLMACH	RECFORM (x,M)
		CTLASA	RECFORM (x,A)
LOCATE-und READ SET-Steuerung		SCALARVARYING	

1) Blocklänge b wird ignoriert
2) 1: für F-Format
   5: für V-Format
3) Falls r nicht angegeben gilt:
   bei F: r = b
   bei V: r = b - 4

Bild 6-7 Entsprechung von ENVIRONMENT-Attribut und FILE-Kommando

6.4.4 Ermittlung der Datei-Kenndaten

Zu jeder Datei im BS2000 gehören bestimmte Kenndaten, die den Typ und die Struktur der Datei beschreiben. Für eine bestehende Datei liegen die Kenndaten bereits fest; für eine neue Datei müssen sie aus den Angaben des Benutzers erst zusammengestellt werden.

Wird in einem PL/I-Programm eine Datei mit OPEN INPUT oder OPEN UPDATE eröffnet, so wird davon ausgegangen, daß eine zugehörige BS2000-Datei bereits besteht und im Datei-Katalog eingetragen ist. Ferner wird vorausgesetzt, daß die Kenndaten der BS2000-Datei mit dem Attributsatz der PL/I-Datei und mit den Angaben bei ENVIRONMENT verträglich sind. Durch entsprechende Ein-Ausgabe-Anweisungen können in der Datei Datensätze ergänzt, geändert oder gelöscht werden. Die Kenndaten der Datei können dabei nicht geändert werden (außer z.B. SPACE).

Wird eine Datei mit OPEN OUTPUT eröffnet, so wird im allgemeinen davon ausgegangen, daß eine entsprechende Datei nicht existiert, sondern daß eine neue Datei erstellt wird. Ist jedoch eine entsprechende Datei bereits vorhanden, so wird davon ausgegangen, daß sie durch eine neue Datei ersetzt wird, d.h. die alte Datei wird gelöscht.

Unter bestimmten Voraussetzungen kann man durch den Operanden OPEN des FILE-Kommandos erreichen, daß die alte Datei bestehen bleibt. Es wird dann ähnlich wie bei OPEN UPDATE verfahren, d.h. die alten Datensätze bleiben bestehen und die neuen Datensätze werden hinzugefügt.

Wird eine neue Datei erstellt, so müssen die Kenndaten zusammengestellt werden. Der Benutzer kann die gewünschten Werte

- im PL/I-Programm durch die ENVIRONMENT-Angabe oder
- im FILE-Kommando

angeben. Die möglichen Angaben sind in den vorhergehenden Abschnitten erläutert. Sind danach die Kenndaten noch nicht vollständig, so werden - soweit dies möglich und sinnvoll ist - vom PL/I-System vorgegebene Werte eingesetzt, die hier als System-Vorgabe für Dateien bezeichnet werden. Die so ermittelten Kommandos der BS2000-Datei müssen mit dem Attributsatz der PL/I-Datei verträglich sein.

Beim Schließen der Datei werden die Kenndaten in den Datei-Katalog übernommen. Die Datei ist damit katalogisiert und die Kenndaten sind mit ihr verbunden.

Im Bild 6-8 ist eine Übersicht über die hier beschriebenen Vorgänge gegeben. Die genaue Beschreibung ist in den nachfolgenden Abschnitten zu finden.

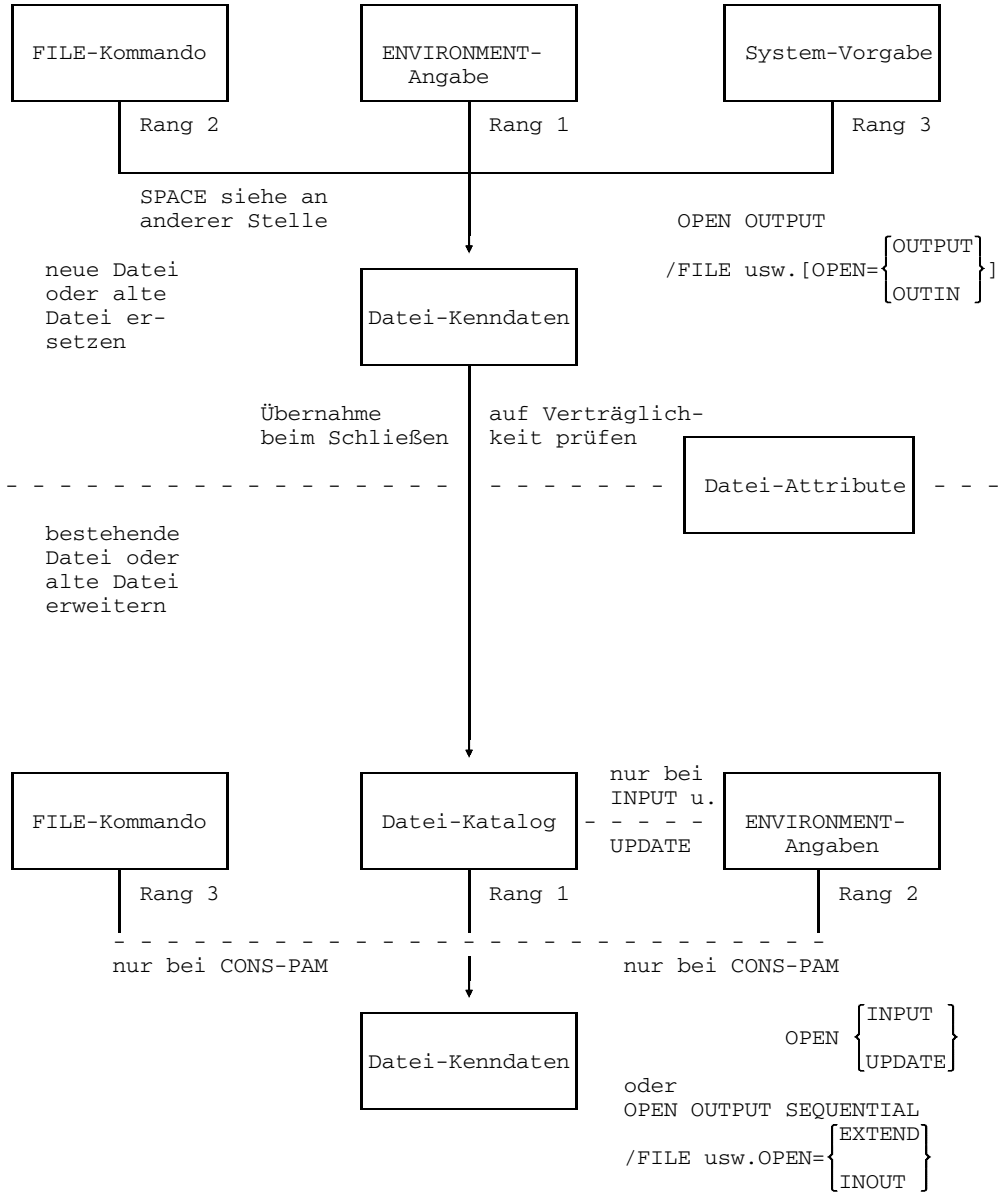


Bild 6-8 Zusammenstellen der Datei-Kenndaten und die Vorrangregeln dazu

6.4.4.1 Kenndaten für eine neue Datei

Wird eine Datei im PL/I-Programm mit dem Datei-Attribut OUTPUT eröffnet, so werden folgende Fälle unterschieden:

- Datei noch nicht vorhanden.
  Gibt es zu dem entsprechenden Dateinamen noch keine Datei, so wird eine neue Datei eingerichtet.
- Datei bereits vorhanden.
  Gibt es zu dem entsprechenden Dateinamen bereits eine Datei, so gibt es zwei Möglichkeiten:
  - Ist im FILE-Kommando durch den OPEN-Parameter festgelegt, daß die Datei erweitert werden soll und ist dies gemäß den Bedingungen, wie sie im folgenden Abschnitt beschrieben sind, möglich, so bleibt die Datei erhalten und es wird ähnlich wie beim Eröffnen mit OPEN UPDATE verfahren (siehe nachfolgende Abschnitte).
  - Trifft das vorstehend gesagte nicht zu, so wird so verfahren, als ob die Datei nicht existiert, d.h. es wird eine neue Datei eingerichtet.

  Ausnahme

  Werden bei OPEN OUTPUT auf eine vorhandene Datei im FILE-Kommando Nullparameter (z.B. RECSIZE=,) angegeben, so werden die entsprechenden FILE-Parameter aus dem Katalog besetzt.

Wird eine neue Datei eingerichtet, so werden die Kenndaten dafür in folgender Reihenfolge ermittelt:

1. Es werden zuerst die Werte aus dem FILE-Kommando in die Kenndaten übernommen, mit Ausnahme von SPACE=, das zu diesem Zeitpunkt bereits ausgewertet und in den Datei-Kenndaten bereits enthalten ist.
2. Fehlen bei den Kenndaten noch erforderliche Angaben, so werden sie danach, soweit vorhanden, aus der ENVIRONMENT-Angabe des PL/I-Programms übernommen.
3. Schließlich werden noch fehlende Werte als System-Vorgabe ergänzt. Diese Werte sind in Bild 6-9 aufgelistet.

Die Angabe RECSIZE(0) bzw. RECSIZE = 0 im Bild 6-9 bedeutet, daß der Wert undefiniert ist und vom Datenverwaltungssystem ein entsprechender Puffer angelegt wird (z.Zt. 2048 Bytes).
Ein Datensatz darf dann maximal diese Länge besitzen. Bei SAM-Dateien werden 4 Bytes für Verwaltungszwecke benötigt, so daß die maximal mögliche Länge des Datensatzes um diese 4 Bytes kleiner ist. Darüberhinaus wird bei SAM- und ISAM-Dateien auf nicht PAM-Key-behafteten Plattendateien ein 12 Byte langes Kontrollfeld vom Datenverwaltungssystem für jeden logischen Block belegt.

Bedingung	System-Vorgabe	
	ENVIRONMENT-Angabe	FILE-Kommando
	CONSECUTIVE	
CONSECUTIVE INDEXED REGIONAL		FCBTYPE = SAM FCBTYPE = ISAM FCBTYPE = PAM
[CONSECUTIVE INDEXED REGIONAL(3)] REGIONAL(1)	V	RECFORM = V RECFORM = F
[V U] F	RECSIZE(0) 3) RECSIZE(r) 1)	RECSIZE = 0 3) RECSIZE = r 1)
V [F U]	KEYLOC(5) KEYLOC(1)	KEYPOS = 5 KEYPOS = 1
	KEYLENGTH(8)	KEYLEN = 8
DEVICE=Band sonst:		BLKSIZE = b 1) BLKSIZE = (STD,n) 2)
OPEN...STREAM PRINT	CTLMACH	RECFORM (x,M)

1) keine Vorgabe; Angabe erforderlich

2) n: so gewählt, daß der Datensatz gemäß RECSIZE gerade hineinpaßt

3) Vorgabe des Datenverwaltungssystems (z.Z. 2048,bzw.SAM 2044 oder 2032) (siehe Text)

Bild 6-9 System-Vorgabe für die Datei-Kenndaten beim Einrichten neuer Dateien in der Reihenfolge von oben nach unten

Sind auf diese Weise die Kenndaten zusammengestellt, so wird geprüft, ob sie mit den im Programm angegebenen Datei-Attributen verträglich sind. Ist dies nicht der Fall, so wird die Bedingung UNDEFINEDFILE gesetzt.

Wird für eine neu eingerichtete Datei explizit eine CLOSE-Anweisung gegeben oder wird die Datei implizit beim Ende des Programmlaufs geschlossen (was im allgemeinen auch bei fehlerhaftem Programmlauf gegeben ist), so werden die Kenndaten der Datei in den Datei-Katalog übernommen und damit für die Datei festgehalten. Die Datei ist damit existent.

Ausnahme

Bei PAM-Dateien werden RECFORM und RECSIZE nicht mit in den Katalog übernommen und müssen daher bei jedem späteren Eröffnen der Datei angegeben werden. Dies ist nur relevant bei der Organgisationsform CONSECUTIVE; bei REGIONAL-Dateien werden diese Daten in einem Verwaltungskopf für die Datei festgehalten.

6.4.4.2 Alte Ausgabe-Datei verlängern (EXTEND)

Wird eine Datei durch OPEN OUTPUT eröffnet, so wird im allgemeinen eine neue Datei erzeugt. Ist bereits eine entsprechende Datei vorhanden, so wird so verfahren, als ob diese Datei nicht existent wäre: Kenndaten und Datensätze der alten Datei stehen dann nicht mehr zu Verfügung. Die Größe des aktuellen Speicherplatzes und die Sekundär-Zuweisung gemäß "/FILE Dateiname, SPACE =" bleiben erhalten.

Durch den Parameter OPEN im FILE-Kommando kann man jedoch bestimmen, daß die alte Datei bestehen bleibt und mit ihr die Kenndaten und die Datensätze, und daß neue Datensätze hinzugefügt werden. Die Datei muß mit dem Datei-Attribut RECORD eröffnet sein. Für die Erweiterung einer bestehenden Ausgabe-Datei müssen also folgende Bedingungen erfüllt sein:

- Die Datei wird mit OPEN OUTPUT [SEQUENTIAL] eröffnet
- Im FILE-Kommando ist angegeben:
  OPEN = EXTEND bei SAM oder ISAM
  OPEN = INOUT bei PAM

In Bezug auf die Kenndaten werden die gleichen Arbeiten durchgeführt, wie sie im folgenden Abschnitt für Dateien beschrieben sind, die mit UPDATE eröffnet werden, jedoch wird nicht auf Verträglichkeit mit den ENVIRONMENT-Angaben geprüft.

Bei CONSECUTIVE-Dateien wird beim Eröffnen auf das Ende der Datei positioniert, so daß nach den allgemeinen Regeln der Ausgabe-Anweisungen neue Datensätze ans Ende der Datei angefügt werden. Bei INDEXED- und REGIONAL-Dateien wird ebenfalls auf das Ende der Datei positioniert; die neuen Datensätze müssen dann mit Schlüsseln angeliefert werden, die größer sind als alle vorhandenen.

6.4.4.3 Kenndaten bei vorhandener Datei

Wird eine Datei mit dem Datei-Attribut INPUT oder UPDATE eröffnet, so muß die entsprechende Datei vorhanden sein. Das gleiche gilt, wenn eine Datei wie im vorigen Abschnitt beschrieben, erweitert werden soll. Die Kenndaten für diese Datei stehen dann bereits im Datei-Katalog und werden daraus übernommen.

Bei Dateien, die mit /FILE STATE = FOREIGN katalogisiert werden, werden die Kenndaten, soweit sie nicht explizit angegeben sind, aus den Kennsätzen entnommen.

Ausnahme

Bei PAM-Dateien werden RECFORM und RECSIZE nicht im Katalog festgehalten und müssen daher im FILE-Kommando oder bei ENVIRONMENT angegeben werden; dabei hat eine Angabe bei ENVIRONMENT Vorrang. Diese ist nur relevant für die Organisationsform CONSECUTIVE.

Sind im PL/I-Programm ENVIRONMENT-Angaben vorhanden, so wird geprüft, ob sie mit den Kenndaten der Datei verträglich sind. Ferner wird geprüft, ob die Kenndaten mit den beim Eröffnen der Datei angebenen Datei-Attribute verträglich sind. Fällt eine dieser Prüfungen negativ aus, so wird die Bedingung UNDEFINEDFILE gesetzt.

Ausnahme

Wird eine Datei mit OUTPUT eröffnet und erweitert, wie das im vorhergehenden Abschnitt beschrieben ist, so wird nicht auf Verträglichkeit mit den ENVIRONMENT-Angaben geprüft.

Das Eröffnen (INPUT oder UPDATE) einer katalogisierten, leeren Datei führt beim ersten Lesen zur Bedingung ENDFILE.

6.5 Stromorientierte Ein- und Ausgabe

Die stromorientierte Ein- und Ausgabe (STREAM) ist die für den Benutzer bequemste Art des Datentransports zwischen PL/I-Programm und externem Datenträger. Sie ist zudem weitgehend unabhängig von unterschiedlichen Rechensystemen und die Datenbestände können zwischen verschiedenen Rechnern nahezu beliebig ausgetauscht werden. Bei der Eingabe werden die externen Daten als kontinuierlicher Strom betrachtet und zeichenweise fortlaufend verarbeitet. Die Zeichen werden vom Eingabesystem den Benutzerwünschen entsprechend aufbereitet und in die interne Wertedarstellung konvertiert. Bei der Ausgabe wird die interne Wertedarstellung den Benutzerangaben entsprechend in Zeichenform konvertiert und zu einem kontinuierlichen Strom von Zeichen zusammengefügt. Dabei kann der Benutzer verschiedene Stufen der Steuerung wählen, entweder die einfache DATA- oder LIST-gesteuerte Aufbereitung, bei denen die Konvertierung in die externe Darstellung durch PL/I festgelegt ist, oder die EDIT-gesteuerte Aufbereitung, bei der der Benutzer die externe Darstellung selbst zeichengenau festlegen kann.

Für die Druckausgabe kann der Datenstrom in Zeilen und Seiten unterteilt werden, was in Abschnitt 6.5.2 näher erläutert wird.

6.5.1 Regeln für die stromorientierte Ein- und Ausgabe

Die von der stromorientierten Ein- und Ausgabe zu bearbeitenden Dateien müssen CONSECUTIVE organisiert sein. Andere Organisationsformen können nicht verwendet werden.

Als BS2000-Zugriffsmethoden können für die stromorientierte Ein- und Ausgabe SAM und ISAM und die Systemdateien verwendet werden. Bei der Verwendung von ISAM gelten bezüglich Schlüsselposition und Schlüssellänge die gleichen Sonderregeln wie bei satzorientierter Verarbeitung mit CONSECUTIVE (Schlüsselposition nur 1 oder 5, Schlüssellänge nur 4 oder 8). Als Satzformat sind F, V und U zugelassen. Dabei wird in Verbindung mit U je Satz genau ein Block transportiert, wobei sich die maximal mögliche Satzlänge aus BLKSIZE ergibt. Die Verwendung des U-Formats kann u.U. zu hohem Verschnitt führen.

Für das FILE-Kommando gelten die gleichen Angaben wie sie im Abschnitt 6.6.1.7 gegeben sind, jedoch ist FCBTYPE = PAM nicht erlaubt.

Für die Vorgänge Öffnen, Schreiben, Lesen und Schließen der Datei gelten die gleiche Regeln wie für die satzorientierte Verarbeitung. Diese Regeln sind unter Abschnitt 6.6.1 zu finden.

Die in PL/I möglichen Dateiattribute für STREAM-Dateien und die anwendbaren Anweisungen zeigt die folgende Zusammenstellung:

Attributsatz			Anweisung	Organisation
STREAM	INPUT1)		GET [SKIP]	CONSECUTIVE
	OUTPUT	LINESIZE	PUT [SKIP]	
	OUTPUT PRINT	PAGESIZE LINESIZE	[PAGE,LINE] PUT[ ] [SKIP]	

1) nicht auf leere Datei

Die Länge eines jeden Datensatzes ergibt sich in erster Linie durch die Anwendung von SKIP in der GET- oder PUT-Anweisung bzw. in der Format-Angabe bei EDIT-Steuerung. Wenn jedoch SKIP nicht vor dem Erreichen der maximalen Satzlänge angewendet wird, erfolgt gemäß der Stromdefinition automatischer Übergang zum neuen Satz.
Die maximale Satzlänge ist festgelegt durch:

1. Attribut LINESIZE bei OPEN (nur für OUTPUT)
2. RECSIZE bzw. BLKSIZE des FILE-Kommandos bzw. des ENVIRONMENT-Attributs
3. Steueranweisung SYSFILE (für Systemdateien)
4. Voreinstellung (siehe Bild 6-10)

Für System-Dateien werden die vorstehenden Angaben in der angegebenen Reihenfolge untersucht, bis ein Wert gefunden wird.

Für die anderen Dateien gelten folgende Zusatzregeln:

- Angaben von RECSIZE (BLKSIZE) und Geräteeigenschaften sind immer vorrangig, wenn diese Angaben kleiner als LINESIZE bei OPEN bzw. SYSFILE-Steueranweisung sind.
- Ist bei Dateien im F-Format RECSIZE > LINESIZE, so wird der Zeichenstrom beim Schreiben bis zur angegebenen Satzlänge mit Leerzeichen aufgefüllt.

Datei	Dialogprozeß	Stapelprozeß
SYSLST	120	120
SYSOUT	Zeilenlänge des Gerätes[1)]	132
SYSDTA	120	120
Datei mit RECFORM=F oder U	RECSIZE[2)] oder 120	
Datei mit RECFORM=V	RECSIZE[2)] oder 120	

1) ggf. 1 Zeichen weniger wegen Wagenrücklauf
2) Bei PRINT-Dateien wird noch ein Zeichen für die Vorschubsteuerung abgezogen

Bild 6-10 Voreinstellungen für LINESIZE bei stromorientierter Ein-/Ausgabe

Die Grenzwerte für Systemdateien gelten auch dann für SYSLST bzw. SYSDTA, wenn diese Dateien mittels SYSFILE-Kommando einer Benutzerdatei zugeordnet wurden. Bei diesen Dateien hat der Benutzer die Möglichkeit, während des Programmlaufs die Satzlänge zu ändern, indem er die Datei schließt (CLOSE), dann erneut eröffnet und in der OPEN-Anweisung eine neue LINESIZE-Angabe gibt.

Der automatische Übergang zum nächsten Datensatz erfolgt bei EDIT-gesteuerter Übertragung genau an der jeweils eingestellten Grenze. Bei der Ausgabe werden also gegebenenfalls Werte aufgebrochen und bei der Eingabe aufeinanderfolgende Zeilen verkettet.

Bei der DATA- oder LIST-gesteuerten Ausgabe werden die einzelnen Ausgabewerte nicht am Satzende aufgebrochen, sondern in den nächsten Satz geschrieben, wenn sie nicht mehr vollständig im aktuellen Datensatz Platz finden; d.h. wenn LINESIZE überschritten würde. Ausgabewerte werden nur dann aufgebrochen, wenn das Element allein größer als LINESIZE ist.

6.5.2 PRINT-Dateien

Dateien, die mit den Attributen STREAM OUTPUT PRINT implizit oder explizit vereinbart werden, sind Druck- bzw. PRINT-Dateien. Sie können nur mit der Anweisung PUT angesprochen werden. Durch die PUT-Anweisung wird eine Zeichenfolge erzeugt, in die Vorschubzeichen für Seiten und Zeilen eingefügt sind. Diese Vorschubzeichen unterteilen die Zeichenfolge in Zeilen, die jeweils einem Satz der externen Datei zugeordnet werden. Dieses Steuerzeichen wird jedem Satz einer PRINT-Datei automatisch vorangestellt. Die Codierung dieses Zeichens kann mit den ENVIRONMENT-Angaben CTLMACH bzw. CTLASA beeinflußt werden (siehe Abschnitt 6.3.4).

Bei der Ausgabe in die Systemdateien SYSLST und SYSOUT werden stets Vorschubsteuerzeichen gemäß CTLMACH erzeugt. Eine anders lautende Angabe wird ignoriert.

Bei der Ausgabe in eine Datei die mit "/FILE...RECFORM=(x,N)" eingerichtet wurde, werden Vorschubsteuerzeichen wie bei "RECFORM=(x,M)" erzeugt.

Wenn die PRINT-Datei nicht auf einem Drucker ausgegeben wird, so hat das Steuerzeichen keine Auswirkung, es wird dann wie ein Teil des Datensatzes behandelt.

Ein neuer Datensatz wird immer dann erzeugt, wenn in der Ausgabe-Anweisung eine SKIP- bzw. PAGE-Angabe verarbeitet wird oder der Datenstrom die durch LINESIZE explizit oder implizit vereinbarte Satzlänge erreicht hat (siehe Abschnitt 6.5.1). Bei der DATA- und LIST-gesteuerten Ausgabe wird der neue Datensatz bereits dann begonnen, wenn der aktuelle Ausgabewert im laufenden Satz nicht mehr komplett angeordnet werden kann.

Für die voreingestellten Satzlängen gelten die Festlegungen aus Abschnitt 6.5.1. Dabei ist aber zu berücksichtigen, daß jeweils der Platz für das generierte Steuerzeichen berücksichtigt werden muß.

Die Seitenlänge für eine PRINT-Datei kann auf folgende Weise vereinbart werden:

1. PAGESIZE-Angabe in der Anweisung OPEN
2. Steueranweisung *RUNOPT SYSFILE = PAGESIZE(x,y,z)
3. Voreinstellung:
   für SYSOUT im Dialog (Sichtgerät): PAGESIZE(1)
   sonst: PAGESIZE(60)

Die Angaben werden in der angegebenen Reihenfolge untersucht bis ein Wert gefunden ist. Beim Erreichen des Seitenendes wird die Bedingung ENDPAGE gesetzt.

Wurde eine PRINT-Datei auf einem externen Datenträger abgespeichert, d.h. sie wurde nicht über SYSLST automatisch ausgedruckt, so kann man sie jederzeit mit dem PRINT-Kommando (SPACE=E) auf einem Drucker ausgeben.

	Drucker (Stapelbetrieb)	Dialoggerät	
		Zeilenmodus 1)	Seitenmodus
PAGESIZE(n)		n=1 1)	n>1
LINESIZE(n)			n≤ Zeilenlänge Gerät. Bei Gerät mit Wagenrücklauf: n≤ Zeilenlänge Gerät-1
SKIP-Angabe	Ausgabe der Zeile	Ausgabe der Zeile	Zeile nach Seitenpuffer
LINE(n)		wie SKIP(1)	
PAGE		wie SKIP(1)	Seitenpuffer ausgegeben
ENDPAGE		wird nie gesetzt	
LINENO		wächst unbegrenzt	
PAGENO		ergibt stets Wert 1	

1) Voreinstellung: Zeilenmodus

Bild 6-11 Besonderheiten für Dateien mit STREAM PRINT für die Systemdatei SYSOUT

6.5.3 Stromorientierte Ein-Ausgabe auf Dialoggerät

Um die Verwendung der STREAM-Ein-Ausgabe bei Dialoggeräten (Systemdatei, SYSOUT, SYSDTA) zu erleichtern, wird in diesem Fall von den PL/I-Regeln in einigen Fällen etwas abgewichen:

- Ist das letzte Zeichen, das auf der Dialogstation ausgegeben wird, ein Doppelpunkt (:), so wird zusätzlich nach der Ausgabe eine SKIP-Funktion ausgeführt (als ob die Anweisung PUT FILE(a) SKIP; folgt).

  Dies gilt für alle Ausgaben, die auf die System-Datei SYSOUT gehen, unabhängig davon, auf welchem Wege diese Datei in PL/I angesprochen wird:

  – PUT FILE (SYSOUT)...

  – OPEN FILE(a) TITLE ('SYSOUT')
    PUT FILE(a)...

  – *RUNOPT SYSFILE = TITLE ('b')
    OPEN FILE(a) TITLE('b');
    PUT FILE(a)...

- Bei der Ausgabe von Zeichenfolgen werden (wie bei PRINT-Dateien) die umschließenden Anführungszeichen nicht ausgegeben.

6.5.3.1 Eingabe vom Dialoggerät (SYSDTA)

Bei der STREAM-Eingabe von SYSDTA gelten folgende Abweichungen gegenüber der Eingabe von Dateien:

- Für LIST- und DATA-Eingabe bildet das Zeilenende einen Trenner für die Elemente. Gehen dem Zeilenende die Trenner Leerzeichen, Komma, Semikolon voraus, so wird das ganze wie ein Trenner behandelt. Für die Grenzwerte gilt somit

  – leere Eingabe: wird ignoriert
  – Zeile nur aus Leerzeichen: wird ignoriert
  – Leerzeichen..., Leerzeichen... Zeilenende: bilden zusammen einen Trenner
  – eine Zeile nur aus einem Trenner: ein leeres Element

- Für EDIT-Eingabe bildet das Zeilenende das Ende der einzugebenden Folge. Ist die Eingabefolge zu kurz, so wird sie mit Leerzeichen auf die geforderte Länge aufgefüllt.

- Ist das letzte Zeichen der Eingabefolge ein Bindestrich (-), wird die Trenner-Funktion des Zeilenendes in dieser Zeile aufgehoben: Die Eingabefolge setzt sich in der nächsten Zeile fort. Der Bindestrich wird aus der Eingabefolge entfernt.

6.5.3.2 Ausgabe auf Dialoggerät (PRINT-Datei auf SYSOUT)

Wird eine Datei mit den Attributen STREAM PRINT in die Systemdatei SYSOUT ausgegeben, so ergeben sich für den Dialogbetrieb einige Besonderheiten gegenüber dem Stapelbetrieb.

Für die Ausgabe auf ein Dialoggerät ist zu unterscheiden zwischen dem Zeilenmodus und dem Seitenmodus.

Im Zeilenmodus wird der auszugebende Text zeilenweise an das Dialoggerät ausgegeben. Der Zeilenmodus ist voreingestellt. Er kann explizit dadurch eingestellt werden, daß die Seitengröße auf den Wert 1 gesetzt wird (PAGESIZE(1)).

Es gelten dann folgende Abweichungen:

- Die Funktion LINE wirkt wie SKIP(1).
- Die Funktion PAGE wirkt wie SKIP(1).
- Die Bedingung ENDPAGE wird nie gesetzt.
- SKIP(n) mit n>3 wird auf SKIP(3) reduziert.
- Die Zeilennummer, die über die eingebaute Funktion LINENO erhalten werden kann, wächst unbegrenzt.
- Die Seitennummer, die über die eingebaute Funktion PAGENO erhalten werden kann, hat stets den Wert 1.

Der Seitenmodus wird dadurch eingestellt, daß die Seitengröße auf einen Wert größer 1 eingestellt wird. In diesem Modus werden die Zeilen einer Seite in einem Seitenpuffer zwischengespeichert. Ein impliziter oder expliziter Seitenvorschub (PAGE) bewirkt, daß der Inhalt des Seitenpuffers auf dem Dialoggerät ausgegeben wird. Bei einem Sichtgerät wird der ganze Bildschirm vorher gelöscht.

Es gelten folgende Abweichungen:

- Der Wert für die Zeilenlänge (LINESIZE) darf nicht größer sein als die Zeilenlänge beim Ausgabegerät; andernfalls wird die Bedingung UNDEFINED FILE gesetzt.

  Bei Ausgabegeräten mit Wagenrücklauf muß die Zeilenlänge (LINESIZE) um 1 kleiner sein als die Zeilenlänge beim Ausgabegerät.
- Wird eine neue Zeile begonnen, so wird die alte Zeile nicht an das Ausgabegerät ausgegeben, sondern im Seitenpuffer abgelegt.
- SKIP(n) mit n>3 wird auf SKIP(3) reduziert.

- Die Funktion PAGE bewirkt, daß der Inhalt des Seitenpuffers an das Ausgabegerät ausgegeben wird.
- Der Wert für die Seitengröße (PAGESIZE) darf für Sichtgeräte nicht größer sein als die für den Bildschirm erlaubte Zeilenzahl-1; andernfalls wird die Bedingung UNDEFINEDFILE gesetzt.
- Wird die eingestellte Seitengröße überschritten, so wird implizit die Funktion PAGE ausgeführt.

6.6 Satzorientierte Ein- und Ausgabe

Die satzorientierte Ein- und Ausgabe wird für solche Dateien angewendet, die im PL/I-Programm das Attribut RECORD besitzen. Dieses Attribut kann der Programmierer der Datei direkt zuordnen oder es wird aufgrund anderer Dateiattribute (z.B. UPDATE, DIRECT usw.) implizit angenommen (siehe Kapitel 8 der PL/I-Sprachbeschreibung [1]). Bei der satzorientierten Ein- und Ausgabe werden die Daten in der rechnerinternen Form übertragen. Dateien, welche in dieser Betriebsart erstellt werden, sind somit zwischen verschiedenen Rechnersystemen nur beschränkt austauschbar.

Bei der satzorientierten Übertragung ist die Transporteinheit aus der Sicht des PL/I-Programms ein Datensatz der externen Datei; d.h. bei jedem READ, REWRITE, LOCATE oder WRITE wird genau ein Satz übertragen. Für die satzorientierte Ein- und Ausgabe sind alle PL/I-Organisationsformen zulässig.

Dieser Abschnitt ist daher nach den Organisationsformen

```
CONSECUTIVE
INDEXED
REGIONAL(1) und
REGIONAL(3)
```

untergliedert.

Eine Übersicht über die zulässigen Ein-/Ausgabeanweisungen für RECORD-Dateien in Abhängigkeit von Dateiattributen und Organisationsform gibt Bild 6-12. Einige Ausnahmen zu diesen Kombinationen entnehme man den folgenden Abschnitten.

Attributsatz			Anweisung	Organisation		
RECORD	SEQUENTIAL	OUTPUT	WRITE FROM LOCATE [SET]	CONS		
		INPUT	READ {INTO / SET / IGNORE}		IND	REG
		UPDATE	READ {INTO / SET / IGNORE} REWRITE [FROM]			
			DELETE			
	SEQUENTIAL KEYED	OUTPUT	WRITE FROM KEYFROM LOCATE [SET] KEYFROM		IND	REG
		INPUT	READ {INTO / SET} [KEY]			REG (1)
			READ {INTO / SET} [KEYTO] READ IGNORE			REG
		UPDATE	READ {INTO / SET} [KEY]			REG (1)
			READ {INTO / SET} [KEYTO] READ IGNORE			REG
			WRITE FROM KEYFROM			
			REWRITE [FROM]			REG
			DELETE [KEY]			1)
	DIRECT KEYED	OUTPUT	WRITE FROM KEYFROM			REG
		INPUT	READ INTO KEY		IND	
		UPDATE	READ INTO KEY WRITE FROM KEYFROM REWRITE FROM KEY DELETE KEY			

1) Ausnahmen siehe Text

[ ] nicht möglich

FCBTYPE	SAM*	ISAM	PAM
RECFORM	FVU	FV	FV

Bild 6-12 Zulässige EA-Anweisungen bei verschiedenen Dateiattributen und Organisationsformen

Als Besonderheit ist bei der satzorientierten Übertragung die SCALARVARYING-Angabe im ENVIRONMENT-Attribut zu beachten. Schreiben und Lesen von skalaren Folgen mit Attribut VARYING im LOCATE-Modus (LOCATE, READ SET) sind nur mit der SCALARVARYING-Angabe sinnvoll. Einzelheiten hierzu findet man im Abschnitt 6.3.5.

Bei satzorientierter Ausgabe ist es ebenfalls möglich, Datenbestände zu erzeugen, die anschließend mit dem PRINT-Kommando ausgedruckt werden sollen. Ist Vorschubsteuerung erwünscht (SPACE=E im PRINT-Kommando), so beachte man folgende Punkte:

- Die Organisationsform muß CONSECUTIVE sein.
- Das Vorschubsteuerzeichen muß vom Programm erzeugt und als erstes Zeichen des Datensatzes ausgegeben werden.
- Die Codierung der Vorschubsteuerzeichen findet man in Abschnitt 6.3.4. Der Typ der Steuerzeichen (insbesondere ASA) sollte im FILE-Kommando oder im ENVIRONMENT-Attribut angegeben werden.

6.6.1 Regeln für CONSECUTIVE-Organisation

CONSECUTIVE-Dateien sind sequentiell in ihrer Zugangs- oder Schlüsselreihenfolge organisiert. Eine CONSECUTIVE-Datei darf nur mit dem Datei-Attribut SEQUENTIAL bearbeitet werden. CONSECUTIVE kann auf die Zugriffsmethoden SAM, PAM und ISAM abgebildet werden, wobei für ISAM die Schlüssel automatisch erzeugt werden. CONSECUTIVE-Dateien können je nach Zugriffsart Sätze fester, variabler oder undefinierter Länge enthalten. Die möglichen Ein-/Ausgabeanweisungen sind in Bild 6-13 zusammengestellt.

Organisation	Attributsatz			Anweisung
CONSECUTIVE SAM (ISAM) (PAM) F V U	RECORD	SEQUENTIAL	OUTPUT	WRITE FROM LOCATE
			INPUT	READ{INTO | SET | IGNORE}
			UPDATE	READ{INTO | SET | IGNORE} REWRITE [FROM]

Bild 6-13 EA-Anweisungen für CONSECUTIVE-Organisation

6.6.1.1 Eröffnen von CONSECUTIVE-Dateien

Werden PAM-, SAM- oder ISAM-Dateien mit dem Datei-Attribut OUTPUT eröffnet, so wird auf den Dateianfang positioniert. In der Datei vorhandene Datensätze gehen verloren. Wird jedoch im FILE-Kommando

```
/FILE...OPEN=INOUT oder
/FILE...OPEN=EXTEND
```

angegeben, so wird auf das Dateiende positioniert und die neuen Datensätze werden der Datei hinzugefügt.

Das Eröffnen mit dem Dateiattribut UPDATE ist nur für Dateien auf Direktzugriffsträgern zulässig. Bei OPEN UPDATE wird vorausgesetzt, daß die Datei schon erstellt ist. Die aktuelle Satzposition steht nach OPEN auf Dateianfang.

Handelt es sich um eine ISAM-Datei, so müssen KEYLEN bzw. KEYLENGTH gleich 4 oder 8 und KEYPOS bzw. KEYLOC gleich 1 oder 5 sein. Andernfalls folgt UNDEFINEDFILE-Bedingung.

6.6.1.2 Schließen von CONSECUTIVE-Dateien

Eine Datei wird entweder explizit durch Angabe der CLOSE-Anweisung oder implizit bei Beendigung des Programms geschlossen. Eventuell noch vorhandener Ausgabe-Puffer wird in die Datei ausgegeben. Die Zuordnung einer PL/I-Datei zur BS2000-Datei wird aufgehoben. Die der Datei im PL/I-Programm zugeordneten Puffer werden freigegeben.

6.6.1.3 Schreiben in eine CONSECUTIVE-Datei

Die Datei muß mit dem OPEN-Attribut OUTPUT eröffnet worden sein. Mit der Anweisung WRITE wird ein Satz ausgegeben. Mit der Anweisung LOCATE wird ein Zeiger auf den nächsten freien Platz im EA-Puffer übergeben. Wenn RECSIZE und die Länge der in der WRITE- bzw. LOCATE-Anweisung spezifizierten Variablen unverträglich sind, wird RECORD-Bedingung gemeldet.

Handelt es sich um eine ISAM-Datei, so wird der Schlüssel vom Ein-/Ausgabesystem automatisch erzeugt. Für KEYLENGTH bzw. KEYLEN kann in diesem Fall nur 4 oder 8 angegeben werden und KEYPOS muß 1 bzw. 5 sein. Ist KEYLEN = 4, so werden binäre Schlüssel (FIXED BIN (31,0)) in der Schrittweite 1 erzeugt (FORTRAN-Form), ist KEYLEN = 8 angegeben, so wird der Schlüssel als 8-stellige Dezimalzahl (PICTURE'(8)9') in der Schrittweite 1 erzeugt (verträglich mit EDT und EDOR).

6.6.1.4 Lesen aus einer CONSECUTIVE-Datei

Das OPEN-Attribut muß INPUT oder UPDATE sein. Bei der Anweisung READ INTO wird ein Satz aus dem Eingabepuffer gelesen. Bei READ SET wird der Zeiger auf den Satz im Eingabepuffer übergeben. Bei der Anweisung READ IGNORE(n) werden so viele Sätze gelesen, aber nicht übergeben, wie durch n angegeben ist.
Bei Unverträglichkeit zwischen der aktuellen Satzlänge der Datei und der in der READ-Anweisung spezifizierten Variablen wird RECORD-Bedingung, bei Übertragungsfehler wird TRANSMIT-Bedingung und bei Erreichen von Dateiende ENDFILE-Bedingung gemeldet.
Der Schlüssel einer ISAM-Datei wird nicht an das Programm ausgeliefert.

6.6.1.5 Überschreiben von Sätzen einer CONSECUTIVE-Datei

Das OPEN-Attribut muß UPDATE sein. Es wird der Satz überschrieben, der unmittelbar zuvor erfolgreich gelesen wurde; d.h. es darf bei READ keine Fehlermeldung aufgetreten sein und es darf nicht READ IGNORE (n) vorangegangen sein. RECORD-Bedingung wird gemeldet, wenn Satz- und Variablenlänge nicht verträglich sind bzw. bei den Zugriffsmethoden SAM und PAM mit variabler Satzlänge das Zurückschreiben verkürzter oder verlängerter Datensätze versucht wird. Bei ISAM ist dies zulässig. Übertragungsfehler führen auf TRANSMIT-Bedingung.

6.6.1.6 Löschen von Sätzen in einer CONSECUTIVE-Datei

Löschen von Sätzen einer CONSECUTIVE-Datei ist nicht möglich.

6.6.1.7 FILE-Kommando für CONSECUTIVE-Dateien

Falls die Datei nicht bereits katalogisiert ist oder das ENVIRONMENT-Attribut entsprechende Angaben enthält, sollte das FILE-Kommando folgende Parameter enthalten.

```
/FILE     Dateiname,  (Name des Datenbestandes)
 LINK   = PL/I-Title,
          ⎧SAM  ⎫
 FCBTYPE=⎨ISAM  ⎬,
          ⎩PAM  ⎭

 RECSIZE= r,           (Anzahl Zeichen inkl. Verwaltungs-Information)
          ⎧F|V|U                  ⎫
 RECFORM=⎨                        ⎬ ,
          ⎩({F|V|U}[,{A|M}])      ⎭

          ⎧STD     ⎫     Puffergröße = 1 PAM-Block
 BLKSIZE=⎨(STD,n)  ⎬,    Puffergröße in n PAM-Blöcken (max. 16)
          ⎩m       ⎭     Puffergröße in Anzahl Zeichen
                         (notwendig für Banddateien)

          ⎧1⎫
 KEYPOS =⎨ ⎬,           bei RECFORM = F       ⎫
          ⎩5⎭           bei RECFORM = V       ⎪
                                              ⎪
          ⎧4⎫                                 ⎬      (nur bei ISAM)
 KEYLEN =⎨ ⎬,                                 ⎪
          ⎩8⎭                                 ⎭

          ⎧e       ⎫    e:  Erstzuweisung/Verlängerung/Verkürzung
 SPACE  =⎨        ⎬
          ⎩(e[,z]) ⎭    z:  Inkrement

          ⎧EXTEND⎫
 OPEN   =⎨      ⎬,
          ⎩INOUT ⎭
```

Regeln für Parameter:

- Für RECSIZE ist die physikalische Satzlänge in Anzahl Zeichen einzutragen; d.h. Satzlängenfeld, Vorschubsteuerzeichen, Satzschlüssel usw. sind ggf. in die Angabe einzubeziehen.
- Für RECFORM ist die Angabe U nur in Verbindung mit Banddateien sinnvoll. Die Angaben A bzw. M werden nur benötigt, wenn die Datei anschließend mit dem PRINT-Kommando ausgedruckt werden soll (siehe Abschnitt 6.5.2).
- Für BLKSIZE muß mit Ausnahme von SAM-Banddateien die Pufferlänge in PAM-Blöcken (max. 16 PAM-Blöcke) angegeben werden.
  Maximalangabe für Banddateien ist 32762 Zeichen.

- Die Werte für den SPACE-Parameter im FILE-Kommando sind entsprechend der einzutragenden Datenmenge in Anzahl PAM-Blöcken anzugeben; sie müssen insbesondere mit dem Wert von BLKSIZE verträglich sein. Bei SAM sollte ferner Erst- und Zweitzuweisungsangabe sowohl durch 3 als auch durch die Anzahl der PAM-Blöcke je logischem Block teilbar sein. Bei ISAM sollte ferner Platz für den Index-Bereich vorgesehen sein. Es ist also mindestens 1 PAM-Block mehr anzugeben als bei BLKSIZE. Bei falscher Angabe wird der Wert beim Eröffnen der Datei auf den nächsthöheren erlaubten Wert gesetzt.

Beispiel

```
/FILE ERGEBNIS,LINK=AUSGABE,FCBTYPE=SAM,RECFORM=V,RECSIZE=84,      -
                BLKSIZE=STD,SPACE=(3,3)
```

unter Berücksichtigung der System-Vorgabe ist dies gleichwertig mit

```
/FILE ERGEBNIS,LINK=AUSGABE,RECSIZE=84
```

Weitere Beispiele für FILE-Kommandos findet man in Abschnitt 6.2.2.5.

6.6.2 Regeln für INDEXED-Organisation

Bei einer INDEXED-organisierten Datei sind die Sätze mit einem Schlüssel versehen. Dieser Schlüssel ist im Datenbestand stets Teil des jeweiligen Datensatzes. Aus Sicht des PL/I-Programms kann er auch vor dem Datensatz stehen. Der Zugriff auf die Sätze ist direkt oder sequentiell nach der Sortierfolge der Schlüssel möglich. Die Datei muß sich auf einem Direktzugriffsträger befinden.
Eine INDEXED-organisierte Datei kann entweder

```
          SEQUENTIAL
oder      SEQUENTIAL KEYED
oder      DIRECT KEYED
```

verarbeitet werden. Die Organisationsform INDEXED ist nur mit der Zugriffsmethode ISAM verträglich. Die Dateien können Sätze mit variabler oder fester Länge haben. Die möglichen Ein-/Ausgabeanweisungen sind in Bild 6-14 zusammengestellt.

Organisation	Attributsatz			Anweisung
INDEXED ISAM F V	RECORD	SEQUENTIAL	INPUT	READ {INTO | SET | IGNORE}
			UPDATE	READ {INTO | SET | IGNORE} REWRITE [FROM] DELETE
		SEQUENTIAL KEYED	OUTPUT	WRITE FROM KEYFROM LOCATE KEYFROM
			INPUT	READ {INTO | SET} {[KEY] | [ ] | [KEYTO]} READ IGNORE
			UPDATE	READ {INTO | SET} {[KEY] | [ ] | [KEYTO]} READ IGNORE WRITE FROM KEYFROM REWRITE [FROM] DELETE [KEY]
		DIRECT KEYED	INPUT	READ INTO KEY
			UPDATE	READ INTO KEY WRITE FROM KEYFROM REWRITE FROM KEY DELETE KEY

Bild 6-14 E/A-Anweisungen für INDEXED-Organisation

6.6.2.1 Schlüsselangabe

Der im Programm bei KEY oder KEYFROM angegebene Schlüssel wird nötigenfalls in den Datentyp CHARACTER konvertiert. Die den Schlüssel bildende Zeichenfolge darf maximal die Länge 255 haben.

Beim Schreiben in eine Datei wird der vom Programm gelieferte Schlüssel in den durch KEYPOS und KEYLEN bzw. KEYLOC und KEYLENGTH definierten Bereich des Datensatzes eingetragen. Er überschreibt dabei den entsprechenden Bereich in der Information, welche mit der Satzvariablen geliefert wird. Man beachte jedoch die besondere Wirkung für
ENV (KEYLOC(0)), welche im Abschnitt 6.3.3 beschrieben ist.

Ist der vom Programm gelieferte Schlüssel kürzer als die Angabe bei KEYLEN bzw. KEYLENGTH, so wird er rechts mit Leerzeichen aufgefüllt. Ist der angelieferte Schlüssel länger, so wird er rechts abgeschnitten.

Beim Lesen wird der im Datensatz stehende Schlüssel in die bei KEYTO genannte Variable ausgeliefert. Hat diese das Attribut CHARACTER und die Länge dieser Folge stimmt nicht mit KEYLEN bzw. KEYLENGTH überein, so wird rechts abgeschnitten oder mit Leerzeichen aufgefüllt. Ist die Variable keine CHARACTER-Folge, kann jede Bedingung auftreten, die auch bei Zuweisungen vorkommt.

Für Dateien, die mit

```
                          ⎡INPUT ⎤
RECORD SEQUENTIAL KEYED   ⎨      ⎬
                          ⎣UPDATE⎦
ENVIRONMENT (INDEXED GENKEY)
```

eröffnet werden, wird durch die Angabe GENKEY bestimmt, daß ein durch READ KEY (Schlüssel) angegebener Schlüssel auch kürzer sein darf, als er für die Datei vereinbart ist. Es wird dann der erste Datensatz gelesen, dessen Schlüssel mit der angegebenen Zeichenfolge beginnt, ist ein solcher Datensatz nicht vorhanden, so wird die Bedingung KEY gesetzt.

Gibt es mehrere Schlüssel, die mit der angegebenen Zeichenfolge beginnen, so kann durch READ KEY (Schlüssel) nur der erste Datensatz (der mit dem niedrigsten Schlüsselwert) gelesen werden; die weiteren können durch sequentielles Lesen (READ bzw. READ KEYTO) erreicht werden. Liegt der Schlüssel außerhalb des Datensatzes (KEYLOC (0)), so kann der Schlüssel des ersten Datensatzes nicht erhalten werden, da KEY und KEYTO nicht gemeinsam in einer Anweisung erlaubt sind.

6.6.2.2 Eröffnen einer INDEXED-Datei

INDEXED-organisierte Dateien dürfen SEQUENTIAL, SEQUENTIAL KEYED oder DIRECT (KEYED) eröffnet werden. Die Attribute DIRECT oder SEQL (ohne KEYED) dürfen nur mit den OPEN-Attributen INPUT oder UPDATE verwendet werden. DIRECT oder SEQUENTIAL ohne KEYED sind also in Verbindung mit dem Attribut OUTPUT nicht zulässig. Die Ersterstellung von Dateien, die mit dem Attribut DIRECT verarbeitet werden sollen, muß mit dem Attribut SEQUENTIAL KEYED erfolgen.

Wird eine Datei mit dem Dateiattribut INPUT oder UPDATE eröffnet, so ist die aktuelle Satzposition bei DIRECT KEYED nach dem Eröffnen undefiniert und steht bei SEQUENTIAL oder SEQUENTIAL KEYED auf dem Dateianfang. Der jeweilige Datenbestand muß mindestens einen Satz enthalten.

Der Zugriff auf "shared update" deklarierte Dateien (für Mehrfachbenutzung vorgesehene Dateien) wird bei Dateien, die mit dem Attribut UPDATE eröffnet werden, unterstützt. Die Mehrfachbenutzung von Dateien wird mit der Angabe von SHARUPD=YES im FILE-Kommando signalisiert. Siehe zu Mehrfachbenutzung von Dateien die Beschreibung Betriebssystem BS2000 Datenverwaltungssystem DVS [7].

6.6.2.3 Schließen einer INDEXED-Datei

Eine Datei wird explizit durch Angabe der CLOSE-Anweisung oder implizit bei Beendigung des Programms geschlossen.

Eventuell noch vorhandene Puffer von OUTPUT- oder UPDATE-eröffneten Dateien werden ausgegeben. Die mit der Datei verbundenen Puffer im PL/I-Programm werden freigegeben und die Zuordnung zur BS2000-Datei wird aufgehoben.

6.6.2.4 Schreiben in eine INDEXED-Datei

Für das Schreiben in eine INDEXED-organisierte Datei kann als OPEN-Attribut OUTPUT oder UPDATE angegeben werden. Bei WRITE wird der Satz geschrieben, bei der Anweisung LOCATE wird der Zeiger auf den reservierten Platz im Ausgabepuffer geliefert. WRITE ist nur erlaubt, wenn die Datei mit OUTPUT SEQUENTIAL KEYED, UPDATE SEQUENTIAL KEYED oder UPDATE DIRECT KEYED eröffnet wurde. Die LOCATE-Anweisung verlangt ein OPEN mit OUTPUT SEQUENTIAL KEYED.
KEY-Bedingung wird gemeldet, wenn ein Satz mit dem gleichen wie dem gelieferten Schlüssel schon vorhanden ist. Ebenfalls wird KEY-Bedingung gemeldet, wenn für den angelieferten Satz kein Platz mehr in der Datei ist. Desgleichen wird KEY-Bedingung bei OUTPUT gemeldet, wenn die Schlüsselreihenfolge nicht aufsteigend ist. Für die einzelnen Fälle gibt es unterschiedliche ONCODE-Werte, welche man dem Anhang entnehmen kann.

RECORD-Bedingung wird gemeldet, wenn Satz- und Variablenlänge unverträglich sind; TRANSMIT-Bedingung wird bei Übertragungsfehlern gemeldet.

Wird eine Datei mit OPEN = EXTEND (siehe Abschnitt 6.4.4.1) eröffnet, so müssen die neuen Datensätze mit Schlüssel angeliefert werden, die größer sind als alle anderen vorhandenen.

6.6.2.5 Lesen aus einer INDEXED-Datei

Für das Lesen aus einer INDEXED-organisierten Datei kann als OPEN-Attribut INPUT oder UPDATE angegeben werden.

Unverträglichkeit von Satz- und Variablenlängen führt auf RECORD-Bedingung.

Wird ein Satz mit dem angegebenen Schlüssel nicht gefunden, so führt das zur KEY-Bedingung.

TRANSMIT-Bedingung wird bei Übertragungsfehlern gemeldet.

Beim Lesen aus "shared update" deklarierten Dateien wird bei UPDATE eröffneten Dateien der Block, in dem sich der zu lesende Satz befindet, für andere Benutzer gesperrt. Diese Sperre wird erst nach einem READ-, REWRITE- oder DELETE-Auftrag gelöscht.

6.6.2.6 Überschreiben von Sätzen einer INDEXED-Datei

Beim Überschreiben von Sätzen einer INDEXED-Datei mit der Anweisung REWRITE muß das OPEN-Attribut UPDATE sein. Es wird der Satz überschrieben, dessen Schlüssel in der REWRITE-Anweisung angegeben ist (bei DIRECT); oder der Satz, der vorher gelesen wurde (bei SEQUENTIAL). KEY-Bedingung wird gemeldet, wenn kein Satz mit dem angegebenen Schlüssel existiert.

Nach dem Zurückschreiben des Satzes in "shared update" deklarierte Dateien wird der Block, in dem der Satz zurückgeschrieben werden soll, wieder für die Mehrfachbenutzung freigegeben.

6.6.2.7 Löschen von Sätzen einer INDEXED-Datei

Beim Löschen von Sätzen einer INDEXED-Datei mit der Anweisung DELETE muß das OPEN-Attribut UPDATE sein. Es wird der Satz gelöscht, dessen Schlüssel in der KEY-Angabe bezeichnet wurde oder der unmittelbar vorher gelesene Satz (bei SEQUENTIAL UPDATE). Existiert kein Satz mit diesem Schlüssel in der Datei so wird KEY-Bedingung gesetzt.

Beim Löschen von Sätzen in "shared update" deklarierten Dateien wird der Block, der den zu löschenden Satz enthält, nach dem Löschen wieder für die Mehrfachbenutzung freigegeben.

6.6.2.8 FILE-Kommando für INDEXED-Dateien

Falls die Datei nicht bereits katalogisiert ist oder das ENVIRONMENT-Attribut entsprechende Angaben enthält, sollte das FILE-Kommando folgende Parameter enthalten:

```
/FILE      Dateiname,        (Name des Datenbestandes)
 LINK    = PL/I-Title,
 FCBTYPE = ISAM,
 RECSIZE = r                 (Anzahl Zeichen)

 RECFORM = { F }
           { V }

 BLKSIZE = { STD     },      (Puffergröße = 1 PAM-Block)
           { (STD,n) },,
                             (Puffergröße in n PAM-Blöcken max. 16)

 KEYPOS  = 1≤ c ≤ (r-k),     (Position Schlüsselanfang)

 KEYLEN  = k,                (Anzahl Zeichen)

 SPACE   = { e       }       e:  Erstzuweisung
           { (e[,z]) }
                             z:  Zweitzuweisung (Verlängerung)

 OPEN    = EXTEND
```

Regeln für die Parameter:

- Für RECSIZE ist die Satzlänge in Anzahl Zeichen einzusetzen. Bei V-Dateien ist das Satzlängenfeld (4 Zeichen) zu berücksichtigen und bei ENV (KEYLOC(0)) muß die Schlüssellänge zur Satzlänge hinzugeordnet werden.
- Bei KEYPOS wird die Position des linken Zeichens des Schlüssels im Satz angegeben. Die kleinste mögliche Angabe ist 1 bei RECFORM = F bzw. 5 bei RECFORM = V.
- Die Werte für den SPACE-Parameter im FILE-Kommando sind entsprechend den einzutragenden Datenbeständen anzugeben; sie müssen insbesondere mit dem Wert von BLKSIZE verträglich sein. Zur Aufnahme des Index-Bereiches ist mindestens 1 PAM-Block mehr anzugeben als bei BLKSIZE.

Bei fehlenden Angaben kann die UNDEFINEDFILE-Bedingung gesetzt werden.

Beispiel

```
/FILE BESTAND,LINK=TYP,FCBTYPE=ISAM,RECSIZE=90,     -
/             RECFORM=V,BLKSIZE=STD,KEYPOS=5,       -
/             KEYLEN=6,SPACE=(18,6)
```

Weitere Beispiele findet man in den Abschnitten 2.2.6 und 6.2.2.5.

6.6.3 Regeln für REGIONAL(1)-Organisation

Eine REGIONAL(1)-Datei ist in Regionen gegliedert, denen je ein numerischer Schlüssel zugeordnet ist. REGIONAL(1)-Dateien werden mit der Zugriffsmethode PAM realisiert. Eine REGIONAL(1)-Datei kann nur Sätze fester Länge enthalten. Jede Region der Datei enthält nur einen Satz. Jede Region-Nummer entspricht somit einer relativen Satz-Position in der Datei. Der Satzschlüssel (KEY) ist gleichbedeutend mit der Region-Nummer und wird nicht in der Datei aufgezeichnet. Auf die Sätze kann man sequentiell oder direkt zugreifen.

Die Sätze werden eng hintereinander angeordnet, wie ein kontinuierlicher Strom, so daß ein PAM-Block einen oder mehrere Sätze, aber auch einen Teil eines Satzes enthalten kann. Aus der Region-Nummer (KEY) und der Satzlänge wird vom Ein-/Ausgabesystem die Block-Nummer und die relative Position eines Satzes innerhalb dieses Blockes bestimmt.

Die für REGIONAL(1) zulässigen Ein-/Ausgabe-Anweisungen sind in Bild 6-15 zusammengestellt.

Organisation	Attributsatz			Anweisung
REGIONAL PAM F V[2)]	RECORD	SEQUENTIAL	INPUT	READ {INTO / SET / IGNORE}
			UPDATE	READ {INTO / SET / IGNORE} REWRITE [FROM] DELETE
		SEQUENTIAL KEYED	OUTPUT	WRITE FROM KEYFROM LOCATE KEYFROM
			INPUT	READ {INTO / SET} [KEYTO][1)] READ IGNORE
			UPDATE	READ {INTO / SET} [KEYTO][1)] READ IGNORE REWRITE [FROM] DELETE
		DIRECT KEYED	OUTPUT	WRITE FROM KEYFROM
			INPUT	READ INTO KEY
			UPDATE	READ INTO KEY WRITE FROM KEYFROM REWRITE FROM KEY DELETE KEY

1) bei REGIONAL(1) auch mit KEY
2) nur bei REGIONAL(3)

Bild 6-15 EA-Anweisungen für REGIONAL(1)- und REGIONAL(3)-Organisation

6.6.3.1 Schlüsselangabe

Der vom Programm über KEY/KEYFROM gelieferte Schlüssel muß eine Zeichenfolge sein, die nur aus Ziffern (0 bis 9) und Leerzeichen statt führender Nullen bestehen darf. Die Länge sollte 8 Zeichen nicht überschreiten. Führende Leerzeichen werden als Nullen interpretiert. Sind mehr als 8 Ziffern angegeben, so werden nur die rechten 8 Ziffern genommen; sind weniger als 8 Ziffern angegeben, so werden links Leerzeichen angefügt.
Die Interpretation des Schlüssels beginnt innerhalb der 8 Zeichen von links. Zwischen Ziffern eingebettete Leerzeichen begrenzen den Schlüssel; d.h. ein Leerzeichen rechts von einer Ziffer sowie alle weiteren Zeichen rechts davon werden ignoriert. Werden nur Leerzeichen gefunden, so erhält der Schlüssel den Wert Null. Bei Verstoß gegen obige Regeln erfolgt KEY-Bedingung.

Der aus der KEY- bzw. KEYFROM-Angabe so gewonnene Schlüssel dient als Region-Nummer und wird nicht in der Datei aufgezeichnet. Der kleinste mögliche Schlüssel ist 0.

6.6.3.2 Scheinsätze

Eine REGIONAL(1)-Datei enthält gültige Sätze und/oder Scheinsätze. Beim Erstellen einer REGIONAL(1)-Datei werden alle Regionen mit Scheinsätzen vorbesetzt, wenn die Dateieröffnung mit DIRECT OUTPUT erfolgt. Wird mit SEQUENTIAL OUTPUT eröffnet, erfolgt das Eintragen von Scheinsätzen in Verbindung mit Schreibvorgängen.

Ein Scheinsatz wird durch 'FF'B4 im ersten Zeichen gekennzeichnet, der Rest des Satzes ist undefiniert. Der Benutzer kann jederzeit auf Scheinsätze zugreifen und muß diese selbst erkennen können. Das Ein-/Ausgabesystem prüft nicht auf Scheinsätze, d.h. es erfolgt keine KEY-Bedingung bei Zugriff auf solche.

6.6.3.3 Eröffnen einer REGIONAL(1)-Datei

Eine REGIONAL(1)-Datei kann in folgender Weise eröffnet werden:

- Erstmaliges Eröffnen(OUTPUT)
- Verlängern (OUTPUT, UPDATE)
- Bearbeiten (INPUT, UPDATE)

Das erstmalige Eröffnen einer REGIONAL(1)-Datei kann entweder mit DIRECT OUTPUT oder SEQUENTIAL OUTPUT erfolgen.

Für das erstmalige Eröffnen in Verbindung mit DIRECT OUTPUT wird beim OPEN der gesamte primäre Speicherplatz entsprechend der SPACE-Angabe des FILE-Kommandos vom Ein-/Ausgabesystem mit Scheinsätzen vorformatiert. Sekundäre SPACE-Angaben werden nicht berücksichtigt.

Für das erstmalige Eröffnen bei Angabe von SEQUENTIAL OUTPUT müssen die Sätze bezüglich ihrer Region-Nummer in aufsteigender Reihenfolge angeliefert werden. Jede übersprungene Region wird vom Ein-/Ausgabesystem mit einem Scheinsatz belegt.

Das Verlängern einer bestehenden Datei ist nur möglich, wenn im FILE-Kommando die Angaben OPEN = EXTEND (oder OPEN = INOUT) und SPACE = primär angegeben sind. Andernfalls wird eine bestehende Datei gelöscht und eine neue Datei "erstmals eröffnet". Verlängern bedeutet, daß einmal um "primär" PAM-Seiten verlängert wird, wobei "primär" ggf. auf ein Vielfaches von 3 und mindestens auf den Wert von BLKSIZE aufgerundet wird.

Für das Verlängern ist möglich

- SEQUENTIAL KEYED OUTPUT
  Die Schlüssel der neu angelieferten Sätze müssen größer sein als alle bisher vorhandenen.
- SEQUENTIAL KEYED UPDATE
- SEQUENTIAL DIRECT OUTPUT oder UPDATE

Nach der Erstellung einer REGIONAL(1)-Datei kann diese Datei mit den Attributen DIRECT INPUT bzw. DIRECT UPDATE oder mit SEQUENTIAL INPUT bzw. SEQUENTIAL UPDATE eröffnet werden.

6.6.3.4 Schließen einer REGIONAL(1)-Datei

Das Schließen einer Datei erfolgt entweder explizit durch Angabe der CLOSE-Anweisung oder implizit bei Beendigung des Programms.

Außer den allgemeinen üblichen Aktionen beim CLOSE-Vorgang füllt das Ein-/Ausgabesystem bei einer mit OUTPUT SEQUENTIAL eröffneten REGIONAL(1)-Datei den Speicherbereich hinter der aktuellen Position bis zum Dateiende mit Scheinsätzen auf.

6.6.3.5 Schreiben in eine REGIONAL(1)-Datei

Mit der Anweisung WRITE wird die angegebene Variable als Satz in den Ausgabepuffer geschrieben, mit der Anweisung LOCATE wird ein Zeiger auf den im Ausgabepuffer reservierten Platz geliefert. Es erfolgt keine Prüfung, ob der Satz schon vorhanden ist und es kann keine KEY-Bedingung aufgrund eines schon vorhandenen Schlüssels geben. Schließt der Satz bei SEQUENTIAL-Verarbeitung nicht unmittelbar an seinen Vorgänger an, werden alle übersprungenen Regionen mit Scheinsätzen besetzt.

KEY-Bedingung wird gemeldet, wenn der Wert des Schlüssels größer ist als die Nummer der letzten aufgrund der Dateikenndaten möglichen Region oder wenn der Schlüssel syntaktisch falsch ist.

6.6.3.6 Lesen aus einer REGIONAL(1)-Datei

Für das Lesen aus einer REGIONAL(1)-organisierten Datei muß als OPEN-Attribut INPUT oder UPDATE angegeben werden.

Mit der Anweisung READ wird ein Satz in die in der Anweisung spezifizierte Variable gelesen. Bei READ SET wird der Zeiger auf den Anfang des aktuellen Satzes im Eingabepuffer gesetzt. Zu beachten ist, daß auch Scheinsätze von dem Ein-/Ausgabesystem ausgeliefert werden. Der Benutzer muß diese selbst erkennen, es erfolgt beim Lesen von Scheinsätzen keine KEY-Bedingung. KEY-Bedingung wird gemeldet, wenn der Wert des Schlüssels größer ist als die Nummer der letzten beim Erstellen der Datei erzeugten Region oder wenn der Schlüssel syntaktisch falsch ist.

6.6.3.7 Überschreiben von Sätzen einer REGIONAL(1)-Datei

Soll ein Satz mit der Anweisung REWRITE überschrieben werden, so wird auch hier nicht vorher geprüft, ob es sich um einen Scheinsatz handelt oder nicht. Wurde die Datei mit dem Attribut SEQUENTIAL eröffnet, so muß der REWRITE-Anweisung ein erfolgreiches READ vorangegangen sein. Eine KEY-Bedingung kann also nicht erfolgen, es sei denn, der Schlüssel adressiert eine Region außerhalb der Datei oder er ist syntaktisch falsch.

6.6.3.8 Löschen von Sätzen einer REGIONAL(1)-Datei

Beim Löschen eines Satzes einer REGIONAL(1)-Datei mit der Anweisung DELETE wird der entsprechende Satz als Scheinsatz gekennzeichnet. Im 1. Zeichen eines Scheinsatzes stehen (8)'1'B. Wie beim Schreiben erfolgt keine Prüfung, ob der Satz existiert, es wird auch keine KEY-Bedingung gemeldet, es sei denn, der Schlüssel adressiert eine Region außerhalb der Datei oder er ist syntaktisch falsch. DELETE kann nur bei einer Datei verwendet werden, die mit UPDATE eröffnet wurde. Wurde eine Datei mit den Attributen SEQUENTIAL UPDATE eröffnet, so muß eine READ-Anweisung vorausgehen.

6.6.3.9 FILE-Kommando für REGIONAL(1)-Dateien

Falls die Datei nicht bereits katalogisiert ist oder das ENVIRONMENT-Attribut entsprechende Angaben enthält, sollte das FILE-Kommando folgende Parameter enthalten:

```
/FILE          Dateiname,    (Name des Datenbestandes)
 LINK      =   PL/I-Title,
[FCBTYPE   =   PAM,]         (voreingestellt)
 RECSIZE   =   r,            (Anzahl Zeichen)
 RECFORM   =   F,
 BLKSIZE   =   (STD,n),
 SPACE     =   e,            (Erstzuweisung bei OUTPUT/
                              Zweitzuweisung bei EXTEND oder INOUT)
```

Regeln für das FILE-Kommando:

- Bei RECSIZE wird als Satzlänge nur die tatsächliche Zeichenzahl eingetragen, der Schlüssel beeinflußt die Länge eines Satzes nicht und tritt im FILE-Kommando nicht in Erscheinung.
- In der SPACE-Angabe muß 1 PAM-Seite für Verwaltungsinformation berücksichtigt werden.
- Durch RECSIZE wird gleichzeitig die Puffergröße bestimmt. Eine BLKSIZE-Angabe wird nicht benötigt.
- SPACE legt den erforderlichen Speicherbedarf in Anzahl PAM-Blöcken fest und muß mit RECSIZE in Einklang stehen. Angabe einer Zweitzuweisung wird ignoriert.

Beispiel

Datei für 2000 Regionen

```
/FILE TELEFON,LINK=NR,FCBTYPE=PAM,RECSIZE=200,       -
/              RECFORM=F,SPACE=198
```

6.6.4 Regeln für REGIONAL(3)-Organgisation

Eine REGIONAL(3)-Datei muß auf einem Direktzugriffsträger liegen. Sie erlaubt Sätze fester oder variabler Länge, denen ein Schlüssel vorangestellt ist. Die Verarbeitung ist sequentiell oder direkt möglich. Es wird die Zugriffsmethode PAM verwendet.

Eine REGIONAL(3)-Datei ist in Regionen zu je n PAM-Seiten unterteilt, wobei $1 \leq n \leq 16$ gilt. Die Regiongröße wird durch den BLKSIZE-Parameter im FILE-Kommando festgelegt. Jede Region erhält eine Region-Nummer, die der ersten Region ist 0, die der zweiten Region ist 1 usw. Die erste PAM-Seite der Datei dient der Aufnahme von Verwaltungsinformation, so daß die Region mit der Region-Nummer r bei der PAM-Seite r mal n + 2 anfängt.

Jede Region enthält einen oder mehrere Sätze, abhängig von der aktuellen Länge der Sätze. Vor jedem Satz wird ein Schlüssel aufgezeichnet; seine Länge ist bei RECSIZE nicht zu berücksichtigen. Der vor jedem Satz stehende Schlüssel wird als regionspezifischer Satzschlüssel bezeichnet.
Die für REGIONAL(3)-Dateien zulässigen Ein-/Ausgabeanweisungen sind in Bild 6-15 zusammengestellt.

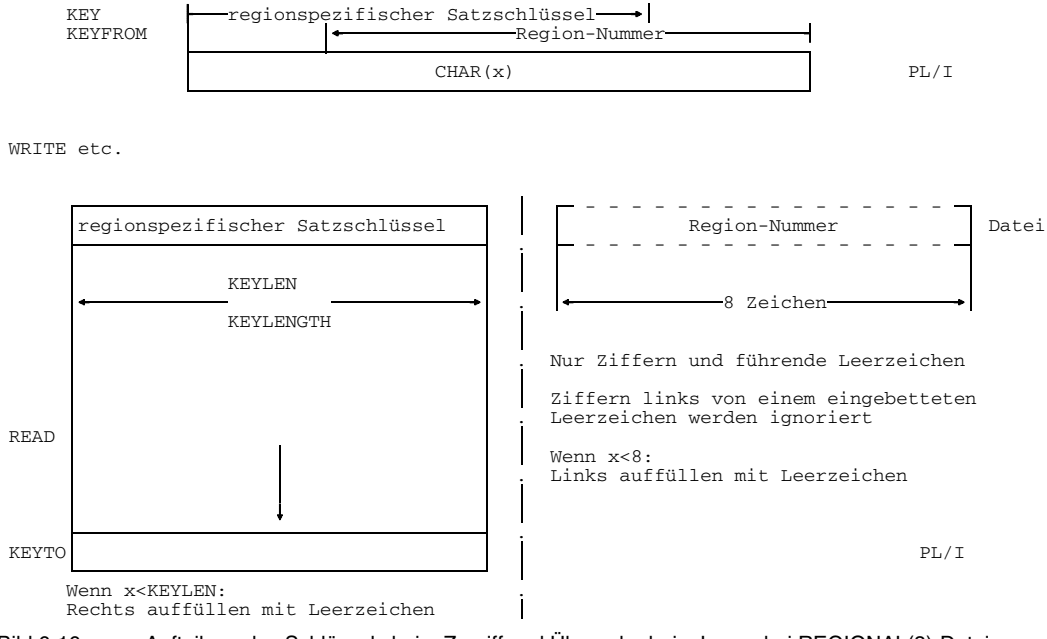


Bild 6-16 Aufteilung des Schlüssels beim Zugriff und Übergabe beim Lesen bei REGIONAL(3)-Dateien

6.6.4.1 Schlüsselangabe

Der im Quellprogramm angegebene Schlüssel wird als Quellenschlüssel bezeichnet. Er wird als Zusammensetzung von Region-Nummer und eines innerhalb der Region gültigen und dort aufgezeichneten Schlüssels betrachtet. Die ersten 8 Zeichen von rechts im Quellenschlüssel werden für die Region-Nummer herangezogen; diese maximal 8 Zeichen dürfen nur aus Ziffern (0 bis 9) und führenden Leerzeichen bestehen. Führende Leerzeichen werden als Null interpretiert, das erste eingebettete Leerzeichen beendet wie bei REGIONAL(1) die Region-Nummer. Sind überhaupt keine Ziffernzeichen oder ungültige Zeichen im Quellenschlüssel, so wird KEY-Bedingung gemeldet. Die gewonnene Zahl wird zur Adressierung der angesprochenen Region (Holen oder Zurückschreiben der betroffenen n PAM-Blöcke) herangezogen.

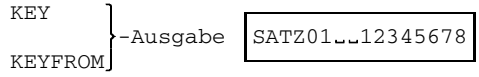
Aus dem Quellenschlüssel werden von links so viele Zeichen für den regionspezifischen Satzschlüssel genommen, wie in der KEYLEN-Angabe des FILE-Kommandos bzw. bei KEYLENGTH im ENVIRONMENT-Attribut angegeben sind. Ist die Länge des Quellenschlüssels kürzer als die KEYLEN-Angabe, so wird der regionspezifische Satzschlüssel rechts mit Leerzeichen aufgefüllt. Er enthält in diesem Fall insbesondere auch alle Ziffern der Region-Nummer. Der regionspezifische Satzschlüssel wird als Zeichenfolge seinem Datensatz vorangestellt (siehe Bild 6-16).

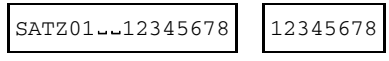
Beispiel 1

```
PL/I-Angabe:        KEY = ('SATZ01␣␣12345678')
FILE-Kommando:      KEYLEN = 6
Interpretationen:   Quellenschlüssel                  ≙SATZ01␣␣12345678
                    regionspezifischer Satzschlüssel  ≙SATZ01
                    Regionnummer                      ≙12345678

  KEY      ⎫
           ⎬-Ausgabe  SATZ01␣␣12345678
  KEYFROM  ⎭

                      SATZ01    12345678

     regionsspezifischer     Region-Nummer
     Schlüssel
     Länge:6                 Länge:8
```

Beispiel 2

```
PL/I-Angabe:        KEY = ('SATZ01␣␣12345678')
FILE-Kommando:      KEYLEN = 16
Interpretationen:   Quellenschlüssel                  ≙SATZ01␣␣12345678
                    regionspezifischer Satzschlüssel  ≙SATZ02␣␣12345678
                    Regionnummer                      ≙12345678
```

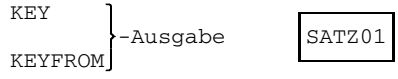


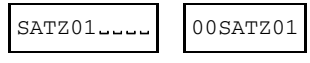



```
   regionsspezifischer     Region-Nummer
   Schlüssel
   Länge:16                Länge:8
```

Beispiel 3

```
PL/I-Angabe:        KEY = ('SATZ01')
FILE-Kommando:      KEYLEN = 10
Interpretationen:   Quellenschlüssel                  ≙SATZ01
                    regionspezifischer Satzschlüssel  ≙SATZ01␣␣␣␣
                    Regionnummer                      ≙KEY-Bedingung
```






```
   regionsspezifischer     Region-Nummer
   Schlüssel
   Länge:10                Länge:8
                           da nicht numerisch:
                              KEY-Bedingung
```

Beispiel 4

```
PL/I-Angabe:        KEY = ('SATZ␣A␣REGION␣B')
FILE-Kommando:      KEYLEN = 6
Interpretationen:   Quellenschlüssel                    ≙SATZ␣A␣REGION␣B
                    regionspezifischer Satzschlüssel    ≙SATZ␣A
                    Regionnummer                        ≙KEY-Bedingung

  KEY       ┐
            }-Ausgabe  ┌──────────────┐
  KEYFROM   ┘          │SATZ␣A␣REGION␣B│
                       └──────────────┘

                       ┌──────┐  ┌─────────┐
                       │SATZ␣A│  │0000000B │
                       └──────┘  └─────────┘
     regionsspezifischer      Region-Nummer
     Schlüssel
     Länge:6                  Länge:8
                              da nicht numerisch:
                                 KEY-Bedingung
```

Anmerkung zu doppelten Schlüsseln

Da bei WRITE keine Prüfung des regionspezifischen Satzschlüssels erfolgt, ob in der angegebenen Region bereits ein Satz mit diesem Schlüssel vorhanden ist, können Sätze mit doppeltem Schlüssel vorkommen.
Im DIRECT-Zugriff wird immer nur der erste dieser Sätze gefunden, weitere werden erst dann wiedergefunden, wenn alle vorhergehenden Sätze mit diesem Schlüssel in der Region gelöscht sind.

Im Normalfall wird dann, wenn in der angegebenen Region kein Platz für einen neuen Datensatz gefunden wird, bzw. wenn ein zu lesender Datensatz nicht in der angegebenen Region vorhanden ist, die Bedingung KEY gesetzt.

Mit Hilfe der Angabe

```
ENVIRONMENT (LIMCT (n))
```

kann bestimmt werden, daß nicht nur die im Schlüssel angegebene Region betrachtet werden soll, sondern noch n weitere. Erst wenn in 1 + n Regionen kein Platz für einen Datensatz gefunden wird (Ausgabe), bzw. der Datensatz nicht gefunden wird (Eingabe), wird die Bedingung KEY gesetzt. Wird hierbei das Ende der Datei erreicht, so wird am Dateianfang fortgefahren.

Voreinstellung ist LIMCT (0).

6.6.4.2 Scheinsätze

Bei OPEN OUTPUT DIRECT wird die gesamte Datei, Region für Region, mit Scheinsätzen vorformatiert. Bei ihnen wird das erste Zeichen jedes Schlüsselfeldes und das 1. Zeichen des Datensatzes mit 'FF'B4 aufgefüllt. Der Anwender muß selbst verantworten, daß seine Schlüssel nicht mit dem Zeichen 'FF'B4 beginnen. Wenn mit OUTPUT SEQUENTIAL eröffnet wird, erfolgt das Eintragen von Scheinsätzen in Verbindung mit dem Schreibvorgang.

6.6.4.3 Eröffnen einer REGIONAL(3)-Datei

Bei einer REGIONAL(3)-organisierten Datei findet beim OPEN-Vorgang folgendes statt:

Bei OPEN OUTPUT DIRECT wird der gesamte, der Datei im SPACE-Parameter des FILE-Kommandos zugewiesene Primärspeicher als in Regionen unterteilt betrachtet und jede Region wird mit Scheinsätzen vorformatiert. Eine Sekundärzuweisung wird nicht berücksichtigt.

Bei OPEN OUTPUT SEQUENTIAL KEYED wird nicht vorformatiert. Jedoch bedingt OPEN OUTPUT SEQUENTIAL KEYED, daß bei den nachfolgenden WRITE-Anweisungen die Sätze in aufsteigender Folge der Region-Nummer angeliefert werden müssen. Die regionspezifischen Satzschlüssel werden nicht auf Einhaltung einer Sequenz geprüft, d.h. daß die Sätze innerhalb einer Region nicht nach den regionspezifischen Satzschlüsseln sortiert sind. Regionen, deren Nummern übersprungen werden, werden mit Scheinsätzen maximaler Länge vorformatiert, ebenso freibleibender Platz in einer Region.
Für das Verlängern einer bestehenden REGIONAL(3)-Datei gilt sinngemäß das gleiche wie für REGIONAL(1)-Dateien (siehe Abschnitt 6.6.3.3).

6.6.4.4 Schließen einer REGIONAL(3)-Datei

Das Schließen einer Datei erfolgt entweder explizit durch Angabe der CLOSE-Anweisung oder implizit bei Beendigung des Programms.

Bei vorangegengenen OPEN OUTPUT SEQUENTIAL werden alle Regionen mit Scheinsätzen aufgefüllt, die noch nicht vorformatiert oder mit relevanten Sätzen gefüllt sind, so daß bei nachfolgender INPUT- oder UPDATE-Verarbeitung keine undefinierten Regionen vorhanden sind.

6.6.4.5 Schreiben in eine REGIONAL(3)-Datei

Aus dem Quellenschlüssel wird die Region-Nummer in beschriebener Weise gewonnen und in dieser Region der erste freie Scheinsatz gesucht. In diesen wird der neue Satz (und sein regionspezifischer Satzschlüssel) eingetragen bzw. bei LOCATE dem Programm der Zeiger auf den Satzanfang geliefert und der regionspezifische Satzschlüssel eingetragen.
Ist der Satz bei SEQUENTIAL-Verarbeitung nicht für die gleiche Region bestimmt wie sein Vorgänger, wird der übersprungene Bereich mit Scheinsätzen aufgefüllt (siehe Abschnitt 6.6.4.3).
Ist die Region-Nummer zu groß oder kein freier Satz in der vorhandenen Region, wird eine KEY-Bedingung gemeldet (siehe Abschnitt 6.6.4.1, LIMCNT).

6.6.4.6 Lesen aus einer REGIONAL(3)-Datei

Es wird unterschieden zwischen DIRECT- und SEQUENTIAL-Verarbeitung. Bei DIRECT werden aus dem Quellenschlüssel die Region-Nummer und der regionspezifische Satzschlüssel gewonnen. Die Region wird in den EA-Puffer eingelesen (oder bei Überschreitung KEY-Bedingung gemeldet) und die Region sequentiell von vorn nach hinten nach dem Satz durchsucht, der als erster den gesuchten regionspezifischen Satzschlüssel hat. Ist ein solcher nicht vorhanden, wird KEY-Bedingung gemeldet (siehe Abschnitt 6.6.4.1, LIMCNT). Andernfalls wird der Satz dem Programm geliefert bzw. bei READ SET der Zeiger auf seinen Anfang gesetzt.
Bei SEQUENTIAL wird der nächste definierte Satz aufgesucht und geliefert bzw. bei READ SET zeigt der Zeiger auf ihn. Bei READ IGNORE werden so viele definierte Sätze übergangen, wie bei IGNORE angegeben. Gegebenenfalls wird ENDFILE-Bedingung gemeldet. Scheinsätze gelten als nicht definierte Sätze und werden nicht geliefert.
Ist KEYTO angegeben, wird nur der regionspezifische Satzschlüssel der dort spezifizierten Variablen zugewiesen (siehe Bild 6-16).

6.6.4.7 Überschreiben einer REGIONAL(3)-Datei

Dem REWRITE muß bei SEQUENTIAL UPDATE ein erfolgreiches READ vorangegangen sein.

Der im Quellenschlüssel angegebene Satz (bei DIRECT) muß vorhanden sein, sonst erfolgt KEY-Bedingung. Die Suche erfolgt wie beim Lesen aus einer REGIONAL(3)-Datei. Der gefundene Satz wird mit dem neuen Inhalt überschrieben. Es wird der erste Satz in der angegebenen Region überschrieben, der den gewünschten Schlüssel besitzt.

6.6.4.8 Löschen eines Satzes in einer REGIONAL(3)-Datei

Der Satz wird aufgrund des Quellenschlüssels adressiert und in einen Scheinsatz gewandelt. Ist er nicht vorhanden, wird keine KEY-Bedingung gemeldet.

KEY-Bedingung wird ausgelöst, wenn die Region-Nummer zu groß ist.

6.6.4.9 FILE-Kommando für REGIONAL(3)-Dateien

Falls die Datei nicht bereits katalogisiert ist oder das ENVIRONMENT-Attribut entsprechende Angaben enthält, muß das FILE-Kommando mindestens folgende Parameter enthalten:

```
/FILE      Dateiname,    (Name des Datenbestandes)
 LINK    = PLI-Title,
[FCBTYPE = PAM,]         (voreingestellt)
 RECSIZE = r,            (maximale Länge des Datensatzes)
           ⎧F⎫
 RECFORM = ⎨ ⎬,
           ⎩V⎭

           ⎧STD     ⎫      Puffergröße = 1 PAM-Block
 BLKSIZE = ⎨        ⎬,
           ⎩(STD,n) ⎭      Puffergröße in n PAM-Blöcken (max. 16)
 KEYLEN  = k,              (Schlüssellänge in Anzahl Zeichen)
                           (1 bis 255)

 SPACE   = e               (Erstzuweisung bei OUTPUT/
                           Zweitzuweisung bei EXTEND oder INOUT
```

Regeln für das FILE-Kommando:

- Bei der RECSIZE-Angabe darf die Schlüssellänge k nicht mit berücksichtigt werden, wohl aber bei der Berechnung, wieviele Sätze in eine Region aufgenommen werden können.
- Die Größe einer Region wird durch BLKSIZE festgelegt. Die Regiongröße muß so bemessen sein, daß mindestens ein Satz einschließlich Schlüssel aufgenommen werden kann.
- Jede Region kann einen oder mehrere Sätze aufnehmen, abhängig vom Verhältnis BLKSIZE/RECSIZE und von KEYLEN.
- Bei KEYLEN wird die Länge des regionspezifischen Satzschlüssels angegeben; für k ist 1 bis 255 möglich.
- In der SPACE-Angabe muß 1 PAM-Seite für Verwaltungsinformation berücksichtigt werden.

- Die SPACE-Angabe sollte in Einklang mit RECSIZE und KEYLEN stehen. Eine automatische Verlängerung des bei der Erstellung der Datei vorgegebenen Speicherbedarfs ist nicht möglich. Die Angabe eines Inkrements wird daher ignoriert.

  Es sollte angegeben werden: e = a * n + 1 wobei
  a = Anzahl gewünschter Regionen
  n = Angabe bei BLKSIZE (1 bis 16 möglich)

Beispiel

Datei für 1000 Regionen, je Region mit max. 17 Sätze, je Satz max. 100 Bytes, Schlüssellänge 18 Bytes

```
/FILE ARTIKEL,LINK=NR,FCBTYPE=PAM,RECSIZE=100,                -
              RECFORM=F,BLKSIZE=STD,KEYLEN=18,SPACE=1002
```

6.7 Magnetband

Die Verarbeitung von Magnetbändern ist ausführlich im Manual BS2000 DVS, Bandverarbeitung [17] beschrieben. In der Sprache PL/I gibt es mit Ausnahme vom Attribut BACKWARDS keine speziellen Sprachelemente für Banddateien; für alle Dateien gibt es einheitliche Sprachmittel. Auf Grund der Eigenschaften des Speichermediums Magnetband gibt es einige Einschränkungen und einige zusätzliche Steuerungsmöglichkeiten, die in den nachfolgenden Abschnitten erläutert sind.

6.7.1 Zugriffsmethode für Banddateien

Auf Banddateien darf nur mit SAM zugegriffen werden.

```
/FILE...FCBTYPE=SAM (Ausnahme siehe 6.7.4)
```

6.7.2 Datei-Attribute

Bei der Vereinbarung einer Banddatei bzw. beim Eröffnen einer Banddatei sind folgende Attribut-Sätze möglich:

```
⎧ STREAM[PRINT]          ⎧INPUT ⎫          ⎫
⎪                        ⎨      ⎬          ⎪
⎩                        ⎩OUTPUT⎭          ⎭
⎧ RECORD SEQUENTIAL      ⎧INPUT[BACKWARDS]⎫⎫
⎪                        ⎨                ⎬⎪
⎩                        ⎩OUTPUT          ⎭⎭
```

Die Organisationsform der Datei muß CONSECUTIVE sein. Sie kann bei der Vereinbarung mit

```
[ENVIRONMENT (CONSECUTIVE)]
```

angegeben werden. Diese Organisationsform ist voreingestellt und kann daher entfallen.

6.7.3 Zugriffe

Ein Magnetband ist ein Speichermedium, das nur sequentiell beschrieben oder gelesen werden kann. Dies entspricht der Organisationsform CONSECUTIVE. Von den hierfür allgemein zugelassenen Zugriffsarten ist UPDATE nicht möglich. Im Bild 6-17 ist eine Übersicht über die erlaubten Attributsätze und die erlaubten Anweisungen für das Lesen und Schreiben von Datensätzen.

Organisation	Attributsatz			Anweisung
CONSECUTIVE SAM F V U	STREAM	PRINT	INPUT	GET [SKIP]
			OUTPUT	PUT [SKIP]
				PUT [[PAGE,LINE] / [SKIP]]
	RECORD	SEQUENTIAL	OUTPUT	WRITE FROM LOCATE
			INPUT [BACK-WARDS]	READ { INTO / SET / IGNORE }
			UPDATE	nicht möglich bei Magnetband

Bild 6-17 Attributsatz und Zugriffe beim Magnetband

Einzelheiten zum Zugriff sind den Abschnitten 6.5 und 6.6.1 zu entnehmen. Darüber hinaus ist es möglich, beim Lesen mit dem letzten Datensatz der Datei zu beginnen und mit dem ersten zu enden. Dies ist nur möglich mit dem Attributsatz

```
RECORD SEQUENTIAL INPUT BACKWARDS
```

6.7.4 Schließen der Datei

Beim Schließen einer Magnetbanddatei kann man zusätzlich bestimmen, welche Position das Magnetband einnehmen soll. Folgende Angaben sind möglich:

```
                        [              [UNLOAD]  ]
CLOSE FILE (Banddatei)[ENVIRONMENT ( {       } )]
                        [              [LEAVE ]  ]
```

Die Angaben bei ENVIRONMENT haben folgende Bedeutung:

keine Angabe
: Das Magnetband wird bis zum Anfang des Bandes zurückgespult.

UNLOAD
: Das Magnetband wird zurückgespult und entladen.
Das Gerät gehört weiterhin dem Prozeß.

LEAVE
: Wurde das Magnetband vorwärts gelesen, so wird auf das Dateiende positioniert oder auf das Magnetbandende, wenn die Datei auf einer weiteren Spule fortgesetzt ist.

  Wurde das Magnetband rückwärts gelesen (BACKWARDS), so wird auf den Dateianfang positioniert oder auf den Magnetbandanfang, wenn die Datei auf einer anderen Spule beginnt.

  Hinweis

  LEAVE führt nur dann zum gewünschten Verhalten, wenn die Magnetbanddatei mit CLOSE LEAVE geschlossen und anschließend zum Lesen mit OPEN SINOUT eröffnet wird, wobei

  ```
  /FILE...,FCBTYPE=BTAM
  ```

  vereinbart sein muß.

7 Prozedur-Schnittstelle

In der Sprache PL/I ist festgelegt, daß Teile eines vollständigen Programms getrennt übersetzt werden können. Es sind dies die externen Prozeduren. Eine externe Prozedur kann von einer anderen externen Prozedur aus aufgerufen werden. Die dafür in PL/I vorhandenen Möglichkeiten sind ausführlich im Kapitel 6 der Sprachbeschreibung [1] beschrieben.

Im Abschnitt 7.1 wird erläutert, wie der Aufruf von Prozeduren in der maschinennahen Form verwirklicht ist. Dabei werden alle die Fakten beschrieben, die zum Verständnis erforderlich sind. Ferner wird auf folgende Steuerungsmöglichkeiten näher eingegangen:

- Art der Parameter-Übergabe (VARIABLE, ASSEMBLER)
- Aufruf von Bibliotheks-Moduln (LIBRARY)
- Bindersteuerung (WXTRN)

Die Kenntnis der internen Aufruftechnik ist von Bedeutung, wenn ein Speicherauszug interpretiert werden muß oder wenn man beabsichtigt, eine Assembler-Prozedur zu erstellen, die wie eine PL/I-Prozedur aufgebaut ist.

In einem weiteren Abschnitt ist dargestellt unter welchen Bedingungen Verbindungen zwischen PL/I-Prozeduren und Prozeduren, die in Assembler und anderen Sprachen geschrieben sind, hergestellt werden können.

7.1 PL/I-Schnittstellen

In den nachfolgenden Abschnitten wird beschrieben, wie eine externe PL/I-Prozedur aufgerufen wird, wie ggf. Parameter übergeben werden, wie ggf. ein Ergebnis zurückgegeben wird und wie zum rufenden Block zurückgekehrt wird. Zum Verständnis dieses Abschnitts sind Kenntnisse über die interne Darstellung von Datenelementen und Kenntnisse der Assembler-Sprache erforderlich.

Die hier beschriebenen Einzelheiten beziehen sich auf die Version 3.xx des Übersetzers. Änderungen sind vorbehalten.

7.1.1 Aufruf-Schnittstelle

Wird eine Prozedur aufgerufen, so werden einmal auf der rufenden Seite bestimmte Operationen ausgelöst. Dies ist im Abschnitt 7.1.1.1. beschrieben.

Auf der gerufenen Seite sind weitere Operationen erforderlich. Die hier erforderliche Anfangsbehandlung ist im Abschnitt 7.1.1.2 ausführlich erläutert.

7.1.1.1 Rufende Prozedur

Für den Aufruf eines Blockes wird im Register 15 die Adresse abgelegt, bei der das zu rufende Programm gestartet werden soll. Es wird mit dem Befehl

```
BALR 14,15
```

die auf den Befehl folgende Adresse im Register 14 als Rückkehradresse hinterlegt und dann auf die im Register 15 angegebene Zieladresse gesprungen.

Im Register 13 wird die Anfangsadresse des eigenen Aktivierungssatzes übergeben.

Im Register 12 steht ein Verweis auf den Pseudoregister-Vektor. Dieses Register darf nicht verändert werden (siehe Bild 7-1).

Register	Inhalt
1-4	Angaben zu den ersten 4 Parametern, soweit vorhanden Weitere Parameter-Angaben stehen im Aktivierungssatz
12	interne Information; darf nicht verändert werden
13	Anfangsadresse des Aktivierungssatzes
14	Rücksprungadresse
15	Hinsprungadresse

Bild 7-1 Registerinhalte bei Aufruf einer externen PL/I-Prozedur oder eines Assembler-Moduls nach PLI1-Aufrufkonventionen

7.1.1.2 Anfangsbehandlung

In der gerufenen Prozedur werden in der Anfangsbehandlung einige Operationen durchgeführt, die im wesentlichen darin bestehen, daß die aktuellen Werte der Register im Aktivierungssatz der rufenden Prozedur sichergestellt werden und daß der eigene Aktivierungssatz aufgebaut wird.

Im einzelnen werden folgende Funktionen ausgeführt (siehe dazu auch das allgemeine Beispiel in Bild 7-2).

```
Programm        CSECT                          Eingang in Assembler-Programm
                USING Programm, 15             Für Konstanten
                B   Start                      Konstanten überspringen
                DS  OF
                DC  FL1'n'                     }
                                               } eigener Name der Länge n
                DC  CL7'Programm'              }

Länge           DC  F'96'                      }
                                               } für eigenen Aktiv.Satz
Kopf            DC  X'00010000'                }
Startadresse    DC  A (Start)                  für USING
                USING Aktivierungssatz, 13     für fremden Aktiv.Satz
Start           STM 14,12, Register            Register 14 bis 12 sichern
                L   10,Startadresse            }
                                               } für Befehle
                USING Start, 10                }
                L    9,Kopf                    Zwischenspeicher
                L    8,Tempende                Ende Vorgänger-Aktiv.Satz
                L    7,Länge                   Länge eigener Aktiv.Satz
                ALR  7,8                       Ende eigener Aktiv.Satz

                LH   6,Tempsegmentnr           }  nur wenn: *COMOPT OPTIONS=XS
                ICM  6,1,Tempsegmentnr         }  Segmentnummern angleichen
                CLM  6,2,4072(12)              }  aktuelle Segmentnummer
                BNE  Speicherfordern           }  gleich

                CL   7,444(,12)                prüfen ob Platz im Stack
                BNH  Weiter
Speicherfor-    EQU  *
dern
                L    15,432(,12)               }  für Stack
                BALR 14,15                     }  Speicher nachfordern
Weiter          EQU  *
                DROP 13
                USING Aktivierungssatz, 8      für eigenen Aktiv-Satz
                STH   6,Tempsegmentnr          nur wenn: *COMOPT OPTIONS
                                               = XS
                LR    6,7                      }
                STM   6,7 Tempende             }
                ST    9, Kopfwort              }  Satz
                ST   13, Vorgänger             }
                LA   13, Aktivierungssatz      Anfangsadresse eigener
                DROP  8                        Aktiv.Satz für
                USING Aktivierungssatz, 13     eigenen Aktiv.Satz
```

Bild 7-2 Ablaufschema einer Anfangsbehandlung

- Als erster Befehl steht ein Sprungbefehl, der die am Anfang stehenden Konstanten überspringt. Er muß auf Wortgrenze beginnen.

- Nach dem obigen Sprungbefehl steht in zwei Ganzwörtern der Name der Assembler-Prozedur, unter dem sie aufgerufen wird. Er hat folgendes Format:
  - 1 Byte Anzahl der signifikanten Zeichen des Namens
  - 7 Bytes Name, rechts ggf. mit Leerzeichen aufgefüllt

```
rel. Adresse | Aktivierungssatz   DSECT
        +0   | Kopfwort          DS   1F
        +4   | Vorgänger         DS   1F
        +8   |                   DS   1F
        +12  | Register          DS  14F
        +68  |                   DS   2F
        +76  | Tempende          DS   1F
        +80  | Permende          DS   1F
        +84  |                   DS   2F
        +92  | Tempsegmentnr     DS   CL1
        +93  | Permsegmentnr     DS   CL1
        +94  |                   DS   CL2
        +96  |
```

Bild 7-3 Struktur des Aktivierungssatzes für das Programm im Bild 7-2

- Die Inhalte der Register 14 bis 15 und 0 bis 12 werden im Aktivierungssatz der rufenden Prozedur sichergestellt.
  Die Anfangsadresse des Aktivierungssatzes steht im Register 13.
- In der gerufenen Prozedur wird möglichst im Anschluß an den Aktivierungssatz der rufenden Prozedur ein eigener Aktivierungssatz eingerichtet. Dazu wird geprüft, ob noch genügend Platz für den Aktivierungssatz vorhanden ist. Ist dies nicht der Fall, so wird Speicher nachgefordert. Hierzu wird das Register 12 benötigt. Zur Übergabe und Rückgabe der Werte werden die Register 7 und 8 verwendet. Alle Register außer 7,8,14 und 15 werden nicht verändert.
- Im eigenen Aktivierungssatz werden folgende Werte eingetragen (siehe dazu die Beschreibung des Aktivierungssatzes im Kapitel 10):
  - Block-Typ (+1)
  - Markierung (+2)
  - Vorgänger-Aktivierungssatz (+4)
    Es wird der Wert des Registers 13 eingetragen.
  - permanentes Ende (+80)
    Es wird die erste Adresse hinter dem eigenen Aktivierungssatz abgespeichert. Dieser Wert ergibt sich im Normalfall aus dem gleichen Feld im Aktivierungssatz des Vorgängers zuzüglich der Länge des eigenen Aktivierungssatzes.
    Ist jedoch hinter dem Aktivierungssatz des Vorgängers nicht genügend Platz vorhanden, so wird bei der Nachforderung von Speicherplatz dieses Feld entsprechend geändert.

  - temporäres Ende (+76)
    Es wird der gleiche Wert abgespeichert wie beim permanenten Ende (+80)

- In das Register 13 wird die Anfangsadresse des eigenen Aktivierungssatzes geladen. Dieses Register wird erst wieder vor der Rückkehr zur rufenden Prozedur zurückgesetzt.
- Der Inhalt des Registers 12 darf nicht verändert werden.

Die Aufteilung des Aktivierungssatzes ist im Kapitel 10 ausführlich beschrieben. Im Bild 7-3 ist die Struktur des Aktivierungssatzes gezeigt, wie er für das Assembler-Programm in Bild 7-2 verwendet werden kann.

7.1.2 Parameter-Übergabe

Werden beim Aufruf einer Prozedur Parameter angegeben, so müssen deren Werte der aufgerufenen Prozedur zur Verfügung gestellt werden. Im Normalfall geschieht dies dadurch, daß man pro Parameter einen Zeiger übergibt, der auf den Anfang des Speicherplatzes weist, in dem der Wert steht.

Moduln, die mit "*COMOPT OPTIONS = XS" übersetzt wurden, weisen folgende Besonderheiten auf. Bei Parametern vom Typ BIT UNAL weist der übergebene Zeiger auf einen Absolut-Bitzeiger, der auf den Parameterwert zeigt.

In einigen Fällen wird außer dem Parameter-Wert auch die zugehörige Datenbeschreibung benötigt.

Für die Parameter-Übergabe gibt es mehrere Möglichkeiten. Sie können vom Benutzer für externe Prozeduren durch die OPTIONS-Angabe gesteuert werden:

```
DCL Prozedur ENTRY usw. OPTIONS(Angabe)
```

Die externe Prozedur muß die Parameter entsprechend verarbeiten können. Hierauf wird in den folgenden Abschnitten eingegangen.

Im Abschnitt 7.1.4 ist beschrieben, daß unter bestimmten Bedingungen auch das Ergebnis eines Funktions-Aufrufes wie ein zusätzlicher Parameter behandelt wird.

Wird ein Aktual-Parameter nicht "durch Bezug", sondern "durch Zuweisung" übergeben (siehe Abschnitt 6.2.4.2 in der Sprachbeschreibung [1]), so weist ein übergebener Zeiger auf den Speicherplatz der eingerichteten Hilfs-Variable und nicht auf den Speicherplatz des Aktual-Parameters. Entsprechend wird auch die Datenbeschreibung für die Hilfs-Variable übergeben.

7.1.2.1 Normalfall (PL/I)

Wird bei der Vereinbarung einer externen Prozedur durch eine DECLARE-Anweisung keine OPTIONS-Angabe zur Steuerung der Parameter-Übergabe gemacht, so gelten die nachstehenden allgemeinen Regeln für alle Prozeduren. Diese Regeln ergeben die günstigsten Bedingungen in Bezug auf Zeit- und Speicherplatz-Optimierung.

Für jeden Parameter wird ein Zeiger übergeben, der auf den Anfang des Speicherplatzes weist, in dem der Wert des Parameters gespeichert ist.

Im Anschluß an die Parameterwerte werden unter Umständen die zu den Parametern gehörenden Datenbeschreibungen übergeben.

Für die Übergabe der Datenbeschreibung zu den Parametern gelten folgende Regeln:

- Ein Zeiger auf die Datenbeschreibung wird übergeben,
  - wenn im Formal-Parameter eine *-Angabe enthalten ist (AREA, BIT, CHARACTER),
  - und wenn das Attribut DIMENSION oder STRUCTURE vorhanden ist.
- Eine Längenangabe wird übergeben,
  - wenn im Formal-Parameter eine *-Angabe vorhanden ist (AREA, BIT, CHARACTER),
  - und wenn der Parameter skalar ist.
- Ein undefinierter Wert wird übergeben,
  - wenn das Attribut AREA, BIT, CHARACTER oder DIMENSION vorhanden ist
  - aber im Formal-Parameter keine *-Angabe vorhanden ist.
- In allen anderen Fällen wird keine Datenbeschreibung übergeben.

Wenn der rufenden Seite die Datenbeschreibung des Formalparameters nicht bekannt ist, so wird angenommen, daß er eine *-Angabe enthält.

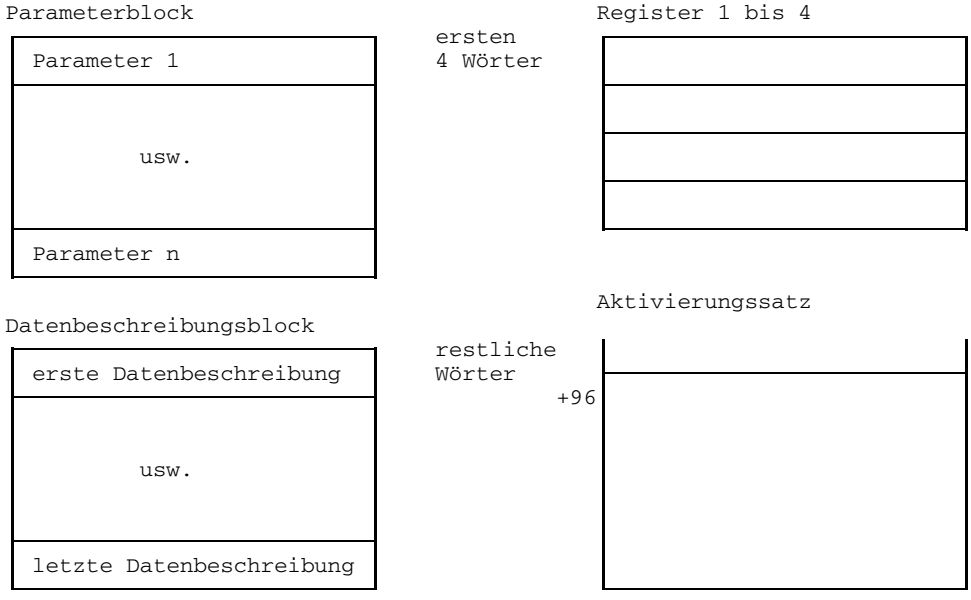


Bild 7-4 Übergabe der Parameter (Normalfall)

Listet man die Zeiger auf die Parameterwerte in der Reihenfolge auf, wie sie im Quellprogramm niedergeschrieben sind und danach die für die Datenbeschreibung übergebenen Werte in der gleichen Reihenfolge, so werden

- die ersten vier Wörter in den Registern 1 bis 4 und
- die weiteren Wörter im Aktivierungsblock der rufenden Prozedur ab Relativ-Adresse +96

an die gerufene Prozedur übergeben. Die Anfangsadresse des Aktivierungssatzes wird im Register 13 übergeben.

Die Datenbeschreibung ist ausführlich im Kapitel 10 beschrieben.

In einigen Fällen kann für die Rückgabe eines Ergebnisses beim Funktions-Aufruf ein weiterer Parameter übergeben werden. Dies ist im Abschnitt 7.1.4 beschrieben.

Durch die Angabe des Attributes PARAMETER (INPUT) (siehe Sprachbeschreibung [1] Abschnitt 4.2, PARAMETER) kann angegeben werden, daß mit diesem Parameter nur Werte an die gerufene Prozedur übergeben werden sollen, nicht jedoch zurück an die rufende Prozedur (Übergabe durch Zuweisung; passing by value; siehe Sprachbeschreibung [1] Abschnitt 6.2.4.2). Sind für den Parameter folgende Bedingungen erfüllt, so wird statt des Zeigers auf den Parameterwert, der Wert selbst übergeben:

- skalarer Parameter
- keine *-Angabe
- nicht größer als ein Ganzwort.

Belegt der Wert weniger als ein Ganzwort, so werden Größen mit den Attributen REAL BINARY rechtsbündig, alle anderen linksbündig, im Ganzwort abgelegt.

Bei Parametern von externen Prozeduren muß die Angabe sowohl auf der rufenden als auch auf der gerufenen Seite für den gleichen Parameter gemacht werden:

Beispiel

rufende Seite

```
DCL  Eingang    ENTRY (CHAR(1),         PARAMETER (INPUT),
                       FIXED BINARY(15) PARAMETER (INPUT),
                       CHAR(30);
```

gerufene Seite

```
Eingang:  PROCEDURE (A,B,C);

DCL     A       CHAR(1)           PARAMETER (INPUT),

        B       FIXED BINARY(15)  PARAMETER (INPUT),

        C       CHAR(30)          PARAMETER;
```

Bei Parametern für interne Prozeduren genügt die Angabe, wie sie für die gerufene Seite angegeben ist.

Die Angaben PARAMETER (UPDATE) und PARAMETER (OUTPUT) bewirken keine Veränderung der Parameter-Übergabe.

7.1.2.2 Allgemeine Assembler-Konvention (VARIABLE)

In Sonderfällen kann es erforderlich sein, bei der Parameter-Übergabe für alle Parameter die Datenbeschreibungen mit zu übergeben. Dies kann erreicht werden durch die Angabe OPTIONS (VARIABLE) bei der Vereinbarung des Eingangs. Es ist also zu vereinbaren:

```
DCL Eingang ENTRY usw.
            OPTIONS (VARIABLE);
```

Diese Form der Parameter-Übergabe ist für den Aufruf von Assembler-Prozeduren gedacht. Sie kann für PL/I-Prozeduren nicht verwendet werden.

Ist OPTIONS (VARIABLE) angegeben, so werden die Parameter nicht direkt übergeben, sondern sie werden mit den Datenbeschreibungen in einem Versorgungsblock übergeben. Der Versorgungsblock hat den folgenden Aufbau (siehe dazu auch Bild 7-5):

- Das erste Wort ist ein Kopfwort und enthält in den rechten 16 Bits die Anzahl der übergebenen Parameter. Die linken 16 Bits sind undefiniert.
- Es folgt für jeden Aktual-Parameter in der Reihenfolge der Niederschrift des Quellprogramms ein Zeiger, der auf die Speicheradresse weist, bei der der Parameterwert gespeichert ist.
- Falls es sich um eine Prozedur handelt, die durch einen Funktions-Aufruf angesprochen wird, erfolgt die Rückgabe des Ergebnisses über einen weiteren Parameter, wie dies im Abschnitt 7.1.4 beschrieben ist.
- Für jeden Parameter und ggf. für das Ergebnis folgt ein Zeiger, der auf die Datenbeschreibung weist, die zu den Parametern und ggf. zum Ergebnis gehört.
- Für Datenbeschreibungen, die das Attribut PICTURE besitzen, wird nicht der Datentyp X'3A' bzw. X'3B' übergeben, sondern der Datentyp X'00' bzw. X'01', der eine erweiterte Datenbeschreibung besitzt, d.h. sie enthält zusätzliche eine Maskenbeschreibung. Die Datenbeschreibung und die Maskenbeschreibung sind ausführlich im Abschnitt 10.6 beschrieben.

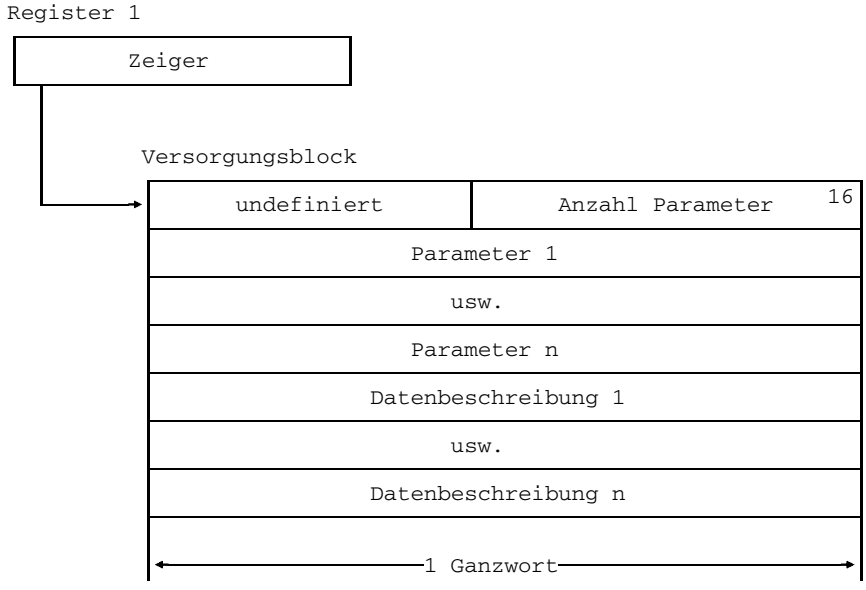


Bild 7-5 Parameter-Übergabe im Versorgungsblock bei OPTIONS (VARIABLE)

7.1.2.3 Standard-Assembler-Konventionen (ASSEMBLER)

Wird OPTIONS (ASSEMBLER) angegeben, so werden die Parameter nach Standard-Assembler Konventionen dem Industriestandard entsprechend übergeben. Die Vereinbarung lautet:

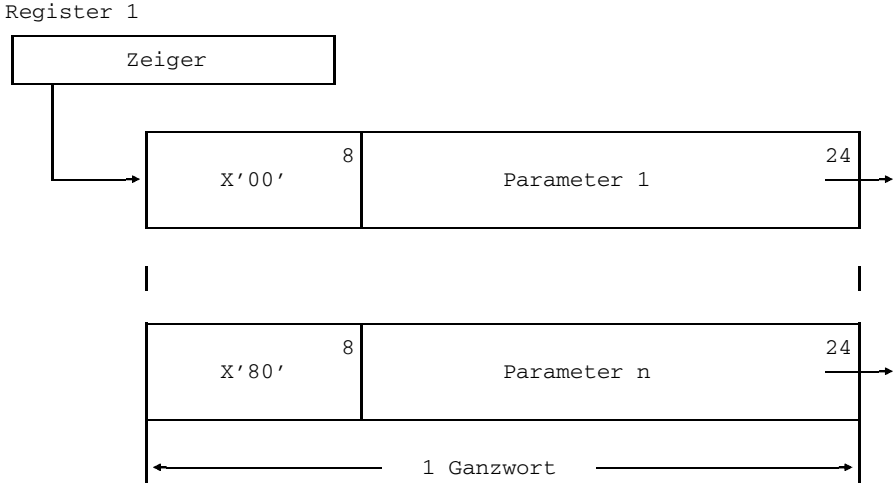
```
DCL Eingang ENTRY usw. OPTIONS ( { ASSEMBLER } [INTER]);
                               { ASM       }
```

Die vorstehend hinter OPTIONS angegebenen Möglichkeiten haben alle die gleiche Bedeutung.

Für die Übergabe wird ein Versorgungsblock aufgebaut und seine Adresse im Register 1 übergeben. Der Versorgungsblock hat folgenden Aufbau (siehe dazu Bild 7-6):

- Für jeden Aktual-Parameter wird in der Reihenfolge der Niederschrift im Quellprogramm ein Zeiger angegeben, der auf die Speicheradresse weist, bei der der Parameterwert gespeichert ist. Die links stehenden ersten 8 Bits haben alle den Wert ’0’B.
- Für den letzten Parameter hat das erste linksstehende Bit nicht den Wert ’0’B, sondern den Wert ’1’B.
- Werden keine Parameter übergeben, so haben im Register 1 alle Bits den Wert ’0’B.

Datenbeschreibungen werden nicht übergeben.




```
VARYING:  Zeiger weist auf Längenangabe
BIT:      Zeiger weist auf Zeichen mit erstem Bit
```

Bild 7-6 Parameter-Übergabe im Versorgungsblock bei OPTIONS (ASSEMBLER[INTER])

Es gilt folgende Besonderheit:

- Falls ein Formal-Parameter das Attribut VARYING besitzt, weist der Zeiger nicht auf den Anfang der Daten sondern auf die vor den Daten stehende Längenangabe (siehe Intern-Darstellung im Kapitel 10).
- Als Zeiger wird stets eine Byte-Adresse übergeben. Dies muß beachtet werden, wenn der Parameter das Attribut BIT hat und das erste Bit nicht bei einer Byte-grenze beginnt.

7.1.3 Problembearbeitung

Nach der Anfangsbehandlung folgt die eigentliche Problemlösung.
Dabei gilt:

- Im Register 13 steht stets die Anfangsadresse des eigenen Aktivierungssatzes.
- Der Inhalt des Registers 12 wird nie verändert.

Register	Inhalt
12	interne Information; wird nie verändert
13	Anfangsadresse des eigenen Aktivierungssatzes Umsetzen in der Aufruf-bzw. Rückkehrphase

Bild 7-7 Register die bei PL/I-Blöcken einen bestimmten Inhalt haben

7.1.4 Ergebnis-Rückgabe

Wird eine Prozedur durch einen Funktions-Aufruf aufgerufen, so wird bei der Rückkehr ein Ergebnis an die rufende Prozedur zurückgegeben. Die Rückgabe geschieht auf folgende Art:

- Ein realer skalarer Gleitpunktwert in den Gleitpunktregistern F0 bis F3
- Ein max. 1 Ganzwort langer Wert im Register 1.
- Alle anderen Werte werden über einen zusätzlichen Parameter zurückgegeben, als Sonderfall wird dabei ein Rückgabewert mit *-Angabe behandelt.

Siehe dazu die folgenden Abschnitte.

7.1.4.1 Rückgabe im Register 1

Der Wert des Ergebnisses wird im Register 1 zurückgegeben, wenn eine skalare Größe vorliegt und einer der folgenden Datentypen gegeben ist:

- BIT(n) NONVARYING mit n ≤ 32; der Wert steht rechtsbündig im Register 1
- REAL FIXED BINARY (immer wie PREC(31,x))
- POINTER
- OFFSET

7.1.4.2 Rückgabe in Gleitpunktregistern

Der Wert des Ergebnisses wird in Gleitpunktregistern zurückgegeben, wenn die Größe skalar ist und folgende Datentypen vorliegen.

- REAL FLOAT BINARY PRECISION (g)
- REAL FLOAT DECIMAL PRECISION (g)

Die Registerbelegung ergibt sich wie folgt:

Register F0 linke Hälfte	für DECIMAL g = 1 bis 6 für BINARY g = 1 bis 21
Register F0	für DECIMAL g = 7 bis 16 für BINARY g = 22 bis 53
Register F0 und F2	für DECIMAL g = 17 bis 33 für BINARY g = 54 bis 109

7.1.4.3 Rückgabe über Parameter

Trifft keiner der Fälle zu, die in den Abschnitten 7.1.4.1 und 7.1.4.2 beschrieben sind, so wird der Wert über einen zusätzlichen Parameter, den Ergebnis-Parameter, zurückgegeben.

Der Ergebnis-Parameter ist der letzte Parameter in der Parameter-Liste. Beim Aufruf ist der Zeiger definiert, der Wert des Parameters jedoch undefiniert. Das aufgerufene Programm legt das Ergebnis in den Speicherbereich ab, der durch den Zeiger des Ergebnis-Parameters bestimmt ist. Der Aufbau des Parameters und die Zuordnung der Datenbeschreibung ist die gleiche wie bei anderen Parametern und kann im Abschnitt 7.1.2 nachgelesen werden.

Sind für die Prozedur keine Parameter vereinbart, so ist der Ergebnis-Parameter der einzige Parameter.

Handelt es sich bei dem Ergebnis um einen Wert mit *-Angabe, so erfolgt die Übergabe nach den im Abschnitt 7.1.4.4 beschriebenen Regeln.

7.1.4.4 Rückgabe bei *-Angabe

Ist bei der RETURNS-Angabe der Anweisung PROCEDURE oder ENTRY eine *-Angabe vorhanden, so wird für das Ergebnis wie vorstehend an die Liste der Parameter ein weiterer Parameter hinzugefügt, der Ergebnis-Parameter.

Zu diesem Ergebnis-Parameter gibt es stets eine Datenbeschreibung.

Der Ergebnis-Parameter wird bei der Parameter-Übergabe wie ein normaler Parameter behandelt. Sein Wert ist beim Aufruf undefiniert und wird erst in der gerufenen Prozedur bestimmt.

Der Ergebnis-Parameter besteht aus einem Zeiger (indirekter Zeiger), der auf einen Zeiger im Aktivierungssatz der rufenden Prozedur weist (direkter Zeiger). Der Wert des direkten Zeigers ist beim Aufruf undefiniert.

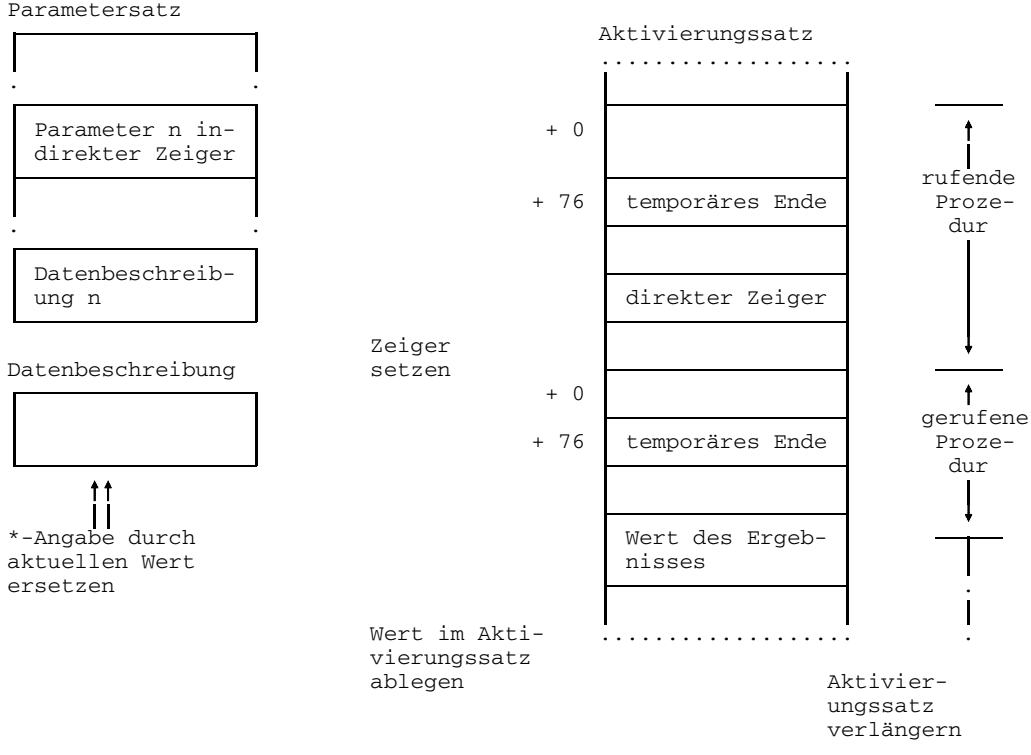


Bild 7-8 Rückgabe eines Wertes bei *-Angabe

Von der gerufenen Prozedur werden folgende Funktionen durchgeführt (siehe auch das Bild 7-8):

- In der Datenbeschreibung wird die *-Angabe durch den aktuellen Wert ersetzt (Punkt 1 im Bild).
- Der aktuelle Aktivierungssatz wird temporär um den für das Ergebnis benötigten Speicherplatz erweitert und das Ergebnis dort abgelegt (Punkt 2 im Bild).
- Ein Zeiger auf den Speicherplatz des Ergebnisses wird als direkter Zeiger in die Speicherzelle abgelegt, auf die der als Parameter übergebene indirekte Zeiger weist. Die Speicherzelle des direkten Zeigers liegt im Aktivierungssatz der rufenden Prozedur (Punkt 3 im Bild).

  Abweichend vom allgemeinen Fall weist der Zeiger bei Variablen mit dem Attribut VARYING nicht auf die Längenangabe, sondern auf das erste Byte hinter der Längenangabe.
- Bei der Rückkehr aus der gerufenen Prozedur wird deren (verlängerter) Aktivierungssatz nicht - wie in den anderen Fällen - freigegeben, sondern dem Aktivierungssatz der rufenden Prozedur zugeschlagen. Er wird von der rufenden Prozedur dann freigegeben, wenn das Ergebnis des Funktionsaufrufes verarbeitet ist (Punkt 4 im Bild).

Die Zusammenhänge sind im Bild 7-8 skizziert. Der Aufbau des Aktivierungssatzes ist in Kapitel 10 beschrieben und die Parameter-Übergabe im Abschnitt 7.1.2.

7.1.5 Beendigung einer Prozedur

Es gibt zwei Möglichkeiten, die gerufene Prozedur wieder zu verlassen.

- Rückkehr
  Es wird zu der Stelle zurückgekehrt, von der aus die Prozedur aufgerufen wurde (RETURN,END).
- Rücksprung
  Es wird über eine GOTO-Anweisung an eine beliebige Stelle einer dynamisch vorhergehenden Prozedur gesprungen.

7.1.5.1 Rückkehr

Wird von der gerufenen Prozedur an die Stelle zurückgekehrt, von der aus sie aufgerufen wurde, so werden folgende Bedingungen hergestellt:

- In das Register 13 wird die Anfangsadresse des Aktivierungssatzes vom Vorgänger geladen. Sie steht im eigenen Aktivierungssatz (+4).
- Die Register 2 bis 11 werden auf den Wert zurückgesetzt, den sie beim Aufruf hatten.
- Wird im Register 1 kein Ergebnis zurückgegeben, so wird es auf den Wert gesetzt, den es beim Aufruf hatte.

Die Rückkehradresse war beim Aufruf im Register 14 hinterlegt und wurde in der Anfangsbearbeitung im Aktivierungssatz des Vorgängers (+12) gesichert. Wird das Register 14 ebenfalls auf seinen alten Wert zurückgesetzt, so kann mit dem Befehl BR 14 zurückgekehrt werden.
Falls der Aufruf ein Funktions-Aufruf war, wird ein Ergebnis zurückgegeben. Dies ist im Abschnitt 7.1.4 beschrieben.

Register	Inhalt
0	beliebig
1	Funktionswert (Ergebnis) bzw. unverändert
2-11	unverändert
12	unverändert; darf auch nicht vorübergehend geändert werden
13	Unverändert (Zeiger auf den Aktivierungssatz der rufenden Prozedur)
14-15	beliebig (Register 14 enthält in der Regel die Rücksprungadresse)

Bild 7-9 Registerinhalte für die Rückkehr zu einer PL/I-Prozedur

```
Rückkehr    EQU *
            USING Aktivierungssatz,13     für eigenen Aktiv.Satz
            L   13,Vorgänger              Adresse Vorgänger-Aktiv.Satz
            LM  1,11,Register+12 1)       Register 1 bis 11 restaurieren 1)
            LM  2,11,Register+16 1)       Register 2 bis 11 restaurieren 1)
            L   14,Register+0             Register 14 restaurieren
            BR  14                        Rücksprung
```

1) Register 1 bei Funktions-Aufruf nicht restaurieren.

Bild 7-10 Beispiel für die Rückkehr zur aufrufenden Stelle

7.1.5.2 Rücksprung

Eine Prozedur kann über eine GOTO-Anweisung verlassen werden. Das Sprungziel liegt in diesem Fall in einem dynamisch vorangehenden Block.

Für den Rücksprung werden in der gerufenen Prozedur folgende Funktionen ausgeführt:

- Das dritte Ganzwort des Markenwertes enthält die Adresse des Aktivierungssatzes, der zum Sprungziel gehört. Diese wird ins Register 13 gebracht. Damit ist die Umgebung der Ziel-Prozedur eingestellt.
- Die Register 3 bis 11 werden auf die alten Werte zurückgesetzt, die im Aktivierungssatz des Sprungziels (relativ 32) sichergestellt waren.
- In das Register 10 wird die Basisadresse für das Sprungziel gebracht. Sie steht im zweiten Ganzwort des Markenwertes.
- Es wird auf die Adresse des Sprungziels gesprungen, die im ersten Ganzwort des Markenwertes steht.

Siehe dazu auch die Bilder 7-11 und 7-12.

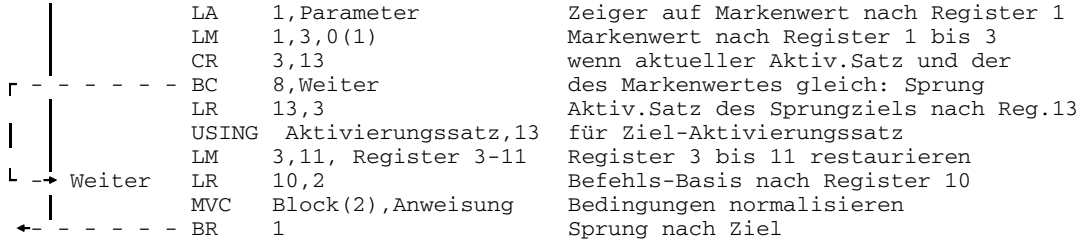


Bild 7-11 Ablaufschema für einen Sprung auf eine Marke, die als Parameter übergeben wird.

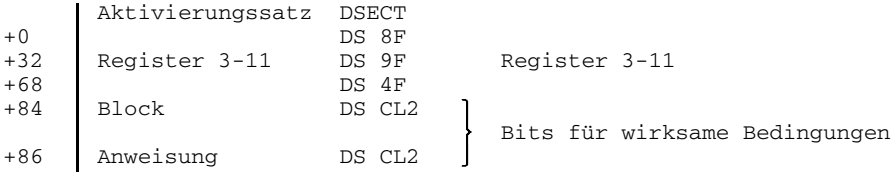


Bild 7-12 Struktur des Aktivierungssatzes zu Bild 7-11

7.1.6 Bibliotheks-Prozedur (LIBRARY)

Dem Benutzer stehen bestimmte Dienstleistungen zur Verfügung in Form von vorgefertigten und bereits übersetzten externen Prozeduren. Der Leistungsumfang dieser Prozeduren ist im Kapitel 11 getrennt für jede Prozedur beschrieben. Die allgemeinen Zusammenhänge sind in entsprechenden Abschnitten zu finden.

Soll eine dieser Prozeduren aufgerufen werden, so muß in der rufenden Prozedur der Eingang durch die Anweisung DECLARE vereinbart werden, wie dies bei der Beschreibung der Prozedur angegeben ist.

Für einen Teil der Prozeduren muß bei der Vereinbarung des Eingangs die Angabe

```
OPTIONS (LIBRARY) bzw.
OPTIONS (LIB)
```

gemacht werden. Sie bewirkt, daß dem Eingangsnamen, der gemäß den allgemeinen Regeln für externe Namen aus maximal 7 Zeichen bestehen kann, soviel Nummernzeichen (#) angehängt werden, bis der Name 8 Zeichen umfaßt. Unter dem so entstandenen Namen ist die Prozedur im PLI1-Laufzeitsystem enthalten.

Dies bedeutet für den Benutzer, daß er den gleichen Namen auch für eigene externe Prozeduren verwenden kann, so lange er nicht im Gegensatz zu den PL/I-Regeln für die Verwendung von Namen steht (siehe Beispiel in in Bild 7-13).

Die Angabe OPTIONS (LIBRARY) darf nicht verwendet werden, wenn der Eingang eine vom Benutzer erstellte und übersetzte externe Prozedur bezeichnet.

```
DCL ERROUT ENTRY OPTIONS (LIBRARY);
CALL ERROUT; ..........................  ruft ERROUT## in der
BEGIN;                                   Bibliothek
   DCL ERROUT ENTRY;
   CALL ERROUT; .......................  ruft eigene externe Prozedur
                                         ERROUT
```

Bild 7-13 Beispiel für den gleichen Eingangsnamen mit und ohne OPTIONS (LIBRARY)

7.1.7 Binden (WXTRN)

Wird in einer externen Prozedur der Eingang einer anderen externen Prozedur vereinbart, so wird diese beim Binden automatisch mit angebunden, ohne daß es hierzu einer expliziten Angabe bedarf.

Soll das automatische Anbinden unterbleiben, so kann das durch die Angabe OPTIONS (WXTRN) bei der Vereinbarung des Eingangs erreicht werden:

```
DCL    Eingang ENTRY usw. OPTIONS (WXTRN);
```

Ein solcher Eingang wird nur dann in das Programm eingebunden, wenn dies explizit durch die INCLUDE-Angabe beim Binden gefordert wird oder wenn er auf Grund einer anderen zum Programm gehörigen externen Prozedur automatisch angebunden wird (siehe Abschnitt 3.3.4).

7.2 Assembler-Prozeduren

In bestimmten Fällen kann es wünschenswert oder erforderlich sein, statt einer externen PL/I-Prozedur eine Prozedur zu verwenden, die in der Assembler-Sprache geschrieben ist. Gründe hierfür können sein:

- Die Leistung eines bestehenden Assembler-Programms soll in einem PL/I-Programm verwendet werden.
- Eine nur über die Assembler-Sprache erreichbare Leistung soll in einem PL/I-Programm verwendet werden.
- Eine PL/I-Prozedur soll durch eine gleichwertige Assembler-Prozedur mit verbesserter Effizienz ersetzt werden.

Um dies zu verwirklichen stehen drei Möglichkeiten zur Verfügung:

- Ein Assembler-Programm wird so aufgebaut, daß es sich wie eine PL/I-Prozedur verhält.
- Es wird eine nach Standard-Assembler-Konventionen (siehe Abschnitt 7.1.2.3) geschriebene Assembler-Prozedur verwendet.
- Falls Datenbeschreibungen der Parameter gewünscht sind, kann die Allgemeine Assembler-Konvention (siehe Abschnitt 7.1.2.2) angewendet werden.

In Kapitel 13 sind ASSEMBLER-Makros beschrieben, die den Anschluß von ASSEMBLER-Programmen an PL/I-Programme und umgekehrt vereinfachen.

Dieses ist in den folgenden Abschnitten näher erläutert.

7.2.1 Assembler-Prozedur nach PLI1-Konventionen

Die nachstehende Art, Assembler-Prozeduren in PL/I-Programmen zu verwenden, ist besonders dann geeignet, wenn Assembler-Prozeduren neu geschrieben werden und speziell für die Zusammenarbeit mit PL/I gedacht sind. Sie können dann optimal angepaßt werden.

Im Abschnitt 7.1 ist ausführlich beschrieben, wie der Aufruf von PL/I-Prozeduren auf der Assembler-Ebene durchgeführt wird. Eine Assembler-Prozedur, die von einem PL/I-Programm aufgerufen wird, muß sich für die Anfangsbearbeitung nach dem Aufruf, für die Übernahme der Parameter, für die Rückgabe eines Ergebnisses und für die Rückkehr genau so verhalten, wie eine PL/I-Prozedur.

Soll von einer Assembler-Prozedur, die von einem PL/I-Programm aufgerufen wurde, wiederum eine PL/I-Prozedur aufgerufen werden, so sind auch noch der Aufruf, die Übergabe von Parametern und ggf. die Übernahme von Ergebnissen so durchzuführen, wie dies bei PL/I-Prozeduren der Fall ist.

Wird der Lauf eines PL/I-Programmes beendet, so werden noch gewisse Abschlußarbeiten durchgeführt. Dazu gehört zum Beispiel, daß der Inhalt von Ausgabe-Puffern ausgegeben wird, daß offene Dateien abgeschlossen werden, usw. Dies wird als Abschlußbehandlung bezeichnet. Sie ist vergleichbar mit der in der PL/I-Sprache gegebenen Möglichkeit "ON FINISH ON-Einheit;".

Wird die Assembler-Prozedur nach den Konventionen von PL/I aufgebaut, so gilt auch für sie die Abschlußbehandlung von PL/I.

Tritt ein Fehler in der Assembler-Prozedur auf, so tritt die entsprechende Fehlerbehandlung von PL/I in Kraft und es wird ggf. die entsprechende wirksame ON-Einheit aufgerufen.

Um bei einem Fehler in der Assembler-Prozedur sicherzustellen, daß die Fehlerbehandlung, die im PL/I-Laufzeitsystem zur Verfügung steht, auch in diesem Fall zur Verfügung steht, ist es vorteilhaft, den Eingang des Assemblerprogramms in der rufenden PL/I-Prozedur mit

```
DCL    Eingang usw. OPTIONS (PLI1)
```

zu vereinbaren. Es wird dann bei jedem Aufruf das für die Fehlerbehandlung wichtige Register R13 sichergestellt und bei der Rückkehr wird die Sicherstellung wieder gelöscht.

7.2.2 Assembler-Prozeduren nach Standard-Assembler-Konventionen (ASSEMBLER)

Sollen Assembler-Programme, die nach Standard-Assembler-Konventionen aufgebaut sind, von PL/I-Prozeduren aufgerufen werden, so muß bei der Vereinbarung im PL/I-Programm für den Eingang der Assembler-Prozedur die Angabe

```
OPTIONS ( ⎧ASSEMBLER⎫ [INTER])
         ⎨         ⎬
         ⎩ASM      ⎭
```

gemacht werden. Die Übergabe der Parameter erfolgt dann in der im Abschnitt 7.1.2.3 beschriebenen Form. Die Angabe INTER hat keine zusätzliche Bedeutung.

Dies setzt jedoch voraus, daß im Assembler-Unterprogramm einige Einschränkungen eingehalten werden:

- keine eigene Unterbrechungsbehandlung
- Register 12 wird nicht verändert
- kein Aufruf einer PL/I-Prozedur
- kein Auftreten von Bedingungen
- das Assembler-Unterprogramm kann nicht durch einen Funktions-Aufruf angesprochen werden.

7.2.3 Assembler-Prozeduren nach allgemeiner Assembler-Konvention (VARIABLE)

Es gelten sinngemäß alle Punkte von Abschnitt 7.2.2.
Die entsprechende Vereinbarung ist

```
OPTIONS (VARIABLE)
```

Die Parameterübergabe ist in Abschnitt 7.1.2.2 beschrieben.
Die Rückgabe eines Funktionswertes ist im Gegensatz zu OPTIONS (ASSEMBLER) zulässig, wird aber wie ein zusätzlicher Parameter gehandhabt.
Besondere Leistung dieser Art von Parameterübergabe ist, daß zu jedem Parameter auch der entsprechende PL/I-Deskriptor erzeugt bzw. ausgewertet wird und daß wegen der gesonderten Parameter-Anzahl-Bestimmung auch Parameter vom Typ BIT UNALIGNED verwendet werden dürfen.

7.2.4 Aufruf von PL/I-Prozeduren aus Assembler-Programmen

Sollen PL/I-Prozeduren von Assembler-Programmen aufgerufen werden, so muß der Eingang in eine PL/I-Prozedur folgendermaßen deklariert werden:

```
Eingang: { PROCEDURE } OPTIONS(ASSEMBLER)
         { ENTRY     }
```

Folgendes ist bei der Deklaration zu beachten:

- Die Angaben ASSEMBLER und MAIN können nicht zusammen verwendet werden.
- Die Angabe RETURNS ist nicht zulässig.
- Es ist sichergestellt, daß während des Ablaufes der PL/I-Prozedur die PL/I-Programmunterbrechungsbehandlung aktiviert ist.
- Für den Aufruf einer PL/I-Prozedur sind im rufenden Assembler-Programm folgende Aktionen vorzunehmen:
  Im Register 1 wird die Adresse der Parameterliste erwartet. Die Parameterliste ist nach Standard-Assembler-Konventionen (siehe Abschnitt 7.1.2.3) anzulegen. Im Register 13 ist die Adresse des Sicherungsbereiches einzutragen. Der Sicherungsbereich muß auf Wortgrenze beginnen und eine Länge von 18 Worten haben.
  In Register 15 muß die Adresse der aufzurufenden PL/I-Prozedur stehen. Der Aufruf muß mit dem Befehl BALR R14, R15 erfolgen.
  Der Rücksprung aus der PL/I-Prozedur erfolgt über Register 14.

7.3 FORTRAN- und COBOL-Prozeduren

7.3.1 Allgemeines

Die PLI1-Sprachverknüpfung erlaubt es, zur Laufzeit mit FORTRAN- und COBOL-Prozeduren zu kommunizieren. Die fremdsprachlichen Prozeduren müssen von den Übersetzern

FORTRAN-Übersetzer FOR1 ab V1.40
COBOL-Übersetzer COB1 ab V1.21

generiert worden sein. Die Kommunikation zwischen PLI1 und der fremdsprachlichen Prozedur geschieht

- bei FORTRAN mittels Parametern und EXTERNAL-Daten
- bei COBOL nur mittels Parametern

Eine PLI1-Prozedur ruft eine COBOL-Prozedur durch eine CALL-Anweisung, eine FORTRAN-Prozedur durch eine CALL-Anweisung oder als Funktions-Prozedur auf. Parameter werden beim Aufruf mitgegeben, Funktionswerte durch Funktionen zurückgeliefert.

Ein COMMON-Block in FORTRAN und eine PLI1-Variable mit dem Attribut STATIC EXTERNAL werden im statischen Speicher abgelegt. Haben beide den gleichen Namen, so liegen sie übereinander und wirken wie zwei identische STATIC EXTERNAL-Variable in PLI1: Zuweisungen eines Wertes an eine Variable bedeutet, daß gleichzeitig auch der anderen der gleiche Wert zugewiesen wird.

Es gibt keine vergleichbare Möglichkeit in COBOL; nur Parameter können in PLI1 und COBOL den gleichen Speicherplatz belegen.

Die gesamte Sprachverknüpfung wird über OPTIONS-Angaben vom PLI1-Übersetzer aus ermöglicht. Existierende FORTRAN- und COBOL-Prozeduren müssen im allgemeinen nicht geändert oder neu übersetzt werden, neue Prozeduren können erstellt werden ohne einen evtl. Gebrauch im Sprachenverbund zu berücksichtigen.

Eine Prozedur bedeutet bei COBOL ein Unterprogramm und bei FORTRAN ein Unterprogramm oder eine Funktion. Die Konventionen der jeweiligen Sprache bleiben durch den Anschluß unberührt.

Die Bearbeitung einer Datei muß vor Aufruf einer fremdsprachlichen Prozedur mit einer CLOSE-Anweisung abgeschlossen werden, wenn in der fremdsprachlichen Prozedur die gleiche Datei bearbeitet werden soll.

REGIONAL-Dateien sind nur in PLI1-Prozeduren bearbeitbar.

Bei CONSECUTIVE-Zugriff auf ISAM-Dateien bearbeitet PLI1 nur 4 Byte lange BINARY-Keys oder 8 Byte lange CHARACTER-Keys.

7.3.2 Übereinstimmung der Daten

Eine detaillierte Kenntnis der Sprachen COBOL bzw. FORTRAN ist für die Sprachverknüpfung nicht notwendig, wohl aber Wissen über die Darstellung der Daten. Die interne Darstellung des Aktual-Parameters in PL/I und die interne Darstellung des zugehörigen Formal-Parameters in FORTRAN bzw. COBOL müssen miteinander verträglich sein. Wie bei der Parameter-Übergabe zwischen externen PL/I-Prozeduren, so wird auch hier vom Übersetzer die Verträglichkeit nicht geprüft. Dies liegt in der Verantwortung des Benutzers.

Falls es für den Datentyp eines Formal-Parameters in PL/I keinen entsprechenden Datentyp in FORTRAN oder COBOL gibt, wird vom Übersetzer eine Warnung ausgegeben. Resultate zur Laufzeit sind dann undefiniert und das Programm wird möglicherweise abnormal beendet. PL/I hat mehr Datentypen als FORTRAN oder COBOL und nicht alle haben einen entsprechenden Parallel-Typ in einer der beiden Sprachen.

Außer den Datentypen ist auch die Ausrichtung der Daten im Speicher entscheidend. Hier gilt folgende Entsprechung:

	ausgerichtet	nicht ausgerichtet
PL/I	ALIGNED	UNALIGNED
FORTRAN	Normalfall	-
COBOL	SYNCHRONIZED	nicht SYNCHRONIZED

Die Ausrichtung eines Aktual-Parameters bestimmt sich, wie auch der Datentyp, aus der Parameterbeschreibung oder aus dem Parameter selbst. Nur ALIGNED-Parameter dürfen an SYNCHRONIZED-Parameter bei COBOL oder an FORTRAN-Parameter übergeben werden. Sowohl ALIGNED- als auch UNALIGNED-Parameter dürfen an COBOL-Parameter übergeben werden, die nicht mit SYNCHRONIZED vereinbart sind. Für die Übereinstimmung von Aktual- und Formal-Parameter ist der Benutzer selbst verantwortlich.

FORTRAN

Die Datentypen, die zwischen PL/I und FORTRAN verträglich sind, sind in Bild 7-14 aufgelistet.

FORTRAN	PL/I nur ALIGNED-Daten	
INTEGER * 2	FIXED BINARY (p,q)	mit 0<p≤15, q=0
INTEGER * 4	FIXED BINARY (p,q)	mit 15<p≤31, q=0
REAL * 4	FLOAT BINARY (p) FLOAT DECIMAL(p)	mit p ≤ 21 mit p ≤ 6
REAL * 8	FLOAT BINARY (p) FLOAT DECIMAL(p)	mit 21 < p ≤ 53 mit 6 < p ≤ 16
REAL * 16	FLOAT BINARY (p) FLOAT DECIMAL(p)	mit 53 < p ≤ 109 mit 16 < p ≤ 33
COMPLEX * 8	COMPLEX FLOAT BIN(p) COMPLEX FLOAT DEC(p)	mit p ≤ 21 mit p ≤ 6
COMPLEX * 16	COMPLEX FLOAT BIN(p) COMPLEX FLOAT DEC(p)	mit 21 < p ≤ 53 mit 6 < p ≤ 16
COMPLEX * 32	COMPLEX FLOAT BIN(p) COMPLEX FLOAT DEC(p)	mit 53 < p ≤ 109 mit 16 < p ≤ 33
LOGICAL * 1	BIT(8)	
LOGICAL * 4	BIT(32)	
CHARACTER n feste Länge	CHARACTER (n) NONVARYING	
CHARACTER n variable Länge	CHARACTER (n) VARYING	

Bild 7-14 Verträgliche Datentypen zwischen PL/I und FORTRAN

PL/I-Matrizen (DIMENSION), soweit sie einen verbundenen Speicher besitzen, können an FORTRAN-Felder übergeben werden. Sind die Elemente Zeichenfolgen, so müssen sie das Attribut NONVARYING haben. Felder können nicht als Funktionswert zurückgegeben werden.

Mehrdimensionale Felder in FORTRAN werden im Gegensatz zu den Matrizen in PLI1, die seitenweise abgelegt werden, spaltenweise abgelegt.
Durch eine Index-Überlagerung einer Matrix kann man nun erreichen, daß in PL/I z.B. mit B (i,j,k) auf dasselbe Element zugegriffen wird wie in FORTRAN mit A (i,j,k) (siehe Abschnitt 4.2 unter DEFINED-Attribut: iSUB-Defining in der PL/I-Beschreibung).

Beispiel

1. PL/I-Routine ruft FORTRAN-Unterprogramm mit Parameterübergabe auf:

```
PLIROUT:        PROC;

       DCL    FORUP ENTRY (DIM(4,5,6)...)OPTIONS(FORTRAN);
       DCL A  DIMENSION (4,5,6)...;
       DCL B  DIMENSION (6,5,4)DEF A (3SUB,2SUB,1SUB)...;

       /* IN PLI WIRD MIT B GEARBEITET */
       CALL FORUP (A);

END;
```

2. PL/I-Unterprogramm wird von einer FORTRAN-Routine mit Parameterübergabe aufgerufen:

```
PLIUP:    PROC (A)   OPTIONS (FORTRAN);
            .

       DCL A  DIMENSION (4,5,6) PARAMETER...;
       DCL B  DIM (6,5,4) DEF A (3SUB,2SUB,1SUB)...;

       /* IN PLI WIRD MIT B GEARBEITET */
          .
END;
```

In beiden Fällen werden die Felder in FORTRAN mit

```
      DIMENSION A (6,5,4)
```

deklariert.

Achtung

DEFINED-Variable mit iSUB-Angabe können *nicht* bei GET DATA und PUT DATA verwendet werden.

Vereinbarungen mit * sind an zulässigen Stellen auch bei der Parameter-Übergabe an FORTRAN erlaubt. Es wird dann dafür gesorgt, daß aktuelle Werte in die zu übergebenden Datenbeschreibungen eingesetzt werden.

COBOL

Die Datentypen, die zwischen PL/I und COBOL verträglich sind, sind im Bild 7-15 aufgeführt.

Bei der Ausrichtung sind folgende Angaben verträglich:

```
ALIGNED   und SYNCHRONIZED
UNALIGNED und keine SYNCHRONIZED-Angabe
```

Werden Matrizen übergeben, so wird eine Warnung ausgegeben, da sie nur beschränkt mit einer über die OCCURS-Klausel definierten COBOL-Tabelle verträglich sind.

COB1	PLI1
COMPUTATIONAL 1-4 Ziffernstellen	FIXED BINARY (p,0) mit 0<p≤15
5-9 Ziffernstellen	FIXED BINARY (p,0) mit 15<p≤31
COMPUTATIONAL-1	FLOAT DECIMAL (p) mit p≤ 6 FLOAT BINARY (p) mit p≤21
COMPUTATIONAL-2	FLOAT DECIMAL (p) mit 6<p≤16 FLOAT BINARY (p) mit 21<p≤53
COMPUTATIONAL-3 (n)	FIXED DECIMAL (n)
DISPLAY	CHARACTER (n)

SYNCHRONYZED	ALIGNED
ohne diese Angabe	UNALIGNED

Bild 7-15 Verträgliche Datentypen zwischen PL/I und COBOL

Strukturen können übergeben werden (sie entsprechen auf COBOL-Seite durch Stufennummern gegliederten logischen Datensätzen), wenn sichergestellt ist, daß die interne Datenstruktur in PL/I und COBOL übereinstimmt. Dies ist im allgemeinen bis zur Stufennummer 2 der Fall. Einzelheiten zur internen Darstellung für PL/I sind im Kapitel 10 zu finden.
Bei ALIGNED-Strukturen mit ausgerichteten Datentypen (BINARY, FLOAT, usw.) bestehen Unverträglichkeiten.

7.3.3 Vereinbarung, Aufruf

Der Eingang in eine FORTRAN- bzw. COBOL-Prozedur muß durch

```
DCL Eingang ENTRY usw.

          ⎡FORTRAN [INTER]⎤
OPTIONS (⎨                ⎬)
          ⎣COBOL          ⎦
```

vereinbart werden.

Die Angaben haben folgende Bedeutung (siehe auch Abschnitt 7.3.4):

FORTRAN — Es wird eine FORTRAN-Prozedur aufgerufen. Programmunterbrechungen, die während des Laufs der FORTRAN-Prozedur auftreten, werden von PL/I behandelt.

FORTRAN INTER — Es wird eine FORTRAN-Prozedur aufgerufen. Programmunterbrechungen, die während des Laufs der FORTRAN-Prozedur auftreten, werden von FORTRAN bearbeitet.

COBOL — Es wird eine COBOL-Prozedur aufgerufen.

Folgendes ist bei der Vereinbarung des Eingangs zu beachten:

- Bei COBOL gibt es keine Funktions-Prozeduren; folglich kann auch das Attribut RETURNS nicht angegeben werden.
- Die Angabe OPTIONS ist auch bei Eingangs-Variablen erlaubt.
- Ein Eingang, wie oben definiert, kann auch bei GENERIC verwendet werden.
- Die in Programmen entsprechend dem Industriestandard bei OPTIONS erlaubten Angaben NOMAPIN, NOMAPOUT, ARGi werden ignoriert.
  Es wird wie bei NOMAP verfahren.

7.3.4 Unterbrechungsbehandlung

In FORTRAN werden nur einige der zur Laufzeit möglichen Unterbrechungen behandelt. Bei der Sprachverknüpfung PL/I-FORTRAN ist es möglich, nicht behandelte Unterbrechungen an PL/I weiterzuleiten. Hierzu muß der Benutzer die Angabe OPTIONS (FORTRAN INTER) machen. Dadurch werden die von der aufgerufenen FORTRAN-Prozedur nicht bearbeiteten Unterbrechungen in PL/I durch eine ON-Einheit bzw. durch die ON-Einheit des PL/I-Systems behandelt. Falls INTER nicht angegeben wurde, wird eine auftretende Unterbrechung nur durch PL/I behandelt.

Es gilt zur Zeit das Folgende:

- Die INTER-Angabe ist vorhanden:
  Vor jedem Aufruf der FORTRAN-Prozedur wird die Unterbrechungsbehandlung von FOR1 eingestellt. Die PLI1-Fehlerbehandlung ist abgeschaltet. Es ist jedoch sichergestellt, daß bei Fehlerabbruch in einer FORTRAN-Prozedur auch die PLI1-Endebehandlung noch durchgeführt wird.

  Bei der Rückkehr aus der FORTRAN-Prozedur wird die PLI1-Unterbrechungsbehandlung wieder eingeschaltet.

  Der nicht unerhebliche Rechenzeitaufwand für das Hin- und Herschalten der Unterbrechungsbehandlung bei jedem Aufruf (einige tausend Maschinenbefehle) ist nur bei verhältnismäßig großen FORTRAN-Prozeduren und ggf. in der Testphase gegen die verbesserte Fehlerdiagnose aufzuwiegen.

- Die INTER-Angabe ist nicht vorhanden:
  Alle Unterbrechungen werden von der PL/I-Unterbrechungsbehandlung bearbeitet. Es findet kein Hin- und Herschalten zwischen verschiedenen Unterbrechungsbehandlungen statt, d.h.

  – alle vom FORTRAN-System erkannten Fehler werden von diesem behandelt und anschließend in die PLI1-Endebehandlung verzweigt.

  – die von der Hardware bzw. vom Betriebssystem gemeldeten Unterbrechungen (STXIT-Behandlungen) werden direkt von der PLI1-Fehlerbehandlung bearbeitet, was unter Umständen zur Zuordnung der Unterbrechung zur auslösenden Anweisung die Auswertung der Objektliste und des Speicherauszugs erfordert.
    Abhilfe: Wiederholung des Programmlaufs mit neu übersetztem Programm mit der Angabe OPTIONS (FORTRAN INTER) und/oder Einsatz der Testhilfe IDA in der FORTRAN-Prozedur.

Bei COBOL ist die Angabe INTER wirkungslos, da es keine COBOL-Unterbrechungsbehandlung über STXIT-Behandlung gibt. Es gilt hier dasselbe wie bei OPTIONS (FORTRAN), d.h. ohne die Angabe INTER.

Ausnahme

Nach der Behandlung der vom COBOL-System erkannten Fehler wird nicht zu PL/I zurückverzweigt. Abhilfe siehe Abschnitt 7.3.5.

7.3.5 Programmende

- FORTRAN
  - Verhalten am Ende des PL/I-Programms
    Der Benutzer, der ein PL/I-Programm mit Sprachtransfer schreibt, braucht keine besonderen Vorkehrungen bei Abschluß des PL/I-Programms zu treffen, um die Aufrufe nach FORTRAN abzuschließen.
  - Verhalten bei Ende der FORTRAN-Prozedur
    Es ist sichergestellt, daß bei Beendigung der FORTRAN-Prozedur über STOP oder Fehlerabbruch die PLI1-Endebehandlung angesteuert wird. Über die Bedingung FINISH kann der PL/I-Programmlauf sogar unter Umständen fortgesetzt werden.
- COBOL

  Es ist nicht möglich, beim COB1-Laufzeitsystem eine Endebehandlung anzumelden. Das bedeutet, wenn die COBOL-Prozedur abbricht, z.B. auf Grund der Anweisung STOP RUN oder auf Grund eines Programmfehlers, der vom COBOL-System erkannt wird, wird das PL/I-Programm nicht ordnungsgemäß abgeschlossen. Unter Umständen gehen Daten für Dateien verloren.

7.3.6 Aufruf von PL/I-Prozeduren aus FORTRAN- und COBOL-Programmen

Sollen PL/I-Prozeduren von FORTRAN- oder COBOL-Programmen aufgerufen werden, so ist der Eingang in eine PL/I-Prozedur folgendermaßen zu deklarieren:

```
Eingang:{PROCEDURE} OPTIONS({FORTRAN})
        {       }          {      }
        {ENTRY  }          {COBOL }
```

Die Angaben haben folgende Bedeutung:

FORTRAN Die PL/I-Prozedur wird von einem FORTRAN-Programm aufgerufen.

COBOL Die PL/I-Prozedur wird von einem COBOL-Programm aufgerufen.

Folgendes ist bei der Deklaration zu beachten:

- Die Angaben COBOL und FORTRAN schließen sich gegenseitig aus. Sie können bei einem Eingang auch nicht zusammen mit der Angabe MAIN verwendet werden.
- Die Angabe RETURNS kann nicht zusammen mit der Angabe COBOL aufgeführt werden, da es keine COBOL-Funktionsprozeduren gibt.
- Für Parameter mit den Attributen AREA, BIT, CHAR oder DIMENSION sind als Längen- oder Dimensionsangaben nur ganzzahlige Konstanten erlaubt.
- Die bei dem Industriestandard entsprechend Angaben NOMAP (p), NOMAPIN und NOMAPOUT werden mit einer Warnung versehen und ignoriert. Für alle Parameter gilt als Voreinstellung NOMAP.
- Es ist sichergestellt, daß während des Ablaufes der PL/I-Prozedur die PL/I-Programmunterbrechungsbehandlung aktiviert ist.
- Im übrigen gilt für die Unterbrechungsbehandlung das unter Abschnitt 7.3.4 erläuterte.
- Bei Programmende wird verfahren wie bei Abschnitt 7.3.5 beschrieben.

7.4 ILCS-Prozeduren

7.4.1 Allgemeines

Die PLI1-Sprachverknüpfung erlaubt zur Laufzeit auch die Programm-Kommunikation nach den Konventionen der Inter Language Communication Services (ILCS). Diese Konventionen ermöglichen die beliebige Verknüpfung von in unterschiedlichen Programmiersprachen geschriebenen Programmen. Weitere Einzelheiten können der Freigabemitteilung von ILCS entnommen werden. Die ILCS-Fähigkeit der verschiedenen Sprachübersetzer ist in deren Dokumentation zu erfahren.

Die Kommunikation zwischen ILCS-fähigen Routinen geschieht mittels Argumenten 'per reference', d.h. es werden nur die Verweise auf die Argumente übergeben. Die Routinen können als Prozedur oder als Funktion (soweit dies in der jeweiligen Sprache erlaubt ist) aufgerufen werden.

Die ILCS-Sprachverknüpfung wird auf PLI1-Seite über OPTIONS-Angaben ermöglicht. Die ILCS-Anweisungen in anderen Programmsprachen sind in deren Dokumentation einzusehen.

7.4.2 Anfangsbehandlung

Jedes ILCS-fähige Hauptprogramm ruft zunächst die ILCS-Anfangsbehandlung auf, diese wiederum ruft alle beteiligten sprachspezifischen Anfangsbehandlungen auf, so daß sichergestellt ist, daß alle beteiligten Sprachumgebungen eingerichtet und damit funktionsfähig sind.

7.4.3 Deklaration, Aufruf

Ein ILCS-Eingang in eine PLI1-Prozedur oder eine PLI1-Funktion wird folgendermaßen mit der OPTIONS-Option gekennzeichnet:

```
n :  { PROCEDURE } [(p)] OPTIONS (ILCS) [ RETURNS ( { FIXED } ) ]
     { ENTRY     }                                 { FLOAT }
```

Ein Aufruf einer ILCS-Prozedur oder ILCS-Funktion aus einer PLI1-Prozedur wird folgendermaßen mit der OPTIONS-Option deklariert:

```
DCL n ENTRY [ ( p ) ] OPTIONS (ILCS) [ RETURNS ( { FIXED } ) ]
                                               { FLOAT }
```

Dabei bedeuten: n Name der aufzurufenden Prozedur oder Funktion.
p Liste der Argumente der Aufzurufenden Prozedur oder Funktion.

Der Aufruf einer ILCS-Prozedur oder ILCS-Funktion aus einer PLI1-Prozedur wird wie der Aufruf einer PLI1-Prozedur mit der CALL-Anweisung durchgeführt.

7.4.4 Mapping der Dateien

Die ILCS-Konventionen unterstützen nur die folgenden Parametertypen:

BINARY FIXED (31)
BINARY FLOAT (21)
BINARY FLOAT (53)
DECIMAL FLOAT (6)
DECIMAL FLOAT (16)
CHARACTER (i)

Die Argumente müssen normale Ausrichtung haben (Ganzwort- oder Byte-Grenze, je nach Typ), d.h. auf PLI1-Seite mit dem Attribut ALIGNED versehen sein. Die entsprechenden Deklarationen in den anderen Sprachen sind dort nachzusehen.

Da die ILCS-Sprachverknüpfung keine Deskriptoren für die Argumente kennt ist es nicht möglich, die Argumente in den aufrufenden wie in den gerufenen Routinen abzuprüfen. Es liegt in der Verantwortung des Benutzers, sicherzustellen, daß die Argumente in der rufenden Routine bezüglich Datentyp und Mapping den Parametern in der aufgerufenen Routine entsprechen.

7.4.5 Interrupt-Behandlung

Bei der Sprachverknüpfung von ILCS-Programmen mit PLI1-Programmen wird beim Eintritt in ein PLI1-Programm stets die ILCS-Unterbrechungsbehandlung ausgeschaltet und die PLI1-Unterbrechungsbehandlung eingestellt. Treten Fehler in diesem PLI1-Programm auf, so wird die PLI1-Fehlerbehandlung durchlaufen. Beim Verlassen eines PLI1-Programmes wird die PLI1-Unterbrechungsbehandlung ausgeschaltet und die ILCS-Unterbrechungsbehandlung aktiviert.

7.4.6 Endebehandlung

Führt ein PLI1-Programm, das von einem ILCS-Programm aufgerufen wurde, auf Programmende, so wird die ILCS-Endebehandlung aufgerufen, die die Endebehandlungen aller beteiligten ILCS-Programme und damit auch die PLI1-Endebehandlung aufruft und schließlich das Programm beendet.

8 Optimierungen

Im Abschnitt 8.1 wird ein Überblick gegeben über die verschiedenen Arten der Optimierung, wie sie durch den Übersetzer und seine Bibliotheken gegeben sind.

Im Abschnitt 8.2 werden Hinweise gegeben, wie sich durch Umstellungen im Programm eine günstige Leistung ergeben kann. Die Hinweise sind nach steigendem Aufwand für die Verwirklichung geordnet. Der Abschnitt enthält außerdem Hinweise für die Programmierung im virtuellen Speicher und endet mit einer Betrachtung über die Vorteile der modularen Programmierung.

Im Abschnitt 8.3 wird ausführlich erläutert, wann eine bestimmte Operation durch eingefügten Code (in-line code) oder durch Bibliotheks-Aufrufe verwirklicht wird. Mit Hilfe dieser Information ist es häufig möglich, den Zusatzaufwand für einen Bibliotheks-Aufruf zu vermeiden.

Im Abschnitt 8.4 werden die verschiedenen Optimierungen erläutert, die vom Übersetzer durchgeführt werden, wenn dies durch die Steuerangabe "COMOPT OPTIMIZE=" gefordert wird. Des weiteren sind Hinweise gegeben, was der Benutzer tun kann, um die Vorteile dieser Optimierung voll ausnutzen zu können. Dies dürfte besonders für den wissenschaftlichen Programmierer von Bedeutung sein.

Im Abschnitt 8.5 wird aufgezeigt, wie die vom Übersetzer durchgeführte Optimierung vom Benutzer beeinflußt werden kann.

Im Abschnitt 8.6 werden schließlich einige Programmierhinweise gegeben, die dem Anfänger eine Hilfe sein können.

8.1 Überblick

8.1.1 Übersetzer

Es ist Hauptaufgabe der Optimierung, Objekt-Programme zu erzeugen, die eine möglichst kurze Laufzeit haben und dabei möglichst wenig Speicherplatz belegen. Dies bedeutet, daß für die PL/I-Anweisungen ein möglichst günstiger Code erzeugt wird, aber auch daß ggf. die Reihenfolge der Anweisungen verändert wird, so daß sich bei gleichem Ergebnis die Leistung verbessert.

Folgende Arten von Optimierungen führt der Übersetzer aus:

- Eliminierung gemeinsamer Ausdrücke
- Auslagerung invarianter Ausdrücke aus DO-Schleifen
- Reduktion linearer Ausdrücke in DO-Schleifen
- Eliminierung gemeinsamer Ausdrücke und Auslagerung invarianter Ausdrücke bei reduziblen Funktionen
- Vereinfachung von Ausdrücken
- In-line-code bei Konvertierungen
- In-line-code für Folgenbearbeitung
- In-line-code für die meisten eingebauten Funktionen
- Spezieller Code für Matrix- und Struktur-Zuweisungen
- Register und Adressen-Optimierung. Das schließt ein, daß Werte möglichst lange in Registern gehalten werden und daß auf Grund einer Analyse des Programmablaufs und der Bezugszähler eine günstige Adreßrechnung erzeugt wird.
- Eliminierung gemeinsamer Konstanten und gemeinsamer Datenbeschreibungen um eine günstige Speicherbelegung zu erreichen.
- Spezialcode für den Aufruf interner Prozeduren und für die Rückkehr aus diesen Prozeduren.

Einige dieser Optimierungen werden auch dann durchgeführt, wenn sie nicht auf Grund der Steueranweisung "COMOPT OPTIMIZE=" gefordert werden, andere wiederum werden nur auf Anforderung durchgeführt.

8.1.2 Das Laufzeitsystem

Das PLI1-Laufzeitsystem besteht aus einer Menge von Moduln, die nach logischen Gesichtspunkten gebildet wurden. Diese sind zu zwei vorgebundenen Moduln zusammengefaßt, die von allen PL/I-Programmen gemeinsam benutzt werden können.

Beim Binden besteht die Möglichkeit zu wählen, ob die vorgebundenen Moduln oder die Einzelmodule verwendet werden sollen.

Im ersteren Fall werden die vorgebundenen Module erst zur Laufzeit dynamisch nachgeladen. Sie können vom Systemverwalter als gemeinsam benutzbar erklärt werden. Dadurch werden sie von allen PL/I-Programmen gemeinsam benutzt und damit wird Arbeitsspeicherplatz gespart, wenn mehrere PL/I-Programme gleichzeitig laufen. Es ergibt sich eine Zeitersparnis beim Start des Programms.

Im zweiten Fall entscheidet der Übersetzer, welche Einzelmodule notwendig sind, und wählt somit eine minimale Untermenge von Moduln aus, die mit dem erzeugten Objektmodul zusammengebunden werden.

8.2 Manuelle Optimierungen

Dank der Modularität der PL/I-Bibliotheken und der umfangreichen Optimierung durch den Übersetzer werden viele Anwendungsprogramme eine zufriedenstellende Leistung erbringen und keine speziellen Programmumstellungen erfordern.

Für Programme, die nicht zu den oben erwähnten gehören, werden in diesem Abschnitt Maßnahmen erläutert, die eine Verbesserung der Leistung erbringen können. Die in Abschnitt 8.2.1 aufgeführten Maßnahmen erfordern einen geringeren Aufwand als die in Abschnitt 8.2.2.

Es wird vorausgesetzt, daß System-Betrachtungen gelöst sind (z.B. die Organisation der PL/I-Bibliotheken). Ferner wird vorausgesetzt, daß dem Leser die Steueranweisungen für die Übersetzung (COMOPT) und für den Lauf (RUNOPT) bekannt sind.

8.2.1 Programmumstellung Stufe 1

Alle Testhilfen aus dem Programm entfernen. Daß durch die Verwendung von Testhilfen in einigen Fällen ein zusätzlicher Aufwand entsteht, ist offensichtlich, da sie den Trend haben, große Mengen Ausgaben zu erzeugen. Einige Testhilfen, wie z.B. die Bedingungen SUBSCRIPTRANGE und STRINGRANGE, erzeugen zwar nur dann eine Ausgabe, wenn ein Fehler auftritt, benötigt aber für die Prüfung erheblich mehr Zeit und Speicher.

PUT DATA-Anweisungen sollten ebenfalls aus dem Programm entfernt werden und hier besonders die, für die keine Datenliste angegeben ist. Diese Anweisungen benötigen Information über die Variablen und die Konvertierungsmoduln der Bibliothek. Hierfür ist zusätzlicher Speicherplatz erforderlich.

Die Testhilfe "COMOPT DEBUG = STMT" erhöht nicht die Laufzeit aber den Speicherbedarf um etwa 8 Bytes für jede Anweisung.

8.2.2 Programmumstellung Stufe 2

In PL/I gibt es häufig für ein Problem mehrere Lösungsmöglichkeiten. Eine davon wird im allgemeinen die günstigste sein. Welche dies ist, ist abhängig von der Implementierung der Sprache. Der Unterschied kann einen oder mehrere Maschinenbefehle bedeuten, aber auch mehrere hundert.

Die zweite Stufe der Programmumstellung befaßt sich mit den Sprachelementen, für die vom Übersetzer ein großer Aufwand getrieben werden muß und für die es alternative Sprachelemente gibt. Die Beachtung dieser Fakten kann in häufig durchlaufenen Programmtexten große Vorteile bringen.

Dabei sollte man sich bewußt sein, daß die Verwendung eines bestimmten Sprachelements nicht notwendigerweise deswegen schlecht ist, weil es weniger effizient ist als ein anderes; es muß hier vielmehr auch beachtet werden, was vom Programm im Augenblick gefordert wird und was von ihm in der Zukunft gefordert werden könnte.

Nachstehend sind einige Beispiele für Sprachelemente gegeben, die sehr aufwendig sind:

1. Die Verwendung selbstdefinierender Strukturen (REFER) erlaubt es, Daten sehr kompakt darzustellen. Dabei können jedoch Darstellungen entstehen, die nicht optimal sind. In dem Beispiel

   ```
   DCL 1 Strukturen   BASED (Zeiger),
         2 Länge,
         2 Vorher,
         2 Tabelle    DIM (x REFER (Länge)),
         2 Nachher;
   ```

   wird für den Zugriff auf die Variable "Vorher" ein Befehl benötigt und für den Zugriff auf die Variable "Nachher" etwa 4 Befehle.

   Für eine selbstdefinierende Struktur ist es am günstigsten, wenn alle Glieder mit einer bekannten Länge vor denen stehen, bei denen die Länge eingestellt werden kann. Unter den Gliedern einstellbarer Länge sollten die am weitesten vorne stehen, auf die am häufigsten zugegriffen wird. Beim vorstehenden Beispiel wäre es günstiger die Variable "Nachher" vor die Variable "Tabelle" anzuordnen.

2. Unnötige Strukturierung des Programms sollte vermieden werden. Prozeduren, BEGIN-Blöcke und ON-Einheiten benötigen für Aufruf, Verwaltung und Rückkehr zusätzliche Zeit und zusätzlichen Speicher.

   Dort wo eine DO-Gruppe (DO;...;END;) ausreicht ist sie günstiger als ein BEGIN-Block (BEGIN;...;END;).

   Diese Empfehlung sollte betrachtet werden im Zusammenhang mit Bemerkungen über modulare Programmierung weiter unten in diesem Abschnitt.

3. Die folgenden Maßnahmen betreffen Konvertierungen, die grundsätzlich auf ein Minimum reduziert werden sollten.

   a) im Abschnitt 8.3 ist beschrieben, welche Konvertierungen in-line erfolgen.

   b) Durch zusätzliche Variable können Konvertierungen vermieden werden. In dem Beispiel

   ```
   DCL Folge CHAR(8);
   Folge = Folge + 1;
   ```

   werden zwei Konvertierungen benötigt, bei der für die eine ein Bibliotheks-Aufruf erforderlich ist, während in dem gleichwertigen Beispiel

   ```
   DCL Folge  CHAR(8);
   DCL Zähler DECIMAL FIXED;
   Zähler = Zähler + 1;
   Folge  = Zähler;
   ```

   nur eine Konvertierung ohne Bibliotheksaufruf erforderlich ist.

   c) Sollen Daten von einer Struktur zu einer anderen transportiert werden, so sollten diese in ihrem Aufbau übereinstimmen, damit sie als Gesamtheit transportiert werden kann.

   d) In einem arithmetischen Ausdruck sollten unterschiedliche Datentypen vermieden werden, insbesondere Zeichenfolgen sind hier ungünstig.

   e) Maskierte Zeichenfolgen (PICTURE) sind gegenüber Zeichenfolgen (CHAR) zu bevorzugen. Soll z.B. ein Eingabeelement aus drei Dezimalzeichen bestehen und für die Korrektur falscher Werte soll weder ONSOURCE noch ONCHAR verwendet werden, so ist das Programmstück

   ```
   DCL  Folge  CHAR(3),
        Zahl   FIXED DECIMAL(5,0);
   ON CONVERSION GOTO Fehler;
   Zahl = Folge;
   ```

   weniger günstig als das Programmstück

   ```
   DCL  Folge  CHAR(3),
        Wert   PIC '999' DEFINED Folge,
        Zahl   FIXED DECIMAL(5,0);
   IF VERIFY   (Folge, '123456780')   = 0
        THEN GOTO Fehler;
   Zahl = Wert;
   ```

   f) Interne Schalter, Zähler und Variable, die für die Indizierung von Matrix-Elementen verwendet werden, sollten mit FIXED BINARY vereinbart sein. Daten, die für die Ausgabe bestimmt sind sollten DECIMAL sein.

   g) Bei der Vereinbarung der Genauigkeit für Variable, die in Ausdrücken verwendet werden ist Vorsicht geboten. Unterschiedliche Genauigkeiten können zu zusätzlichen Befehlen für die Bildung von Zwischenwerten führen.

4. Die folgenden Maßnahmen beziehen sich auf Folgen.

   a) Im Abschnitt 8.3 ist beschrieben, welche Operationen in-line erfolgen.

   b) Bitfolgen sollten mit ALIGNED vereinbart werden, sofern nicht wichtigere Gründe für eine möglichst dichte Speicherung sprechen.

   Wenn dichte Speicherung erforderlich ist, so ist Vorsicht geboten, wenn ein Datenverbund vorliegt, in dem nur Bitfolgen mit dem Attribut UNALIGNED vorkommen und die das Attribut BASED oder PARAMETER haben. Falls es möglich ist, sollte man sicherstellen, daß solche Datenverbunde auf einer Bytegrenze beginnen, indem man mindestens eines der Glieder mit dem Attribut ALIGNED vereinbart. Im anderen Fall nimmt der Übersetzer an, daß der Datenverbund auf einer beliebigen Bitgrenze beginnen kann. Dies erfordert zusätzlich Befehle.

   So sollte z.B. statt der Vereinbarung

```
DCL 1   Struktur   BASED,
        2 Type     BIT(1),
        2 Füller   BIT(7),
        2 Bit      BIT(1);
```

   besser geschrieben werden

```
DCL 1   Struktur   BASED,
        2 Type     BIT(1) ALIGNED,
        2 Bit      BIT(1);
```

   c) Man beachte, daß Verkettungen von Bitfolgen zeitaufwendig sind. Es ist günstiger die Pseudo-Variable SUBSTR zu verwenden.

   d) Folgen mit dem Attribut NONVARYING sind ungünstig, wenn die Länge zur Übersetzungszeit noch nicht bekannt ist, wie z.B. in

```
DCL      Folge  CHAR(Länge)
```

   e) Die eingebaute Funktion DATE ist zeitaufwendig. Man sollte dies in einem Programm nur einmal verwenden.

5. Die folgenden Maßnahmen beziehen sich auf Ein- und Ausgabe:

   a) Große Blocklängen (BLKSIZE) sind zeitgünstiger als kleine.

   b) Im FILE-Kommando sollte die Angabe SPACE möglichst so groß gewählt werden, wie die Datei vermutlich wird, um Speichernachforderungen zu vermeiden.

   c) Bei der Strom-Ein-Ausgabe ist eine lange Datenliste günstiger als mehrere kurze.

d) Durch Überlagerung kann die Ein-Ausgabe von Zeichenfolgen vereinfacht werden. In dem Beispiel

```
DCL 1   Eingabe,
        2 Typ       CHAR(2),
        2 Satz,
          3 Feld 1  CHAR(5),
          3 Feld 2  CHAR(7),
          3 Feld 3  CHAR(66);
GET EDIT (Eingabe) (A(2), A(5), A(7), A(66));
```

werden für jede Eingabe 4 Felder bearbeitet. Es ist günstiger, die ganze Struktur als ein Feld einzulesen und diesem Feld die Struktur zu überlagern:

```
DCL  Feld CHAR(80)  DEFINED (Eingabe);
GET  EDIT(Feld) (A(80));
```

Noch günstiger wäre eine READ-Anweisung.

8.2.3 Programmumstellung für virtuellen Speicher

Die Ausgabe des PLI1-Übersetzers ist gut angepaßt an die Erfordernisse eines virtuellen Speichersystems. Der auszuführende Code und die Konstanten sind schreibgeschützt und von den Daten getrennt. Im allgemeinen gibt es wenig Grund, das Programm umzustellen, um den Seitenwechsel (paging) gering zu halten. Wo eine solche Umstellung wichtig ist, kann eine Reihe von Schritten unternommen werden; die Auswirkungen sind jedoch im allgemeinen nur gering.

Die Ziele der Programmanpassung an ein virtuelles Speichersystem sind, den Seitenwechsel gering zu halten, d.h. die Anzahl der Datentransporte vom Hilfsspeicher in den Hauptspeicher und umgekehrt soll auf ein Minimum reduziert werden. Dies kann dadurch erreicht werden, daß man Felder, die gleichzeitig angesprochen werden, zusammenhängend ablegt und daß man möglichst wenig Seiten schafft, die verändert werden können.

Dies kann dadurch erreicht werden, daß man das Quellprogramm so schreibt, daß der Übersetzer daraus die günstigste Anpassung an ein virtuelles Speichersystem erstellt. Am meisten wird durch die Steuermöglichkeiten erreicht, die beim Binden der übersetzten Moduln zur Verfügung stehen, so daß bestimmte Moduln in bestimmten Seiten angeordnet werden.

Beim Entwurf und bei der Programmierung modularer Programme können weitere Anpassungen an ein virtuelles Speichersystem erreicht werden.

Damit der Übersetzer das beste Ergebnis erreichen kann, muß man sowohl darauf achten, wie das Quellprogramm geschrieben wird, als auch darauf, wie die Daten vereinbart werden.

Bei der Vereinbarung von Daten ist besonders auf Datenverbunde zu achten, die wesentlich größer sind als eine Seite. Felder innerhalb eines Datenverbundes, auf die gemeinsam zugegriffen wird, sollten auch beieinander stehen. In diesem Fall kann auch die Wahl zwischen einer Matrix, deren Elemente Strukturen sind und einer Struktur, deren Elemente Matrizen sind, von Bedeutung sein.

So sind in dem Beispiel

```
DCL 1 Struktur   DIMENSION (3000),
      2 Name     CHAR (20),
      2 Nummer   FIXED BINARY;
```

für jedes Matrix-Element Name und Nummer zusammenhängend abgelegt und können daher günstig gemeinsam adressiert werden. In der gleichwertigen Vereinbarung

```
DCL 1 Struktur,
      2 Name     CHAR (20)    DIMENSION (3000),
      2 Nummer   FIXED BINARY DIMENSION (3000);
```

werden die Namen zusammenhängend abgelegt, während der Name und die zugehörige Nummer einen großen Abstand haben.

Die Entscheidung zwischen der Speicherklasse STATIC INTERNAL und AUTOMATIC hat wenig Einfluß auf den Seitenwechsel. Dies trifft nicht für die Speicherklassen CONTROLLED und BASED zu.

Eine vollständige Kontrolle über die Anordnung von Speicherplatz für Variable kann man dadurch erreichen, daß man BASED-Variable verwendet und diese in einen Bereich (AREA) anordnet. Alle Variablen eines Bereichs werden zusammenhängend angeordnet.

Eine weitere Verbesserung ist möglich, wenn man die Anzahl der nicht schreibgeschützten Moduln verringert. Dazu muß man folgende Einzelheiten wissen:

Variable, bei denen der Übersetzer erkennen kann, daß sie nicht verändert werden, werden als Konstanten betrachtet und in einem schreibgeschützten Modul abgelegt. Dies sind

- Variable mit dem Attribut STATIC (CONSTANT) INTERNAL INITIAL (Angabe)
- Variable mit dem Attribut STATIC INTERNAL (Angabe), die folgenden Bedingungen genügt:
  - nicht Zielvariable in einer Zuweisung
  - nicht Aktual-Parameter bei einem Aufruf
  - nicht Ziel-Variable für eine Eingabe
  - nicht Aktual-Parameter für den Aufruf einer eingebauten Funktion

Der Übersetzer erzeugt für jede externe Prozedur folgende Moduln:

- Einen schreibgeschützten Modul mit CSECT, der die ausführbaren Anweisungen, alle Konstanten und die Variablen enthält, die als Konstante betrachtet werden (siehe oben).
- Optional einen nicht schreibgeschützten Modul für
  - STATIC-Variable, Anker für CONTROLLED-Variable, Datei-Konstante, usw.

Diese Module sind die Einheiten, die vom Binder weiterverarbeitet werden. Im Normalfall werden die Module in der zufällig anfallenden Reihenfolge abgelegt, was zu unnötigem Speicherverschnitt und erhöhtem Speicherbedarf führen kann.

Der Binder kann jedoch durch Steueranweisungen veranlaßt werden, bestimmte Module zusammenhängend oder einen bestimmten Modul am Anfang einer Seite abzulegen. Siehe dazu die Beschreibung des Binders.

Wenn ein Programm in günstiger Weise in Moduln unterteilt ist, kann man die Verwendung der Moduln analysieren und sie so anordnen, daß sich ein günstiger Seitenwechsel ergibt. Es sei jedoch darauf hingewiesen, daß dies eine schwierige und zeitaufwendige Arbeit ist.

8.2.4 Modulare Programmierung

Obwohl es möglich ist, ein Programm zu schreiben, das nur aus einer externen Prozedur besteht, ist es sinnvoller, ein Programm in Module aufzuteilen.
In PL/I ist die Basis-Einheit für die Modularisierung die Prozedur und der BEGIN-Block.

Der Vorteil der modularen Programmierung und der Gesichtspunkt der strukturierten Programmierung ist allgemein bekannt, jedoch gibt es auch Gesichtspunkte in Bezug auf effektive Programme.

Einige allgemeine Vorteile der modularen Programmierung sind nachstehend aufgeführt:

- Die Programmgröße beeinflußt Zeit- und Speicherbedarf für die Übersetzung. Ganz allgemein erhöht sich die Übersetzungszeit mehr als linear mit Programmgröße. Ferner wird bei Änderung in kleineren Quellprozeduren die Zeit für die Neuübersetzung geringer sein als bei großen Quellprozeduren.
- Eine Prozedur, die eine einfache Funktion erfüllen soll, braucht nur die Daten, die für diese Funktion erforderlich sind. Wegen der Eigenschaften der AUTOMATIC-Variablen besteht weniger die Gefahr, daß Daten anderer Funktionen dabei zerstört werden.
- Wenn eine Prozedur nur eine einzelne Funktion zu erfüllen hat, so kann sie viel einfacher durch eine geänderte Version ersetzt werden. Außerdem kann eine solche Funktion vielleicht auch in anderen Anwendungen verwendet werden.
- Der Speicherplatz für alle AUTOMATIC-Variablen einer Prozedur wird dann zugeordnet, wenn die Prozedur an einem ihrer Eingangspunkte aufgerufen wird. Verringert man die Anzahl der Funktionen, die durch eine Prozedur ausgeführt werden, so ist es häufig möglich, auch die Anzahl der Variablen in der Prozedur zu verringern. Dies kann wiederum den Gesamtbedarf an Speicherplatz für AUTOMATIC-Variable reduzieren.

Noch wichtiger in Bezug auf effektives Programmieren sind folgende Betrachtungen:

1. Wenn die statische CSECT oder der Aktivierungssatz größer wird als 4096 Bytes, dann muß der Übersetzer zusätzlichen Code einfügen, um den entfernten Speicher anzusprechen.
2. Wenn der erzeugte Code für eine Prozedur 4096 Bytes übersteigt, so muß das Basisregister häufiger umgesetzt werden.

Zusätzliche Aufrufe von Prozeduren erhöhen zwar die Ablaufzeit, aber die modulare Programmierung kann dies dadurch wieder ausgleichen, daß wesentlich weniger Befehle ausgeführt werden müssen.

8.3 In-line-Operationen

Viele Operationen werden in-line durchgeführt. Es zahlt sich daher für den Benutzer aus, wenn er sich darüber informiert, welche Operationen in-line durchgeführt und welche einen Bibliotheks-Aufruf benötigen und er sein Programm so abändert, daß möglichst wenig Bibliotheks-Aufrufe verwendet werden. Die meisten in-line-Operationen beziehen sich auf die Konvertierung und die Folgebearbeitung.

8.3.1 Daten-Konvertierung

Die Daten-Konvertierungen, die in-line durchgeführt werden, sind in Bild 8-1 aufgelistet. Eine Konvertierung außerhalb des angegebenen Bereiches oder der angegebenen Bedingung wird als Bibliotheks-Aufruf durchgeführt.

Bei arithmetischen maskierten Zeichenfolgen wird dann in-line konvertiert, wenn in der Maske nur die folgenden Zeichen enthalten sind:

V und 9
nicht gleitende Zeichen + - S $
gleitende Zeichen +... -... S... $...
Unterdrückungszeichen * Z
Einfügungszeichen , . / B

Masken, die nur die vorstehenden Zeichen enthalten, werden in Bezug auf in-line-Konvertierung in drei Gruppen unterteilt.

Typ 1: Masken, die nur 9 enthalten und wahlweise ein V und ein Vorzeichen vorn oder hinten.
Beispiele: ’99V99’, ’99’ ’S99V9’, ’99V+’, ’S999’

Typ 2: Masken mit Unterdrückungszeichen, ein Vorzeichen und Einfügungszeichen oder Masken nach Typ 1 mit Einfügungszeichen.
Beispiele: ’ZZZ’, ’**/**9’, ’ZZZ9V.99’, ’+ZZZ.ZZ’, ’S///99’, ’9.9’

Typ 3: Masken mit gleitenden Zeichen, Einfügungszeichen und Vorzeichen.
Beispiele: ’$$$$’, ’-,--9’, ’S/SS/S9’, ’+++9V.9’, ’$$$9-’

Die nachstehend aufgeführten Gründe können dazu führen, daß in den in folgenden Bildern angegebenen Fällen die Konvertierung nicht in-line durchgeführt wird.

- Die Bedingung SIZE ist wirksam und kann auftreten.
- Die Ziffernpositionen bei Ziel und Quelle überlappen sich nicht.
  So wird z.B. DECIMAL (6,8) oder DECIMAL (5,-3) nach PIC ’999V99’ nicht in-line konvertiert.

- Die Maske enthält eine bestimmte Kombination, die schwierig in-line zu konvertieren ist. So z.B.
  - Wenn zwischen gleitenden Zeichen * bzw. Z und der ersten 9 kein V steht, z.B. in ’ZZ.99’
  - Wenn gleitende Zeichen oder Unterdrückungszeichen rechts von einem Dezimalpunkt stehen, z.B. in ’ZZZV.ZZ’, ’++V++’.

Der Übersetzer gibt für jeden generierten Sprung im Laufzeitsystem eine Informationsmeldung aus, wenn *COMOPT DIAGNOST = INFORMATION angegeben wurde.

Konvertierung		Bedingung und Kommentar	
Quelle	Ziel		
FIXED BINARY	FIXED BINARY		
	FIXED DECIMAL		
	FLOAT	Ziel: Ganzwort oder Doppelwort	
	BIT (n)	n≤32,NONVARYING n≤2088,NONVARYING ALIGNED STRINGSIZE unwirksam	
	CHARACTER (n)	n≤256; STRINGSIZE unwirksam; über FIXED DECIMAL	
	PICTURE numerisch	Länge≤256;über FIXED DECIMAL; Maskentyp 1,2 und 3;SIZE unwirksam	
FIXED DECIMAL (g,k)	FIXED BINARY		
	FIXED DECIMAL		
	FLOAT	p+q≤75;Ziel: Ganzwort oder Doppelwort	
	BIT (n)	Länge≤2088;NONVARYING ALIGNED; STRINGSIZE unwirksam	
	CHARACTER (n)	n≤256; wenn g = k dann gradzahlig; STRINGSIZE unwirksam	
	PICTURE numerisch	Maskentyp 1,2 und 3	
FLOAT Ganzwort oder Doppelwort	FIXED BINARY		
	FIXED DECIMAL	k≤80;SIZE unwirksam	
	FLOAT	Ziel,Quelle: Ganzwort oder Doppelwort	
	BIT (n)	n≤2088;NONVARYING ALIGNED STRINGSIZE unwirksam	
BIT (n)	FIXED BINARY	n≤32	Quelle: NONVARYING
	FIXED DECIMAL	n≤32	Quelle: NONVARYING ALIGNED
	FLOAT		
	CHARACTER	n = 1	

Bild 8-1 (Teil 1) Implizite Konvertierungen, die in-line durchgeführt werden

Konvertierung		Bedingung und Kommentar
Quelle	Ziel	
CHARACTER (n)	BIT	n = 1;CONVERSION unwirksam
	FIXED DECIMAL	
	FLOAT	
	FIXED BINARY	
PICTURE alphanumerisch	CHARACTER (n)	n≤256;NONVARYING
	PICTURE alphanumerisch	Masken müssen identisch sein
PICTUREnume-risch,Maskentyp 1,2 und 3 außer$...	FIXED BINARY	über FIXED DECIMAL;SIZE unwirksam
	FIXED DECIMAL	Maskentyp 2 ohne * und Einfügungszeichen,./B SIZE unwirksam
	FLOAT	über FIXED DECIMAL;SIZE unwirksam
	PICTURE	Maskentyp 1,2 oder 3;SIZE unwirksam
LABEL	LABEL	
Zeiger	Zeiger	

Bild 8-1 (Teil 2) Implizite Konvertierungen, die in-line durchgeführt werden

Folgen-Operation	Bedingungen für die Operanden
Zuweisung	CHARACTER
	BIT mit konstanter Länge≤32
	BIT mit konstanter Länge beginnend auf Bytegrenze
Boolesche Operationen	wie bei Zuweisung
Vergleich	wie bei Zuweisung
Verketten	CHARACTER

Bild 8-2 Operation für Folgen, die unter den angegebenen Bedingungen in-line ausgeführt werden

Folgen-Funktion	Bedingungen
AFTER	nie
BEFORE	nie
BIT	immer
BOOL (a, b, c)	wenn c eine Konstante dann wie bei Boolesche Operationen
CHAR	immer
COLLATE	immer
COPY (a, b)	wenn a konstante Länge≤4096 und b Konstante≤4095
DECAT	nie
HIGH (a)	wenn a Konstante≤4095
INDEX	CHARACTER
LENGTH	immer
LOW (a)	wenn a Konstante≤4095
REPEAT (a, b)	wenn a konstante Länge≤4096 und b Konstante≤4095
REVERSE (a)	nur im Zusammenhang INDEX (REVERSE (a),b) wenn a CHARACTER und b CHAR (1) oder b Konstante mit der Länge≤256
SEARCH	nie
STRING	wenn Füllzeichen vorhanden wegen VARYING oder BIT ALIGNED oder unverbundenem Speicher dann wie Verketten, sonst wie Zuweisung
SUBSTR	wenn STRINGRANGE unwirksam
TRANSLATE (a,b,c)	wenn a konstante Maximal-Länge≤256 und b und c konstante Länge≤256
UNSPEC	immer
VALID	nie
VERIFY	CHARACTER

Bild 8-3 Eingebaute Folge-Funktionen, die unter den angegebenen Bedingungen in-line ausgeführt werden

8.3.2 Folgen-Bearbeitung

Die Funktionen und Operationen, die in-line ausgeführt werden, sind in den Bildern 8-2 und 8-3 aufgelistet. Einige dieser Funktionen werden allerdings in bestimmten Zusammenhängen als Bibliotheks-Aufruf ausgeführt.
Wenn z.B. die Ausdrücke in den Funktionen BIT oder CHAR eine implizite Konvertierung erfordern, die nicht in-line erfolgt, so wird die entsprechende Bibliotheks-Prozedur aufgerufen.

8.4 Globale Optimierungen

8.4.1 Gemeinsame Ausdrücke

Der Begriff "gemeinsame Ausdrücke" wird für Ausdrücke wie z.B. "B C" in

```
A = B * C + D1
   .
   .
   .
D = B * C + D2
```

verwendet, wobei die Variablen B und C zwischen den beiden Ausdrücken nicht verändert werden. In diesem Falle ist es nicht nötig, den Ausdruck "B * C" mehrmals zu berechnen.

Der Vorgang, daß bei solchen Ausdrücken die mehrfache Berechnung vermieden wird, wird als "Eliminierung gemeinsamer Ausdrücke" bezeichnet.

Eine wichtige Anwendung für die Eliminierung gemeinsamer Ausdrücke finden wir in Anweisungen, in denen indizierte Variable verwendet werden und bei denen der gleiche Indexwert für mehrere Variable verwendet wird.

Beispiel

```
PREIS (ARTIKEL) = MENGE (ARTIKEL) *PREIS (ARTIKEL)
```

Der Wert des Index ARTIKEL wird nur einmal berechnet. Dabei wird, wenn die Berechnung des Index einen Dezimalwert ergibt, die Konvertierung in einen Dualwert mit durchgeführt.

8.4.1.1 Unterbrechungsbehandlung

Die Reihenfolge der meisten Operationen in einer PL/I-Anweisung hängt von der Priorität der betroffenen Operatoren ab. Jedoch ist die Reihenfolge der Auswertung der Teilausdrücke, deren Ergebnis den Operanden für die Operatoren niedrigerer Priorität bilden, nur insoweit definiert, daß ein Operand vollständig ausgewertet ist, bevor sein Wert für eine weitere Operation verwendet wird. Solch ein Teilausdruck kann z.B. ein Index-Ausdruck, ein Zeiger-Ausdruck oder ein Funktion-Bezug sein.

Aus dem vorstehend genannten Grund können ON-Einheiten, die im Zusammenhang mit Teilausdrücken angesprochen werden, in einer nicht voraussehbaren Reihenfolge aufgerufen werden. Als Folge davon kann die Auswertung eines Ausdruckes verschiedene Werte ergeben.

Das Ergebnis kann davon abhängen, in welcher Reihenfolge die ON-Einheiten aufgerufen und welche Anweisungen dort ausgeführt werden. Wird eine ON-Einheit für eine berechenbare Bedingung aufgerufen, so gilt:

1. Alle Variablen enthalten den Wert, der ihnen bei der Ausführung vorhergehender Anweisungen zugewiesen wurde. Diese Werte können in der ON-Einheit verwendet werden. So kann z.B. die Anweisung PUT DATA verwendet werden, um den Wert einer Variablen auszugeben, den sie beim Eintritt in die ON-Einheit besitzen.
2. Wird in einer ON-Einheit, die durch das Setzen einer berechenbaren Bedingung aufgerufen wurde, Variablen ein Wert zugewiesen, so ist dies für jeden folgenden Programmteil der aktuelle Wert der Variablen.

Immer, wenn die Möglichkeit besteht, daß Variable auf Grund einer auftretenden berechenbaren Bedingung verändert werden, sei es durch die entsprechende ON-Einheit oder auf Grund eines Sprungs aus der ON-Einheit, unterbleibt die Eliminierung gemeinsamer Ausdrücke.

Dazu ein Beispiel:

```
ON   ZERODIVIDE B,C = 1;
  .
  .
  .
X = A * B + B/C;
Y = A * B + D;
```

Der Teilausdruck A * B ist zwar in den beiden Zuweisungen gemeinsam, aber er wird nicht eliminiert. Falls nämlich die Bedingung ZERODIVIDE auftritt, kann der gleiche Teilausdruck in der zweiten Zuweisung einen anderen Wert haben als in der ersten.

Die vorstehenden Aussagen gelten nur, wenn die Angabe ORDER gegeben ist, explizit oder implizit. Sind die vorstehenden Einschränkungen nicht erwünscht, so kann dies durch die Angabe REORDER erreicht werden; die Eliminierung gemeinsamer Ausdrücke wird dann nicht eingeschränkt.
ORDER und REORDER sind in weiteren Abschnitten beschrieben.

8.4.2 Auslagerung invarianter Ausdrücke oder Anweisungen aus DO-Schleifen

Steht ein Ausdruck in einer Schleife und kann der Übersetzer feststellen, daß bei jedem Schleifendurchlauf das gleiche Ergebnis entsteht, so handelt es sich um einen invarianten Ausdruck. Das entsprechende gilt auch für Anweisungen.

Ein invarianter Ausdruck oder eine invariante Anweisung kann aus der Schleife herausgenommen und vor die Schleife gesetzt werden, so daß statt einer Auswertung bei jedem Schleifendurchlauf nur eine einzige Auswertung vor dem Eintritt in die Schleife nötig ist. Dazu ein Beispiel:

```
DO  I = 1 TO N:
   .
   .
   .
J = 3;
   .
   .
   .
END;
```

Die Zuweisung J = 3 kann unter bestimmten Bedingungen invariant sein und dann aus der Schleife ausgelagert werden. Unter bestimmten Bedingungen kann sie auch innerhalb der Schleife vor- oder zurückverlegt werden, wenn dies Vorteile ergibt.

Wird die Auslagerung invarianter Teile gewünscht, so muß bei einem die Schleife umfassenden Block die Angabe REORDER explizit oder implizit vorhanden sein.

8.4.3 Reduktion linearer Ausdrücke in DO-Schleifen

Die Multiplikation einer Laufvariablen mit einer Konstanten ist die einfachste Form eines linearen Ausdrucks. Wo es möglich ist, wird eine Multiplikation in eine Addition überführt, die schneller ausgeführt werden kann.

Beispiel

```
DO  I = M TO N BY 2;
    .
    .
    .
    A(I) = I * 4;
    .
    .
    .
    END;
```

Wenn bei diesem Beispiel innerhalb der Schleife die Variable I nicht verändert wird, so wird ein Code erzeugt, der dem folgenden Programmteil entspricht:

```
       I = M;
       IF I > N THEN GOTO Ende;
       TEMP = 4 * I;

Anfang:.
       .
       .
       I = I + 2:
       TEMP = TEMP + 8;
       IF I < N THEN GOTO Anfang;
Ende:  .
       .
       .
```

In diesem Beispiel ist nicht gezeigt, daß auch die Adreßrechnung für den indizierten Zugriff A (I) zusätzlich optimiert wird. Hier liegt in Wirklichkeit die Stärke dieser Art von Optimierung.

Wünscht man die in diesem Abschnitt beschriebene Art von Optimierung, so muß REORDER angegeben sein.

8.4.4 ORDER- und REORDER-Angabe

Die Angaben ORDER und REORDER werden zur Steuerung der Optimierung verwendet. Sie können bei den Anweisungen PROCEDURE und BEGIN angegeben werden und gelten für den ganzen Block. Die Angaben vererben sich auf untergeordnete Blöcke, sofern dort nicht eine explizite Angabe vorhanden ist. Die Voreinstellung ist ORDER.

8.4.4.1 ORDER-Angabe

Wenn gefordert wird, daß in ON-Einheiten, die auf Grund einer berechenbaren Bedingung aufgerufen werden, eine Variable stets den zuletzt im Block zugewiesenen Wert enthält, so muß für den Prozedur- bzw. BEGIN-Block die Angabe ORDER gemacht werden.

Auch in einem Block mit der Angabe ORDER können gleichwertige Ausdrücke eliminiert werden. In diesem Falle kann es sein, daß berechenbare Bedingungen seltener auftreten, als wenn gleichwertige Ausdrücke nicht eliminiert werden. Falls jedoch eine berechenbare Bedingung in einem Block mit der Angabe ORDER auftritt, so ist der Wert der Variablen in Anweisungen, die der Stelle vorangehen, bei der die Bedingung gesetzt wird, auf jeden Fall der letzte Wert, der der Variablen zugewiesen wurde, falls in der ON-Einheit ein Bezug auf diese Variable vorhanden ist.

In einem Block mit der Angabe ORDER sind auch andere Optimierungen erlaubt, mit Ausnahme der Auslagerung von Ausdrücken für die eine Bedingung auftreten kann. Da dies nur dann vermieden werden kann, wenn alle betroffenen berechenbaren Bedingungen unwirksam gesetzt werden, bedeutet die Verwendung von ORDER praktisch, daß keine Auslagerung invarianter Teile aus Schleifen stattfindet.

8.4.4.2 REORDER-Angabe

Ist REORDER angegeben, so werden vom Übersetzer alle die Optimierungen vorgenommen, die die Programmlogik, wie sie im Quellprogramm niedergeschrieben ist, unverändert lassen, solange beim Programmlauf keine Fehler auftreten. Auslagerung invarianter Teile vor oder hinter eine Schleife wird durchgeführt, so daß diese nur einmal vor oder hinter der Schleife ausgeführt werden.

Kürzere Ausführungszeiten bei Schleifen können dadurch erreicht werden, daß man Werte von Variablen, die sehr häufig in der Schleife verändert werden in Registern hält. Bei fehlerfreiem Programmlauf können Werte in Registern gehalten und eine beträchtliche Zeitersparnis dadurch erreicht werden, daß das Holen und Speichern der Werte im Speicher entfällt. Falls der letzte Wert der Variablen hinter der Schleife benötigt wird, so wird beim Verlassen der Schleife der Wert der Variablen ihrem Speicherplatz im Speicher zugewiesen.

Registerzuordnungen können wirksamer durchgeführt werden, wenn REORDER angegeben ist. Jedoch wird nicht garantiert, daß Variable, die im Block verändert werden, ihren letzten Wert haben, wenn eine berechenbare Bedingung auftritt, da der letzte Wert im Register und nicht im Speicher stehen kann. Aus diesem Grunde sollten in einer ON-Einheit, die auf Grund einer berechenbaren Bedingung aufgerufen wird, kein Bezug auf eine Variable stehen, die im Block mit der Angabe REORDER verändert wird. Die Verwendung der eingebauten Funktionen ONSOURCE und ONCHAR ist in diesem Zusammenhang jedoch gültig.

Ein Programm ist fehlerhaft, wenn während des Laufs eine berechenbare Bedingung

oder die Bedingungen ERROR oder ATTENTION in einem Block mit der Angabe REORDER auftritt und dadurch eine Variable verwendet wird, deren Wert nicht garantiert ist.

Da diese Einschränkungen die Korrektur fehlerhafter Daten ausschließt, ausgenommen die Korrektur durch ONSOURCE und ONCHAR in einer ON-Einheit für die Bedingung CONVERSION, ist der Benutzer angewiesen auf die System-Einheit für die Bedingung, d.h. Beendigung des Programmlaufs oder er muß die ON-Einheit zur Fehlerkorrektur verwenden und mit neuen Daten den betroffenen Programmteil neu starten.

Hierzu ein Beispiel:

```
ON OVERFLOW  PUT DATA;
DO J = 1  TO M;
   DO I = 1 TO N;
      X(I,J) = Y(I) + Z(J) * L + SQRT(W);
      P = I * J;
      END;
   END;
```

Wenn die obige Befehlsfolge in einem Block mit der Angabe REORDER steht, so entspricht das übersetzte Programmstück den folgenden Anweisungen:

```
ON OVERFLOW  PUT DATA;
Temp1 =SQRT(W);
DO J = 1 TO M;
   Temp2  =J;
   DO I = 1 TO N;
      X(I,J) = Y(I) + Z(J) * L + Temp1;
      P = Temp2;
      Temp2 =Temp2 + J;
      END;
   END;
```

Temp1 und Temp2 sind temporäre Variable, die den Wert von Ausdrücken aufnehmen, die vor die Schleife ausgelagert sind. Die Multiplikation I * J ist auf die Addition zurückgeführt (Temp2 + J). Die Zuweisung P = I * J könnte durch P = N * M außerhalb der Schleife ersetzt werden.

Wenn z.B. die Bedingung OVERFLOW auftritt, wird in diesem Fall nicht garantiert, daß die Variablen den aktuellen Wert besitzen, da diese ggf. in Registern gehalten werden und nicht im Speicherplatz der Variablen, auf den in der ON-Einheit zugegriffen werden kann.

In dem obigen Beispiel ist nicht gezeigt, daß auch I,J,P die Adreßrechnungen für die indizierten Zugriffe X(I,J), Y(I) und Z(I) optimiert werden.

8.4.5 Vermeiden von Seiteneffekten / Reduzible Funktionen

Eine Funktion, deren Aufruf keine Seiteneffekte produziert und die einen Wert zurückliefert, der nur von den Werten der Aktual-Parameter abhängt, ist reduzibel.

Ein Seiteneffekt ist jede Veränderung irgendeiner Variablen, die außerhalb der gerufenen Funktion bekannt ist, sowie jeder Ein-Ausgabe-Vorgang. Der Übersetzer betrachtet eine Funktion dann und nur dann als reduzibel, wenn sie mit dem Attribut REDUCIBLE vereinbart ist. Für reduzible Funktionen werden gemeinsame Ausdrücke eliminiert und invariante Ausdrücke ausgelagert, wie dies für Operationen und eingebaute Funktionen erläutert ist. Abgesehen davon ist es allgemein günstig, reduzible Funktionen REDUCIBLE zu erklären, da der Übersetzer besser optimieren kann, wenn Seiteneffekte ausgeschlossen sind.

8.4.6 Optimierung Boolescher Ausdrücke

Treten in einem Ausdruck, der in einer Bedingung für eine Verzweigung steht, komplexe Boolesche Ausdrücke auf, so werden diese in eine Reihe von Verzweigungen mit einfachen Booleschen Ausdrücken aufgelöst. Dabei werden nur soviel Boolesche Ausdrücke ausgewertet, bis das Ergebnis eindeutig ermittelt ist. Der restliche Teil wird ignoriert. Irreduzible Funktionen werden jedoch in jedem Fall aufgerufen. So wird z.B. für die Anweisung

```
IF A < B | C > D THEN GOTO L;
```

so getan, als ob hier folgende Anweisungen stehen

```
IF A < B THEN GOTO L;
IF C > D THEN GOTO L;
```

Der Code für die Anweisung

```
IF A <B & C> D THEN X = Y;
```

entspricht den Anweisungen

```
   IF A ≥ B THEN GOTO L;
   IF C ≤ D THEN GOTO L;
   X = Y;
L: ...
```

8.4.7 Vereinfachung von Ausdrücken

Eine Vereinfachung von Ausdrücken bedeutet, daß die Form eines Ausdruckes geändert wird, ohne daß sich das erwartete Ergebnis ändert, um einen günstigeren Code zu erzeugen. Eine wesentliche Vereinfachung betrifft Ausdrücke, in denen arithmetische Konstante vorkommen.

Konstante Ausdrücke der Form $C_1 + C_2$, $C_1 - C_2$, $C_1 * C_2$, wobei C_1 und C_2 ganzzahlige Konstanten sind, werden durch äquivalente Konstanten ersetzt. So wird z.B. der Ausdruck 2 + 5 durch die Konstante 7 ersetzt.

Bei Ausdrücken, die bei der Adressierung von Matrix-Elementen und elementaren Struktur-Gliedern entstehen, versucht der Übersetzer die konstanten Teile herauszuziehen.

Ausdrücke der Form

C_1 * (exp + C_2), C_1 * (exp - C_2), C_1 * (C_2 + exp)

werden umgeformt in äquivalente Ausdrücke der Form

C_3 * exp + C_4

wobei C_1, C_2, C_3, C_4 ganzzahlige Konstanten sind und exp ein ganzzahliger Ausdruck ist.

Dadurch wird z.B. folgendes erreicht:

```
DCL A DIMENSION (20,20);
A (C1 * I + C2, C3 * K + C4)
```

wird genauso effektiv adressiert wie A (I,K).

8.4.8 Initialisierung von Datenverbunden

Diese Art der Optimierung betrifft Datenverbunde, die mit AUTOMATIC, BASED oder CONTROLLED vereinbart sind.

Wenn alle Elemente einer Matrix mit demselben Wert initialisiert werden, wird Code erzeugt, der das erste Element initialisiert und den Rest der Initialisierung durch einen einzigen Transportbefehl oder, im Fall nicht konstanter Grenzen, durch ein spezielles Unterprogramm leistet.

Beispiel

```
DCL A DIMENSION (20,20) FIXED BINARY INIT ((400)0);
```

Die Matrix A wird in der beschriebenen Weise initialisiert. Nicht angewendet wird diese Art der Optimierung für eine Matrix deren Elemente mit CHAR VARYING oder BIT VARYING vereinbart sind.

Diese Art der Optimierung wird auch angewendet, wenn es sich um Zuweisung eines Element-Wertes an eine Matrix handelt, wenn die Matrix einen verbundenen Speicher besitzt.

Wenn der zu initialisierende Datenverbund, sei es nun eine Matrix oder eine Struktur, konstante Längen und konstante Initialwerte hat, so wird eine Initialisierungs-Konstante für den gesamten Datenverbund abgelegt, der dann mit einem einzigen Transportbefehl initialisiert wird. Nicht angewendet wird diese Art der Optimierung, wenn der Speicherbedarf für die Initialisierungs-Konstante größer wäre als für den allgemeinen Initialisierungscode.

Beispiel

```
DCL A DIMENSION (3) BINARY FIXED INIT (1,2,3);
```

Die Matrix A wird in dieser Weise initialisiert.

8.4.9 Spezialcode bei Datenverbund-Zuweisung

Der Übersetzer realisiert Zuweisungen an Matrizen und Strukturen, wenn möglich, durch einen einzigen Datentransport. Dies ist der Fall, wenn Quell- und Ziel-Datenverbund Datenbeschreibungen haben, die, abgesehen von Speicherklassen, übereinstimmen und verbundenen Speicher besitzen.

8.4.10 Verwendung der Register bei der DO-Anweisung

Soweit das möglich ist, werden für DO-Schleifen die für die Schleifenbildung relevanten Größen während des Schleifendurchlaufs in Registern gehalten. So wird z.B. der Übersetzer versuchen, bei der Anweisung

```
DO Laufvariable = Anfangswert BY Schrittweite TO Endwert;
```

die Werte für die Laufvariable, für die Schrittweite und für den Endwert in Registern zu halten. Dies ergibt eine sehr wirksame Optimierung.

8.4.11 Aufruf interner Prozeduren

Der Übersetzer versucht solche interne Prozeduren zu erkennen, die nur von ihrem umfassenden Block aufgerufen und nicht rekursiv verwendet werden und deren automatischer Speicherbereich eine konstante Größe hat.

Für solche Prozeduren können für den Aufruf und für die Rückkehr kürzere Codefolgen generiert werden als im allgemeinen Fall.

Die gleiche Aussage gilt auch für BEGIN-Blöcke, deren automatischer Speicherbereich ein konstante Größe hat.

8.4.12 Ausnutzung der globalen Optimierung

Dieser Abschnitt enthält Einzelheiten über Codierungs-Praktiken, die beachtet oder vermieden werden sollten, wenn man die Vorteile der globalen Optimierung ausnutzen will, die durch die Steueranweisung * COMOPT OPTIMIZE = TIME eingestellt wird.

8.4.12.1 Eliminierung gemeinsamer Ausdrücke

Die Eliminierung gemeinsamer Ausdrücke unterbleibt in folgenden Fällen:

1. Bei Ausdrücken, in denen Variable verwendet werden, deren Wert in einer Ein- Ausgabe-Bedingung oder in einer berechenbaren Bedingung verändert wird.
2. Wenn eine BASED-Variable einer Variablen überlagert wird, die in einem gleichwertigen Ausdruck verwendet und der BASED-Variablen zwischen den gleichwertigen Ausdrücken ein neuer Wert zugewiesen wird, so unterbleibt die Optimierung.

   So wird z.B. in dem folgenden Programmstück der gleichwertige Ausdruck X + Z nicht eliminiert, weil die BASED-Variable A der Variablen X überlagert (P = ADDR(X)) und der Variablen A zwischen den beiden gleichwertigen Ausdrükken ein Wert zugewiesen wird.

```
DCL A BASED (P)
P = ADDR (X);
  .
  .
  .
P = ADDR (Y);
  .
  .
  .
B = X + Z;
P -> A = 2
C = X + Z;
```

3. Bei der Verwendung von Alias-Variablen.
   Eine Alias-Variable ist eine Variable, deren Wert durch einen Bezug auf einen Bezeichner verändert werden kann, der nicht der eigene Bezeichner ist. Beispiele dafür sind DEFINED-Variable und die zugehörige Basis-Variable, Aktual- und Formal-Parameter und BASED-Variable, und die von ihnen überlagerten Variablen.

   Variable, deren Adressen einer externen Prozedur über Zeiger bekannt sind, und die entweder als externe Variable oder als Aktual-Parameter verwendet werden, werden ebenfalls als Alias-Variable betrachtet.
   Eine Alias-Variable verhindert die Eliminierung gleichwertiger Ausdrücke nicht vollständig, aber sie schränkt sie ein.

   Wenn ein gleichwertiger Ausdruck eine Alias-Variable enthält, so werden die möglichen Ablaufwege, in denen gemeinsame Ausdrücke vorkommen können, nach Zuweisungen durchsucht, in denen als Zielvariable entweder die Variable selbst verwendet wird oder eine Alias-Variable dieser Variablen.

   Wenn ein Programm eine externe Zeiger-Variable enthält, so wird angenommen, daß dieser Zeiger auf alle Variablen gesetzt werden kann, deren Adresse externen Prozeduren bekannt ist. Dies sind alle Variablen, deren Adresse dem externen Zeiger zugeordnet werden könnte. Das bedeutet, daß sich alle Variablen, die durch externe Zeiger adressiert werden oder durch andere Zeiger, denen der Wert eines externen

Zeigers zugewiesen wurde, auf externe Variable beziehen können.

4. Bei veränderter Form eines Ausdruckes.
   Wenn der Teilausdruck B + C als gleichwertiger Ausdruck behandelt wird, so würde der Übersetzer nicht in der Lage sein, diesen als gleichwertigen Ausdruck in der folgenden Anweisung zu entdecken:

   ```
   D = A + B + C;
   ```

   Der Übersetzer verarbeitet den Ausdruck von links nach rechts. Folglich erkennt er den Ausdruck A + B und (A + B) + C. Wird der Aufruf jedoch D = A + (B + C) geschrieben, so kann der Benutzer sicher sein, daß B + C als gleichwertiger Ausdruck betrachtet wird, da der Übersetzer den Ausdruck mit der höchsten Priorität zuerst verarbeiten muß.

5. Abhängig vom Bereich gemeinsamer Ausdrücke.
   Um gemeinsame Ausdrücke erkennen zu können, wird das Programm analysiert und es werden Fluß-Einheiten ermittelt. Eine Fluß-Einheit ist ein Programmteil, der nur am Anfang betreten werden kann und stets am Ende verlassen wird. Eine Fluß-Einheit kann mehrere PL/I-Anweisungen enthalten und umgekehrt kann eine PL/I-Anweisung mehrere Fluß-Einheiten enthalten.

   Gleichwertige Ausdrücke werden über mehrere individuelle Fluß-Einheiten erkannt. Wenn der Ablauf zwischen den Fluß-Einheiten jedoch komplex wird, wird die Erkennung von gleichwertigen Ausdrücken über die Grenzen von Fluß-Einheiten unterdrückt.

Die Eliminierung gemeinsamer Audrücke wird durch die nachstehend aufgeführten Punkte unterstützt:

1. Variable in Ausdrücken sollten weder external sein, noch verbunden sein mit externen Zeigern noch in der eingebauten Funktion ADDR als Parameter verwendet werden.
2. Das Quellprogramm sollte keine externen Prozeduren, keine externen Marken-Variablen sowie keine Marken-Konstanten, die externen Prozeduren bekannt sind, verwenden.
3. Variable in Ausdrücken sollten in ON-Einheiten weder verändert werden, noch sollte auf sie zugegriffen werden.
4. REORDER für den Block angeben.
5. Reduzible Funktion mit REDUCIBLE vereinbaren.

8.4.12.2 Auslagerung invarianter Ausdrücke

Die Auslagerung invarianter Ausdrücke aus Schleifen wird verhindert durch:

1. ORDER-Angabe für den Block. Die Auslagerung wird jedoch nicht in jedem Fall durch die ORDER-Angabe verhindert, sondern nur für solche Operationen, die eine berechenbare Bedingung setzen können.
2. Die Verwendung von Variablen, deren Wert durch Ein-Ausgabe-Anweisungen gesetzt oder verwendet werden können.
3. Die Verwendung von Variablen, die in ON-Einheiten für Ein-Ausgabe-Bedingungen oder berechenbare Bedingungen gesetzt werden können oder die Alias-Variable sind.
4. Ein komplexer Programmfluß, der externe Prozeduren, externe Variable oder Marken-Konstante einschließt.

Die Auslagerung invarianter Ausdrücke aus Schleifen wird unterstützt durch:

1. REORDER für den Block angeben.
2. Vermeidung der weiter oben angegebenen Punkte 2 bis 4.
3. Reduzible Funktionen mit REDUCIBLE vereinbaren.

8.4.12.3 Reduktion linearer Ausdrücke in Schleifen

Diese Art von Optimierung ist nur dann möglich, wenn die Laufvariable innerhalb der Schleife nicht verändert wird. Es gelten weiterhin die gleichen Bedingungen wie bei der Auslagerung invarianter Ausdrücke, wie dies weiter oben beschrieben ist.

8.4.12.4 Register- und Adreß-Optimierung

Für diese Art der Optimierung gelten die gleichen Bedingungen wie für die weiter oben beschriebene Eliminierung gemeinsamer Audrücke.

8.4.12.5 Verwendung der Register bei DO-Anweisungen

Hier gilt das gleiche wie bei der weiter oben beschriebenen Reduktion linearer Ausdrücke in Schleifen. Zusätzlich müssen folgende Bedingungen erfüllt sein:

- Während der Ausführung der Schleife darf auf die Laufvariable weder in einer ON-Einheit noch in einer Prozedur Bezug genommen werden.
- Es darf nicht aus der Schleife herausgesprungen werden.

8.5 Steuerung der Optimierung (OPTIMIZE)

Die Optimierung, die der Übersetzer durchführt, kann durch die Steueranweisung

```
*COMOPT OPTIMIZE = Spezifikation
```

beeinflußt werden. Die Spezifikationen sind in den nachfolgenden Unterabschnitten erläutert. Diese Spezifikationen können auch bei PROCEDURE OPTIONS (Angabe) angegeben werden.

8.5.1 Zeitoptimierung (TIME)

Es werden alle die Optimierungen durchgeführt, die im Abschnitt 8.1.1 angegeben sind.

8.5.2 Bedingungen unwirksam setzen (ENABLING)

Durch diese Spezifikation wird für alle berechenbaren Bedingungen außer OVERFLOW, UNDERFLOW und ZERODIVIDE die Voreinstellung auf "unwirksam" gesetzt. Danach bleiben nur diejenigen Bedingungen per Voreinstellung wirksam, für deren Abprüfung der Übersetzer in keinem Fall spezielle Befehle zu erzeugen braucht, da die Abprüfung von den Maschinenbefehlen selbst geleistet wird.

8.5.3 Reihenfolge der Anweisungen modifizierbar (REORDER)

Durch diese Spezifikation wird die System-Vorgabe von ORDER auf REORDER umgestellt.

Es wird empfohlen für Produktionsläufe stets OPTIMIZE = (TIME, ENABLING, REORDER) zu verwenden und in der Quellprozedur dort wo es nötig ist, den entsprechenden Bedingungs-Vorspann vor die Anweisung zu setzen und die ORDER-Angabe explizit anzugeben.

8.5.4 Überlappung (OVERLAP)

Ist diese Spezifikation angegeben, so geht der Übersetzer davon aus, daß sich Ziel- und Quellbereich einer Zuweisung nicht überlappen oder identisch sind, sofern er eine Überlappung nicht eindeutig erkennen kann, denn Transporte zwischen sich nicht überlappenden Daten sind schneller. In diesem Falle gibt der Übersetzer eine Warnung aus. Ist diese Spezifikation nicht angegeben, so entscheidet der Übersetzer in Zweifelsfällen für "überlappend" und erzeugt sicheren Code. Dadurch wird die Laufzeit erhöht.

Diese Spezifikation berührt nicht den Fall, daß auf der rechten Seite ein echter Ausdruck steht, außer wenn es sich um SUBSTR, UNSPEC oder STRING mit einem variablen Bezug handelt.

8.6 Programmierhinweise

In diesem Abschnitt sind einige Fehler und Fallen aufgelistet, die beim Schreiben eines Programms sehr häufig auftreten. Sie beruhen darauf, daß Regeln mißverstanden oder übersehen werden oder weil Konventionen und Einschränkungen der Implementierung nicht beachtet werden.

8.6.1 Quellprogramm und allgemeine Syntax

1. Wenn bei der manuellen Niederschrift des Quellprogramms die nachfolgenden Zeichen nicht sauber dargestellt werden, kann es bei der Eingabe der Quelltexte durch eine zweite Person zu Schreibfehlern kommen.

   ```
   1 (Ziffer), I (Buchstabe), | (senkrechter Strich)
   / (Schrägstrich), ' (Apostroph)
   7 (Ziffer), > (größer als)
   L (Buchstabe) < (kleiner als)
   O (Buchstabe), 0 (Ziffer)
   S (Buchstabe), 5 (Ziffer)
   Z (Buchstabe), 2 (Ziffer)
   _ (Unterstrich), - (Minuszeichen)
   ```

2. Es ist darauf zu achten, daß die Quellzeilen nur innerhalb des Bereichs geschrieben werden, der durch

   ```
   COMOPT MARGINS = TEXT (a, b)
   ```

   gegeben ist. Voreinstellung ist TEXT (2,72).

3. Werden bestimmte Zeichen vergessen, so kann dies zu Fehlern führen, die oft nur schwer erkennbar sind. Dazu gehört

   - Der abschließende Apostroph bei Literalen fehlt
   - Die schließende Klammer fehlt oder es gibt mehr öffnende als schließende Klammern.
   - Bei einem Kommentar fehlt die abschließende Zeichenfolge */ oder sie ist falsch geschrieben /*.

   Literale und Kommentare, die über mehr als eine Zeile gehen, erkennt man im Übersetzungs-Protokoll an einem Stern hinter der Zeilennummer.

4. Reservierte Schlüsselwörter im 48-Zeichen-Satz (z.B. GT, CAT) müssen in jedem Fall beidseitig in Leerzeichen oder Kommentare eingeschlossen sein.

   Achtung

   Versehentliche Verwendung dieser Schlüsselwörter kann zu schwer verständlichen Fehlermeldungen führen. Zum Beispiel wird "DCL NL" als Syntaxfehler interpretiert.

5. Es ist darauf zu achten, daß genügend END-Anweisungen für den Abschluß einer DO-Gruppe, einer Prozedur und eines BEGIN-Blockes vorhanden sind. Dies gilt besonders dann, wenn die Form "END Mark;" gewählt wird: Auf Grund der hierfür festgelegten Regeln kann der Übersetzer nicht in jedem Fall erkennen, ob eine END-Anweisung zu wenig ist.
6. In einigen Zusammenhängen ist es nicht ohne weiteres einsichtig, daß eine Angabe in Klammern eingeschlossen werden muß. Das gilt besonders für Ausdrücke hinter den Schlüsselwörtern WHILE und RETURN.

8.6.2 Programm-Steuerung

Die Haupt-Prozedur, das ist die Prozedur, bei der das Programm gestartet werden soll, muß die Angabe

```
PROCEDURE OPTIONS (MAIN)
```

enthalten.

Haben mehrere externe Prozeduren die Angabe MAIN, so wird die vom Binder als erste gefundene Prozedur dafür verwendet.

8.6.3 Vereinbarungen und Attribute

1. Für Variable mit dem Attribut AUTOMATIC wird der Speicherplatz beim Eintritt in den Block zugewiesen. Werden dabei für die notwendigen Berechnungen Variable verwendet, so müssen deren Werte bereits vor dem Eintritt in den Block feststehen.

   Beispiel

   ```
   Name:  PROCEDURE;
          Länge = 4;
          DCL Folge CHAR (Länge);
   ```

   Dieses Beispiel führt zum Fehler, da der Speicherplatz zugeordnet wird, bevor die Zuweisung ausgeführt ist.

2. In DECLARE-Anweisungen sind fehlende Kommas eine Quelle von Fehlern. So muß z.B. in einer Struktur-Vereinbarung jedes Strukturglied mit einem Komma abgeschlossen werden.

3. Externe Namen (EXTERNAL) sollten nicht länger als 7 Zeichen sein, andernfalls werden sie auf die ersten 4 und die letzten 3 Zeichen gekürzt.

4. In einer Vereinbarung mit dem Attribut PICTURE gibt das Maskenzeichen V die Trennung an zwischen ganzzahligem und gebrochenen Teil. Er belegt keine Zeichenposition und erzeugt bei der Angabe auch nicht das Trennzeichen Punkt oder Komma. Dies ist nur durch das Maskenzeichen Punkt (.) bzw. Komma (,) möglich. Diese beiden Maskenzeichen wiederum sind reine Einfügungszeichen und sagen nichts darüber aus, welche Ziffernstellen zum ganzzahligen Teil gehören und welche zum gebrochenen Teil.

   Beispiel

   ```
   DCL  A  PIC '99,9',
        B  PIC '99V9',
        C  PIC '99,V9';
   A,B,C = 45.6;
   PUT LIST (A,B,C);
   ```

   Auf Grund des obigen Beispiels werden folgende Zahlen ausgegeben:

   ```
   04,5      456    45,6
   ```

   Werden diese Werte wieder durch GET LIST (A,B,C) eingelesen, so werden in den Variablen A,B und C stehen

   ```
   Zeichenfolge    '04,5'    '560'    '45,6'
   Wert               45       56      45.6
   ```

Werden wiederum durch PUT LIST (A,B,C) die Werte der Variablen ausgegeben, so erhält man die Ausgabe

```
04,5        560      45,6
```

5. Verschiedene Vereinbarungen für den gleichen Bezeichner mit dem Attribut EXTERNAL müssen nach der Ergänzung durch die Vorgabe-Regeln den gleichen Attributsatz ergeben, da sonst Fehler entstehen können. Die Übereinstimmung kann vom Übersetzer nicht geprüft werden.

   Ein INITIAL-Attribut braucht nur einmal angegeben zu werden. Wird es mehrmals angegeben, so müssen alle gleiche Werte ergeben.

6. Ein Bezeichner kann innerhalb seines Bedeutungsbereichs nur für einen einzigen Zweck verwendet werden. Das nachstehende Beispiel ist fehlerhaft, da der Bezeichner X mit unterschiedlicher Bedeutung verwendet wird.

```
   PUT  FILE (X) LIST (A)   X ist Datei
   X = Y + Z;               X ist Variable
X: M = N;                   X ist Marke
```

7. Werden Konstanten als Parameter an externe Prozeduren übergeben, so muß auf die Genauigkeit geachtet werden.

Beispiel

```
           DCL  Prozedur ENTRY EXTERNAL;
           CALL Prozedur (6);      Genauigkeit (1,0)
Prozedur:  PROCEDURE (x);
           DCL x FIXED DECIMAL;    Genauigkeit (5,0)
```

Aktual- und Formal-Parameter haben hier unterschiedliche Genauigkeit.

8. Ein Formal-Parameter wird abhängig von bestimmten Bedingungen entweder dem Aktual-Parameter überlagert (Übergabe durch Bezug) oder er erhält einen eigenen Hilfs-Speicherplatz, dem der Wert des Aktual-Parameters zugewiesen wird (Übergabe durch Wert). Im letzteren Fall wird durch eine Änderung des Formal-Parameters der Aktual-Parameter nicht geändert. Eine Rückgabe eines Wertes ist in diesem Fall nicht möglich.

Beispiel

```
     DCL  A    FIXED BIN,
          B    FLOAT;
     CALL P    (A,B);
P:   PROCEDURE  (X,Y);
     DCL (X,Y) FIXED BIN;
     X = 3;                /* A wird verändert */
     Y = 4;                /* B wird nicht verändert*/
```

9. Wenn für einen Bezeichner bei der Vereinbarung nicht alle Attribute angegeben werden, so werden die fehlenden gemäß den Vorgabe-Regeln ergänzt. Dabei sollte auf folgende Regeln besonders geachtet werden.

   REAL FLOAT DECIMAL (6) wird für die arithmetische Variable ergänzt, sofern der Bezeichner nicht mit einem Buchstaben von I bis N beginnt oder gemäß PL/I-Norm übersetzt wird. In diesen Fällen wird FIXED BINARY (15,0) ergänzt.

   Wird eines der Attribute angegeben, die den Datentyp bestimmen, so wird ein fehlendes Attribut aus dem Satz REAL/FLOAT/DECIMAL ergänzt und wenn gemäß PL/I-Norm übersetzt wird aus dem Satz REAL/FIXED/BINARY.

   Dazu folgende Beispiele (NOISO):

```
DCL I;            ergänzt:  REAL FIXED BINARY  (15,0)
DCL J REAL;       ergänzt:  REAL FLOAT DECIMAL (6)
DCL K STATIC;     ergänzt:  REAL FIXED BINARY  (15,0)
DCL L FIXED;      ergänzt:  REAL FIXED DECIMAL ( 5,0)
```

10. Die Genauigkeit eines komplexen Ausdruckes ist nicht ohne weiteres einleuchtend. Sie folgt den Regeln für die Auswertung eines Ausdruckes. So sind z.B. für den Ausdruck

```
1 * 2I
```

    die Attribute COMPLEX DECIMAL PRECISION (2,0).

11. Wenn eine Prozedur mehrere Eingänge hat, die unterschiedliche Parameter haben, so ist sicherzustellen, daß nur auf solche Parameter zugegriffen wird, die zum aktuellen Eingang gehören.

    Beispiel

```
A:  PROCEDURE (P,Q);
    P = Q + 8; RETURN;
B:  ENTRY (R,S);
    R = P + S;
    END;
```

    In der Zuweisung R = P + S ist der Bezug auf P falsch, da P beim Eintritt in B keinen definierten Wert besitzt.

8.6.4 Zuweisung und Initialisierung

1. Wenn auf eine Variable zugegriffen wird, so wird erwartet, daß sie einen Wert besitzt, der mit dem Attributsatz der Variablen in Übereinstimmung ist. Wenn dies nicht der Fall ist, so kann entweder das Programm mit einem fehlerhaften Wert weiterlaufen oder aber es wird eine der Bedingungen gesetzt. Die Ursache für einen solchen fehlerhaften Wert kann sein, daß der Variablen bisher noch kein Wert zugewiesen wurde, ihr Wert also undefiniert ist, oder aber daß der Variablen durch eine der nachfolgenden Aktionen ein falscher Wert zugewiesen wurde. So z.B.

   a) durch die eingebaute Funktion UNSPEC

   b) durch Eingabe eines Datensatzes (RECORD)

   c) durch Überlagerung einer maskierten Zeichenfolge (PICTURE) über eine Zeichenfolge (CHAR) mit einer Zuweisung an die Zeichenfolge und nachfolgende Zugriffe über die maskierte Zeichenfolge.

   d) Übergabe eines Parameters an eine andere externe Prozedur ohne daß die Attributsätze von Aktual- und Formal-Paramter übereinstimmen.

   e) Zuweisung eines Wertes an eine BASED-Variable und Zugriff zu diesem Wert über eine BASED-Variable mit anderem Attributsatz.

   Wird einer Variablen kein Wert zugewiesen, so ist ihr Wert undefiniert und auch der weitere Programmablauf. Man kann nicht annehmen, daß die Variable in diesem Fall den Wert 0 hat.

   Wird ein Index verwendet, dessen Wert undefiniert ist, so kann dieser Fehler dadurch entdeckt werden, daß die Bedingung SUBSCRIPTRANGE wirksam gesetzt wird, vorausgesetzt der Index liegt nicht zufällig im vereinbarten Bereich.

2. Der Versuch, eine Variable mit undefiniertem Wert auszugeben, kann zu einer Unterbrechung führen. In dem Beispiel

   ```
   DCL A   DIM(10)  FIXED DECIMAL;
   A(1) = 13;
   PUT LIST (A);
   ```

   führt zu einem Fehler. Dies kann vermieden werden, wenn die Matrix mit dem Wert 0 vorbesetzt wird, z.B. durch

   ```
   A = 0;
   ```

3. Man beachte den Unterschied zwischen dem Zuweisungszeichen = und dem Vergleichszeichen =.

   Die Anweisung

   ```
   A = B = C;
   ```

   bedeutet, daß die Variablen B und C auf Gleichheit verglichen werden und das Ergebnis ('0'B oder '1'B) der Variablen A zugewiesen wird.

4. Wird bei der Initialisierung oder Zuweisung einer Folgen-Variablen fester Länge eine Folge zugewiesen, die kürzer ist als das Ziel, so werden bei CHAR rechts Leerzeichen und bei BIT rechts Bits mit dem Wert '0'B hinzugefügt. Nach der Zuweisung in dem Beispiel

   ```
   DCL A   CHAR(6),
       B   CHAR(3)  INIT ('CR');
   A = B;
   ```

   enthalten die Variablen folgende Werte:

   ```
   A = 'CR␣␣␣␣' und B = 'CR␣'
   ```

5. Wenn die Bedingung SIZE unwirksam ist und es tritt eine solche Bedingung auf, so ist das Ergebnis unbestimmt.

   FIXED DECIMAL: Es können Bytes unterdrückt werden, ohne daß eine Unterbrechung erfolgt. Falls die Zielgenauigkeit gradzahlig ist, kann an höchster Stelle ein Byte eingefügt werden.

8.6.5 Arithmetische und Boolesche Ausdrücke und Konvertierungen

1. Die Regeln für die Auswertung von Ausdrücken sollten genau beachtet werden und dabei speziell die Reihenfolge der Operationen. Die folgenden Beispiele zeigen die Art der Fehler, die dabei auftreten können.

```
X > Y | Z      entspricht      (X > Y) | Z
               entspricht nicht X > Y | X > Z

X > Y > Z      entspricht      (X > Y) > Z
               entspricht nicht X > Y & Y > Z
```

   Alle Operationen gleicher Priorität werden von links nach rechts ausgeführt mit Ausnahmen von **, Vorzeichen +, Vorzeichen - und, die von rechts nach links ausgeführt werden.

   Das Beispiel

```
A = B ** - C ** D;
```

   ist gleichwertig mit

```
A=B**(-(C**D))
```

   Die Aufgabe der Klammern ist es, die vorgegebenen Regeln zu modifizieren. Es kann jedoch günstig sein, redundante Klammern hinzuzufügen um sicher zu gehen oder um die Lesbarbeit zu erhöhen.

2. Für die Konvertierung gibt es umfassende Regeln. Um unnötigen Ärger zu vermeiden, sollten sie genauestens studiert werden. Nachstehend ein Beispiel dazu:

   a) DECIMAL FIXED nach BINARY FIXED kann unerwartete Ergebnisse liefern, wenn gebrochene Zahlen betroffen sind:

```
DCL  I FIXED BIN(31,5) INIT(1);
I = I + 0.1;
```

      Der Wert von I ist danach 1.0625, weil 0.1 nach FIXED BINARY(5,4) konvertiert wird und danach den dualen Wert 0.0001B hat (ohne Bedeutung). Dieser ergibt den dezimalen Wert 0.0625. Diesen Rundungsfehler kann man dadurch verringern, daß man für die Konstante mehr Stellen angibt (0.1000).

   b) Wenn arithmetische Operationen mit Zeichenfolgen durchgeführt werden, so werden die Zwischenwerte in Hilfs-Variablen mit den Attributen FIXED DECIMAL (15,0) gehalten, also ohne gebrochene Stellen. In dem Beispiel

```
DCL  A  CHAR(6) INIT('123.45'),
     B  FIXED(5,2);

B = A;        /* Wert von B 123.45 */
B = A + A;    /* Wert von B 246.00 */
```

      werden also bei der zweiten Zuweisung die gebrochenen Stellen unterdrückt.

c) Die Regeln für die Konvertierung eines arithmetischen Wertes nach BIT beeinflußt die Zuweisung einer dezimalen Konstante an eine Bit-Variable.

Beispiel

```
DCL  A  BIT (1),
     D  BIT (5);

A = 1;          /* Wert von A:  '0'B     */
D = 1;          /* Wert von D:  '00010'B */
D = '1'B;       /* Wert von D:  '10000'B */
```

d) Bei der Zuweisung eines arithmetischen Wertes an eine Zeichenfolge (CHAR) ergeben die Konvertierungsregeln manchmal unerwartete Ergebnisse.

Beispiel 1

```
DCL  A  CHAR(4),
     B  CHAR(7);

A = '0';          /* Wert von A: '0␣␣␣'    */
A = 0;            /* Wert von A: '␣␣␣0'    */
B = 1234567;      /* Wert von B: '␣␣␣1234' */
A = -0.7;         /* Wert von A: '-0.7'    */
```

Die drei Zeichenstellen bei der zweiten und dritten Zuweisung werden nur dann mit Zeichen belegt, wenn ein negatives Vorzeichen, ein Dezimalpunkt und eine einzelne Null vor dem Dezimalpunkt vorhanden sind (siehe vierte Zuweisung).

Beispiel 2

```
DCL    Nummer CHAR(8) INIT('0');
DO I = 1 TO 100;
   Nummer = Nummer + 1;
   END;
```

In dem vorstehenden Beispiel entsteht ein Konvertierungsfehler auf Grund der nachstehenden Funktionen, die für die Zuweisung ausgeführt werden:

- Der Initialwert '0␣␣␣␣␣␣␣' wird nach FIXED DECIMAL (15,0) konvertiert und hat den Wert 0.
- Die dezimale Konstante 1 hat die Attribute FIXED DECIMAL (1,0).
- Bei der Addition entsteht eine Hilfs-Variable mit dem Attribut FIXED DECIMAL (15,0) mit dem Wert 1.
- Dieser Wert wird nach CHAR (18) konvertiert; die Zeichenfolge besteht dann aus 17 Leerzeichen und der Ziffer 1.

– Da die Zielvariable das Attribut CHAR (8) hat, werden nur die ersten 8 Zeichen zugewiesen. Dies sind 8 Leerzeichen, die bei der Konvertierung beim nächsten Durchlauf (siehe oben) zu einem Konvertierungsfehler führen.

e) Die Division kann bei FIXED-Größen zu einem unerwarteten Abschneiden relevanter Ziffern führen oder zur Bedingung FIXEDOVERFLOW.

Beispiel

```
25 + 1/3
```

Dieser Ausdruck wird wie folgt ausgewertet:

```
Operand           Genauigkeit           Ergebnis

   1                 (1,0)                 1
   3                 (1,0)                 3
  1/3               (15,14)                0.333...
  25                 (2,0)                25
 25 + 1/3           (15,14)               2 5.333...
```

Wegen der auf Grund der Regeln errechneten Genauigkeit wird die Ziffer 2 abgeschnitten. Ist die Bedingung FIXEDOVERFLOW wirksam, so wird sie gesetzt.

Hier ist es notwendig den ganzzahligen Teil des Ergebnisses der Division zu erhöhen. Dies kann z.B. erreicht werden durch

```
25 + 01/3
25 + PREC (1/3, 15,13)
25 + DIVIDE (1, 3, 15, 13)
```

oder wenn man das Ergebnis der Division einer eigenen Variablen zuweist, die mit der gewünschten Genauigkeit vereinbart ist.

f) Der Wert einer maskierten Zeichenfolge (PICTURE) wird nur bei der Zuweisung auf Richtigkeit geprüft.

Beispiel

```
DCL A  PIC '999999',
    B  CHAR(6)      DEFINED A,
    C  CHAR(6);

B = 'ABCDEF';          /* Wert auch in A   */
C = A;                 /* keine CONVERSION */
A = C;                 /* CONVERSION       */

A = 123456;            /* Wert von A: 123456
                       /* Wert von B: '123456'  */
C = 123456;            /* Wert von C: '␣␣␣123' */
C = A;                 /* Wert von C: '123456'  */
```

g) Eine Größe mit den Attributen FIXED DECIMAL PRECISION (g,k) hat eine interne Darstellung mit der Genauigkeit g + 1, wenn g gradzahlig ist. Möchte man sicher gehen, daß die Genauigkeit g in diesem Fall nicht überschritten wird, so kann die Bedingung SIZE wirksam gesetzt werden.

Beispiel

```
DCL  (A,B,C) FIXED DECIMAL PRECISION (6,0) INIT (500000);

(SIZE): A = B + C;
```

Es tritt die SIZE-Bedingung auf. Der Wert von A ist jedoch korrekt (1 000 000), da DECIMAL FIXED (6,0) intern die gleiche Darstellung hat wie DECIMAL FIXED (7,0).

8.6.6 DO-Gruppen

1. Wird vor einer DO-Gruppe ein Bedingungs-Vorspann angegeben, so gilt dieser nur für die DO-Anweisung, nicht aber für die gesamte DO-Gruppe.
2. Ist die Bedingung für das Verlassen der DO-Gruppe bereits beim ersten Eintritt in die DO-Gruppe erfüllt, wird die DO-Gruppe nicht ausgeführt.

   Beispiel

   ```
   I = 6;
   DO J = I TO 4;
      X = X + J;
      END;
   ```

   In diesem Beispiel wird die Schleife nicht durchlaufen, weil die Laufvariable bereits beim ersten Schritt in die DO-Gruppe den Wert 6 und damit den Endwert 4 bereits überschritten hat.
3. Ausdrücke in einer DO-Anweisung werden in einer Hilfs-Variablen abgelegt. Ihre Darstellung ergibt sich aus den Regeln für die Auswertung des Ausdruckes. Beim Vergleich mit der Lauf-Variablen sind dann ggf. weitere Konvertierungen nötig.

   Beispiel

   ```
   DCL A  DECIMAL FIXED (5,0);
   A = 10;
   DO I = 1 TO A/2;
      PUT LIST (I);
      END;
   ```

   Die Schleife im vorstehenden Beispiel ist fehlerhaft, wie sich aus den folgenden Funktionen ergibt.

   ```
   Ausdruck            Attribut            Wert

    A                  DEC  (5,0)           10
    A/2                DEC (15,10)           5
    A/2 konvertiert    BIN (31,34)         SIZE-Bedingung
   ```

   Für den Vergleich mit der Lauf-Variablen I muß der Wert A/2 nach BINARY konvertiert werden. Dabei tritt die Bedingung SIZE auf. Ist sie nicht wirksam, wird mit einem undefinierten Wert weitergearbeitet.

   Die Schleife wird 5 mal durchlaufen, wenn man sie wie folgt abändert:

   ```
        Hilf = A/2
        DO I = 1 TO Hilf;
   oder
        DO I = 1 TO PREC (A/2,5,0);
   ```

4. Eine DO-Gruppe kann nicht als ON-Einheit verwendet werden. Sollen für eine ON-Einheit mehr als eine Anweisung angegeben werden, so muß ein BEGIN-Block verwendet werden. In diesem kann eine DO-Gruppe enthalten sein.

5. In der DO-Anweisung

```
DO x = a BY b TO c
```

werden die Ausdrücke a,b und c nur beim Eintritt in die DO-Gruppe berechnet und die ermittelten Werte in einer Hilfs-Variablen abgelegt. Werden Variable, die in diesen Ausdrücken stehen, innerhalb der DO-Gruppe verändert, so hat dies keinen Einfluß auf die Werte von a,b und c.

Beispiel

```
Schritt = 1;
Ende    = 5;
DO  I   = 1 BY Schritt TO Ende;
    Schritt = 100;
    Ende    =   0;
    END;
```

Diese Schleife wird genau 5 mal durchlaufen. Dagegen wird die Schleife im folgenden Beispiel nur einmal durchlaufen.

```
DO I= 1 BY 1 TO 5;
   I= 5;
   END;
```

6. Eine DO-Gruppe mit einer Lauf-Angabe und einer WHILE-Angabe wird nur dann mehrmals durchlaufen, wenn dies aus der Lauf-Angabe zu entnehmen ist. So wird in dem Beispiel

```
DO I  = 1 WHILE (X > Y);
   .
   .
   .
   END;
```

die Schleife nur einmal durchlaufen, wenn die Bedingung X > Y erfüllt ist; im anderen Fall wird sie garnicht durchlaufen.

7. Der Bezeichner I wird sehr häufig ohne Vereinbarung als Lauf-Variable in DO-Schleifen verwendet.

   Beispiel

   ```
   DO I    = 1 TO 10;
   ```

   Innerhalb des Geltungsbereichs der Variablen I könnte implizit eine andere Variable den gleichen Namen erhalten.

   Beispiel

   ```
   DCL X BASED (I);
   ```

   Diese beiden Anweisungen stehen im Gegensatz zueinander und es wird eine Fehlermeldung geben. Wenn I ein Zeiger ist, so kann er in einer DO-Gruppe nur in einer der folgenden Arten verwendet werden:

   ```
   a) DCL  (I, IA, IB, IC) POINTER;
      .
      .
      .
      DO I  = IA, IB, IC;

   b) DCL  (I, IA) POINTER;
      .
      .
      .
      DO WHILE (I = IA);
   ```

8. Wenn die Lauf-Variable einer DO-Gruppe als Index verwendet wird, so muß darauf geachtet werden, daß ihr Wert nicht die Grenzen der Matrix überschreitet.

   Beispiel

   ```
   DCL A  DIM(10);
   DO I = 1 TO N;
      A(I) = X;
      END;
   ```

   Wird N größer als 10, kann durch die Zuweisung der Wert anderer Variablen zerstört werden. Solch ein Fehler ist schwer zu finden, besonders wenn dabei Objektcode zerstört wird. Der Fehler kann dadurch festgestellt werden, daß die Bedingung SUBSCRIPTRANGE wirksam gesetzt wird.

8.6.7 Datenverbund

1. Werden in einer Zuweisung Matrix-Ausdrücke verwendet und wird ein Element der Zielgröße auch auf der rechten Seite der Zuweisung verwendet, so ist zu beachten, daß gemäß der PL/I-Norm zuerst alle Elemente der rechten Seite berechnet werden und der dabei ermittelte Wert einer Hilfs-Variablen zugewiesen wird. Danach erst wird der Wert der Hilfs-Variablen der Ziel-Variablen zugewiesen.

   Beispiel

   ```
   DCL A DIM (10,20);
   A = A + A (1,1);
   ```

   Die Wirkung entspricht folgenden Anweisungen

   ```
   DCL A    DIM (10,20),
       Hilf DIM (10,20);
   Hilf = A + A (1,1);
   A = Hilf;
   ```

   Damit wird allen Elementen der Matrix A der Wert des Elementes A(1,1) zugewiesen.

   Es sei darauf hingewiesen, daß das folgende Programmstück ein anderes Ergebnis hat:

   ```
   DCL A  DIM(10,20);
   DO  I = 1 TO 10;
       DO J = 1 TO 20;
          A (I,J) = A (I,J) + A (1,1);
          END;
       END;
   ```

   Hier wird der Wert des Elements A(1,1) verdoppelt und danach dieser verdoppelte Wert zum Wert der anderen Elemente (außer A(1,1)) addiert.

   Wenn der Übersetzer erkennen kann, daß eine solche Überlappung zwischen den beiden Seiten der Zuweisung nicht besteht, wird auch keine Hilfs-Variable angelegt und es wird direkt in die Ziel-Variable abgelegt.

2. Bei der Multiplikation zweier Matrizen wird die Multiplikation nur auf die Elemente mit den gleichen Indizes angewendet.

   Beispiel

   ```
   DCL (A,B,C)  Dimension(10,10);
   A = B * C;
   ```

   Dies entspricht folgenden Anweisungen:

   ```
   DO  I = 1 TO 10;
       DO J = 1 TO 10;
          A(I,J) =B(I,J) * C(I,J);
          END
       END;
   ```

8.6.8 Folgen

1. Wird einer Folgen-Variablen mit dem Attribut VARYING ein Wert mit Hilfe der Pseudo-Variablen SUBSTR zugewiesen, so wird dadurch die Länge der in der Variablen abgelegten Folge nicht beeinflußt. Ist die Länge der Variablen undefiniert oder wird auf Teile der Variablen außerhalb der definierten Länge referiert, so ist das Ergebnis undefiniert. Ein solcher Fehler wird erkannt, wenn die Bedingung STRINGRANGE wirksam ist.
2. Es ist darauf zu achten, daß auch bei Zwischenergebnissen die Länge einer Bit- oder Zeichenfolge nicht länger als 32767 Zeichen bzw. Bits wird. Hierzu gibt es im Objektprogramm keine Abprüfung.

8.6.9 Funktionen und Pseudo-Variable

Wenn bei einer Zuweisung als Ziel-Variable die Pseudo-Variable UNSPEC verwendet wird, so hat diese das Attribut BIT und der rechts stehende Ausdruck wird nach BIT konvertiert. Es ist darauf zu achten, daß diese Konvertierung möglich ist.

8.6.10 Bedingungen und ON-Einheiten

1. Man beachte die richtige Stellung für die On-Anweisung. Soll beim Auftreten einer bestimmten Bedingung eine ON-Einheit ausgeführt werden, so muß die ON-Anweisung dann zeitlich vor dem Auftreten der Bedingung ausgeführt sein. In dem Beispiel

   ```
   GET FILE (Datei) LIST (a,b,c);
   ON ENDFILE (Datei)  GOTO Dateiende;
   ```

   würde der Lauf des Programms mit Fehler beendet, wenn die Datei leer wäre; der Sprung nach "Dateiende" würde nicht durchgeführt, da die ON-Anweisung beim ersten Zugriff noch nicht ausgeführt wurde. Ferner wird hier jedesmal nachdem die GET-Anweisung ausgeführt wird, die ON-Anweisung erneut ausgeführt, stets mit dem gleichen Ergebnis.

2. Eine ON-Einheit wird entweder beim Auftreten der zugehörigen Bedingung ausgeführt oder auf Grund einer SIGNAL-Anweisung. Ein Sprung in eine ON-Einheit ist nicht möglich.

3. ON-Einheiten für die Bedingung CONVERSION sollten entweder durch eine GOTO-Anweisung verlassen oder es sollten die falschen Zeichen durch die Pseudo-Variablen ONSOURCE oder ONCHAR korrigiert werden. Diese Forderung gilt nicht, wenn die ON-Einheit durch SIGNAL aufgerufen wurde.

4. Bei der normalen Rückkehr aus einer ON-Einheit zur AREA-Bedingung wird die Zuordnung erneut versucht, was zu einer unendlichen Schleife führt, wenn man in der ON-Einheit keine Vorkehrung trifft.
   Die ON-Einheit kann die Tatsache ausnutzen, daß bei der erneuten Zuordnung der Umfang der Speicheranforderung, sowie der Bezug auf den Bereich neu berechnet werden und diesen verändern, oder sie kann für genügend Platz im ursprünglichen Bereich sorgen.

5. ON-Einheiten sollten nicht als Ersatz für Programmteile dienen. Man sollte sie vielmehr nur dort verwenden, wo man eine Ausnahme-Bedingung abfangen möchte, da der Aufruf von ON-Einheiten sehr aufwendig ist. Für die anderen Fälle sollten entsprechende Prüfungen programmiert werden und man sollte nicht die Fehlererkennung des PL/I-Systems dazu verwenden.

   Wenn z.B. in einem Programm Schlüssel für den Zugriff auf eine Datei verwendet werden und einige der Schlüssel ausgelassen werden sollen, weil sie nicht in die Grenzen der Datei passen, könnte man dies über die KEY-Bedingung lösen. Es ist jedoch günstiger die Schlüssel durch entsprechende Abprüfungen zu ermitteln und in einem solchen Fall keinen Zugriff durchzuführen.

8.6.11 Ein-Ausgabe

1. Die Bedingung UNDEFINEDFILE wird nicht nur gesetzt, wenn die durch PL/I gegebenen Attribute unverträglich sind, sondern auch in den folgenden Fällen:

   a) Blocklänge kleiner als Datensatzlänge.

   b) KEYLENGTH Null oder undefiniert für INDEXED- und REGIONAL-Dateien.

   c) Wenn für INDEXED-Dateien die Summe der Werte von KEYLENGTH und KEYLOC größer ist als die Datensatzlänge.

   d) Wenn für das Datensatz-Format VB die logische Datensatzlänge nicht wenigstens um 4 Bytes kleiner ist als die Blocklänge.

2. Wenn eine Datei sowohl für die Eingabe als auch für die Ausgabe benutzt wird, so darf sie nicht mit dem Attribut INPUT oder OUTPUT vereinbart werden. Dies kann bei der Anweisung OPEN angegeben werden.

3. Ein- und Ausgabelisten müssen in runde Klammern eingeschlossen werden. Das gleiche gilt für die Wiederholungsangabe. Somit müssen in dem folgenden Beispiel zwei Paar äußere Klammern verwendet werden:

   ```
   GET LIST ((A(I) DO I = 1 TO N));
   ```

4. Die letzten 8 Bytes des Schlüssels mit dem eine REGIONAL-Datei angesprochen wird, muß eine Zeichenfolge sein, die eine ganze Dezimalzahl darstellt. Wenn dieser Schlüssel erstellt wird, sollten die Regeln für die Konvertierung von arithmetic nach CHAR beachtet werden. So ist z.B. der nachstehende Programmausschnitt fehlerhaft.

   ```
   DCL Schlüssel  CHAR (8);
   DO I = 1 TO 10;
      Schlüssel = I;
      WRITE FILE (Datei) FROM (Variable) KEYFROM (Schlüssel);
      END;
   ```

   Die Vorgabe für I ist FIXED BINARY (15,0). Bei der Konvertierung entsteht daraus eine Zeichenfolge von 9 Zeichen. Entsprechend wird bei der Zuweisung zu "Schlüssel" die rechte Ziffer abgeschnitten..

5. Wird in einer Ein-Ausgabe-Anweisung eine der Angaben KEY, KEYFROM oder KEYTO gemacht, so muß die zugehörige Datei das Attribut KEYED haben.

6. Die Namen SYSIN und SYSPRINT werden nur bei der Anweisung GET bzw. PUT impliziert. In allen anderen Fällen, wie z.B. bei der ON-Anweisung und bei anderen Ein-Ausgabe-Anweisungen müssen diese Namen explizit angegeben werden.

7. PAGESIZE und LINESIZE sind keine Datei-Attribute, d.h. sie können nicht bei Vereinbarung einer Datei sondern nur bei OPEN-Anweisung verwendet werden.

8. Wenn bei GET EDIT oder PUT EDIT alle Elemente einer Datenliste abgearbeitet sind, werden keine weiteren Elemente der Formatliste mehr bearbeitet, auch wenn diese kein Datenelement benötigen.

   Beispiel

   ```
   GET EDIT (A,B) (F(5), F(5), X(70));
   PUT EDIT (A,B) (A(3), F(5), SKIP);
   ```

   Das Format X(70) bleibt unbearbeitet. Will man z.B. die restlichen 70 Stellen einer Lochkarte überspringen, so muß man bei der nächsten Anweisung X(70) oder SKIP voranstellen. Bei der PUT-Anweisung wird das SKIP-Format nicht ausgeführt. Es muß in der nächsten PUT-Anweisung stehen.

9. Wird in einer Datenliste eine Matrix oder eine Struktur angegeben, so bildet jedes Element der Matrix und jedes Elementarglied der Struktur ein Element der Datenliste und gehört damit auch zu einem Element der Formatliste.

   Beispiel

   ```
   DCL 1   A,
           2   B   CHAR(5),
           2   C   FIXED(5,2);
   PUT EDIT (A)    (A(5), F(5,2));
   ```

   Zum Elementarglied B gehört das Format A(5) und zu C das Format F(5,2).

10. Matrix-Elemente werden in der Reihenfolge bearbeitet, in der sich der rechte Index am schnellsten ändert:

    ```
    A(1,1), A(1,2), A(1,3), A(2,1) usw.
    ```

11. Zeichenfolgen, die durch GET LIST oder GET DATA eingelesen werden, müssen in Apostrophe eingeschlossen werden.

12. Die Darstellung des Semikolons im 48-Zeichen-Satz (.,) wird bei der Eingabe durch GET DATA nicht als Trennzeichen Semikolon anerkannt.

13. Wird die Anweisung PUT DATA ohne eine Datenliste verwendet, so werden alle die Datennamen ergänzt, die an der Stelle der Niederschrift auf Grund der Regeln für die Gültigkeit von Namen bekannt sind. Dieser Gültigkeitsbereich wird statisch ermittelt.

    Verwendet man PUT DATA in einer ON-Einheit, so gilt das gleiche, nur wird die ON-Einheit dynamisch aufgerufen. Wenn die Stelle, an der die ON-Einheit aufgerufen wird, in einem Block parallel und untergeordnet zur ON-Einheit ist, so werden Daten, die nur in diesem Block gültig sind, nicht mit ausgegeben.

Für die Testphase kann es daher z.B. von Vorteil sein, in allen inneren Blöcken die Anweisung

```
ON ERROR PUT DATA;
```

am Anfang zu wiederholen.

Achtung

Tritt in einer ON-Einheit für die Bedingung ERROR wiederum die Bedingung ERROR auf, so entsteht eine Endlosschleife.

Mit "PUT DATA;" werden auch solche Variable ausgegeben, die noch keinen definierten Wert haben. Das kann dazu führen, daß die Bedingung CONVERSION gesetzt wird und ggf. als Folge davon die Bedingung ERROR.

Um eine Endlosschleife zu vermeiden, können folgende Anweisungen verwendet werden:

```
ON ERROR  SNAP BEGIN;
            ON ERROR SNAP SYSTEM;
            PUT DATA;
            END;
```

14. Ein Zeiger, der durch READ SET oder LOCATE gesetzt wurde, hat nur bis zur nächsten Anweisung auf die gleiche Datei Gültigkeit. In Ausgabe-Dateien können die Anweisungen WRITE und LOCATE gemischt verwendet werden.

8.6.12 Funktions-Prozeduren mit mehreren Eingängen

Bei einem Funktions-Aufruf kann eine Prozedur über alle Eingänge betreten werden, die eine RETURNS-Angabe haben und über alle RETURN-Anweisungen verlassen werden, bei denen ein Wert angegeben ist. Bei n Eingängen mit RETURNS-Angabe und m RETURN-Anweisungen gibt es theoretisch n mal m Möglichkeiten den Wert des Ausdruckes bei RETURN nach dem Attributsatz gemäß RETURNS-Angabe zu konvertieren. Welche der Konvertierungen notwendig sein wird, hängt vom Programmablauf ab und kann vom Übersetzer nicht festgestellt werden. Dies wird erst bei der Ausführung der RETURN-Anweisungen festgestellt, d.h. dynamisch. Bei Verwendung von mehreren RETURNS-Angaben mit unterschiedlichen Attributsätzen der Ausdrücke wird deshalb für alle möglichen Fälle Vorsorge getroffen. In diesem Fall wird daher der Speicherbedarf und die Rechenzeit merklich ansteigen.

```
                                                                  Ergebnisse
PUT SKIP LIST ( ZEICHEN ()    );                                      3
PUT SKIP LIST ( FESTPUNKT () );                                      3,0
PUT SKIP LIST ( BIT ()        );                                    '0011'B

ZEICHEN:    PROCEDURE  RETURNS ( CHAR (*)  );
            .
            .                                                    3 Eingänge
            .
            RETURN ('7');
FESTPUNKT:  ENTRY      RETURNS ( FIXED(5,2) );                   3 Rückkehr-
            .                                                      punkte
            .
            .
            .
            RETURN (7);                                          = 3 x 3 Kon-
BIT:        ENTRY      RETURNS ( BIT(*) );                       vertierungen
            .
            .
            .                                                     dynamische
            RETURN ('111'B);                                      Entscheidung
            END ZEICHEN;                                          über Art der
                                                                  Konvertierung
```

Attributsatz RETURN-Anweisung		theoretisch erforderliche Konver-tierungen	Attributsatz RETURNS-Angabe
'7'	CHAR (1) NONVARYING		CHAR (*) NONVARYING
7	REAL FIXED DEC (1,0)		REAL FIXED BIN (5,2)
'111'B	BIT (3) NONVARYING		BIT (*) NONVARYING

Bild 8-4 Prozedur mit mehreren Eingängen mit RETURNS-Angabe und mehreren RETURN-Anweisungen mit Ausdruck

8.6.13 Variable Längenangabe

Werden bei der Vereinbarung von Längen (AREA, BIT, CHAR) oder Dimensionen (DIMENSION) keine konstanten Werte, sondern variable Werte oder Ausdrücke verwendet, so kann der Zugriff auf solche Größen, besonders in Verbindung mit Strukturen und Matrizen, aufwendig werden. Dort, wo es möglich ist, sollten sie vermieden werden. Gegebenenfalls ist zu prüfen, ob die gewünschte Wirkung nicht auch über Vorübersetzer-Variable erreicht werden kann, die dann wie Konstante wirken können.

8.6.14 Parameter-Übergabe

Erfüllt ein Parameter die Bedingungen:

- nur Wertübergabe an die gerufene Prozedur und nicht zurück an die rufende Prozedur
- skalar
- keine *-Angabe
- Speicherbedarf ≤ 1 Ganzwort

so wird durch die Attribut-Angabe PARAMETER (INPUT) eine günstige Übergabe des Wertes erreicht. Statt des Zeigers auf den Wert wird dann der Wert selbst übergeben (siehe Abschnitt 7.1.2.1).

8.6.15 Absolut-Bitzeiger bei XS

Soll der Adressenraum oberhalb von 16 M-Bytes (XS) benutzt werden, so sind bei Absolut-Zeigern, die Bit-genau vorweisen sollen, einige Besonderheiten zu beachten. Diese Absolut-Bitzeiger können in folgenden Fällen entstehen:

- bei ADDR auf eine Bitfolge
- bei Übergabe einer Bitfolge als Parameter

Dabei trifft aber ein Absolut-Bitzeiger wegen der allgemeinen Ausrichtungsregeln nur dann auf, wenn alle nachfolgenden Bedingungen gleichzeitig erfüllt sind:

- Vereinbarung einer Bitfolge
- Vereinbarung innerhalb einer Struktur
- es geht innerhalb der Struktur unmittelbar eine Vereinbarung voraus, die ebenfalls alle diese Bedingungen erfüllt und deren Länge nicht ein Vielfaches von 8 ist
- Vereinbarung mit UNALIGNED

Wie ein Absolut-Bitzeiger vereinbart wird, ist im Kapitel 4 beschrieben.

Es muß beachtet werden, daß ein Absolut-Bitzeiger einen größeren Speicherplatz benötigt als ein Absolut-Zeiger: Wird er also innerhalb von Strukturen verwendet, so verschieben sich nachfolgende Felder. Es ist zu prüfen, ob an die interne Anordnung der Felder der Struktur (siehe Kapitel 10) Bedingungen gestellt werden, die diese nicht erlauben.

9 Testhilfen

Testhilfen dienen dem Zweck, Fehler im Programm mittels geeigneter Testmethoden leichter oder überhaupt finden zu können.

Die Semantik eines Programms soll durch den Einbezug von Testhilfen nicht beeinflußt werden.

Die Steuerung des zur Verfügung stehenden Testhilfe-Instrumentariums erfolgt durch Steueranweisungen für den Übersetzer (*COMOPT) und für das Programm (*RUNOPT), sowie über Kontrollanweisungen als Reaktion auf Kontrollereignisse.

Erfahrungsgemäß müssen Testhilfen bequem zu handhaben sein, um in breitem Umfang eingesetzt zu werden.

Folgende Steuerungsmöglichkeiten stehen zur Verfügung:

- Steueranweisungen für den Übersetzer
- Steueranweisungen für das Programm
- Setzen von Kontrollpunkten
- Kontrollanweisungen, wenn ein Kontrollpunkt erreicht ist.

Folgende Arten von Testhilfen stehen zur Verfügung:

- Ablaufverfolgung (trace) für
  - Eintritt in Prozeduren (PROCEDURE)
  - Aufruf von Prozeduren (CALL)
  - Programmsprung (GOTO)
  - Marken (label)
  - Rückkehr (RETURN)
- Rückverfolgung (SNAP)
- Binärdump
- Anschluß an die Testhilfe AID

9.1 Steuerung für den Übersetzer

Dem Übersetzer PLI1 wird durch die Steueranweisung

```
*COMOPT DEBUG = Spezifikation
```

mitgeteilt, daß er bestimmte Testhilfen in das Programm einarbeiten soll.

Folgende Spezifikationen sind möglich:

PROCTRACE	Ausgabe einer Protokollzeile beim Eintritt in eine Prozedur.
CALLTRACE	beim Ausführen einer CALL-Anweisung oder eines Funktions-Aufrufes.
GOTOTRACE	beim Ausführen einer GOTO-Anweisung.
LABTRACE	beim Erreichen einer Marke.
RETURNTRACE	bei der Rückkehr aus einer Prozedur.
BREAKPOINT(a,...)	Vor den Zeilen a, ... werden Kontrollpunkte gesetzt.
STMT	Bei Laufzeit-Fehlern wird die Nummer der Quellzeile im Fehlertext mit ausgegeben.

Für die Zeilenangabe a bei BREAKPOINT gilt das Format

```
[i-] z [:s]
```

dabei ist

i: lfd. Nummer der INCLUDE-Anweisung ganze Dezimalzahl 0...256
Wert 0: Text nicht aus INCLUDE-Datei; voreingestellt

z: Zeilennummer (Lfd.-Nr.) des Quell-Protokolls ganze Dezimalzahl < 10^7

s: lfd. Nummer der Anweisung in der Quellzeile ganze Dezimalzahl < 32
Voreinstellung: 1;

```
/EXEC     $PLI1
          *COMOPT DEBUG = CALLTRACE
          *END
```

```
/EXEC     $PLI1
          *COMOPT DEBUG = BREAKPOINT (50.2)
          *END

Kontrollpunkt in Zeile 50 vor der zweiten Anweisung.
```

Bild 9-1 Beispiele für die Steuerung des Übersetzers

9.2 Steuerung für das Programm

Zur Steuerung der Testhilfen für den Programmlauf stehen folgende Steueranweisungen zur Verfügung:

```
*RUNOPT DUMP  = Spezifikation
*RUNOPT TRACE = Spezifikation
```

Zur dynamischen Steuerung von Testhilfen während des Programmlaufs können Kontrollanweisungen gegeben werden, wenn ein eingestellter Kontrollpunkt erreicht ist.

Um einen Kontrollpunkt aktiv oder passiv zu setzen, steht folgende Steueranweisung zur Verfügung:

```
*RUNOPT ACTIVE = YES oder NO
```

Als Kontrollanweisungen beim Erreichen eines Kontrollpunktes sind dieselben Angaben möglich, wie bei den oben erwähnten Spezifikationen für DUMP, TRACE und ACTIVE.

Wird ein aktiver Kontrollpunkt beim Programmlauf erreicht, so wird folgende Meldung ausgegeben:

```
* BKPT/Prozedur/[i-]z[:s]
```

wobei die Angaben folgende Bedeutung haben:

Prozedur	Name der internen oder externen Prozedur, in der die Anweisung steht
i	Nummer der INCLUDE-Datei; entfällt für die Nummer 0
z	Nummer der Quellzeile im Quellprotokoll
s	Lfd. Nummer der Anweisung in der Quellzeile; entfällt beim Wert 1

Es wird dann eine Eingabe erwartet. Dies kann sein

- eine leere Anweisung
- eine Kontrollanweisung

Sie wird sofort ausgeführt. Ist sie fehlerhaft, so wird auf SYSOUT und SYSLST die Meldung "ERROR" und der noch nicht ausgeführte Teil der Kontrollanweisung ausgegeben. Dieser Teil muß, nachdem er berichtigt wurde, wieder eingegeben werden.

Es können bei *RUNOPT bzw. als Kontrollanweisungen folgende Angaben gemacht werden:

- TRACE =
  - PROCTRACE, NOPROCTRACE
  - CALLTRACE, NOCALLTRACE
  - GOTOTRACE, NOGOTOTRACE
  - LABELTRACE, NOLABELTRACE
  - RETURNTRACE, NORETURNTRACE
  - TERMINAL, NOTERMINAL
  - ALL, NO

  Weitere Einzelheiten können dem Abschnitt 9.1 und dem Kapitel 5 entnommen werden.
- ACTIVE = YES oder NO
  Alle Kontrollpunkte werden aktiv bzw. passiv gesetzt.
- SYSTEM
  Sprung in den Kommando-Modus (break point); nur als Kontrollanweisung sinnvoll.
- DUMP =
  - STACK Stapelspeicher
  - AREA Standard-Bereich
  - RANGE(a,e) Bereich von (sedezimale Adressen) a bis e
  - SNAP Aufrufverschachtelung
  - COND nur im Fehlerfall

  Weitere Einzelheiten sind im Kapitel 5 zu finden.

```
/EXEC      Programm
           *RUNOPT TRACE = CALLTRACE
           *END
```

```
*BKPT      /Programm/50.2       Ausgabe nach SYSOUT
TRACE = CALLTRACE               Eingabe von SYSDTA
```

Der Kontrollpunkt in Zeile 50 bei der zweiten Anweisung hat sich gemeldet. Es wird die Ablaufverfolgung für Prozedur-Aufrufe eingeschaltet.

Bild 9-2 Beispiele für die Steuerung der Testhilfen beim Programmlauf

9.3 Ausgaben der Ablaufverfolgung

Die Information der Ablaufverfolgung wird nach SYSLST ausgegeben (bei TRACE = TERMINAL auch nach SYSOUT).
Die Zeilen haben bei den einzelnen Angaben nachstehenden Aufbau:

```
bei PROCTRACE:    *P: p LEVEL: n r
bei CALLTRACE:    *C: p [i-] z [:s]
bei GOTOTRACE:    *G: l [i-] z [:s]
bei LABTRACE:     *L: l [i-] z [:s]
bei RETURNTRACE:  *R: e [i-] z [:s]
```

wobei die Angaben folgende Bedeutung haben:

p	Prozedur- bzw. Eingangs-Namen oder Name der Eingangs-Variablen
n	Tiefe der Aufrufverschachtelung
r	Inhalte der Register R1 bis R4, R14, R15
i	Nummer der INCLUDE-Datei oder leer
z	Nummer der Quellzeile im Quellprotokoll
s	Nummer der Anweisung in der Quellzeile
l	Name der Marke ggf. mit Index
e	Primärer Eingangs-Name

9.4 Aktivieren der Kontrollpunkte

Die Steueranweisung *RUNOPT ACTIVE = YES oder NO bzw. die Kontrollanweisung ACTIVE = YES oder NO bewirken, daß die Kontrollpunkte aktiv bzw. passiv gesetzt werden.

9.5 Programmunterbrechung

Eine Programmunterbrechung kann erreicht werden durch

- die Kontrollanweisung SYSTEM an einem Kontrollpunkt.
- die Break-/Escape-Taste am Sichtgerät. (Sie ist auf den einzelnen Dialogstationen unterschiedlich realisiert.)
- das Kommando /BREAK, wenn /SYSFILE SYSDTA = (SYSCMD) gilt (in DO-Prozeduren sowie in Stapelprozessen), sofern das Programm Daten von SYSDTA liest.

Das Programm kann anschließend mit /RESUME oder mit /INTR wieder gestartet werden, wobei durch /INTR die Bedingung ATTENTION gesetzt wird.

Werden die Steueranweisungen für das Programm mit

```
*END/
```

abgeschlossen, so wird in den Kommandomodus verzweigt. Es kann dann z.B. SYSDTA umgesteuert werden. Mit dem Kommando /RESUME wird das Programm fortgesetzt.

9.6 Speicherauszug

Ein Speicherauszug kann durch die Steueranweisung *RUNOPT DUMP =, durch einen Aufruf einer im Kapitel 11 beschriebenen Prozedur oder durch die Kontrollanweisung DUMP angefordert werden. Im Dialog wird in einem solchen Fall ausgegeben.

```
%IN 45 DUMP DESIRED? REPLY (Y = YES; N = NO)?
```

Wird die Antwort Y gegeben, so wird der Speicherauszug nach SYSLST ausgegeben.

Bei DUMP = STACK und DUMP = AREA wird die obige Anfrage zuerst für die Ausgabe der Pseudoregister-Vektoren und danach für den gewünschten Speicherbereich ausgegeben.

9.7 Rückverfolgung (SNAP)

Die Ausgabe erfolgt sowohl auf SYSLST als auch auf SYSOUT. Die Ausgabezeilen haben folgendes Format:

Name Typ Adresse Quellzeile

Die Quellzeilen-Angabe wird nur ausgegeben, wenn mit DEBUG = STMT übersetzt wurde. Im einzelnen gilt für "Name" abhängig vom "Typ" folgendes:

TYP	Bedeutung des Namens
ENTRY	Eingangs-Name einer Prozedur
SYSTEM	Eingangs-Name einer Bibliotheks-Prozedur
BEGIN	Lfd. Nummer eines BEGIN-Blockes. Diese Nummer ist im Adreßbuch zu finden; vergleiche LIST = MAP, Abschnitt 3.8.8.
ON	Name der Bedingung

Namen, die länger als 7 Zeichen sind, werden auf die ersten 4 und die letzten 3 Zeichen gekürzt.

Die Quellzeilenangabe hat die Form

```
[i-]z[:s]
```

Siehe dazu Abschnitt 9.2

```
***************************** SNAP *****************************

START OF PRINTING OF NESTED SUBROUTINES

CALLED FROM     TYPE       ON ADDRESS              SOURCE-REFERENCE

  SNAP          SYSTEM       0020D2
  FEHLER        ENTRY        000360                    26   1
  ZEROIDE       ON           000374                    14   1
  ER$INTR       SYSTEM       0107EA
  ER$PUB        SYSTEM       010EAC
  SR$STXT       SYSTEM       00FD04
  ##00004       BEGIN        00045A                    20   1
  X97           ENTRY        000254                    12   1
```

Bild 9-3 Beispiel für die Ausgabe bei der Rückverfolgung

Durch die Angabe *COMOPT OPTIMIZE = TIME können bestimmte interne Prozeduren und Blöcke in einer vereinfachten Form codiert werden (siehe Abschnitte 8.4.11 und 8.5.1). Diese Prozeduren und Blöcke sind in der Speicherbelegungsliste (*COMOPT LIST = MAP) durch die Angabe "(QUICK)" gekennzeichnet. Sie werden in der Rückverfolgungsliste nicht mit ausgegeben.

9.8 Anschluß an die Testhilfe AID

Zum symbolischen Testen mit der Testhilfe AID muß beim Übersetzen des Quellprogramms zusätzliche Testinformation erzeugt werden. Dies geschieht durch Angabe der Compiler-Option "SYMTEST=ALL".

Die Testinformation wird dann entweder durch Angabe von "SYMTEST=ALL" beim Binden/Laden statisch in das geladene Programm übernommen, oder sie kann von AID dynamisch nachgeladen werden, falls der Bindemodul in einer PLAM-Bibliothek abgelegt ist.

Nähere Einzelheiten zum Testen mit AID können dem Manual "AID-Testen von PL/I-Programmen" [18] entnommen werden.

10 Interne Darstellung

Im allgemeinen braucht sich der Benutzer der Programmiersprache PL/I nicht um die interne Darstellung seiner Daten im Speicher des Rechners zu kümmern. In folgenden Zusammenhängen kann es jedoch von Vorteil sein, die interne Darstellung der Daten zu betrachten:

- Zur Analyse von Fehlern im Objekt kann man sich die aktuellen Werte von Variablen ansehen, z.B. durch

  – UNSPEC (a)
  – HEXDEC (UNSPEC (a))
  – CALL ADUMP oder SDUMP oder RDUMP

  und damit bitgenau deren Zustände feststellen.

- Bei der Datensatzausgabe ergibt sich die Länge des Datensatzes aus einem Datenanteil und einem Verwaltungsanteil. Die Länge des Datenanteils ist vom Typ der Quellvariablen und seinem aktuellen Inhalt abhängig. Wie sie ermittelt wird, ist im vorliegenden Abschnitt beschrieben. Sie kann z.B. auch mit Hilfe der eingebauten Funktion SIZE erhalten werden. Ob ein zusätzlicher Verwaltungsanteil benötigt wird, hängt von der Zieldatei ab. Dies ist in Kapitel 6 beschrieben.

- Beim Lesen von Datensätzen aus Dateien, die von einem fremden Programm erstellt sind, muß man wissen, ob die Struktur, aus der die Datensätze einmal geschrieben wurden, und die Struktur der Ziel-Variablen miteinander verträglich sind. Dies kann für den Datenanteil mit Hilfe der Beschreibung der vorliegenden Abschnitte ermittelt werden. Für den Verwaltungsanteil kann dies aus der Beschreibung in Kapitel 6 ermittelt werden.

Für die interne Darstellung von skalaren Variablen, Matrix-Elementen und Struktur-Elementen, aber auch für die Ausrichtung von Strukturen gibt es bestimmte Forderungen an die Anordnung der Daten im Speicher. Diese betreffen einmal die Menge des Speicherplatzes der benötigt wird und zum anderen die Ausrichtung auf eine bestimmte Adressierungsgrenze. Dies ist in den folgenden Abschnitten erläutert. Dabei werden folgende Begriffe für den Speicherbedarf verwendet:

- Bits
  Es wird die angegebene Anzahl von Bits benötigt. Die Variable kann auf jeder Bit-Adresse beginnen.

- Bytes
  Es wird die angegebene Anzahl von Bytes (Zeichen) benötigt. Die Variable kann auf jeder Byte-Adresse beginnen.
- Halbwörter
  Es wird die angegebene Anzahl Halbwörter benötigt. Jedes Halbwort besteht aus 2 Bytes. Die Variable kann auf jeder geraden Byte-Adresse beginnen.
- Ganzwörter
  Es wird die angegebene Anzahl Ganzwörter benötigt. Ein Wort umfaßt 4 Bytes. Die Variable kann auf jeder Byte-Adresse beginnen, die durch 4 teilbar ist.
- Doppelwörter
  Es wird die angegebene Anzahl Doppelwörter benötigt. Ein Doppelwort umfaßt 8 Bytes. Die Variable kann auf jeder Byte-Adresse beginnen, die durch 8 teilbar ist.

Werden also z.B. für eine Variable 2 Ganzwörter benötigt, so beginnt der Speicherplatz auf einer Ganzwort-Adresse. Werden jedoch 8 Bytes benötigt, so beginnt der Speicherplatz auf einer Byte-Adresse. Dies kann zufällig auch eine Ganzwortadresse sein. Die Länge des Speicherplatzes ist in beiden Fällen (bei 8 Byte und 2 Ganzwörtern) gleich.

Der Speicherplatz einer skalaren Variablen, einer vollständigen Matrix und einer Haupt-Struktur beginnt mindestens auf einer Bytegrenze. Dies gilt nicht, wenn eine Variable durch Überlagerung entsteht (z.B. durch das Attribut DEFINED oder BASED) und der Teil der überlagert wird, eine Teil-Matrix oder eine Unter-Struktur ist, die an einer Bitgrenze ausgerichtet ist.
Wird einer Variablen mit dem Attribut BASED oder CONTROLLED Speicherplatz zugeordnet, so geschieht dies stets in Vielfachen von Doppelwörtern und der Speicherplatz beginnt in diesem Fall stets auf Doppelwortgrenze.

10.1 Arithmetische Variable

Im Bild 10-1 ist für die arithmetischen Variablen eine kurzgefaßte Übersicht gegeben über

- Vorgabe und Maximalwert der Genauigkeit (PRECISION)
- Speicherbedarf in Abhängigkeit von der Genauigkeit g
- Adressierungsgrenze in Abhängigkeit von der Ausrichtung (ALIGNED/UNALIGNED)

Ausführliche Angaben dazu sind in den nachfolgenden Abschnitten zu finden.

Die maskierten arithmetischen Zeichenfolgen sind unter den Zeichenfolgen in Abschnitt 10.2 behandelt.

Datentyp			PRECISION		Speicherbedarf		
Modus	Skalierung	Basis	Vorgabe	Maximal	für g	ALIGNED	UNALIGNED
REAL	FIXED	DECIMAL	(5,0)	(15,k)	CEIL ((g + 1)/2) Bytes		
		BINARY	(15,0)	(31,k)	1...15	1 Halbwort	2 Bytes
					16...31	1 Ganzwort	4 Bytes
	FLOAT	DECIMAL	(6)	(33)	1...6	1 Ganzwort	4 Bytes
					7...16	1 Doppelwort	8 Bytes
					17...33	2 Doppelwörter	16 Bytes
		BINARY	(21)	(109)	1...21	1 Ganzwort	4 Bytes
					22...53	1 Doppelwort	8 Bytes
					54...109	2 Doppelwörter	16 Bytes
COMPLEX	Real-und Imaginärteil wie bei REAL; Realteil zuerst						

k:+127...-128

Bild 10-1 Übersicht über Speicherbedarf und Adressierungsgrenze bei arithmetischen Variablen

10.1.1 Duale Festpunkt-Variable (FIXED BINARY)

Duale Festpunkt-Variable haben die Attribute FIXED BINARY PRECISION (g,k), wobei g die Mindestgenauigkeit in Binärstellen angibt und k die Stellung des Dualpunktes bestimmt. Für die interne Darstellung werden

16 Bits für g = 1 bis 15 entspricht 2 Bytes bzw. 1 Halbwort

32 Bits für g = 16 bis 32 entspricht 4 Bytes bzw. 1 Ganzwort

verwendet. Alle internen Berechnungen werden mit dieser ggf. aufgerundeten Genauigkeit durchgeführt. Der Speicherbereich beginnt

- bei ALIGNED auf einer Halb- bzw. Ganzwortgrenze
- bei UNALIGNED auf einer Byte-Grenze

Der für die Ermittlung des Wertes erforderliche gedachte Dualpunkt steht bei k = 0 rechts vom ganz rechts stehenden Bit. Bei positivem k verschiebt er sich um k Dualstellen nach links und bei negativem um |k| Dualstellen nach rechts.

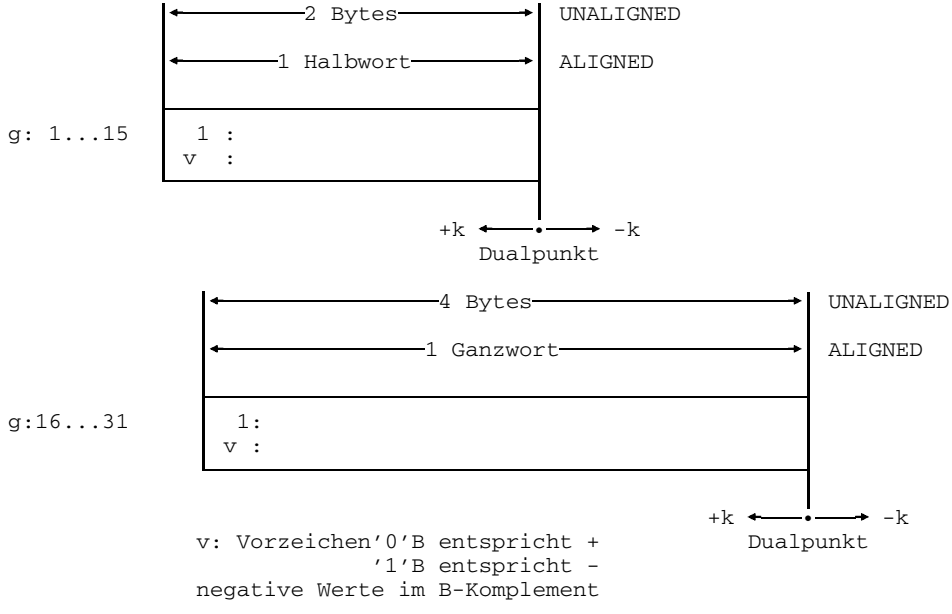


Bild 10-2 Interne Darstellung von Variablen mit den Attributen REAL

```
DCL A   FIXED BINARY PRECISION(10,0)  ALIGNED;
DCL B   FIXED BINARY PRECISION(15,0)  ALIGNED;
DCL C   FIXED BINARY PRECISION(25,0)  ALIGNED;
DCL D   FIXED BINARY PRECISION(31,0)  ALIGNED;

DCL M   FIXED BINARY PRECISION(10,0)  UNALIGNED;
DCL N   FIXED BINARY PRECISION(15,0)  UNALIGNED;
DCL O   FIXED BINARY PRECISION(25,0)  UNALIGNED;
DCL P   FIXED BINARY PRECISION(31,0)  UNALIGNED;
```

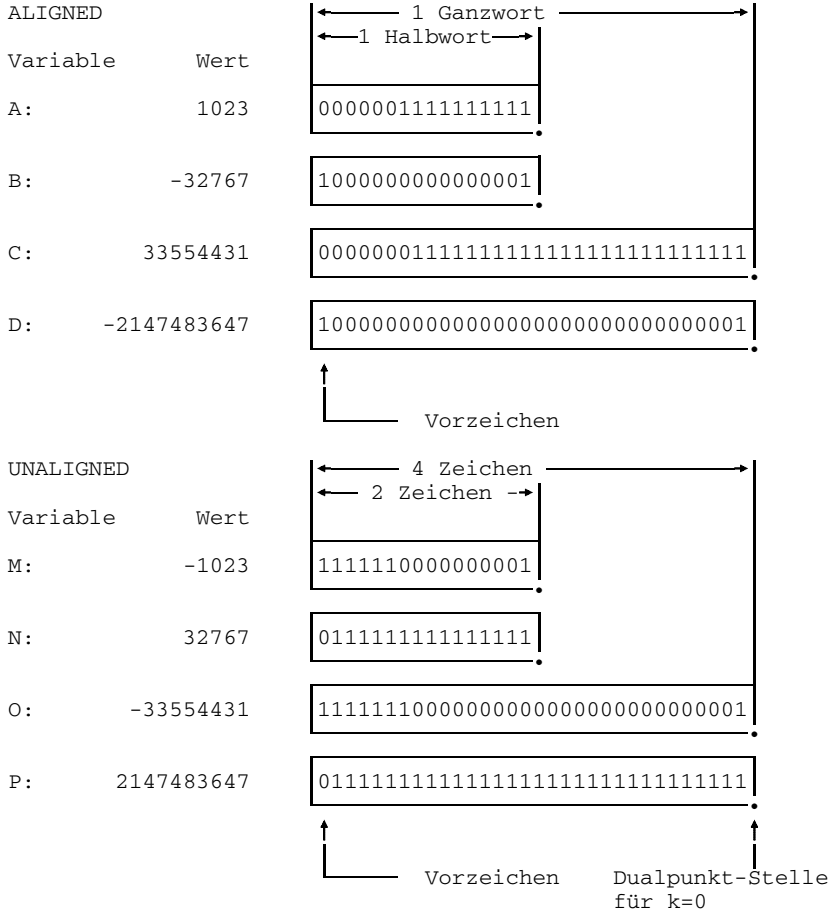


Bild 10-3 Interne Darstellung von Festpunkt-Variablen mit unterschiedlicher Ausrichtung und Genauigkeit und Genauigkeit beim jeweiligen Maximalwert

Das ganz links stehende Bit gibt an, ob es sich um einen positiven (’0’B) oder negativen Wert (’1’B) handelt. Negative Werte werden im B-Komplement dargestellt. Um bei einem negativen Wert den Betrag zu ermitteln, kann wie folgt vorgegangen werden:

- Alle Bits einschließlich Vorzeichenbits werden invertiert.
- Ganz rechts wird ein Bit ’1’B addiert. Ein evtl. in das Vorzeichenbit überlaufendes Bit wird ignoriert.

Der so entstandene Wert stellt den (positiven) Betrag dar.
Im Bild 10-4 sind dazu einige Beispiele gezeigt.

negativer Wert	1 0101=-11	1 1111=-1	1 0000=-0
invertiert	0 1010	0 0000	0 1111
invertiert +1	0 1011=11	0 0001=1	0 0000=0
			1

Bild 10-4 Beispiele zur Ermittlung des Betrages bei negativen Werten für k = 0

10.1.2 Dezimale Festpunkt-Variable (FIXED DECIMAL)

Dezimale Festpunkt-Variable haben die Attribute FIXED DECIMAL PRECISION (g,k), wobei g die Mindestgenauigkeit in Dezimalstellen angibt und k die Stellung des Dezimalpunktes bestimmt. Für die interne Darstellung wird rechts eine Vorzeichenstelle ergänzt und, wenn dies eine ungerade Anzahl Stellen ergibt, links eine Ziffernstelle ergänzt. Ziffernstellen und Vorzeichenstelle werden in je 1/2 Bytes gespeichert. Der Speicherbedarf ergibt sich damit zu

$$\text{CEIL}\left(\frac{g+1}{2}\right) \text{ Bytes}$$

Alle internen Berechnungen werden mit dieser ggf. aufgerundeten Genauigkeit durchgeführt. Der Speicherbereich beginnt sowohl bei ALIGNED als auch bei UNALIGNED auf einer Bytegrenze

Der für die Ermittlung des Wertes erforderlich gedachte Dezimalpunkt steht bei k = 0 rechts von der ganz rechts stehenden Ziffernstelle. Bei positivem k verschiebt er sich um k Dezimalstellen nach links und bei negativem um |k| Dezimalstellen nach rechts.

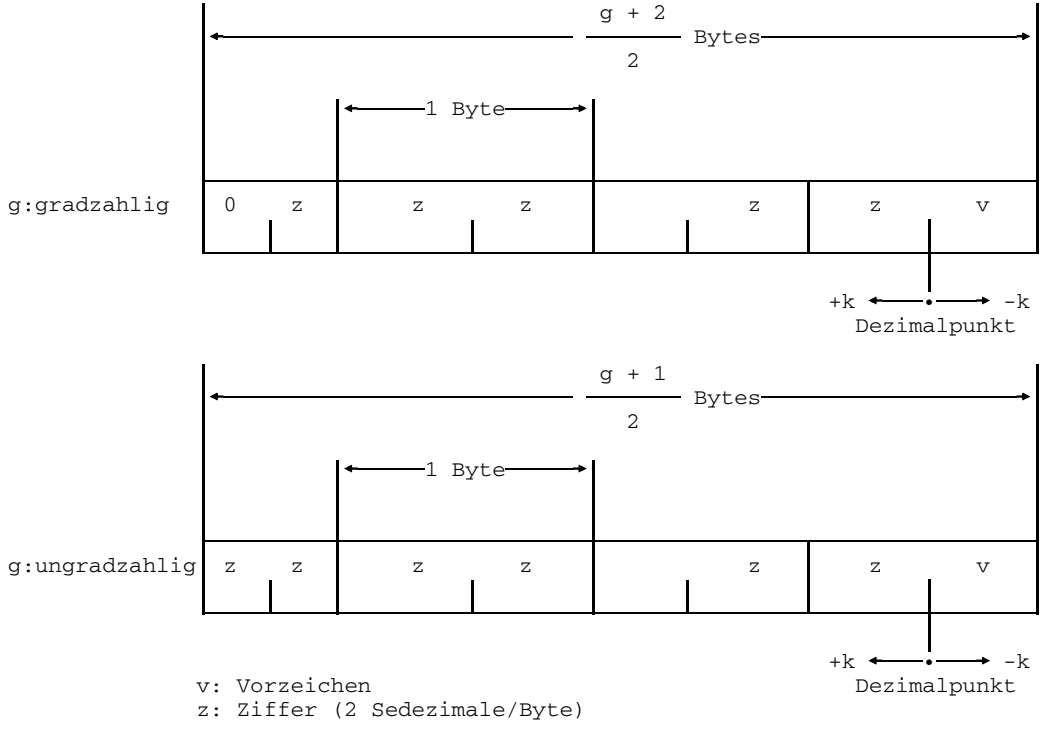


Bild 10-5 Interne Darstellung von Variablen mit den Attributen REAL FIXED DECIMAL PRECISION (g,k)

```
DCL A   FIXED DECIMAL PRECISION(02,0)   ALIGNED;
DCL B   FIXED DECIMAL PRECISION(05,0)   ALIGNED;
DCL C   FIXED DECIMAL PRECISION(10,0)   ALIGNED;
DCL D   FIXED DECIMAL PRECISION(15,0)   ALIGNED;

DCL M   FIXED DECIMAL PRECISION(02,0)   UNALIGNED;
DCL N   FIXED DECIMAL PRECISION(05,0)   UNALIGNED;
DCL O   FIXED DECIMAL PRECISION(10,0)   UNALIGNED;
DCL P   FIXED DECIMAL PRECISION(15,0)   UNALIGNED;
```

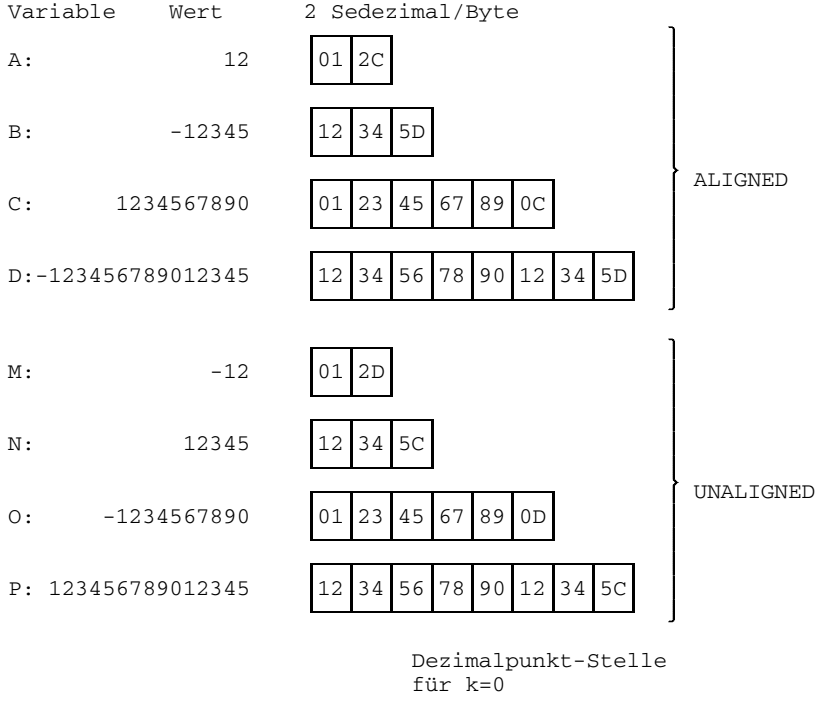


Bild 10-6 Beispiel für die interne Darstellung von dezimalen Festpunkt-Variablen

Die interne Darstellung der Dezimalziffern und der Vorzeichen ist in Bild 10-7 gezeigt.

Vorzeichen	dual	sedezimal
+	1010	A
	1100	C
	1110	E
	1111	F
-	1011	B
	1101	D

Ziffer	dual	sedezimal
0	0000	0
1	0001	1
2	0010	2
3	0011	3
4	0100	4
5	0101	5
6	0110	6
7	0111	7
8	1000	8
9	1001	9

Bild 10-7 Interne Darstellung von Vorzeichen und Ziffern bei dezimalen Festpunkt-Variablen in 4 Bits (dual und sedezimal)

10.1.3 Gleitpunkt-Variable (FLOAT)

Gleitpunkt-Variable haben die Attribute FLOAT PRECISION (g), wobei g die Mindesgenauigkeit der Mantisse ist, bei BINARY in Dualstellen und bei DECIMAL in Dezimalstellen. Intern wird in beiden Fällen die Darstellung

$m \cdot 16^e$

verwendet, wobei die Mantisse m stets ein echter dualer Bruch ist (Dualpunkt ganz links). Der Exponent ist eine ganze Dualzahl (Dualpunkt ganz rechts) bezogen auf die Basis 16.

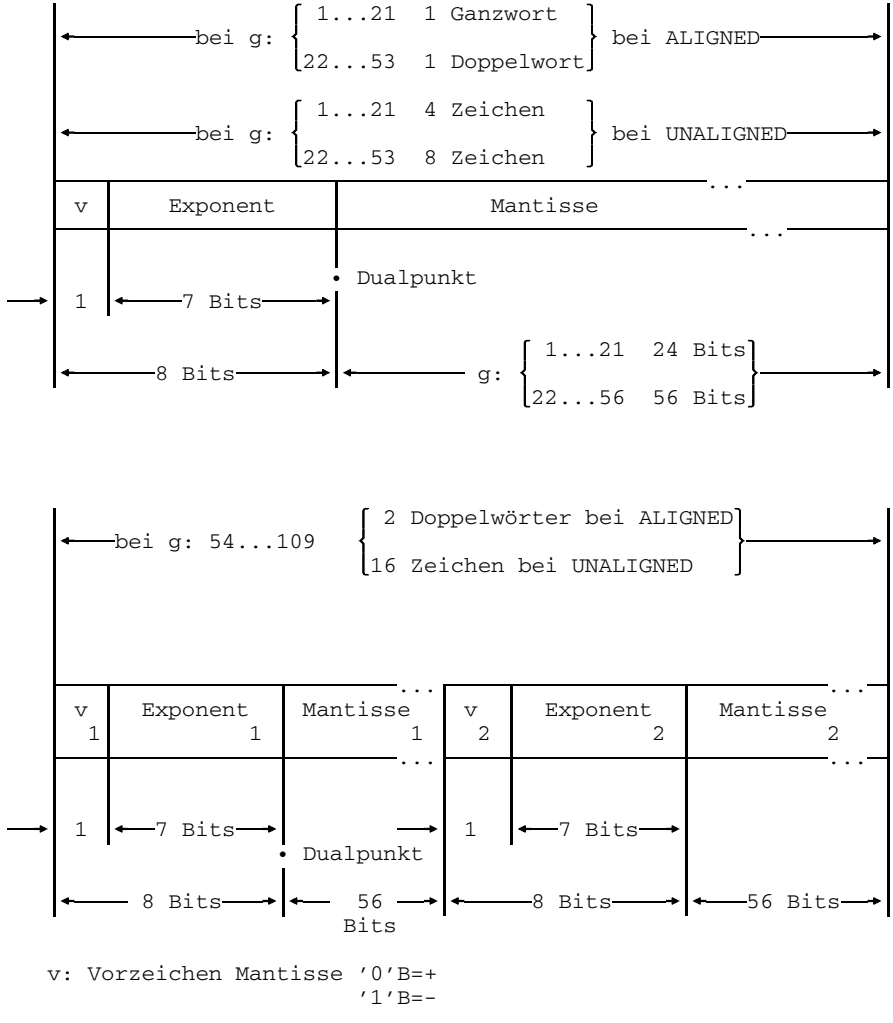


Bild 10-8 Interne Darstellung von Variablen mit den Attributen REAL FLOAT BINARY PRECISION (g)

Das erste links stehende Bit ist das Vorzeichen der Mantisse wobei ’0’B einen positven und ’1’B negativen Wert darstellt. Die folgenden 7 Bits stellen den Wert des Exponenten dar. Für die Ermittlung des Exponentenwertes muß der Dualwert des Bitmusters um den Wert 64 verringert werden. Im Bild 10-9 sind dazu Beispiele gegeben.

dual	dezimal	Exponent
0 000000	0	-64
0 000001	1	-63
0 000010	2	-62
0 111110	62	- 2
0 111111	63	- 1
1 000000	64	0
1 000001	65	+ 1
1 000010	66	+ 2
1 111101	125	+61
1 111110	126	+62
1 111111	127	+63

Exponent = Dezimalwert -64

Bild 10-9 Beispiele für die interne Darstellung des Exponenten bei Gleitpunktvariablen

Die Länge der Mantisse hängt von der Genauigkeit wie folgt ab:

	BINARY (g)	DECIMAL (g)	Mantisse	Gesamt
kurz	1...21	1...6	24 Bits	32 Bits
lang	22...53	7...16	56 Bits	64 Bits
erweitert	54...109	17...33	112 Bits	128 Bits

Bei der erweiterten Gleitpunktzahl wird zweimal hintereinander die Darstellung der langen Gleitpunktzahl verwendet; Vorzeichenbit und Exponentenbits im zweiten Teil sind für den Wert bedeutungslos.

Die interne Darstellung der Mantisse ist die gleiche wie bei einer dualen Festpunkt-Variablen mit dem Dualpunkt links von der ganz links stehenden Dualziffer.

Gleitpunktwerte sind stets normalisiert, d.h. die Mantisse wird unter entsprechender Korrektur des Exponenten so oft um 4 Bits geschiftet, daß mindestens eines der 4 ganz links stehenden Bits ungleich ’0’B ist.

Interne Berechnungen werden mit der Genauigkeit durchgeführt, die durch die interne Darstellung gegeben ist.

```
DCL A   FLOAT BINARY PRECISION(10)   ALIGNED;
DCL B   FLOAT BINARY PRECISION(21)   ALIGNED;
DCL C   FLOAT BINARY PRECISION(53)   ALIGNED;
DCL D   FLOAT BINARY PRECISION(109)  ALIGNED;

DCL M   FLOAT BINARY PRECISION(10)   UNALIGNED;
DCL N   FLOAT BINARY PRECISION(21)   UNALIGNED;
DCL O   FLOAT BINARY PRECISION(53)   UNALIGNED;
DCL P   FLOAT BINARY PRECISION(109)  UNALIGNED;
```

```
interne Darstellung (sedezimal)     Wert

A:42100000                           1.600E+01
B:FFFFFFFF                          -7.237005E+75
C:7FFFFFFFFFFFFFFF                   7.237005577332260E+75
D:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF  -7.23700557733226221397318656304297E+75

M:40100000                           6.250E-02
N:00FFFFFF                           8.636168E-78
O:80FFFFFFFFFFFFFF                  -8.636168555094444E-78
P:00FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF   8.63616855509444462538635186280038E-78
```

Bild 10-10 Beispiel für die interne Darstellung von dualen Gleitpunkt-Variablen

```
DCL A   FLOAT DECIMAL PRECISION(03)   ALIGNED;
DCL B   FLOAT DECIMAL PRECISION(06)   ALIGNED;
DCL C   FLOAT DECIMAL PRECISION(16)   ALIGNED;
DCL D   FLOAT DECIMAL PRECISION(33)   ALIGNED;

DCL M   FLOAT DECIMAL PRECISION(03)   UNALIGNED;
DCL N   FLOAT DECIMAL PRECISION(06)   UNALIGNED;
DCL O   FLOAT DECIMAL PRECISION(16)   UNALIGNED;
DCL P   FLOAT DECIMAL PRECISION(33)   UNALIGNED;
```

```
interne Darstellung (sedezimal)     Wert

A:42100000                           1.60E+01
B:FFFFFFFF                          -7.23700E+75
C:7FFFFFFFFFFFFFFF                   7.237005577332260E+75
D:FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF  -7.23700557733226221397318656304297E+75

M:40100000                           6.25E-02
N:00FFFFFF                           8.63616E-78
O:80FFFFFFFFFFFFFF                  -8.636168555094444E-78
P:00FFFFFFFFFFFFFFFFFFFFFFFFFFFFFF   8.63616855509444462538635186280038E-78
```

Bild 10-11 Beispiele für die interne Darstellung von dezimalen Gleitpunkt-Variablen

10.1.4 Maskierte Zeichenfolge-Variable (PICTURE)

Arithmetische Zeichenfolge-Variable mit dem Attribut PICTURE werden intern stets als Zeichenfolgen dargestellt. Dies ist in Abschnitt 10.2.3 beschrieben.

10.1.5 Komplexe Variable

Komplexe Variable bestehen stets aus einem Realteil und einem Imaginärteil, die beide intern die gleiche Darstellung haben. Der Imaginärteil steht unmittelbar hinter dem Realteil.

10.2 Folgen-Variable

Im Bild 10-12 ist für die Folgen-Variable eine kurzgefaßte Übersicht gegeben über

- Vorgabe und Maximalwert der Längenangabe bei Zeichenfolgen (CHARACTER) und Bitfolgen (BIT)
- Speicherbedarf
- Adressierungsgrenze

Ausführliche Angaben dazu sind in den nachfolgenden Abschnitten zu finden.

Folge		UNALIGNED	ALIGNED
CHAR (n)	NONVARYING	n Bytes	
	VARYING	2 + n Bytes	1 Halbwort + n Bytes
BIT (n)	NONVARYING	n Bits	$\text{CEIL}\left(\frac{n}{8}\right)$ Bytes
	VARYING	$\text{CEIL}\left(2+\frac{n}{8}\right)$ Bytes	1 Halbwort + $\text{CEIL}\left(\frac{n}{8}\right)$ Bytes
PIC'...'	wie bei CHAR (m) NONVARYING m=Anzahl Symbole ohne V,F,K		
PIC'...' COMPLEX	Real- und Imaginärteil wie bei PIC'...'; Realteil zuerst		

n: Vorgabe = 1
Maximum = 32767

m: numerisch max. 255
alphanumerisch max. 511

Bild 10-12 Übersicht über Speicherbedarf und Adressierungsgrenze bei Folgen-Variablen

10.2.1 Bitfolge-Variable (BIT)

Für Bitfolge-Variable mit dem Attribut BIT(n) wird folgender Speicherplatz benötigt (x von 0 bis 7):

```
NONVARYING UNALIGNED               n Bits
NONVARYING   ALIGNED               n Bits + x Füllbits
VARYING    UNALIGNED  2 Bytes    + n Bits + x Füllbits
VARYING      ALIGNED  1 Halbwort + n Bits + x Füllbits
```

Dabei umfaßt x soviel Füllbits, bis die Gesamtzahl der Bits (n+x) durch 8 teilbar ist, also genau eine bestimmte Anzahl Bytes füllen.

Der Speicherbereich beginnt bei

```
NONVARYING UNALIGNED  auf einer Bit-Grenze
NONVARYING   ALIGNED  auf einer Byte-Grenze
VARYING    UNALIGNED  auf einer Byte-Grenze
VARYING      ALIGNED  auf einer Halbwortgrenze
```

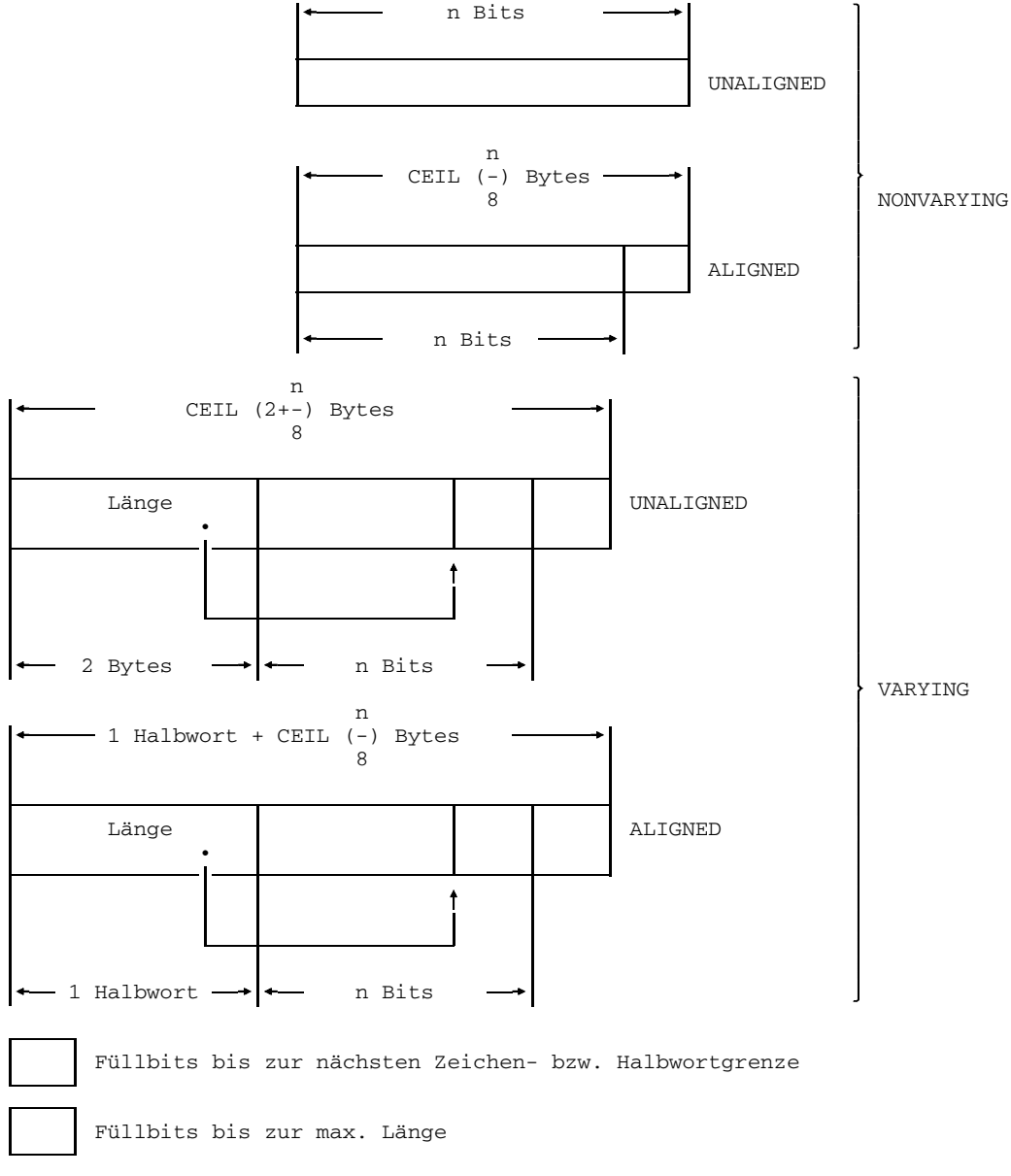


Bild 10-13 Interne Darstellung von Variablen mit dem Attribut BIT (n)

```
DCL A   BIT(4)   NONVARYING   ALIGNED;
DCL B   BIT(4)   NONVARYING   UNALIGNED;
DCL C   BIT(8)   NONVARYING   ALIGNED;
DCL D   BIT(8)   NONVARYING   UNALIGNED;

DCL M   BIT(4)   VARYING      UNALIGNED;
DCL N   BIT(4)   VARYING      ALIGNED;
DCL O   BIT(8)   VARYING      UNALIGNED;
DCL P   BIT(8)   VARYING      ALIGNED;
```

```
Wert             interne Darstellung (dual)

A: 1100          1100
B: 1111          1111
C: 11111100      11111100
D: 11111111      11111111

M: 11            0000000000000001011

N: 1111          00000000000001001111

O: 11111         000000000000110111111

P: 11111111      0000000000000100011111111

                    Längenangabe

   Füllbits bis zur max. Länge
```

Bild 10-14 Beispiele für die interne Darstellung von Bitfolgen

Für die Datensatz-Ausgabe einer skalaren Variablen mit dem Attribut VARYING werden nur soviel Zeichen ausgegeben, daß die durch die Längenangabe bestimmte Anzahl Bits im Datensatz enthalten sind. Im letzten Zeichen des Datensatzes können also noch 0 bis 7 Füllbits enthalten sein. Ist für die Datei die Angabe ENVIRONMENT (SCALARVARYING) gemacht, so wird auch die Längenangabe mit ausgegeben, im anderen Fall nicht. Entsprechendes gilt für die Datensatz-Eingabe. Dies ist ausführlich im Abschnitt 6.3.5 beschrieben.

Skalare Variable, Haupt-Strukturen und Matrizen beginnen mindestens auf Byte-Grenze. Dies ist für Variable mit dem Attribut BIT NONVARYING UNALIGNED in Verbindung mit der Datensatz-Ausgabe von Bedeutung.
Stehen solche Variable am Anfang oder Ende eines Datensatzes, so kann bei der Ausgabe eine unerwartete Datensatzgliederung entstehen. Beim Einlesen eines Datensatzes können angrenzende Variable unbeabsichtigt verändert werden (siehe Abschnitt 10.5.5).

10.2.2 Zeichenfolge-Variable (CHARACTER)

Für die Zeichenfolge-Variable mit dem Attribut CHAR(n) wird folgender Speicherplatz benötigt:

```
NONVARYING                          n Bytes
VARYING    UNALIGNED 2 Bytes    + n Bytes
VARYING      ALIGNED 1 Halbwort + n Bytes
```

Der Speicherbereich beginnt bei

```
VARYING ALIGNED auf Halbwortgrenze
Sonst           auf Byte-Grenze
```

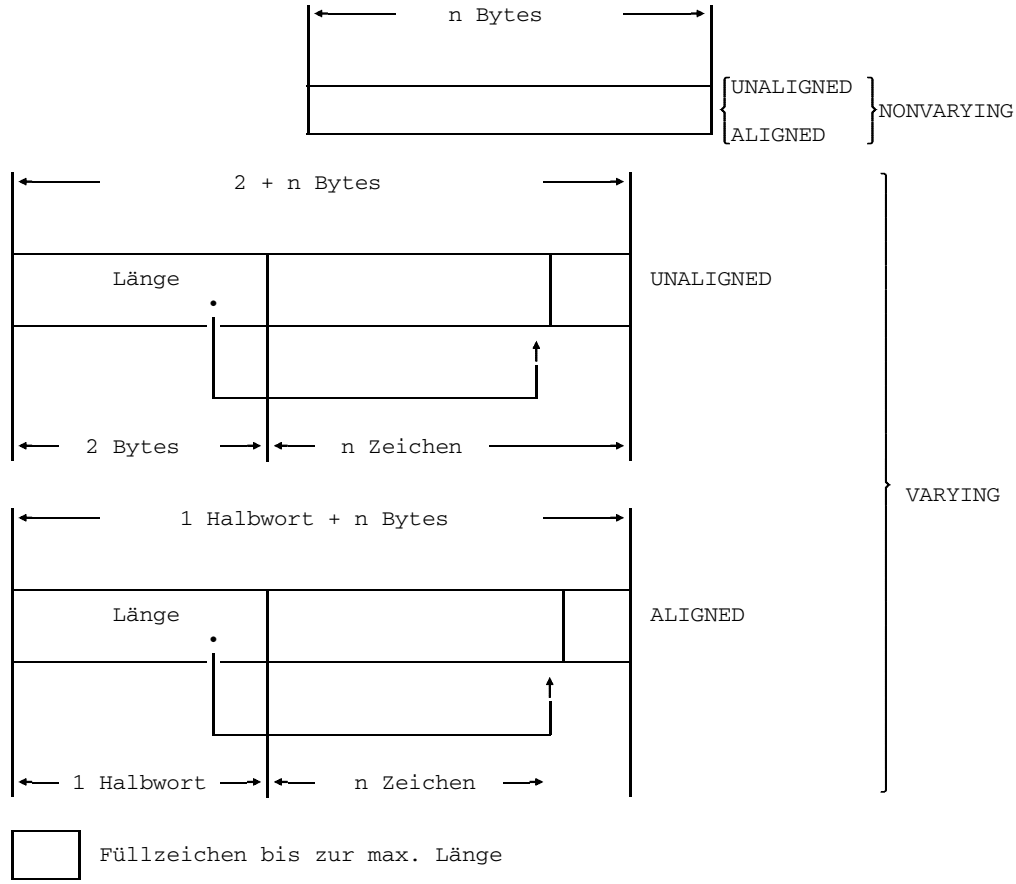


Bild 10-15 Interne Darstellung von Variablen mit dem Attribut CHARACTER(n)

```
DCL A   CHARACTER(05)   NONVARYING   ALIGNED;
DCL B   CHARACTER(05)   NONVARYING   UNALIGNED;
DCL C   CHARACTER(11)   NONVARYING   ALIGNED;
DCL D   CHARACTER(11)   NONVARYING   UNALIGNED;

DCL M   CHARACTER(05)   VARYING      ALIGNED;
DCL N   CHARACTER(05)   VARYING      UNALIGNED;
DCL O   CHARACTER(11)   VARYING      ALIGNED;
DCL P   CHARACTER(11)   VARYING      UNALIGNED;
```

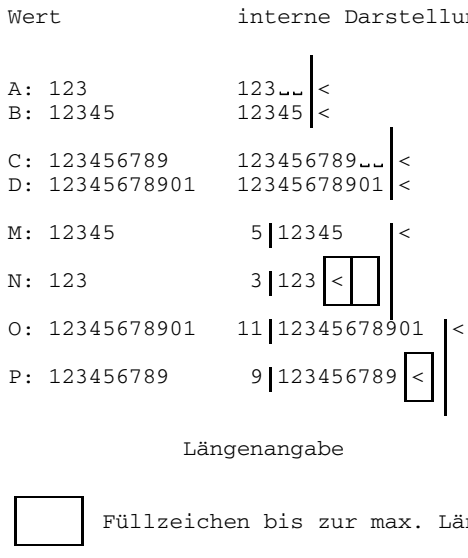


Bild 10-16 Beispiele für die interne Darstellung von Zeichenfolgen

Für die Datensatz-Ausgabe einer skalaren Variablen mit dem Attribut VARYING werden nur soviel Zeichen übertragen, wie die aktuelle Länge angibt. Ist für die Datei die Angabe ENVIRONMENT (SCALARVARYING) gemacht, so wird auch die Längenangabe mit ausgegeben, im anderen Fall nicht. Entsprechendes gilt für die Datensatz-Eingabe. Dies ist ausführlich im Abschnitt 6.3.5 beschrieben.

10.2.3 Maskierte Zeichenfolge-Variable (PICTURE)

In einer Variablen mit dem Attribut PICTURE werden die Werte stets als Zeichenfolge wie bei CHAR(n) NONVARYING dargestellt. Die Länge n der Zeichenfolge ergibt sich aus der Anzahl der Maskensymbole nach Auswertung der Faktoren. Dabei werden die Angaben V, K und F nicht mitgezählt.

Ist das Attribut COMPLEX gegeben, so besteht die Variable aus einem Realteil und einem Imaginärteil, die beide die gleiche Intern-Darstellung haben. Der Imaginärteil folgt unmittelbar auf den Realteil.

```
DCL A   PIC'AX9XXXXX';
DCL B   PIC'*999.V9999';              interne Darstellung
DCL C   PIC'S999ES99';

A = 'A:5A-B';                         A:5A-B

B = 123.456;                          *123.4560

C = 12E73;                            +120E+72
```

Bild 10-17 Interne Darstellung maskierter Zeichenfolge-Variablen

10.3 Programmsteuerungs-Variable

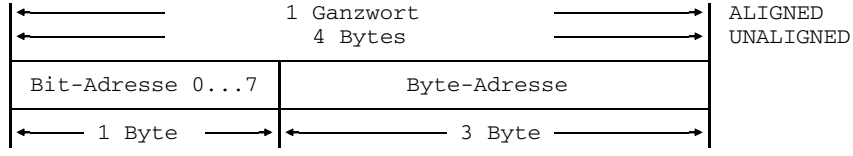
10.3.1 Zeiger (POINTER, OFFSET)

Der Inhalt dieser Variablen ist ein Zeiger, der auf einen Speicherplatz weist.

10.3.1.1 Zeiger bei "*COMOPT OPTIONS = NOXS"

Wurde mit "*COMOPT OPTIONS = NOXS" übersetzt, so haben Relativ- und Absolut-Zeiger die gleiche Intern-Darstellung. Ein Zeiger benötigt 1 Ganzwort bzw. 4 Bytes Speicherplatz. Die drei rechtsstehenden Bytes enthalten die absolute bzw. die relative Speicheradresse. Das linke Byte enthält die Bit-Adresse 0 bis 7; die linken 5 Bits sind immer '0'B.

Ein Leerzeiger hat den sedezimalen Wert 'FFFEFFF8'




```
Leerzeiger (sedezimal):
```

FFFEFFF8

Bild 10-18 Interne Darstellung von Zeigern mit "*COMOPT OPTIONS = NOXS"

```
DCL A   POINTER   ALIGNED;
DCL B   POINTER   UNALIGNED;
DCL C   OFFSET(M)  ALIGNED;
DCL D   OFFSET(U)  UNALIGNED;
```

```
                                    interne Darstellung (sedezimal)

                                         A: 00|0461D8
                                         B: 00|046200
                                         C: 00|000020
                                         D: 00|000020

                                   Bit-Adresse    Byte-Adresse
```

Bild 10-19 Beispiele für die interne Darstellung von Zeiger-Variablen mit "*COMOPT OPTIONS = NOXS"

10.3.1.2 Zeiger bei "*COMOPT OPTIONS = XS"

Wurde mit "*COMOPT OPTIONS = XS" übersetzt, so haben Relativ-Zeiger und Leerzeiger die gleiche Intern-Darstellung, wie sie im Abschnitt 10.3.1.1 beschrieben sind. Für den Absolut-Zeiger gibt es zwei neue Intern-Darstellungsformen:
Der Absolut-Zeiger, der nur bytegenau auf einen Speicherplatz weisen kann: Er benötigt 1 Ganzwort bzw. 4 Bytes Speicherplatz, die die absolute Speicheradresse enthalten. Eine Bit-Adresse ist nicht vorhanden.

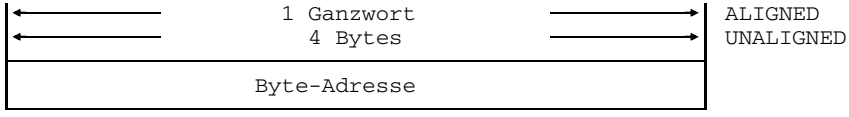


Bild 10-19a Interne Darstellung des Absolut-Zeigers bei "*COMOPT OPTIONS = XS"

Der Absolut-Bitzeiger, der bitgenau auf einen Speicherplatz weisen kann: Er benötigt 1 Ganzwort und 1 Halbwort bzw. 6 Bytes Speicherplatz. Die vier linksstehenden Bytes enthalten die absolute Speicheradresse. Die zwei rechtsstehenden Bytes enthalten die Bit-Adresse 0 bis 7.

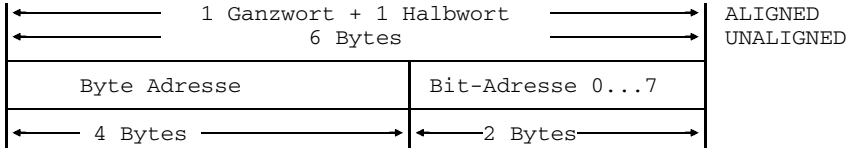


Bild 10-19b Interne Darstellung des Absolut-Bitzeigers bei "*COMOPT OPTIONS = XS"

```
DCL A   POINTER   ALIGNED;
DCL B   POINTER   UNALIGNED;
DCL C   POINTER (BITPTR) ALIGNED;
DCL D   POINTER (BITPTR) UNALIGNED;
```

```
                                        interne Darstellung (sedezimal)

                                             A: 100461D8
                                             B: 00046200
                                             C: 100461D8|0000
                                             D: 00046200|0000

                                        Byte-Adresse   Bit-Adresse
```

Bild 10-19c Beispiele für die interne Darstellung von Absolut-Zeiger- und Absolut-Bitzeiger-Variablen bei "*COMOPT OPTIONS = XS"

10.3.2 Bereich (AREA)

Für eine Bereichs-Variable mit dem Attribut AREA(n) beträgt der

```
                          n
Speicherbedarf CEIL (3 + -) Doppelwörter
                          8
```

Die ersten drei Doppelwörter enthalten Verwaltungsinformationen; die weiteren Doppelwörter stehen für die Zuordnung von BASED-Variablen zur Verfügung. Der Speicherbereich beginnt stets auf Doppelwortgrenze.

Bei der Zuordnung von BASED-Variablen wird der Bereich stets in Doppelwortportionen belegt. Hierdurch kann am Ende der BASED-Variablen Füllinformation entstehen.

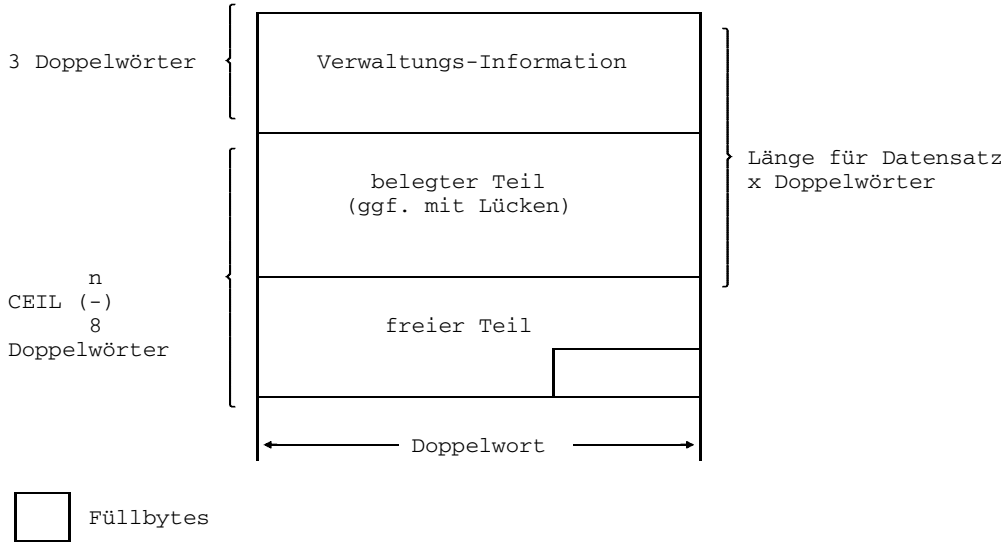


Bild 10-20 Interne Darstellung eines Bereichs mit dem Attribut AREA(n)

Bei der Datensatz-Ausgabe werden die 3 Doppelwörter Verwaltungsinformation und die Doppelwörter des belegten Teils ausgegeben. Die Datensatzlänge ist also stets eine Anzahl von Doppelwörtern (Vielfaches von 8 Bytes). In dem belegten Teil können Lükken enthalten sein. Diese werden mit übertragen; eine Bereinigung der Lücken ist also mit der Ausgabe nicht verbunden. Sowohl bei der Aus- als auch bei der Eingabe wird die Verwaltungsinformation nicht verändert.

Eine detaillierte Beschreibung der internen Struktur des Bereichs und die Verwaltungsstrategie ist in Abschnitt 10.7.4 zu finden.

```
DCL BEREICH      AREA(40);

DCL VARIABLE     CHAR(6) BASED   INIT('   ');

DO I=1 TO 5;
   ALLOCATE VARIABLE IN(BEREICH) SET (Z(I));
   END;                                                    Datensatzlänge

FREE Z(2)->VARIABLE IN(BEREICH);
WRITE FILE(DATEI) FROM(BEREICH);                           64 Bytes
FREE Z(5)->VARIABLE IN(BEREICH);
WRITE FILE(DATEI) FROM(BEREICH);                           56 Bytes
```

Bild 10-21 Beispiel für die Datensatzlänge bei einem Bereich

10.3.3 Marke (LABEL)

Für eine Marken-Variable beträgt der Speicherbedarf 3 Ganzwörter bzw. 12 Bytes.

Dabei enthalten die rechten 4 Bytes die Adresse der Blockaktivierung, die bei rekursivem Aufrufen von spezieller Bedeutung sein kann. Die mittleren 4 Bytes enthalten eine Basisadresse und die linken 4 Bytes die Adresse der Marke, die Zieladresse. Die Speicheradresse beginnt bei ALIGNED auf Ganzwortgrenze, bei UNALIGNED auf Bytegrenze.

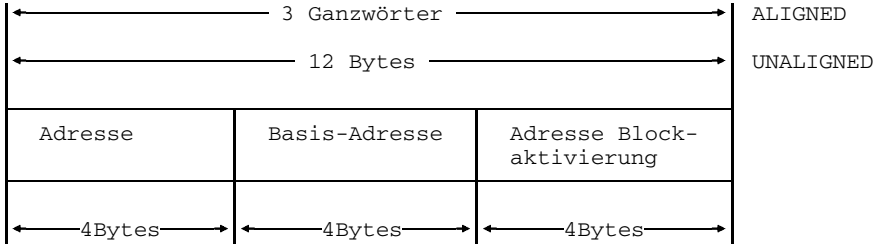


Bild 10-22 Interndarstellung von Variablen mit dem Attribut LABEL

```
DCL MARKE   LABEL VARIABLE;
```

```
interne Darstellung (sedezimal)

00000190|00000000|00046120
00000194|00000000|00046120
00000198|00000000|00046120
0000019C|00000000|00046120

Adresse der          Adresse der
Anweisung            Blockaktivierung

          Basis-Adresse
```

Bild 10-23 Beispiele für die interne Darstellung von Marken-Variablen

10.3.4 Format (FORMAT)

Für eine Format-Variable beträgt der Speicherbedarf 2 Ganzwörter bzw. 8 Bytes

Dabei enthalten die rechten 4 Bytes die Adresse der Blockaktivierung, die bei rekursivem Aufrufen von spezieller Bedeutung sein kann. Die linken 4 Bytes enthalten die Adresse der Formatliste. Die Speicheradresse beginnt bei ALIGNED auf Ganzwortgrenze, bei UNALIGNED auf Bytegrenze.

Bei OPTIONS = NOISO kann ein Format-Wert einer Marken-Variablen zugewiesen werden. In diesem Fall belegt der Formatwert die linken 8 Bytes; die rechten 4 Bytes sind dann undefiniert.

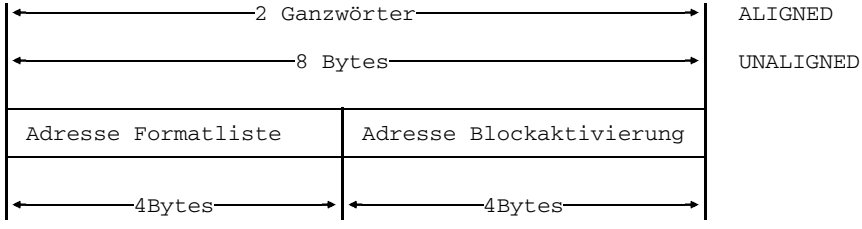


Bild 10-24 Interne Darstellung von Variablen mit dem Attribut FORMAT

```
DCL FORMAT   FORMAT VARIABLE;
```

```
                interne Darstellung (sedezimal)

                     00000468|00046120
                     00000448|00046120
                     00000428|00046120
                             |
                Adresse der  |    Adresse der
                Formatliste  |    Blockaktivierung
```

Bild 10-25 Beispiele für die interne Darstellung von Format-Variablen

10.3.5 Eingang (ENTRY)

Für eine Eingangs-Variable beträgt der Speicherbedarf 2 Ganzwörter bzw. 8 Bytes

Die linken 4 Bytes enthalten die Adresse der Eingangsstelle und die rechten 4 Bytes die Adresse des Aktivierungssatzes einer bestimmten Aktivierung des statischen Vaterblocks.
Die Speicheradresse beginnt bei ALIGNED auf Ganzwortgrenze, bei UNALIGNED auf Bytegrenze.

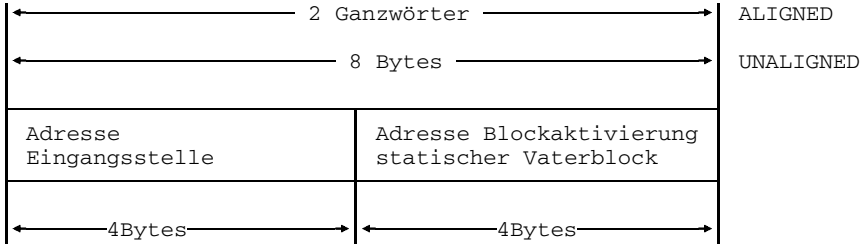


Bild 10-26 Interndarstellung von Variablen mit dem Attribut ENTRY

```
DCL EINGANG   ENTRY VARIABLE;
```

```
                interne Darstellung (sedezimal)

                     00000120 00028120
                     00000180 00028120
                     000001D4 00028120
                     00000234 00028120

               Adresse           Adresse Blockaktivierung
               Eingangsstelle    statischer Vaterblock
```

Bild 10-27 Beispiele für die interne Darstellung von Eingangs-Variablen

10.3.6 Datei (FILE)

Für eine Datei-Variable beträgt der Speicherbedarf 2 Ganzwörter bzw. 8 Bytes

Die linken 4 Bytes enthalten die Adresse des Datei-Attribut-Blocks und die rechten 4 Bytes die Adresse des Datei-Status-Blocks. Die Speicheradresse beginnt bei ALIGNED auf Ganzwortadresse, bei UNALIGNED auf Byteadresse.

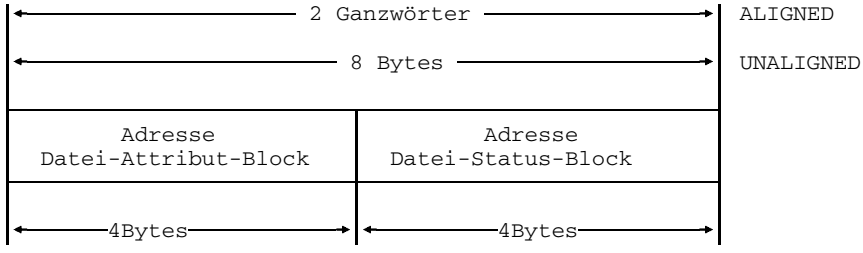


Bild 10-28 Interne Darstellung von Variablen mit dem Attribut FILE

```
DCL VAR   FILE VARIABLE;
```

```
                interne Darstellung (sedezimal)

                     000004F0 00033758
                     00000548 00033B00
                     000004B0 000333B0

                Adresse Datei-  Adresse Datei-
                Attribut-Block  Status-Block
```

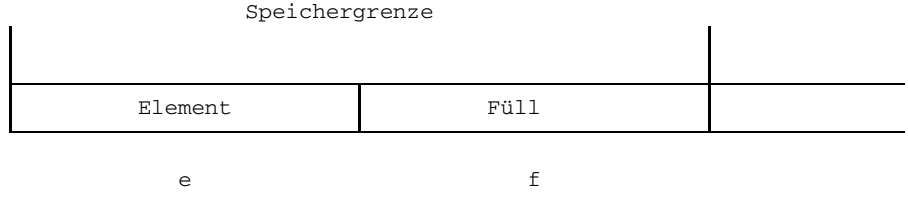
Bild 10-29 Beispiele für die interne Darstellung von Datei-Variablen

10.4 Matrix (DIMENSION)

Eine Variable mit dem Attribut DIMENSION besteht aus einer Anzahl von Matrix-Elementen, die im Speicher hintereinander angeordnet werden. Die Anzahl der Elemente wird durch die Angabe beim Attribut DIMENSION bestimmt.

Ein Matrix-Element kann entweder eine skalare Variable oder eine Struktur sein. Alle Elemente einer Matrix haben den gleichen Aufbau und die gleiche interne Darstellung. Die interne Darstellung, d.h. Speicherbedarf und Adressierungsgrenze für ein Matrix-Element, ist für ein skalares Element im Abschnitt 10.3 beschrieben und für Strukturen in Abschnitt 10.5.

Bei der Anordnung im Speicher müssen Speicherplatz und Adressierungsgrenze für jedes Element erfüllt sein. Das kann dazu führen, daß zwischen den Elementen Füllinformation entsteht, die durch die Anordnung der Elemente in einer Matrix bedingt ist. In einem solchen Fall entsteht auch hinter dem letzten Element der Matrix die gleiche Füllinformation wie zwischen den Elementen; der Speicherplatz dafür gehört mit zum Speicherbedarf der Matrix (siehe Bild 10-30).




```
Speicherbedarf = n (e + f)

mit e  Speicherbedarf eines Elements
    f  Speicherplatz bis zur nächsten Speichergrenze
    n  Anzahl der Matrix-Elemente
```

Bild 10-30 Speicherbedarf für eine Matrix

Die Elemente einer Matrix mit mehr als einer Dimension werden so angeordnet, daß sich von Anfang des Speicherbereichs aus gesehen der rechts stehende Index am schnellsten ändert, also z.B.

```
A(1,1) A(1,2) A(1,3) A(2,1) A2,2) A(2,3)
```

Bei der Verwendung von BIT UNALIGNED in Verbindung mit Datensätzen sind einige Besonderheiten zu beachten (siehe Abschnitt 10.5.5).

Bei der Datensatz-Ausgabe wird die Füllinformation am Ende einer Matrix Bestandteil des Datensatzes. Wird jedoch ein einzelnes Element ausgegeben (skalares Element oder eine Struktur), so wird die Füllinformation am Ende nicht Bestandteil des Datensatzes (siehe Beispiel).

```
                                                  implizit

DCL 1  MATRIX   DIMENSION(1),
       2 FEST      FIXED BINARY,                  ALIGNED
       2 ZEICHEN   CHARACTER(1);                  UNALIGNED

                                                  Ergebnisse
WRITE FILE(DATEI) FROM(MATRIX);                   Datensatzlänge = 4 Byte
WRITE FILE(DATEI) FROM(MATRIX(1));                Datensatzlänge = 3 Byte
```

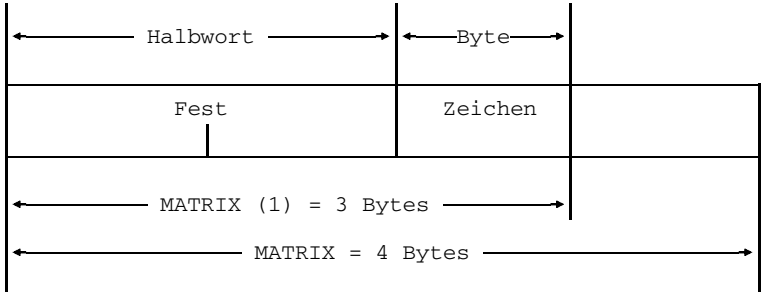


Bild 10-31 Beispiel für Füllinformation am Ende von Matrizen

10.5 Strukturen (STRUCTURE)

Im Abschnitt 10.5.1 ist der Speicherbedarf für Strukturen und die Ausrichtung auf Speichergrenzen beschrieben. In weiteren Abschnitten wird auf die entsprechenden Konsequenzen in einigen Anwendungsbereichen eingegangen.

10.5.1 Speicherbedarf

Bei der Ermittlung des Speicherbedarfs einer Struktur wird bei den untersten Strukturen begonnen und bei der nächst höheren Strukturstufe fortgesetzt, bis die Haupt-Struktur erreicht ist. Dazu werden für jede Unter-Struktur und für die Haupt-Struktur folgende Schritte ausgeführt.

- Die Speichergrenze, auf der eine Struktur-Stufe beginnt, bestimmt sich aus der höchsten Speichergrenze, die bei ihren Gliedern gefordert ist.
- Bei der so ermittelten Speichergrenze beginnend, wird für jedes Glied der Struktur der von ihr benötigte Speicherplatz zugewiesen unter Beachtung der geforderten Speichergrenze.
- Der Speicherplatz endet unmittelbar hinter dem Speicherplatz des letzten Gliedes, d.h. ein Füllfeld am Ende einer Struktur zählt nicht mit zur Länge der Struktur.
- Damit sind für eine Unter-Struktur Speichergrenze und Speicherbedarf bestimmt. Soweit noch eine höhere Struktur-Stufe vorhanden, wird diese Struktur mit diesen Speicherforderungen Glied der höheren Struktur.

Füllfelder werden immer dann eingeschoben, wenn das Ende des vorhergehenden Elements auf einer anderen Speichergrenze endet, wie für das nachfolgende Element gefordert wird. Der Inhalt dieser Füllfelder ist undefiniert.

Wird eine Struktur als Datensatz ausgegeben, so werden stets Vielfache von Bytes ausgegeben. Dies kann in Verbindung mit einem Element, das die Attribute BIT NONVARYING, UNALIGNED besitzt, zu Fehlern führen (siehe Abschnitt 10.5.5).

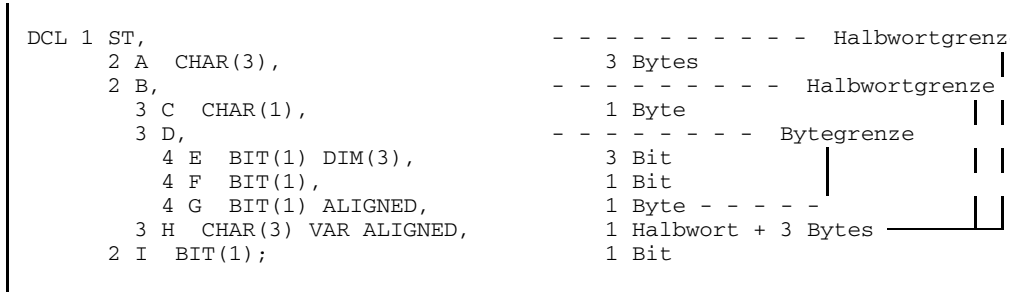


Bild 10-32 Beispiel für die interne Darstellung einer Struktur

Bild 10-33 Interne Darstellung der Struktur gemäß Bild 10-32

10.5.2 Überlagerung

Eine Überlagerung bedeutet allgemein, daß auf einen Speicherplatz mit zwei oder mehr Variablen zugegriffen wird, die jede eine eigene Datenbeschreibung haben. Eine Überlagerung kann in folgenden Fällen auftreten:

1. Mit Hilfe des Attributes DEFINED
2. Durch die eingebauten Funktionen STRING und SUBSTR
3. Durch Variable mit dem Attribut BASED
4. Bei der Parameter-Übergabe
5. Bei der Datensatz-Eingabe
   - durch die Anweisung READ
   - durch die Anweisung READ SET
6. Bei EXTERNAL-Variablen gleichen Namens

Bei der Überlagerung ist es erforderlich, daß die Variablen, die auf den gleichen Speicherplatz zugreifen, auch übereinstimmen. Bei der Datensatzeingabe müssen sie mit der Struktur des Datensatzes übereinstimmen und man kann für die Ermittlung der Übereinstimmung davon ausgehen, daß der Datensatz die gleiche Datenbeschreibung besitzt, wie die Variable aus der er entstanden ist (wie die Variable die ausgegeben wurde).

Bei den Überlagerungen gemäß den Punkten 1, 2 und 4 wird die Übereinstimmung vom Compiler geprüft und ggf. eine Meldung ausgegeben. Beim Punkt 4 gilt diese Prüfung nur innerhalb einer externen Prozedur. Bei der Überlagerung durch BASED-Variable und durch Datensatz-Eingabe gemäß den Punkten 3 und 5 ist eine Prüfung durch den Compiler nicht möglich und für die Übereinstimmung muß der Benutzer selbst sorgen. Solange er dabei die gleichen Regeln einhält, wie bei der Überlagerung durch DEFINED, ist die Übereinstimmung gegeben. In den anderen Fällen muß der Benutzer die Übereinstimmung anhand der internen Darstellung überprüfen.

Die im Abschnitt 10.5.1 beschriebene interne Struktur-Darstellung unterstützt in jedem Fall die in PL/I erlaubte Überlagerung von Haupt- und Unterstruktur, wenn beide gleiche Datenbeschreibungen haben. Dies ist im Bild 10-34 anhand eines Beispiels gezeigt. Die Haupt-Struktur Y und die Unter-Struktur X.B haben gleiche Datenbeschreibungen und können überlagert werden.

```
                                                    implizit        Speicherbedarf

DCL 1 X,
      2 A       CHAR(2),                            ALIGNED        2 Bytes
      2 B,
        3 C     BIT(1),                           UNALIGNED        1 Bit
        3 D     CHAR(3),                          UNALIGNED        3 Bytes
        3 E     FIXED BINARY (31,0);                ALIGNED        1 Ganzwort

DCL 1 Y,
      2 C       BIT(1),                           UNALIGNED        1 Bit
      2 D       CHAR(3),                          UNALIGNED        3 Bytes
      2 E       FIXED BINARY (31,0);                ALIGNED        1 Ganzwort


PUT SKIP(2) LIST (SIZE(X));                        12 }
PUT SKIP(1) LIST (SIZE(Y));                         8 } Ergebnis


WRITE FILE(DATEI) FROM(X.B);                       } identische Datensatzlängen
WRITE FILE(DATEI) FROM(Y);                         } = 2 Ganzwörter
```

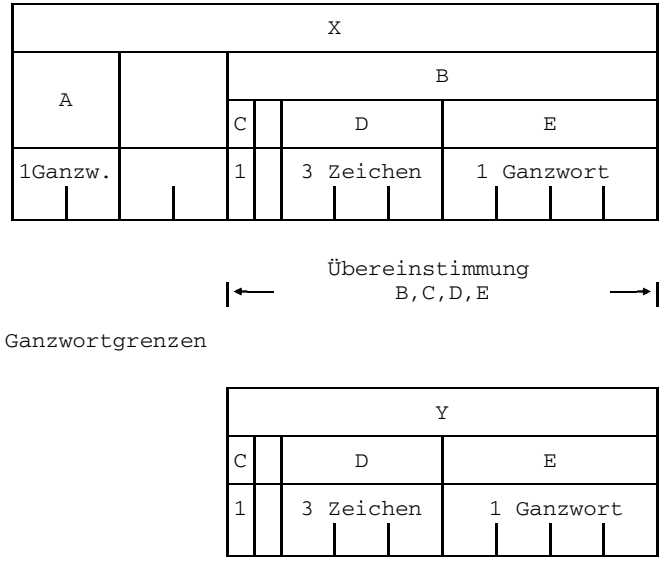


Bild 10-34 Beispiel für die Verträglichkeit von Haupt- und Unterstruktur bei gleicher Datenbeschreibung

10.5.3 Übereinstimmung am Anfang

Die beschriebene interne Darstellung von Strukturen ergibt für den Anfang von Strukturen eine Übereinstimmung, solange für die Strukturen auf der logischen Stufe 2 und deren Glieder gleiche Datenbeschreibungen gegeben sind. Für alle folgenden Strukturen der Stufe 2 und deren Glieder gilt diese Übereinstimmung nicht mehr.
In Bild 10-35 ist dazu ein Beispiel gegeben. In den beiden Haupt-Strukturen X und Y haben die Glieder X.F und Y.F unterschiedliche Datenbeschreibung. Damit gilt für die Strukturen X.C und Y.C der Stufe 2 die Übereinstimmung nicht mehr. Es sind also nur die Teile der Strukturen übereinstimmend, die vor X.C bzw. Y.C stehen. Die Teile der Strukturen, die auf X.C bzw. Y.C folgen (bis zum Ende der Haupt-Struktur) sind nicht mehr übereinstimmend. Dies gilt also auch für die Elemente D und E, obwohl sie gleiche Datenbeschreibungen haben. Dies geht sehr deutlich aus der graphischen Darstellung des Speicherbereichs im Bild 10-35 hervor.

```
                                                    implizit        Speicherbedarf

DCL 1 X,
      2 A       CHAR(3),                            UNALIGNED       3 Bytes
      2 B       CHAR(1),                            UNALIGNED       1 Byte
      2 C,
        3 D     BIT(1),                             UNALIGNED       1 Bit
        3 E     CHAR(3),                            UNALIGNED       3 Bytes
        3 F     FIXED BINARY (31,0);                  ALIGNED       1 Ganzwort

DCL 1 Y                 BASED(Z),
      2 A       CHAR(3),                            UNALIGNED       3 Bytes
      2 B       CHAR(1),                            UNALIGNED       1 Byte
      2 C,
        3 D     BIT(1),                             UNALIGNED       1 Bit
        3 E     CHAR(3),                            UNALIGNED       3 Bytes
        3 F     FLOAT BINARY (51);                    ALIGNED       1 Doppelwort
```

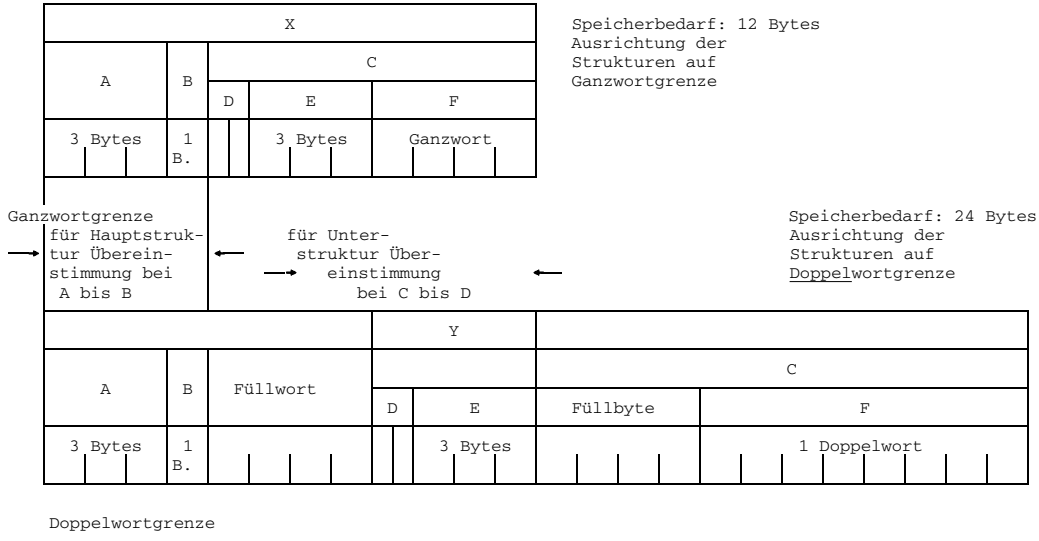


Bild 10-35 Beispiel für die Übereinstimmung des Anfangs einer Struktur

Betrachten wir in Bild 10-35 die Unter-Struktur X.C und Y.C getrennt von den Haupt-Strukturen, so stehen die Elemente D und E auf der logischen Stufe 2 und haben gleiche Datenbeschreibungen. Die Strukturen X.C und Y.C sind also in den Elementen D und E übereinstimmend.

Würden z.B. die Datensätze ausgegeben und eingelesen werden durch

```
WRITE FROM (X)
READ  INTO (Y)
```

so wäre ein Zugriff auf Y.A und Y.B erlaubt und bei

```
WRITE FROM (X.C)
READ  INTO (Y.C)
```

ein Zugriff auf Y.D und Y.E.

10.5.4 Selbstdefinierende Struktur

Strukturen, in denen bei einem oder mehreren Elementargliedern durch die REFER-Angabe vereinbart ist, daß die aktuelle Länge bzw. die aktuelle Dimension in einem anderen Elementarglied abgelegt ist, bezeichnen wir als selbstdefinierende Strukturen.

Hier wird nicht mit mehreren Variablen auf einen Speicherplatz zugegriffen, sondern es kann mit einer Variablen auf mehrere Speicherplätze unterschiedlicher Länge zugegriffen werden. Die Angaben zur Bestimmung der Länge sind in der Struktur selbst gespeichert, so daß ein Zugriff auch bei unterschiedlicher Speicherplatzlänge stets das richtige Ergebnis liefert.

```
                                                    implizit      Speicherbedarf

  DCL 1 S    BASED,
      2 A    CHAR(2),          INIT('——'),        UNALIGNED    2 Bytes
      2 B,
        3 C  FIXED BINARY (31,0),                   ALIGNED    1 Ganzwort
        3 D  CHAR(N REFER (C)) INIT((8)'='),      UNALIGNED    n Bytes
        3 E  FLOAT BINARY (51) INIT(0.5);           ALIGNED    1 Doppelwort


                                                    Ergebnisse
N = 4;    ALLOCATE S SET(Z4);
          PUT SKIP(2) LIST (SIZE(S));               24  ┐
N = 8;    ALLOCATE S SET(Z8);                           } Länge in Bytes
          PUT SKIP(2) LIST (SIZE(S));               32  ┘
N = 0;

                                                    Datensatzlängen

WRITE FILE(DATEI) FROM(Z4->S);                      24 Bytes
WRITE FILE(DATEI) FROM(Z8->S);                      32 Bytes

CLOSE FILE(DATEI);                                  ┐
READ FILE(DATEI) SET(Z);                            } Zugriff durch Z->S
READ FILE(DATEI) SET(Z);                            ┘
```

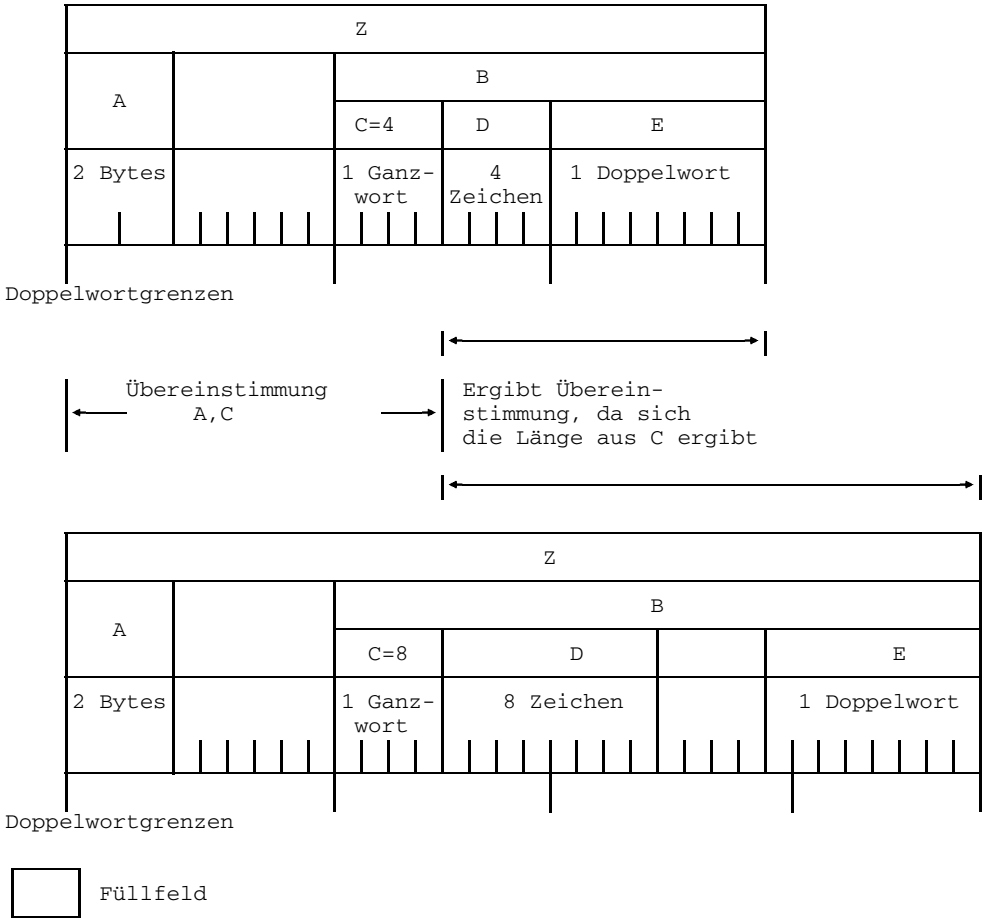


Bild 10-36 Beispiel für selbstdefinierende Struktur

Bei der Datensatz-Ausgabe können durch eine selbstdefinierende Struktur Datensätze unterschiedlicher Länge entstehen. Die Länge errechnet sich jeweils aus den Werten, die beim WRITE bzw. LOCATE aktuell sind.

Bei der Datensatz-Eingabe mit READ SET kann auf jeden Datensatz mit der gleichen Variablen zugegriffen werden, mit der der Datensatz geschrieben wurde, unabhängig von seiner aktuellen Länge. Bei der Eingabe mit READ INTO muß der Variablen soviel Speicherplatz zugewiesen sein, daß sie den längsten Datensatz aufnehmen kann. Es tritt dann ggf. bei kürzeren Datensätzen die Bedingung RECORD auf, die mit "ON RECORD FILE(a);" ignoriert werden kann. Es kann dann auf die Variable korrekt zugegriffen werden.

10.5.5 Datensatz

Bei der Datensatz-Ausgabe durch WRITE FROM wird eine Kopie vom Inhalt des Speicherplatzes als Datensatz ausgegeben. Die Länge des Datensatzes ist stets ein vielfaches von einem Byte. Sie ergibt sich aus den in den Abschnitten 10.1 bis 10.5 gemachten Angaben. Füllinformation am Ende einer Struktur kann nur auftreten, wenn das letzte Element eine Matrix ist, eine skalare Bitfolge mit dem Attribut UNALIGNED oder ein Bereich (AREA), und dieses Element selbst Füllinformation enthält.

Ein so geschriebener Datensatz hat keine explizite Datenbeschreibung. Man kann jedoch davon ausgehen, daß er implizit die gleiche Datenbeschreibung besitzt, wie die Variable, aus der er entstanden ist.

Bei der Datensatz-Eingabe mit READ INTO wird im Speicherplatz der Ziel-Variablen eine Kopie des Datensatzes abgelegt, unabhängig davon, welche Datenbeschreibung die Ziel-Variable hat. Es kann dabei die RECORD-Bedingung auftreten. Bei einem Zugriff wird nun praktisch die Datenbeschreibung der Variablen dem im Speicherplatz abgelegten Datensatz überlagert. Dabei ist es erforderlich, daß die Datenbeschreibung der Variablen und die Datenbeschreibung, die der Datensatz implizit beim Erzeugen mitbekommen hat, gemäß den Regeln übereinstimmt.

Bei der Datensatz-Ausgabe durch LOCATE und der Datensatz-Eingabe durch READ SET gilt entsprechendes. Hier liegt jedoch der Speicherbereich der Variablen im Ein-Ausgabe-Puffer der Datei. Aus diesem Grund können nur Variable mit dem Attribut BASED verwendet werden.

Werden Dateien verarbeitet, die auf einer anderen Datenverarbeitungs-Anlage erstellt wurden, so kann die interne Darstellung von der hier beschriebenen Darstellung abweichen. Es ist dann erforderlich, die interne Darstellung in der Datei zu ermitteln und mit der hier beschriebenen auf Übereinstimmung zu prüfen.

Achtung

Werden Unter-Strukturen als Datensatz ausgegeben, die Elementarglieder mit dem Attribut BIT UNALIGNED besitzen, so wird ggf. Füllinformation bis zur Bytegrenze mit ausgegeben. Bei der Eingabe wird der Transport des Datensatzes in die Variable nicht in Bits, sondern in Bytes durchgeführt. Dabei kann der Inhalt anschließender Variabler der Ziel-Struktur zerstört werden oder es entsteht unverständliche Information.

```
DCL 1 ST,                          ┐
      2 A     BIT(1),              │
      2 B,                         │ implizit
        3 C   BIT(1),              } UNALIGNED
        3 D   BIT(1),              │
      2 E     BIT(1);              ┘
```

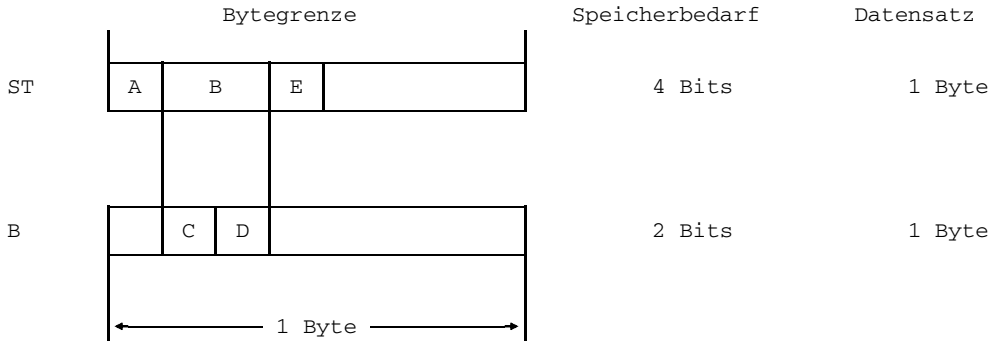


Bild 10-37 Beispiel für einen Datensatz einer Unter-Struktur mit BIT UNALIGNED

Würde bei dem Beispiel in Bild 10-37 mit WRITE FROM (ST.B) ein Datensatz geschrieben und dieser mit READ INTO (ST.B) gelesen, so würden die Variablen ST.A und ST.E mit überschrieben und damit der aktuelle Inhalt zerstört.

Würde der Datensatz in eine Haupt-Struktur gelesen, die die gleiche Struktur hat wie ST.B, so würde der Zugriff ungültige Information ergeben. Das gleiche gilt, wenn die Ziel-Variable skalar wäre und mit BIT (2) vereinbart wäre.

Es ist zu beachten, daß solche Unter-Strukturen auch durch Überlagerung in Verbindung mit DEFINED, BASED oder PARAMETER auftreten können. Ferner kann der vorstehend beschriebene Fall auch bei indizierten Variablen auftreten.

10.6 Beschreibung des Datentyps

Die Datenbeschreibung wird bestimmt durch die Daten-Attribute.
Für die interne Verarbeitung der Daten wird in einigen Fällen beim Programmablauf die Datenbeschreibung benötigt und daher in einer internen Form abgespeichert. Der Aufbau dieser internen Datenbeschreibung ist in Abschnitt 10.6.1 beschrieben.
Für den Datentyp, der mit dem Attribut PICTURE vereinbart ist, wird in einigen Fällen eine interne Darstellung der Maske erforderlich - die Maskenbeschreibung. Ihr Aufbau ist in Abschnitt 10.6.2 beschrieben.

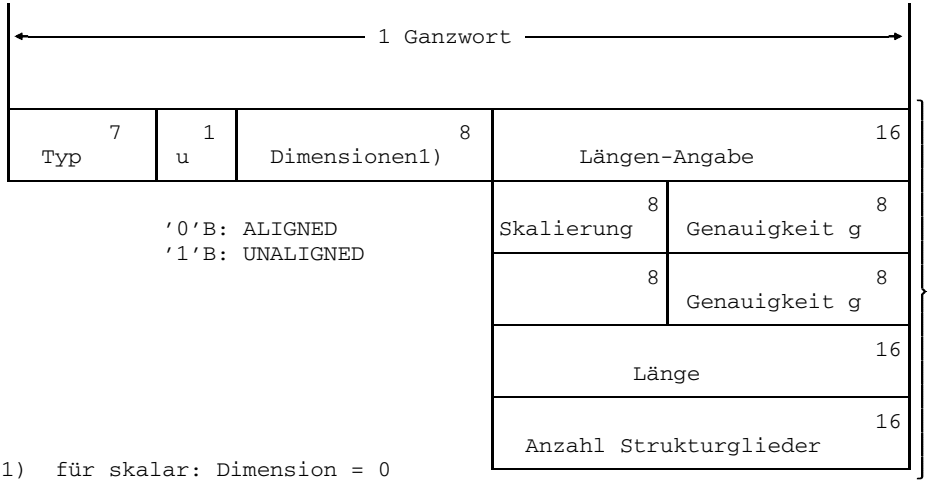
10.6.1 Datenbeschreibung

Die interne Darstellung der Datenbeschreibung ist ggf. bei der Übergabe von Parametern von Interesse. Sie besteht

- bei skalaren Variablen aus einem Kopfwort,
- bei Matrizen aus einem Kopfwort, dem für jede Dimension drei Wörter folgen,
- bei Strukturen aus einer Datenbeschreibung für die Haupt-Struktur, für die Unter-Strukturen und für die Elemente, und zwar in der Reihenfolge, in der sie im Quellprogramm niedergeschrieben wurden. Dabei ist zu beachten, daß sich die DIMENSION-Angabe auf tiefere Stufen vererbt und sich diese daher wie Matrizen verhalten.

Das Format des Kopfwortes ist in Bild 10-38 gezeigt. Die einzelnen Felder haben folgende Bedeutung:

- Typ BIT (7)
  Die Werte sind der Tabelle in Bild 10-39 zu entnehmen. Die weitere Gliederung der Kopfworte hängt vom Typ ab.
- u BIT (1)
  Dieses Bit gibt an, ob für den Datentyp ALIGNED (’0’B) oder UNALIGNED (’1’B) angegeben ist.
- Dimension BIT (8)
  Dieses Feld gibt an, wieviel Dimensionen vereinbart sind.
  Ist DIMENSION nicht angegeben, so hat dieses Feld den Wert (8)’0’B.
- Längen-Angabe
  Das Format und der Inhalt sind abhängig vom Typ. Im Bild 10-39 sind hierzu Angaben gemacht. Die Werte sind die gleichen, die im Quellprogramm angegeben sind, und zwar:
  – Genauigkeit g
  – Skalierung k; linkes Bit ist Vorzeichen (B-Komplement)
  – Länge bzw. max. Länge
  – Anzahl der unmittelbaren Glieder einer Struktur



1) für skalar: Dimension = 0

Bild 10-38 Datenbeschreibung

Beim Typ 58 folgt ein weiteres Ganzwort, das die Adresse der Maskenbeschreibung (siehe Abschnitt 10.6.2) enthält.

Typ mit u (sedez.)		Attribut			PRECISION	Längen-Angabe	
ALIGNED	UNAL						
00	01	PICTURE erweitert (siehe auch Typ = 3A bzw. 3B)					
04	05	BINARY	REAL	FIXED	2 Bytes	k	g
08	09				4 Bytes		
0A	0B			FLOAT	4 Bytes		
0C	0D				8 Bytes		g
0E	0F				16 Bytes		
12	13		COMPLEX	FIXED	2 x 2 Bytes	k	g
16	17				2 x 4 Bytes		
18	19			FLOAT	2 x 4 Bytes		
1A	1B				2 x 8 Bytes		g
1C	1D				2 x 16 Bytes		

Bild 10-39 Datentypen in der Datenbeschreibung (Teil 1)

Typ mit u (sedez.)		Attribut			PRECISION	Längen-Angabe	
ALIGNED	UNAL						
1E	1F	DECIMAL	REAL	FIXED		k	g
26	27			FLOAT	4 Bytes		g
28	29				8 Bytes		
2A	2B				16 Bytes		
2C	2D		COMPLEX	FIXED		k	g
34	35			FLOAT	2 x 4 Bytes		g
36	37				2 x 8 Bytes		
38	39				2 x 16 Bytes		
3A	3B	PICTURE (siehe auch Typ = 00 bzw. 01				Zeichen	
3C	3D	CHARACTER		NONVARYING		Zeichen	
3E	3F			VARYING		max.Zeichen[1)]	
40	41	BIT		NONVARYING		Bits	
42	43			VARYING		max.Bits[1)]	
44	45	POINTER				0	
48	49	OFFSET					
4C	4D	LABEL					
4E	4F	FORMAT					
50	51	ENTRY					
54	55	FILE					
56	57	AREA				Zeichen	
58	59	STRUCTURE				Glieder	

1) ohne Kopfwort (Längenangabe)

k mit Vorzeichen

Bild 10-39 Datentypen in der Datenbeschreibung (Teil 2)

Ist für eine Größe eine Dimension angegeben, so wird in dem Feld "Dimension" die Anzahl der Dimensionen angegeben. Für jede Dimension folgen dann drei weitere Wörter, die folgende Angaben enthalten:

- untere Indexgrenze gemäß Angabe im Quellprogramm,
- obere Indexgrenze gemäß Angabe im Quellprogramm,
- Adressenabstand der Elemente.

  bei BIT NONVAR UNAL: Abstand in Bits
  sonst: Abstand in Bytes

Das Format ist in Bild 10-40 gezeigt. Enthält eine Matrix eine Struktur, so wird diese Angabe für die Struktur und deren Glieder gemacht (siehe Beispiel in Bild 10-41).

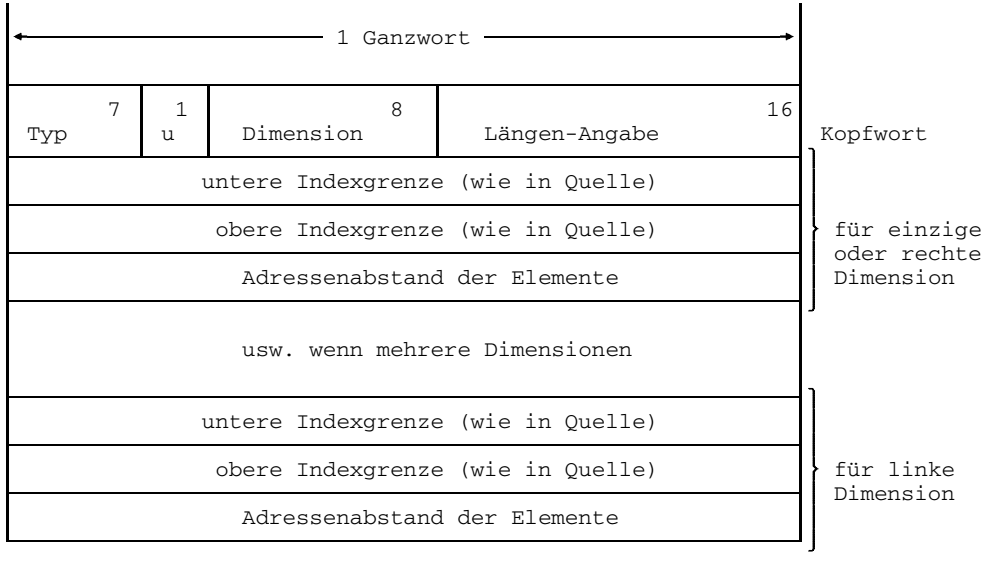


Bild 10-40 Datenbeschreibung für Matrizen

```
DCL  1 Haupt,

       2 Unter1,
       2 Unter2  DIM (3,4),

         3 Element,

       2 Unter3;
```

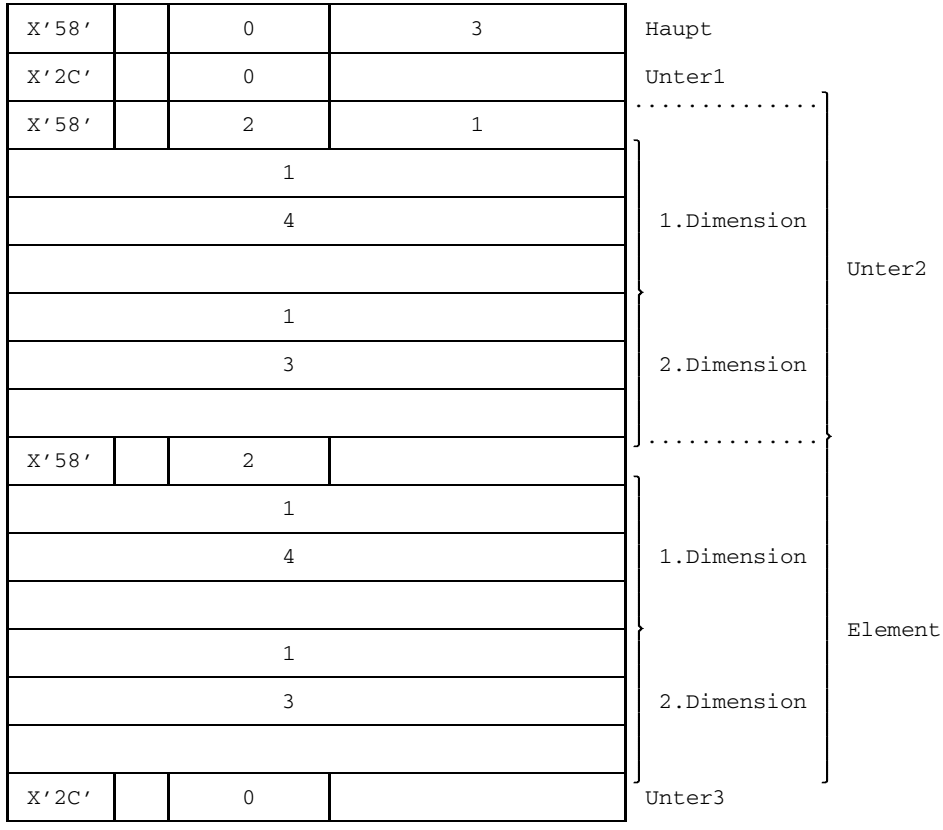
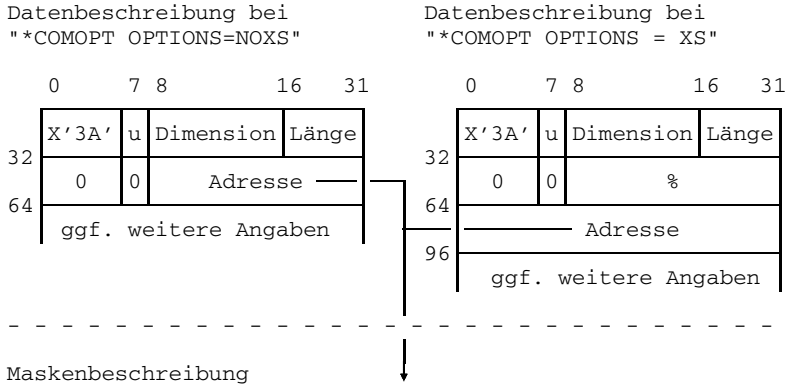


Bild 10-41 Beispiel für die Darstellung der Datenbeschreibung einer Struktur

10.6.2 Maskenbeschreibung

Ist bei der Maskenbeschreibung der Typ 58 angegeben, so folgt bei "*COMOPT OPTIONS = NOXS" im nächsten Ganzwort und bei "*COMOPT OPTIONS = XS" im übernächsten Ganzwort die Adresse einer Maskenbeschreibung. Die Struktur der Maskenbeschreibung ist im Bild 10-42 gezeigt. Die Elemente haben folgende Bedeutung:




```
DCL 1  Maskenbeschreibung  UNALIGNED,
       2  Datentyp         BIT (7),
       2  Ausrichtung      BIT (1),
       2  Maskentyp        BIT (8),
       2  Skalierung       BIT (8),
       2  Maskenlänge      FIXED BIN (15,0),
       2  Länge            BIT (16),
       2  Speicherlänge    BIT (16),
       2  Gleitzeichen,
          3  Mantisse      BIT (8),
          3  Exponent      BIT (8),
       2  Maske            CHAR (x REFER Maskenlänge);
```

Bild 10-42 Datenbeschreibung für Datentyp 0 und zugehörige Maskenbeschreibung

- Datentyp
  Hier wird der mit der Maske verbundene Attributsatz dargestellt:

```
15  REAL DECIMAL FIXED
19  REAL DECIMAL FLOAT mit g: 1 bis 6
20  REAL DECIMAL FLOAT mit g: 7 bis 16
21  REAL DECIMAL FLOAT mit g: 17 bis 33
22  COMPLEX DECIMAL FIXED
26  COMPLEX DECIMAL FLOAT mit g: 1 bis 6
27  COMPLEX DECIMAL FLOAT mit g: 7 bis 16
28  COMPLEX DECIMAL FLOAT mit g: 17 bis 33
30  CHARACTER NONVARYING
```

- Ausrichtung

```
'0'B: ALIGNED
'1'B: UNALIGNED
```

- Maskentyp

  X'01': Nur Maskenzeichen 9 V S + - $
  X'02': Nur Maskenzeichen wie Typ 1 und Z *, . / B
  X'03': Nur Maskenzeichen wie Typ 2 und S... +... -... $...
  X'04': Nur Maskenzeichen X

- Skalierung
  Resultierende Skalierung aus den Ziffernstellen rechts vom Maskenzeichen V und der Angabe F(n).

- Maskenlänge
  Länge der Maske nach Auflösung der Faktoren; jedoch zählen DB und CR jeweils nur als ein Zeichen.

- Länge
  Länge oder Genauigkeit des mit der Maske verbundenen Attributsatzes.

- Speicherlänge
  Länge der Interndarstellung der Variablen in Anzahl Zeichen.

- Gleitzeichen
  Die Bits in den beiden Feldern Mantisse und Exponent haben, wenn sie gesetzt ('1'B) sind, folgende Bedeutung (Bit 1 = linkes Bit).

  Bit 1: unbenutzt
  Bit 2: S... vorhanden
  Bit 3: +... vorhanden
  Bit 4: -... vorhanden
  Bit 5: $... vorhanden
  Bit 6: nur Z oder * vorhanden
  Bit 7: * vorhanden
  Bit 8: unbenutzt

- Maske
  Dieses Feld enthält die Maske nach Auflösung der Faktoren. Die Länge dieses Feldes steht im Feld Maskenlänge. Die Maskenzeichen werden in verschlüsselter Form dargestellt.

  Numerische Masken:

  (t) terminal: letztes Zeichen im gleitenden Teil
  (d) drifting: Zeichen im gleitenden Teil
  (s) static: nicht gleitendes Zeichen

```
X'00'  9      X'54'     $(t)      X'30'     S(t)
X'04'  Y      X'58'     $(d)      X'34'     S(d)
X'08'  Z      X'5C'     $(s)      X'38'     S(s)
X'0C'  *      X'60'     /(t)      X'3C'     +(t)
X'10'  E      X'64'     /(d)      X'40'     +(d)
X'14'  K      X'68'     /(s)      X'44'     +(s)
X'18'  T      X'6C'     .(t)      X'48'     -(t)
X'1C'  I      X'70'     .(d)      X'4C'     -(d)
X'20'  R      X'74'     .(s)      X'50'     -(s)
X'24'  CR     X'78'     ,(t)      X'84'     V
X'28'  DB     X'7C'     ,(d)
X'2C'  B      X'80'     ,(s)
```

  Alphanumerische Masken:

```
X'00'    A      X'04'     9         X'08'     X
```

10.7 Speicherverwaltung

Die in einem Quellprogramm explizit oder implizit vereinbarten Variablen benötigen einen Speicherplatz. Wann und in welchem Speicherbereich ihnen Speicherplatz zugeordnet wird, wird durch das Speicherattribut bestimmt.

Der interne Aufbau und die Verwaltung der verschiedenen Speicherbereiche ist in den folgenden Abschnitten beschrieben.

Speicherklasse		Zuordnung	Stapel	Freigabe bei	Stapel
STATIC	EXTERNAL / INTERNAL	Programmstart		Programmende	
AUTOMATIC	INTERNAL	Blockaktivierung	+1	Blockdeaktivierung	-1
CONTROLLED	EXTERNAL / INTERNAL	ALLOCATE	+1	FREE	-1
BASED(z)	INTERNAL	ALLOCATE		FREE	
		ALLOCATE IN Bereich		FREE IN Bereich	
				FREE Bereich	
		LOCATE SET (x)		LOCATE WRITE, CLOSE	
		READ SET (x)		READ,CLOSE, REWRITE	
PARAMETER	INTERNAL	Aufruf		Rückkehr	
DEFINED(x)	INTERNAL	wie x			
MEMBER	INTERNAL	wie Hauptstruktur			

Bild 10-43 Übersicht über Zuordnung und Freigabe von Speicherplatz

10.7.1 Statische Variable (STATIC)

Variable mit dem Attribut STATIC erhalten ihren Speicherbereich bereits bei der Übersetzung zugeordnet. Eine spezielle Speicherverwaltung ist hier nicht erforderlich.

10.7.2 Aktivierungssätze (stack, AUTOMATIC)

Der Speicherbereich für die Aktivierungssätze (englisch stack) nimmt für jede Blockaktivierung einen Aktivierungssatz auf. Er enthält alle Größen die für die Verwaltung einer Blockaktivierung gespeichert werden müssen. Der Speicherplatz wird automatisch zugeordnet, d.h. ohne Steuerung durch den Benutzer, wenn ein neuer Block aktiviert wird. Wird aus dem Block zurückgekehrt, so wird auch der Speicherplatz freigegeben. Der Aufbau des Speicherbereiches ist in Bild 10-44 gegeben.

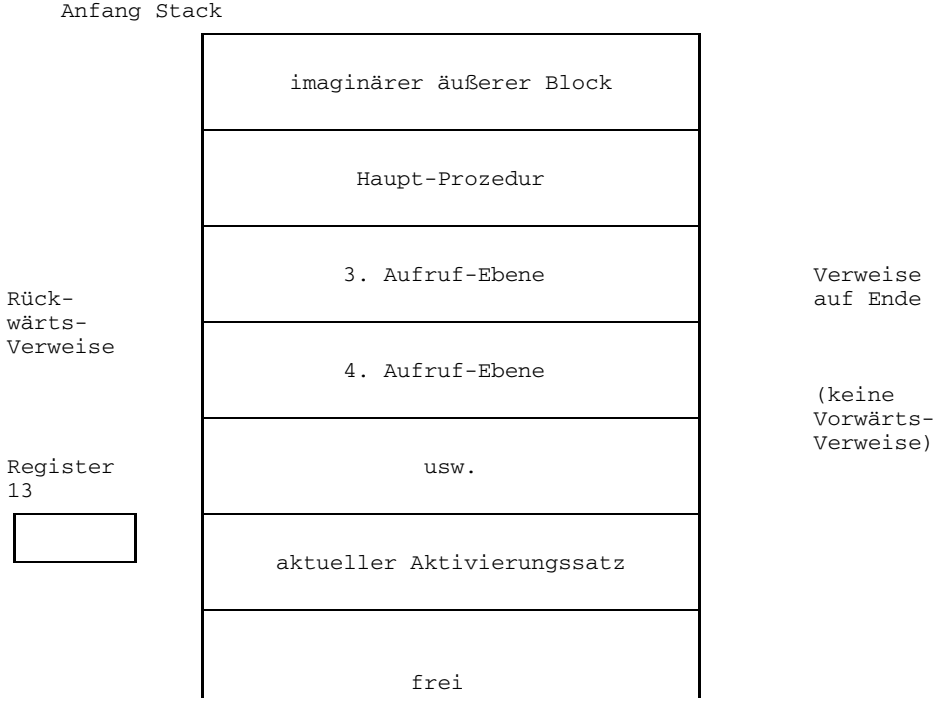


Bild 10-44 Aufbau des Speicherbereichs für die Blockaktivierungen

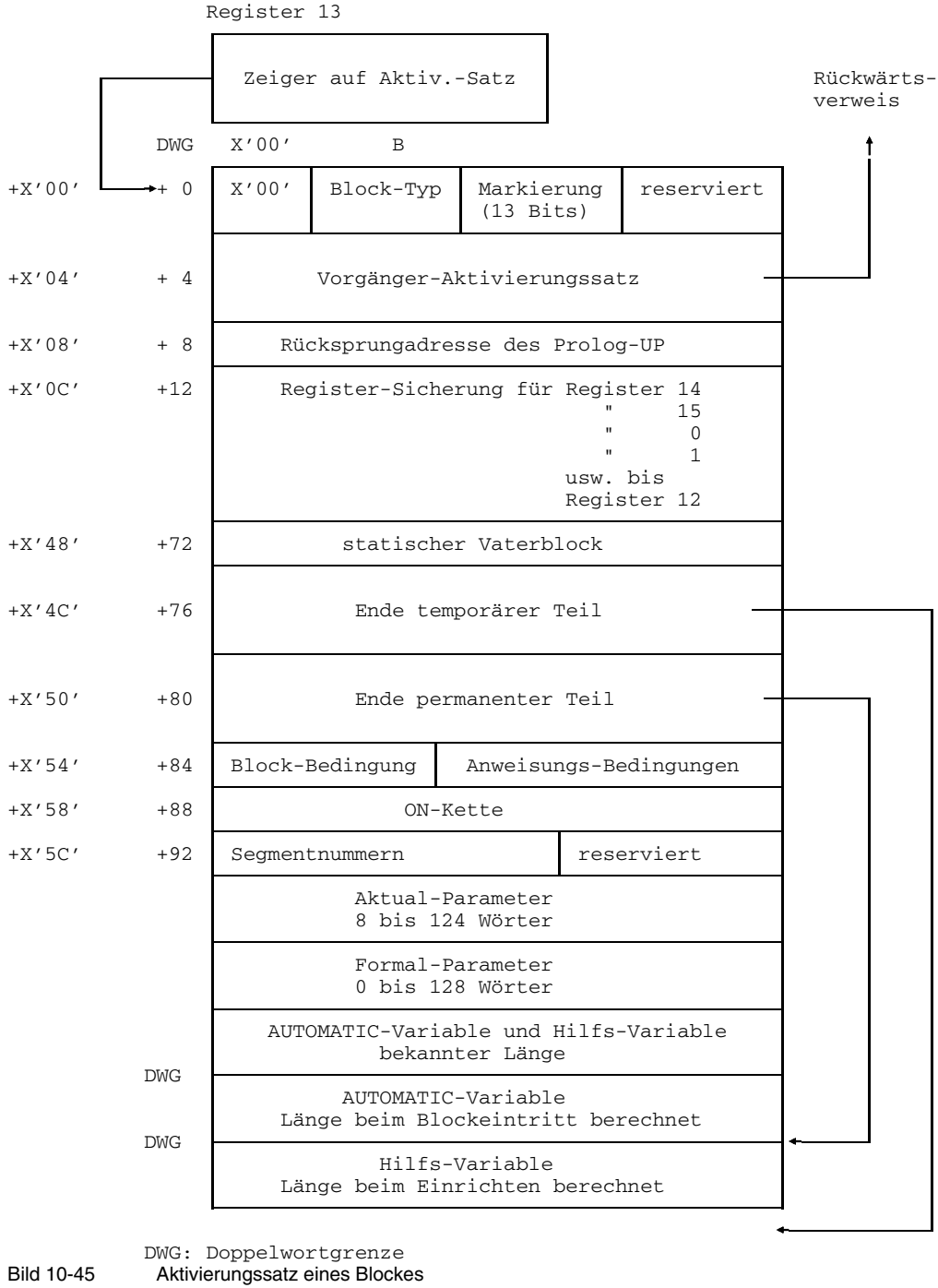


Bild 10-45 Aktivierungssatz eines Blockes

Ein Aktivierungssatz für einen PL/I-Block hat den in Bild 10-45 gezeigten Aufbau. Er beginnt stets auf Doppelwortgrenze. Die Felder haben folgende Bedeutung:

- Block-Typ

  X'01': Externe Prozedur oder PL/I-gerechte Assembler-Prozedur
  X'02': Interne Prozedur
  X'03': BEGIN-Block
  X'04': ON-Einheit

- Markierung
  Ein gesetztes Bit ('1'B) hat folgende Bedeutung (Bit 1 = linkes Bit):

  Bit 1: Initialisierungs-Aktivierung des PL/I-Laufzeit-Rahmens
  Bit 2: Bibliotheks-Prozedur (OPTIONS (LIBRARY))
  Bit 3: Aktivierung auf Grund einer Bedingung
  Bit 4: Der Block enthält ON-Einheiten
  Bit 5: Der Block enthält Anweisung mit Bedingungsvorspann
  Bit 6: Aktivierungen mit speziellem Sicherstellungsbereich (z.B. Fehlerbehandlung)
  Bit 7: Erstauftrag von Konvertier-Routinen
  Bit 8: Es ist eine Liste der Anweisungsnamen vorhanden
  Bit 9: Es wurde mit *COMOPT OPTION=ISO übersetzt
  Bit 10: Es wurde mit *COMOPT OPTIMIZE=ENABLING übersetzt
  Bit 11-13: resereviert

- Vorgänger-Aktivierungssatz
  Die Adresse weist auf den Aktivierungssatz des dynamisch vorangehenden Blockes.

- Rücksprungadresse für Prolog-UP
  Für die Initialisierung von AUTOMATIC-Variablen, wenn mehrere Eingänge vorhanden.

- Register-Sicherung
  Hier werden vom dynamisch nachfolgenden Block die Werte der Register 14 bis 15 und 0 bis 12 abgespeichert, wie sie beim Aufruf des Nachfolgers vorlagen.

- statischer Vaterblock
  Die Adresse weist auf die Aktivierung des Blockes, der in der Niederschrift der Quelle der statisch übergeordnete Block ist (für Zugriff auf dort vereinbarte Größen).

- Ende temporärer Teil
  Die Adresse weist auf das nächste Wort hinter dem Aktivierungssatz. Dies ist das nächste freie Wort oder der nächste Aktivierungssatz. Eventuell vorhandene temporäre Variable (Hilfs-Variable) sind eingeschlossen.

  Wenn mit "*COMOPT OPTIONS = NOXS" übersetzt wurde, so steht im linken Byte die Segmentnummer des temporären Teiles.

- Ende permanenter Teil
  Wie bei "temporäres Ende" jedoch sind die temporären Variablen ausgeschlossen. Sind keine temporären Variablen vorhanden, so enthalten "temporäres Ende" und "permanentes Ende" identische Adressen.

  Wenn mit "*COMOPT OPTION = NOXS" übersetzt wurde, so steht im linken Byte die Segmentnummer des permanenten Teiles.

- Block-Bedingung
  Dieses Feld ist nur von Bedeutung, wenn bei dem Feld Markierung (+2) Bit 5 = ’1’B ist. Ein gesetztes Bit (’1’B) gibt an, daß für den Block die aufgeführte Bedingung wirksam ist (Bit 1 = linkes Bit):

Bit 1 bis 4:	reserviert
Bit 5:	UNDERFLOW
Bit 6:	OVERFLOW
Bit 7:	ZERODIVIDE
Bit 8:	FIXEDOVERFLOW
Bit 9:	CONVERSION
Bit 10:	SIZE
Bit 11:	SUBSCRIPTRANGE
Bit 12:	STRINGRANGE
Bit 13:	STRINGSIZE
Bit 14 bis 16:	reserviert

- Anweisungs-Bedingungen
  Dieses Feld enthält die für die aktuelle Anweisung wirksamen Bedingungen. Beim Blockeintritt enthält er die gleichen Werte wie das Feld "Block-Bedingungen". Wenn für eine Anweisung durch explizite Angaben Abweichungen gegeben sind, so wird dies Feld für die Ausführung der Anweisung auf den für die Anweisung gültigen Stand gebracht und nachher wieder restauriert.

- ON-Kette
  Dieses Feld ist nur von Bedeutung, wenn beim Feld Markierung (+2) Bit 4 = ’1’B ist. Es enthält einen Zeiger auf die Kette der Verwaltungselemente für ON-Einheiten.

- Segmentnummern
  Dieses Feld ist nur von Bedeutung, wenn mit "*COMOPT OPTIONS = XS" übersetzt wurde. Ansonsten ist dieses Feld reserviert (Bit 1 = linkes Bit):

Bit 1-8:	Segmentnummer des temporären Teiles
Bit 9-16:	Segmentnummer des permanenten Teiles
Bit 17-24:	reserviert

- Aktual-Parameter
  Dieses Feld nimmt ggf. Parameter-Angaben auf für den Aufruf von Prozeduren. Der Aufruf ist unter "Parameter-Übergabe" beschrieben. Die Länge richtet sich nach dem

Aufruf mit der umfangreichsten Parameter-Angabe. Um implizite Aufrufe abzudecken, hat dies Feld eine minimale Länge von 8 Wörtern. Maximal sind 124 Wörter möglich. (Für die ersten vier Parameter werden die Register 1 bis 4 verwendet.)

- Formal-Parameter
  Dieses Feld ist nur dann vorhanden, wenn es für den Block Formal-Parameter gibt. Die Angaben in diesem Feld entsprechen denen im vorhergehenden Feld "Aktual-Parameter", jedoch sind in diesem auch die ersten vier Parameter enthalten.
- AUTOMATIC-Variable und Hilfs-Variable bekannter Länge
  Hier liegen die Speicherplätze für die AUTOMATIC-Variable des Blockes und die Hilfs-Variablen, die im Block benötigt werden, soweit deren Länge bei der Übersetzung bekannt ist. Im allgemeinen werden zuerst die AUTOMATIC-Variablen zugeordnet und danach die Hilfs-Variablen. Aus Optimierungsgründen kann eine andere Verteilung auftreten.

  Des weiteren befinden sich hier Zeiger auf die in den zwei nachfolgenden Bereichen stehenden AUTOMATIC- und Hilfs-Variablen.
- AUTOMATIC-Variable
  Hier liegen die Speicherplätze der AUTOMATIC-Variablen, deren Länge erst beim Blockeintritt ermittelt werden kann. Zeiger auf diese Variable werden unter den AUTOMATIC-Variablen bekannter Länge abgelegt.
- Hilfs-Variable
  Hier liegen die Speicherplätze der Hilfs-Variablen, deren Länge erst beim Einrichten ermittelt werden kann. Zeiger auf diese Variable werden unter der Hilfs-Variablen bekannter Länge abgelegt.

10.7.3 Standard-Bereich (CONTROLLED, BASED)

Der Standard-Bereich (standard area) nimmt folgende Größen auf:

- Variable mit dem Attribut CONTROLLED.
- Variable mit dem Attribut BASED, soweit diese nicht in einem benannten Bereich (AREA) angelegt werden.
- Hilfsgrößen, denen während des Objektlaufs Speicherplatz zugeordnet wird, wie z.B. Puffer für die Ein-Ausgabe.

Die Zuordnung von Speicherplatz für Variable mit dem Attribut CONTROLLED oder BASED wird ausschließlich vom Benutzer mit den Anweisungen ALLOCATE und FREE gesteuert. Wird ein Speicherplatz freigegeben, so kann dieser freie Speicherplatz zwischen belegtem Speicherplatz liegen. Dieses sind dann Lücken, die bei einer nachfolgenden Zuordnung wieder belegt werden können, sofern sie groß genug sind.

Der Standard-Bereich besteht mindestens aus einer Erst-Portion, die beim Programmstart festgelegt wird. Das erste Doppelwort ist undefiniert. Die nächsten 5 Doppelwörter nehmen die Verwaltungs-Information für den Standard-Bereich auf. Der restliche Teil steht für Zuordnungen zur Verfügung.

Wird weiterer Speicherplatz benötigt, so werden automatisch weitere Portionen angefordert. Die ersten beiden Ganzwörter jeder Erweiterungs-Portion enthalten Absolut-Adressen auf den Anfang bzw. das Ende der Vorgänger-Portion (siehe Bild 10-46).

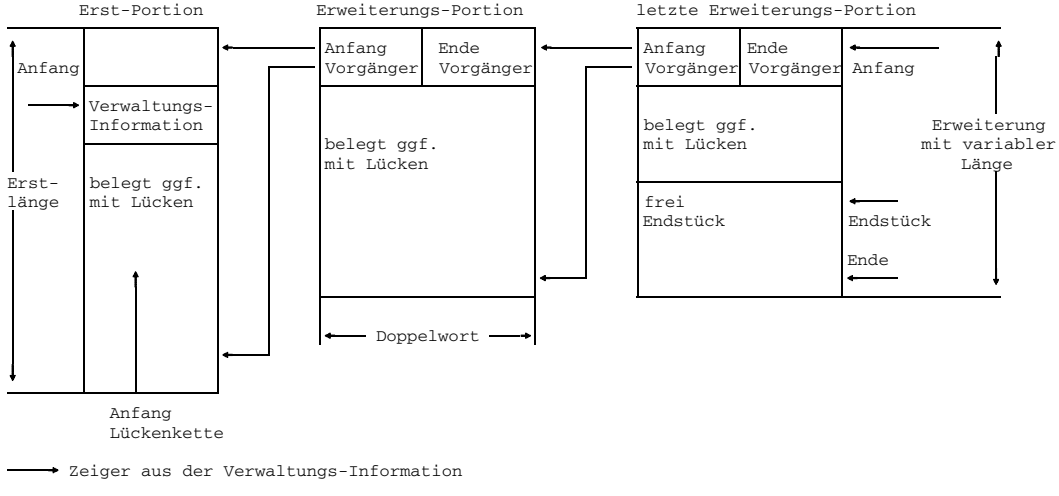


Bild 10-46 Aufbau des Standard-Bereiches (Prinzip)

Der Speicherplatz aller Portionen (ohne das erste Doppelwort jeder Portion und ohne die Verwaltungs-Information) bildet zusammen den für Zuordnungen zur Verfügung stehenden Speicherplatz des Standard-Bereichs. Einmal zugeordnete Portionen bleiben bis zum Ende des Objektlaufs zugeordnet.

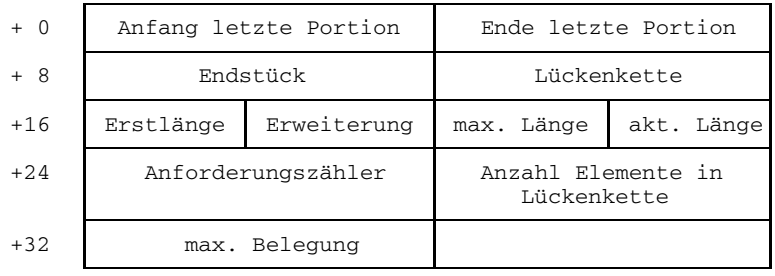


Bild 10-47 Aufbau der Verwaltungsinformation im Standard-Bereich

Die Verwaltungs-Information des Standard-Bereichs hat folgende Bedeutung (siehe dazu Bild 10-47):

- Anfang und Ende der letzten Portion
  Absolute Adressen der letzten Portion, die als letzte zugewiesen wurde. Ist nur die Erst-Portion vorhanden, so ist es diese.
- Endstück
  Absolute Adresse des noch nicht belegten Speichers am Ende des Standard-Bereichs. Das Endstück ist nicht Bestandteil der Lückenkette und kann in jeder Portion beginnen.
- Lückenkette
  Absolute Adressen beginnend mit der ersten Lücke in der Lückenkette. Die Lückenkette ist jeweils nach aufsteigenden Adressen geordnet und ggf. wurden Nachbarlisten vereinigt.

  Eine Lücke ist immer das Vielfache eines Doppelworts (8 Bytes) lang. Sie hat folgenden Aufbau:

  – erstes Ganzwort: Absolut-Adresse auf die nächste Lücke oder ’0’B
  – zweites Ganzwort: Länge der Lücke in Bytes einschließlich der zwei Ganzworte Verwaltungs-Information dieser Lücke
  – Rest: undefiniert
- Erstlänge, Erweiterung, max. Länge
  Diese Angaben sind der Steueranweisung "*RUNOPT STORAGE = AREA (Erstlänge, Erweiterung, max. Länge)" entnommen und bedeuten:

  Erstlänge: Gewünschte Seitenzahl für die erste Portion
  Erweiterung: Gewünschte Seitenzahl für eine Erweiterung

max. Länge: Maximal mögliche Länge

Der als "Erweiterung" angegebene Wert kann überschritten werden, wenn aktuell mehr benötigt wird, und kann unterschritten werden, wenn temporär weniger Speicherplatz vorhanden ist, dieser aber aktuell ausreicht.

- Anforderungszähler
  Dieser Zähler wird bei jeder Anforderung einer neuen Portion um den Wert 1 erhöht. Ist die Steueranweisung "*RUNOPT LIST = SUMMARY" angegeben, so wird dieser Wert am Ende des Programmlaufs ausgegeben.
- Anzahl Elemente in Lückenkette
  Hier wird die Anzahl der Lücken in der Lückenkette festgehalten. Sie dient der Fehlerprüfung.
- Maximale Belegung
  Länge des max. belegten Standardbereichs in Seiten (4K Byte). Ist die Steueranweisung "*RUNOPT LIST = SUMMARY" angegeben, so wird dieser Wert unter "STANDARDAREA:" am Ende des Programms ausgegeben.

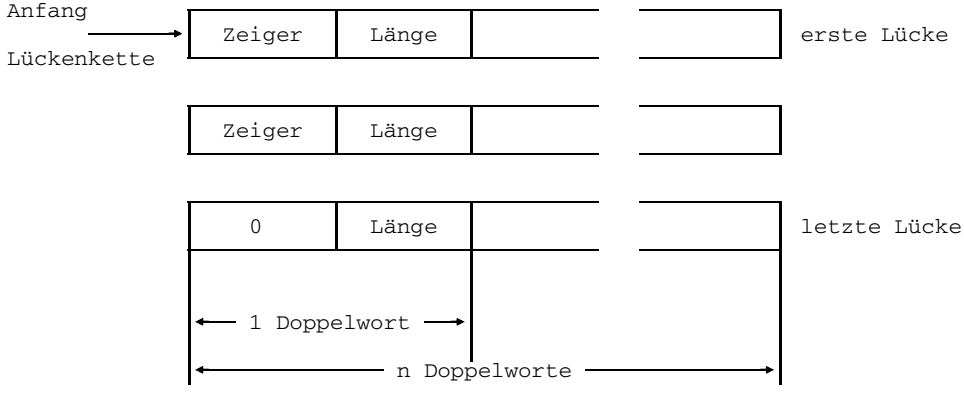


Bild 10-48 Aufbau der Lückenkette (Prinzip)

Der am Ende des Standard-Bereiches liegende freie Speicherplatz wird als Endstück bezeichnet und bildet die Endlücke. Sie gehört nicht zur Lückenkette. Ihre Anfangsadresse steht in der Verwaltungs-Information.

Soll durch die Anweisung ALLOCATE Speicherplatz im Standard-Bereich zugeordnet werden, so werden folgende Schritte in der angegebenen Reihenfolge versucht:

1. Ist noch Platz im Endstück, so wird daraus zugeordnet.
2. Sind für den Standard-Bereich noch nicht 3/4 des Speicherraumes belegt, der maximal dafür vorgegeben ist, so wird eine weitere Portion von soviel Seiten angefordert, wie "Erweiterung" angibt. Sind schon 3/4 des Standard-Bereichs belegt, wird geprüft, ob eine Lückenkette vorhanden ist.

3. Ist eine Lückenkette vorhanden, so wird geprüft, ob die Speicheranforderung aus der nach aufsteigenden Adressen sortierten Lückenkette befriedigt werden kann. Wird eine ausreichend große Lücke gefunden, so wird der Speicherplatz zugeordnet. Entspricht die Lücke genau der Anforderung, so wird sie aus der Lückenliste entfernt, ist sie größer, so wird sie entsprechend verkleinert.
4. Ist keine Lückenkette vorhanden, wird eine weitere Portion von soviel Seiten angefordert, wie "Erweiterung" angibt. Ist dies nicht möglich, so werden soviel Seiten angefordert, wie für die Zuordnung erforderlich ist. Ist einer dieser Versuche erfolgreich, so wird aus dem jetzt größeren Endstück zugeordnet.
5. Sind alle Versuche erfolglos, so wird die Bedingung STORAGE gesetzt.

Soll durch die Anweisung FREE Speicherplatz im Standard-Bereich freigegeben werden, so wird in folgenden Schritten vorgegangen:

1. Grenzt der Speicherplatz an das Endstück, so wird es mit ihm vereinigt.
2. Der freigegebene Speicherplatz wird als Lücke sortiert in die Lückenkette eingehängt und ggf. mit Nachbarlücken vereinigt.

Durch die Funktion ADUMP (siehe Kapitel 11) kann der Inhalt des Standard-Bereichs ausgedruckt werden.

Durch die Steueranweisung "*RUNOPT LIST = SUMMARY" wird erreicht, daß am Ende des Programmlaufs die maximal belegte Größe (x) des Standard-Bereichs in Seiten und Anzahl der Portionen (y) ausgegeben wird:

```
STANDARD AREA: x PAGES; SYSTEM CALLS: y REQM
```

10.7.4 Benannter Bereich (AREA)

Bei einem benannten Bereich handelt es sich um eine vom Benutzer mit dem Attribut AREA vereinbarte Bereichs-Variable. Dabei wird durch das Speicher-Attribut auch vereinbart, in welchem Speicherbereich dieser Bereich angeordnet werden soll.

In einem Bereich kann durch die Anweisung ALLOCATE Variablen mit dem Attribut BASED Speicherplatz zugeordnet werden. Dies kann wiederum ein Bereich sein, so daß sich Bereichs-Verschachtelungen ergeben können.

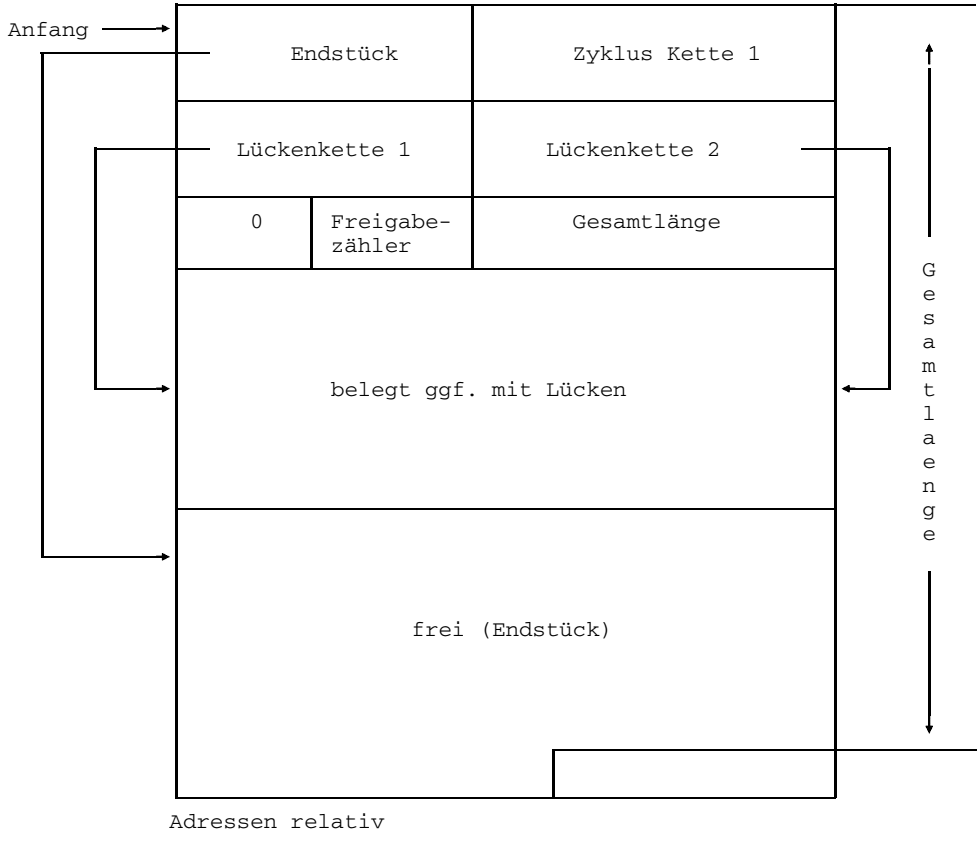


Bild 10-49 Aufbau eines benannten Bereichs

Die Länge eines Bereichs ergibt sich aus der Längenangabe beim Attribut AREA (n) in Byte zuzüglich 24 Bytes für die Verwaltungs-Information. Die Speicherzuordnung erfolgt stets in Vielfachen von einem Doppelwort.

Die Felder des Bereichs haben folgende Bedeutung (siehe dazu Bild 10-49):

- Endstück
  Relative Anfangsadresse des Endstückes
- Zyklus Kette 1
  Die Durchsuchung der Lückenkette 1 beginnt nicht am Anfang, sondern an der Stelle, an der zuletzt Speicherplatz zugeordnet wurde. Dazu wird die Adresse des Vorgängers dieser Lücke festgehalten und die Kette zyklisch durchsucht. Dies stellt eine Optimierung des Suchvorganges dar.
- Lückenkette 1
  Relative Anfangsadresse der Lückenkette 1. Sie enthält alle Lücken, soweit sie nicht in Lückenkette 2 sind. Sie ist nach aufsteigenden Adressen sortiert.
- Lückenkette 2
  Relative Anfangsadresse der Lückenkette 2. Sie enthält die zuletzt entstandene Lücke und ggf. weitere Lücken der gleichen Länge. Die erste Lücke ist die jüngste Lücke.
- Freigabe-Zähler
  Bei jeder Freigabe wird dieser Zähler um 1 erhöht. Beim Auftreten einer AREA-Bedingung wird der Zählerstand festgehalten und nach der Rückkehr aus der AREA-Bedingung wird geprüft, ob sich der Zähler erhöht hat (d.h. eine Freigabe durchgeführt wurde). Ist der Zählerstand unverändert, so wird die Bedingung ERROR gesetzt (Vermeidung von Endlosschleifen).
- Gesamtlänge
  Die Gesamtlänge ergibt sich aus n+24 Bytes, wobei n die Angabe aus dem AREA-Attribut ist.
- Restbereich
  Der restliche Teil des Bereichs steht für Speicherzuordnungen an Variablen zur Verfügung. Er kann aus einem belegten und einem freien Teil bestehen. Im belegten Teil können Lücken enthalten sein. Der freie Teil (Enstück) ist nicht Bestandteil einer Lükkenkette.

Speicherplätze werden stets in Vielfachen von 1 Doppelwort zugeordnet; damit sind auch die Lücken Vielfache eines Doppelwortes. Jede Lücke ist entweder in der Lückenkette 1 oder der Lückenkette 2 enthalten. Alle Lücken einer Lückenkette sind durch Vorwärtsverweise verbunden. Die Adresse der ersten Lücke (der Anker) einer Kette steht in der Verwaltungs-Information. Die Lücken haben den gleichen Aufbau wie in Abschnitt 10.7.3 für den Standard-Bereich beschrieben.

Der am Ende des Bereichs liegende freie Speicherplatz wird als Endstück bezeichnet und bildet die Endlücke. Sie gehört nicht zu einer Lückenkette. Ihre Anfangsadresse steht in der Verwaltungs-Information.

Alle Adressen in der Verwaltungs-Information und in den Lücken sind relativ zum Bereichsanfang. Damit können Bereiche als Gesamtheit anderen Bereichen zugewiesen und in Dateien als Datensatz zwischengespeichert werden.

Soll einer Variablen in einem Bereich Speicherplatz zugeordnet werden (nur durch die Anweisung ALLOCATE IN (Bereich)), so wird in der nachstehenden Reihenfolge vorgegangen:

1. Ist die Lückenkette 2 nicht leer, so wird geprüft, ob die Länge der Lücken (alle Lükken dieser Kette haben die gleiche Länge) der geforderten Länge entspricht. Sind beide gleich lang, so wird der Speicherplatz der ersten Lücke verwendet und diese aus der Lückenkette 2 entfernt.

   Sind die Längen nicht gleich, so wird die Lückenkette 2 aufgelöst und ihre Lücken nach aufsteigenden Adressen in die Lückenkette 1 einsortiert. Dabei werden benachbarte Lücken vereinigt.

2. Es wird in der Lückenkette 1 eine Lücke gesucht, die gleich oder größer der geforderten Länge ist. Wird eine gefunden, so wird deren Speicherplatz der Variablen zugeordnet und die Lücke entfernt bzw. verkürzt.

   Aus Optimierungsgründen wird die Lückenkette 1 zyklisch durchsucht, beginnend an der Stelle, an der beim letzten Mal Speicherplatz aus der Lückenkette zugeordnet wurde. Dazu wird bei der Zuordnung die Adresse des Vorgängers dieser Lücke im Feld "Zyklus Kette 1" festgehalten und mit dem zyklischen Suchen beim Nachfolger dieser Lücke begonnen.

3. Ist das freie Endstück groß genug, so wird dort Speicherplatz zugewiesen.

4. Sind alle Versuche erfolglos, so wird die Bedingung AREA gesetzt.

Soll der Speicherplatz einer Variablen in einem Bereich freigegeben werden, so wird in folgenden Schritten vorgegangen:

1. Grenzt der Speicherplatz an das Endstück, so wird er diesem hinzugefügt. Die Lükkenkette 2 wird aufgelöst, wie unter 4. beschrieben.

2. Ist die Lückenkette 2 leer, so wird der Speicherplatz als neue Lücke eingefügt.

3. Ist die Lückenkette 2 nicht leer und ist die Länge der Lücken gleich der Länge des freizugebenden Speicherplatzes, so wird die neue Lücke am Anfang der Lückenkette eingefügt.

4. Ist die Lückenkette 2 nicht leer und ist die Länge der Lücken ungleich der Länge des freizugebenden Speicherplatzes, so wird die Lückenkette 2 aufgelöst und deren Lücken nach steigenden Adressen in die Lückenkette 1 eingeordnet. Benachbarte Lücken werden zu einer vereinigt.

   Danach wird die neue Lücke in die nunmehr leere Lückenkette eingefügt.

10.7.5 Verweiskette CONTROLLED-Variable

Bei einer Variablen mit dem Attribut CONTROLLED wird die Speicherzuordnung durch die Anweisung ALLOCATE durchgeführt. Wird die Anweisung mehrfach für die gleiche Variable gegeben, so wird jedesmal neuer Speicherplatz zugeordnet; der vorhergehende bleibt dabei erhalten. Der Speicherplatz wird "gestapelt" (englisch: stacked). Bei einem Zugriff gilt immer die letzte Zuordnung.

Durch die Anweisung FREE wird jedoch der zuletzt zugeordnete Speicherplatz freigegeben. Soweit vorhanden wird der davor zugeordnete Speicherplatz zum aktuellen Speicherplatz. Es ist sichergestellt, daß immer genau so viel Speicherplatz freigegeben wird, wie zugeordnet wurde.

Beim Start des Programms wird für jede CONTROLLED-Variable eine STATIC-Variable bzw. bei CONTROLLED (PLI1GLOBAL(n)) ein Pseudoregister-Eintrag mit der Länge 4 Bytes angelegt - der Anker. Ist der CONTROLLED-Variablen Speicherplatz zugeordnet, so enthält der Anker die absolute Adresse des letzten Kettengliedes einer Verweiskette, über die die aktuelle Zuordnung erreicht wird, sonst einen Leer-Zeiger.

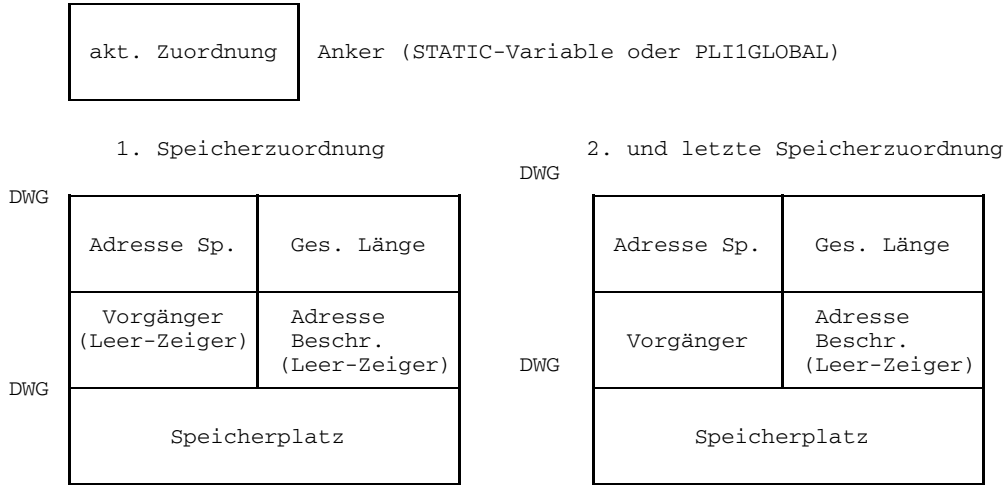
Auf Grund einer ALLOCATE-Anweisung wird einer Variablen mit dem Attribut CONTROLLED im Standard-Bereich ein Speicherplatz zugeordnet, dessen Größe sich aus den Attributen ergibt. Die Größe kann bei jeder Zuordnung einen anderen Wert haben. Enthält der Attributsatz der CONTROLLED-Variablen eine Angabe, die eine variable Größe darstellt (also einen Ausdruck), so wird dieser vor der Speicherzuordnung berechnet. In diesem Fall wird im Standard-Bereich zusätzlich eine Kopie der Datenbeschreibung abgelegt, in der die variablen Größen durch die berechneten Werte ersetzt sind. Sie werden für den Zugriff auf die CONTROLLED-Variable benötigt. Die interne Darstellung der Datenbeschreibung ist in Abschnitt 10.6 beschrieben. Außerdem wird im Standard-Bereich ein Kettenglied mit 2 Doppelwörtern eingerichtet, das letztes Glied der Verweiskette der CONTROLLED-Variablen wird. Der Anker wird neu gesetzt und weist auf das letzte Kettenglied. Ein Kettenglied enthält folgende Verwaltungs-Information:

- Im ersten Ganzwort steht ein Absolut-Zeiger auf den zugeordneten Speicherplatz.
- Das zweite Ganzwort enthält die Gesamtlänge des zugeordneten Speicherplatzes, inclusive Länge des Kettenglieds und, falls eine Datenbeschreibung mitgeführt wird, deren Länge.
- Im dritten Ganzwort steht ein Absolut-Zeiger, der auf das vorhergehende Ketten-Element weist (Rückwärts-Verweis). Beim ersten Glied enthält dieses Ganzwort einen Leerzeiger.
- Das vierte Ganzwort enthält einen Leerzeiger oder einen Absolut-Zeiger auf die Datenbeschreibung.

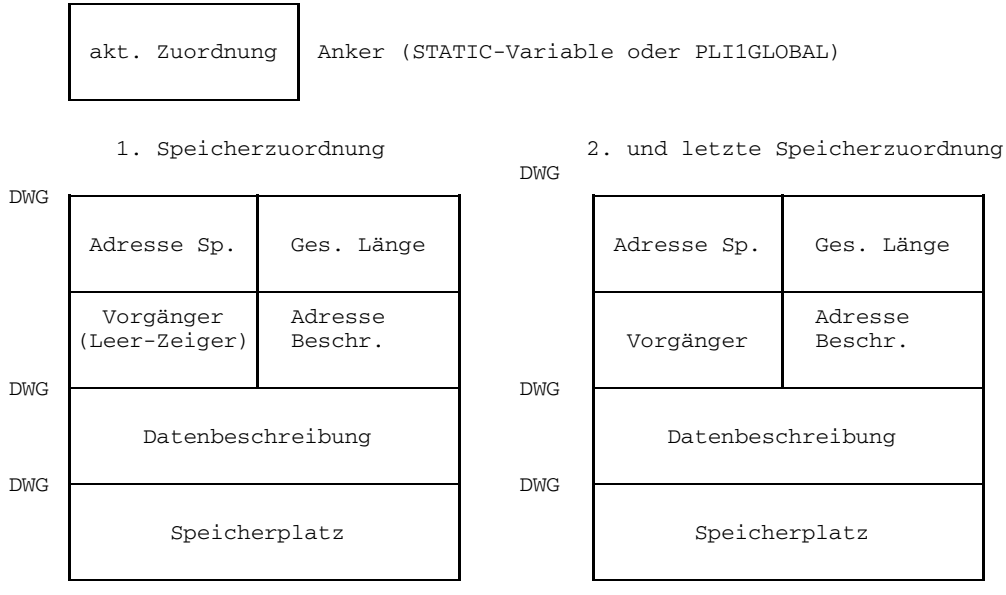
Kettenglied, Datenbeschreibung und Speicherplatz der Variablen werden in einem zusammenhängenden Block zugeordnet.

Verweiskette für CONTROLLED-Variable im Standard-Bereich (Beispiel)

Fall 1: Ohne Datenbeschreibung



Fall 2: Mit Datenbeschreibung



DWG: Doppelwortgrenze

Durch eine FREE-Anweisung wird der aktuelle Speicherplatz freigegeben und, soweit vorhanden, auch der Speicherplatz für die Kopie der Datenbeschreibung. Die Länge des

freizugebenden Speicherplatzes ergibt sich aus der Gesamtlänge, die im Kettenglied festgehalten wurde. Der Anker wird auf den Vorgänger in der Verweiskette gesetzt oder erhält einen Leer-Zeiger, wenn kein Vorgänger mehr vorhanden ist.

Bei einem Bezug auf eine CONTROLLED-Variable wird der Speicherplatz und ggf. die Datenbeschreibung über den Anker und das Kettungsglied der Verweiskette gefunden.

11 Dienstleistungen

In diesem Kapitel sind Leistungen beschrieben, die dem Benutzer in Form vorgefertigter und bereits übersetzter Prozeduren zur Verfügung stehen. Diese Leistungen gehen über den Leistungsumfang von PL/I hinaus und es ist beim Programmaustausch nicht zu erwarten, daß sie auf einem anderen Rechner in gleicher Form erbracht werden.

Die hier beschriebenen Prozeduren können in einem PL/I-Programm als Unterprogramm (CALL) bzw. als Funktion aufgerufen werden; die Prozedur wird beim Binden automatisch in das Programm mit einbezogen.

Die Prozeduren sind unabhängig voneinander beschrieben und in alphabetischer Reihenfolge der Eingangsnamen sortiert. Die Zusammenhänge sind in entsprechenden Abschnitten beschrieben.

Bei den Datenbeschreibungen sind alle die Attribute angegeben, die in dem gegebenen Fall erforderlich sind. Die in eckigen Klammern eingeschlossenen Attribute werden durch die System-Vorgabe von PL/I ergänzt; sie können fortgelassen werden, solange nicht durch DEFAULT-Anweisungen die System-Vorgabe überschrieben wird. Soweit für die vollständige Datenbeschreibung weitere Attribute möglich sind, können diese den Erfordernissen gemäß gewählt werden.

Zu beachten ist, daß bei der Vereinbarung für einige der Prozeduren die Angabe

```
OPTIONS (LIBRARY)
```

zu machen ist. Sie ist im Kapitel 7 beschrieben.

BS2SRTA (a, b, c) BS2SRTB (a, b, c, d) BS2SRTC (a, b, c, e) BS2SRTD (a, b, c, d, e) BS2SRT (h,..., c, d, e)	a: SORT/MERGE-Anweisung CHAR (*) b: RECORD/ALLOC-Anweisung CHAR (*) c: Rückmeldung FIXED BIN PREC (31,0) c = 0: Lauf fehlerfrei c = 16: Fehlerabbruch d: Eingabeprozedur ENTRY ( ) e: Ausgabeprozedur ENTRY ( ) h: Steueranweisung für SORT ab V7.0

RUNTIME	1) 2)	Rechenzeit seit Programmstart in Sekunden
HEXDEC (a)	1) 2)	Bitfolge a nach Sedezimal-Zeichenfolge konvertieren
ERROUT	1)	Fehlertext der aktuellen ON-Einheit nach SYSOUT

TRACE (a) NOTRACE (a)	1)	Ablaufverfolgung ein- bzw. ausschalten: a: Buchstabenfolge P PROCEDURE-Trace C CALL-Trace R RETURN-Trace G GOTO-Trace L LABEL-Trace T auch nach Datenstation leerer String ≙ 'PCRGL'
SNAP	1)	Aufrufverschachtelung nach SYSOUT

RDUMP (a,b)	1)	b Zeichen ab Zeiger a	nach SYSLST
ADUMP	1)	Standard Area	Speicherauszug in Sedezimal- und Zeichendarstellung
SDUMP	1)	Stack	

PLIRETC (a)	1)	Monitor-Job-Variable auf Wert a setzen

CMD (a, b, c)	1)	BS2000-Kommando ausführen a: BS2000-Kommando CHAR (*) b: SYSOUT-Protokoll CHAR (*) c: Rückmeldung FIXED BIN (31)

1) mit OPTIONS (LIBRARY)
2) Funktion

Bild 11-1 Übersicht über die Prozeduren

ADUMP Speicherauszug aus der Standard-Area

```
ADUMP
```

Es wird der Inhalt der Standard-Area (CONTROLLED- und BASED-Variable) nach SYSLST in Sedezimal- und Zeichendarstellung ausgegeben.

Eingang

```
DCL    ADUMP      ENTRY ( )
                  OPTIONS (LIBRARY) [EXTERNAL][CONSTANT];
```

Parameter

Keine

Wirkung

Es wird der aktuelle Inhalt der Standard-Area ausgegeben.
Die Struktur der Standard-Area ist im Kapitel 10 ausführlich beschrieben.

Vor dem obigen Ausdruck wird noch der Pseudoregistervektor (PRV) ausgegeben. Eine Interpretation dieses Ausdrucks ist nur durch Spezialisten möglich.

```
DCL ADUMP ENTRY () OPTIONS (LIBRARY);

DCL FEST FIXED BIN(31,0) CONTROLLED INIT(255);

DCL ZEICHEN CHAR(20) VARYING BASED INIT((8)'4');
DCL P POINTER DIM(3);

DCL Z POINTER DIM(3);

DO I=1 TO 3;

     ALLOCATE FEST; FEST=FEST * (16**I);
     Z(1) = ADDR(FEST); PUT SKIP(2) LIST (UNSPEC(Z(I)));

     ALLOCATE ZEICHEN SET (P(I));
     PUT SKIP(2) LIST (UNSPEC(P(I))); PUT SKIP;

END;

CALL ADUMP;

FREE P(1)->ZEICHEN;
FREE FEST;
ALLOCATE FEST; FEST = FEST * (16**4);

CALL ADUMP;
```

Ergebnis:

```
****** DUMP OF THE STANDARD-AREA: ******

DUMP OF THE PRV:

REPL    VS ADR   MEMORY
        025000   000800D2  F1F7F3F3  D7D3C9C1  D3D3C740
        025020   C2C1C4E4  00000E00  00010001  00000000
        025040   00000002  00017BD4  01017484  02017754
        025060   070176B8  4F009FB2  01017484  00000002
        025080   00025CEC  00000000  00000090  7F000098
           .
           .
           .
           .
DUMP OF ALL LOGICAL STANDARD-AREA SEGMENTS:

REPL   VS ADR   MEMORY
       03E000   00000000  00000000  0003E000  0004E000
       03E020   00000001  00000000  0003E1B8  00000000
       03E040   0003E050  00000014  FFFEFFF8  FFFEFFF8
       03E060   00000000  00000000  00000000  00024008
       03E080   00000000  00000000  00000000  00000000
```

Bild 11-2 Beispiel für den Ablauf von ADUMP

BS2SRT Sortieren/Mischen

```
{BS2SRTA (a,b,c)     }
{BS2SRTB (a,b,c,d)   }
{BS2SRTC (a,b,c,e)   }
{BS2SRTD (a,b,c,d,e) }
```

```
BS2SRT (h1,..,hn,c,d,e)
```

Starten des Sortier/Mischprogramms SORT (siehe Handbuch SORT [10]) ohne oder mit PL/I-Prozeduren für die Bearbeitung der Datensätze vor und/oder nach der Sortierphase.

Eingänge

```
DCL    BS2SRTA  ENTRY    (CHAR (*) [NONVARYING],
                          CHAR (*) [NONVARYING],
                         [REAL]FIXED BINARY PREC (31,0) [ALIGNED])
                [EXTERNAL][CONSTANT];

DCL    BS2SRTB  ENTRY    (CHAR (*) [NONVARYING],
                          CHAR (*) [NONVARYING],
                         [REAL]FIXED BINARY PREC (31,0) [ALIGNED],
                         ENTRY ( ))
                [EXTERNAL][CONSTANT];

DCL    BS2SRTC  ENTRY    (CHAR (*) [NONVARYING],
                          CHAR (*) [NONVARYING],
                         [REAL]FIXED BINARY PREC (31,0) [ALIGNED],
                         ENTRY ( ))
                [EXTERNAL][CONSTANT];

DCL    BS2SRTD  ENTRY    (CHAR (*) [NONVARYING],
                          CHAR (*) [NONVARYING],
                         [REAL]FIXED BINARY PREC (31,0) [ALIGNED],
                         ENTRY ( ),
                         ENTRY ( ))
                [EXTERNAL][CONSTANT];

DCL    BS2SRT ENTRY OPTIONS (VARIABLE);
```

Parameter

a: Eine Zeichenfolge, die eine Steueranweisung SORT oder MERGE für das Programm SORT darstellt.

b: Eine Zeichenfolge, die eine Steueranweisung RECORD oder ALLOC für das Programm SORT darstellt. Wird eine solche Anweisung nicht benötigt, so ist eine leere Folge anzugeben.

c: Es ist nur eine Variable mit den oben angegebenen Attributen erlaubt. Der beim Aufruf übergebene Wert ist bedeutungslos. Bei der Rückkehr wird in den Variablen einer der folgenden Werte übergeben:

c = 0: Der Lauf des Programms SORT war fehlerfrei.
c = 16: Das Programm SORT wurde wegen Fehler abgebrochen.

d: Name einer PL/I-Funktion, mit der jeder Datensatz vor dem Sortiervorgang bearbeitet werden soll; d = 0, wenn nicht vorhanden.

e: Name einer PL/I-Funktion, mit der jeder Datensatz nach dem Sortiervorgang bearbeitet werden soll; e = 0, wenn nicht vorhanden.

h: Eine Zeichenfolge, die eine Steueranweisung für das Programm SORT (z.B. INCLUDE, SORT, SUM, RECORD usw.) darstellt.

Dateien

Für die Dateien, die vom Programm SORT verwendet werden sollen, müssen nach Bedarf folgende LINK-Namen vereinbart sein:

Sortier-Eingabedatei:	SORTIN oder SORTIN01, SORTIN02 usw. bis max. SORTIN99
Misch-Eingabedateien:	MERGE01, MERGE02 usw. bis max. MERGE99
Ausgabedatei:	SORTOUT

Wird der LINK-Name SORTIN und/oder SORTOUT nicht angegeben, so werden die Datensätze nur über die entsprechende PL/I-Funktion zum Sortieren übergeben bzw. vom Sortieren übernommen.

Sofern vom Programm SORT Hilfsdateien benötigt werden, können diese mit folgenden LINK-Namen zur Verfügung gestellt werden.

Arbeitsdatei auf Platte:	SORTWK
Arbeitsdateien auf Band:	SORTWK01 usw. bis max. SORTWK99
Stützpunktdatei:	SORTCKPT

Werden die Hilfsdateien benötigt und sind dafür keine LINK-Namen angegeben, so werden automatisch Dateien unter dem Namen SORTWK bzw. SORTCKPT katalogisiert und eingerichtet. Bei normaler Beendigung des Programms SORT werden diese Dateien gelöscht.

Ausführliche Angaben zum Sortieren und Mischen sind der Beschreibung des Programms SORT zu entnehmen. Die wichtigsten Elemente für die Steuerungsanweisungen SORT, MERGE und RECORD sind in den Bildern 11-3 und 11-4 auszugsweise wiedergegeben.

Beispiele siehe Anhang A.5

```
SORT-Anweisung  ::= SORT FIELDS=({(Anfang,Länge,Folge,[,Format])},...),

MERGE-Anweisung ::= MERGE FIELDS=({Anfang,Länge,Folge,[,Format]},...),

Anfang          ::= Zeichenposition [.Bitposition]

Länge           ::= Zeichenanzahl [.Bitanzahl] "max.256 Zeichen"

Zeichenposition ::= "Anfang des Sortierfeldes; 1 bis 4096 Zeichen"

Bitposition     ::= "Bitposition innerhalb Zeichen; 0 bis 7 Bits; nur für
                     Format BI"

Zeichenanzahl   ::= "Länge Sortierfeld; 1 bis 256 Zeichen"

Bitanzahl       ::= "0 bis 7; nur für Format = BI" "Restlänge Sortierfeld;
                     0 bis 7 Bits; nur für Format BI"

                    ⎡A "aufsteigend"⎤
Folge           ::= ⎨D "absteigend" ⎬
                    ⎣N "Restfeld"   ⎦

                    ⎡BI "Binär"                                 ⎤
                    ⎢CH "Zeichen"                               ⎥
                    ⎢SP "Sonderzeichen"                         ⎥
                    ⎢AA "ISO -> EBCDIC;sortieren; -> ISO"       ⎥
                    ⎢AE "ISO -> EBCDIC;sortieren; -> EBCDIC"    ⎥
Format          ::= ⎨EE "EBCDIC -> ISO;sortieren; -> EBCDIC"    ⎬
                    ⎢EA "EBCDIC -> ISO;sortieren; -> ISO"       ⎥
                    ⎢FI "Festpunkt"                             ⎥
                    ⎢FL "Gleitpunkt"                            ⎥
                    ⎢PD "dezimal,gepackt"                       ⎥
                    ⎣ZD "dezimal,entpackt"                      ⎦
```

Bild 11-3 Auszug aus der Syntax für die Steueranweisungen SORT und MERGE des Programms SORT

```
RECORD-Anweisung ::= RECORD LENGTH (Länge,...),TYPE=Typ

Länge            ::= "ganze Zahl"

                     ⎧F  "feste Satzlänge"    ⎫
Typ              ::= ⎨                        ⎬
                     ⎩V  "variable Satzlänge" ⎭
```

		Voreinstellung	
1. Länge:	max. Länge Eingabesatz	-	
2. Länge:	max. Länge Sortier- bzw. Mischsatz	1. Länge	
3. Länge:	max. Länge Ausgabesatz	2. Länge	
4. Länge:	min. Länge Sortier- bzw. Mischsatz	Länge Steuerfeld	nur bei TYPE = V
5. Länge:	häufigste Länge Sortier- bzw. Mischsatz	(2.Länge+4.Länge)/2	nur bei TYPE = V

Bild 11-4 Kurzfassung der Syntax für die Steueranweisung RECORD des Programms SORT

PL/I-Funktionen für Datensatzbearbeitung

Für die Funktionen zur Bearbeitung der Datensätze gelten folgende Bedingungen:

Eingang

Bezeichner: PROCDURE (s,r)
RETURNS (CHAR (*) [NONVARYING]);
bzw. entsprechende interne Funktion

Der Eingang muß dem Sortierprogramm über Parameter (Parameter d und e von Aufruf BS2SRT) mitgeteilt werden und wird von ihm vor bzw. nach dem Sortier-/Misch-Vorgang für jeden Datensatz einmal aufgerufen.

Parameter

`DCL s CHAR (Länge) NONVARYING UNALIGNED PARAMETER;` Die Länge muß eine Konstante sein. Abhängig von Parameter r enthält der Parameter s einen Datensatz oder einen undefinierten Wert.

```
DCL r REAL FIXED BINARY PREC (31,0) ALIGNED PARAMETER
```

Beim Aufruf wird in r einer der folgenden Werte übergeben:

r = 0: Datensatz wird in s übergeben. Ist keine Eingabedatei angegeben, so ist s undefiniert (muß aber in jedem Fall angegeben werden). Abfrage möglich durch:

```
IF ADDR(s) = NULL( )
   THEN keine Eingabedatei
   ELSE Eingabedatei
```

r = 4: Datensatz wird in s übergeben. Es gibt einen weiteren Datensatz mit dem gleichen Sortierfeld (nur bei der Bearbeitung nach dem Sortiervorgang möglich).

r = 8: Dateiende; der Wert des Parameters s ist undefiniert.

Bei der Rückkehr kann in r einer der folgenden Werte übergeben werden:

r = 0: Datensatz wird übergeben.

r = 4: Datensatz soll vom Programm SORT nicht weiterverwendet werden; übergebener Wert undefiniert. Dieser Wert muß angegeben werden, wenn die Ausgabe der sortierten Sätze von der PL/I-Funktion selbst vorgenommen wird.

r = 8: Ende der Bearbeitung.

r = 12: Zusätzlichen Datensatz vor dem übergebenen Datensatz einfügen. Falls vorhanden wird der übergebene Datensatz vom SORT nochmals geliefert. Dieser Wert muß angegeben werden, wenn die PL/I-Funktion die Eingabe der zu sortierenden Sätze selbst vornimmt.

Ergebnis

Das von der Funktion an den SORT zurückgegebene Ergebnis enthält abhängig vom Wert des Paramters r einen Datensatz oder einen undefinierten Wert:

r = 0: Unveränderter oder veränderter Datensatz

r = 4: Undefinierter Wert

r = 8: Undefinierter Wert

r = 12: Zusätzlicher Datensatz

CMD BS2000-Kommando ausführen

```
CMD (a, b, c)
```

Ausführen eines BS2000-Kommandos

Eingang

```
DCL CMD ENTRY (CHAR(*) [NONVARYING] [UNALIGNED],
               CHAR(*) [NONVARYING] [UNALIGNED],
               [REAL] FIXED BINARY PREC (31,0) [ALIGNED)]
        OPTIONS(LIBRARY) [EXTERNAL] [CONSTANT];
```

Parameter

a: Eine Zeichenfolge, die das auszuführende BS2000-Kommando enthält. Die maximale Länge der Zeichenfolge beträgt 32763.

b: Eine Zeichenfolge, in der das SYSOUT-Protokoll des BS2000-Kommandos eingetragen wird. Jeder Satz des Protokolls enthält in den ersten 4 Bytes sein Satzlängenfeld (Byte 0-1, Byte 2-3 sind reserviert). Die Sätze werden fortlaufend in die Zeichenfolge eingetragen. Die maximale Länge beträgt 32763. Wenn die Länge = 0, wird das Protokoll nach SYSOUT ausgegeben.

c: Eine Festpunktzahl, in der die Rückmeldung über die Ausführung des Kommandos eingetragen wird. Das Feld kann einen der folgenden Werte annehmen :

c = 0: normale Beendigung
c = 4: Speichermangel; keine Anforderung
c = 8: Fehler in der Operandenliste
c = 12: Der Protokollbereich ist für das SYSOUT-Protokoll zu kurz
c = 16: Kommandofehler
c = 20: Ungültiges Kommand

Wirkung

Mit Hilfe des Systemmakros CMD wird das angegebene BS2000-Kommandos ausgeführt. Das Protokoll wird entweder in den Protokollbereich eingetragen oder nach SYSOUT ausgegeben.

Kommandos, die nicht ausgeführt werden können oder die das Programm beendigen, sind in der Beschreibung des Systemmakros CMD in [16] aufgelistet.

```
DCL CMD ENTRY(CHAR(*), CHAR(*), FIXED BIN(31))
        OPTIONS(LIB);
    PROT CHAR(1000),RET_CODE FIXED BIN(31);

/* PROTOKOLL WIRD IN DAS FELD PROT EINGETRAGEN */
CALL CMD ('FSTATUS',PROT,RET_CODE);

/* PROTOKOLL WIRD NACH SYSOUT AUSGEGEBEN */
CALL CMD ('FSTATUS','',RET_CODE);
```

Bild 11-4a Beispiel für die Verwendung von CMD

ERROUT Fehlertext ausgeben

```
ERROUT
```

Ausgabe des aktuellen Fehlertextes nach SYSOUT

Eingang

```
DCL ERROUT ENTRY ( ) OPTIONS (LIBRARY) [EXTERNAL][CONSTANT];
```

Parameter

keine

Wirkung

Wurde eine Bedingung gesetzt und als Folge davon eine ON-Einheit durchlaufen, so kann durch den Aufruf CALL ERROUT der entsprechende Fehlertext ausgegeben werden.

Es handelt sich hier um den gleichen Fehlertext, der von der Systemeinheit ausgegeben würde, wenn keine entsprechende ON-Einheit vorhanden wäre.

```
DCL ERROUT  ENTRY() OPTIONS(LIBRARY);

ON AREA  BEGIN;
         CALL ERROUT;
         IF ONCODE() = 360 THEN GOTO ALLOC_FEHLER;
         IF ONCODE() = 361 THEN GOTO ASSIGN_FEHLER;
         IF ONCODE() = 362 THEN GOTO SIGNAL_FEHLER;
         END;
```

Ausgabe von ERROUT

```
*****AREA-CONDITION, ONCODE=0362 IN LINE 15 IN STATEMENT 1
INSUFFICIENT SPACE IN A NAMED AREA SIGNALED
```

Bild 11-5 Beispiel für die Verwendung von ERROUT

HEXDEC (a) Sedezimalzeichen (hexadezimal)

```
HEXDEC (a)
```

Die Bitfolge a wird in eine Zeichenfolge aus Sedezimalzeichen konvertiert.

Eingang

```
DCL HEXDEC ENTRY (BIT (*) [NONVARYING] [UNALIGNED])
           RETURNS (CHAR (*) [NONVARYING] [UNALIGNED])
           OPTIONS (LIBRARY) [EXTERNAL] [CONSTANT];
```

Parameter

Skalare Bitfolge

Ergebnis

Zeichenfolge die nur die Sedezimalzeichen 0...9 und A...F enthält.

Wirkung

Von links beginnend wird für jeweils 4 Bits ein Sedezimalzeichen erzeugt. Als Sedezimalzeichen werden die Ziffern 0 bis 9 und die Buchstaben A bis F verwendet.

Ist die Länge der Bitfolge nicht ein Vielfaches von 4 Bits, so werden rechts Bits mit dem Wert ’0’B ergänzt. Wird die Ergänzung links gewünscht, so ist beim Parameter eine entsprechende Verkettung anzugeben (siehe Beispiel).

```
DCL HEXDEC  ENTRY    (BIT(*))
            RETURNS  (CHAR(*))     OPTIONS (LIBRARY);

DCL BIT  BIT(7)  INIT ('1111000'B);                          Ergebnis


PUT SKIP(1) LIST (HEXDEC(BIT));                              F0
PUT SKIP(2) LIST (HEXDEC('0'B || BIT));                      78
PUT SKIP(2) LIST (HEXDEC(COPY('0'B,4*CEIL(LENGTH(BIT)/4)
                 -LENGTH(BIT))||BIT));                       78
```

```
Bitmuster und          '0000'B  0
zugehörige             '0001'B  1
Sedezimalziffer        '0010'B  2
                       '0011'B  3
                       '0100'B  4
                       '0101'B  5
                       '0110'B  6
                       '0111'B  7
                       '1000'B  8
                       '1001'B  9
                       '1010'B  A
                       '1011'B  B
                       '1100'B  C
                       '1101'B  D
                       '1110'B  E
                       '1111'B  F
```

Bild 11-6 Beispiele für den Aufruf von HEXDEC

NOTRACE Ablaufverfolgung ausschalten

```
NOTRACE (a)
```

Ausschalten der Ablaufverfolgung

Eingang

```
DCL NOTRACE ENTRY (CHAR(*)) OPTIONS (LIBRARY) [EXTERNAL] [CONSTANT];
```

Parameter

Zeichenfolge, die folgende Buchstaben mit der angegebenen Bedeutung enthalten kann:

P Ablaufverfolgung für Prozeduraufruf (PROCEDURE)
C Ablaufverfolgung für CALL-Anweisungen
R Ablaufverfolgung für Rückkehr von Prozeduraufruf (RETURN)
G Ablaufverfolgung für Sprünge (GOTO)
L Ablaufverfolgung für Marken (LABEL)
T zusätzliche Ausgabe der Ablaufverfolgung auf der Datenstation

Andere Buchstaben werden ignoriert. Der leere String (") entspricht der Angabe 'PCRGL'.

Wirkung

Eine eingeschaltete Ablaufverfolgung (siehe TRACE) kann durch NOTRACE abgeschaltet werden (siehe Kapitel 9).

PLIRETC Returncode setzen

Nur für Anwender mit dem Software-Produkt JV.

```
PLIRETC (a)
```

Die Programminformation des Returncodes wird auf a gesetzt.

Eingang

```
DCL PLIRETC ENTRY ([REAL] FIXED BINARY PREC(31,0) [ALIGNED])
            OPTIONS (LIBRARY) [EXTERNAL][CONSTANT]
```

Parameter

a: Wert zwischen 0 und 999

Wirkung

Die Programminformation (Zeichen 5 bis 7) des Returncodes einer überwachenden Jobvariablen (Monitorjobvariable) erhält bei Ende des Programmlaufs den angegebenen Wert a im Format PIC'999'. Damit kann das Programm eine Information an die Kommandosprachen-Ebene durchreichen.

```
DCL PLIRETC ENTRY(FIXED BINARY(31,0))
            OPTIONS (LIBRARY);

CALL PLIRETC (17);
```

```
/DCLJV JV
/EXEC T.PLIRETC,MONJV=JV
   .
   .
   .
   .
   .
END OF PROGRAM....
/GETJV (JV,5,3)
%017
```

Bild 11-6a Beispiel zum Aufruf von PLIRETC

RDUMP (a,b) Speicherauszug

```
RDUMP (a,b)
```

Beginnend bei der Adresse a werden b Zeichen nach SYSLST ausgegeben.

Eingang

```
DCL RDUMP ENTRY (POINTER [ALIGNED],
                 [REAL] FIXED BINARY PREC (15,0) [ALIGNED])
          OPTIONS (LIBRARY) [EXTERNAL] [CONSTANT];
```

Parameter

a: Absolut-Zeiger auf den Anfang des auszugebenden Speicherbereichs
b: Anzahl Bytes, die auszugeben sind.

Wirkung

Beginnend bei der Adresse a werden mindestens b Zeichen in sedezimaler Darstellung nach SYSLST ausgegeben.

```
DCL RDUMP  ENTRY (POINTER ALIGNED,
                  FIXED BINARY (15,0) ALIGNED)
                  OPTIONS (LIBRARY);

DCL GLEIT  FLOAT DECIMAL (33) INIT (12345E+16);

CALL RDUMP   (ADDR(GLEIT),16);
```

Ausgabe

```
****** DUMP DES BEREICHS 0181A0 - 0181AF: ******

REPL    VS ADR    SPEICHERINHALT
        0181A0    516B1368  0EF11F90  43000000  00000000
```

Bild 11-8 Beispiel für RDUMP

RUNTIME Verbrauchte Rechenzeit

```
RUNTIME
```

Rückgabe der seit dem Programmstart verbrauchten Rechenzeit in Sekunden.

Eingang

```
DCL RUNTIME ENTRY ( )
            RETURNS (CHAR (8))
            OPTIONS (LIBRARY) [EXTERNAL] [CONSTANT];
```

Parameter

Keine

Ergebnis

Rechenzeit in der Form PIC 'ZZZZ9.V99' in Sekunden

Wirkung

Durch den Aufruf dieser Funktions-Prozedur wird die seit dem Programmstart verbrauchte CPU-Zeit ermittelt und in Form einer Zeichenfolge zurückgegeben:

5 Zeichen ganzzahliger Teil
1 Zeichen Dezimalpunkt
2 Zeichen gebrochener Teil

Die Einheit ist die Sekunde. Der Wert ist gerundet.

```
DCL  RUNTIME  ENTRY () RETURNS (CHAR(8)) OPTIONS(LIBRARY);

PUT  SKIP LIST (RUNTIME);


DCL  ZEIT   CHAR(8);

ZEIT = RUNTIME;

PUT  SKIP LIST (ZEIT);
```

Ergebnis

```
0.28
0.29
```

Bild 11-9 Beispiele zum Aufruf von RUNTIME

SDUMP Speicherauszug des Stack

```
SDUMP
```

Es wird der Inhalt des Stack (Blockaktivierungen mit AUTOMATIC-Variablen) nach SYSLST in Sedezimal- und Zeichendarstellung ausgegeben.

Eingang

```
DCL SDUMP ENTRY ( )
          OPTIONS (LIBRARY) [EXTERNAL] [CONSTANT];
```

Parameter

Keine

Wirkung

Es wird der aktuelle Inhalt der Stacksegmente ausgegeben. Die Struktur des Stacks ist im Kapitel 10 ausführlich beschrieben.
Vor den DUMP wird die Anfangsadresse der aktuellen Aktivierung des Stacks ausgegeben, d.i. die Aktivierung des Laufzeitsystems das den DUMP ausgibt.

Vor dem obigen Ausdruck wird noch der Pseudoregistervektor (PRV) ausgegeben. Eine Interpretation dieses Ausdrucks ist nur durch Spezialisten möglich.

```
DCL  SDUMP  ENTRY() OPTIONS(LIBRARY);

DCL  ZEICHEN  CHAR(8) AUTOMATIC INIT('ABCDEFGH');


CALL SDUMP;
```

Ergebnis:

```
***** DUMP OF THE STACK:******

DUMP OF THE PRV:

REPL    VS ADR   MEMORY
        014000   000800D2  F1F7F3F5  D7D3C9C1  D3D3C740
        014020   C2E3C4E4  00000E00  00010001  00000000
        014040   00000002  00010044  0100F8F4  0200FBC4
        014060   0700FB28  4F002422  0100F8F4  00000002



ACTUAL ACTIVATION (REGISTER 13 OF THE DUMPING PROCEDURE): 00019250

DUMP OF ALL LOGICAL STACK SEGMENTS:

REPL    VS ADR   MEMORY
        019000   00000000  00000000  7F019000  7F029000  00000000
        019020   00000010  00000002  00000000  00000014  00000014
        019040   00000000  00000000  00000000  00000000  00000000
```

Bild 11-10 Beispiel für den Aufruf von SDUMP

SNAP Aufrufverschachtelung

```
SNAP
```

Ausgabe der aktuellen Aufrufverschachtelung nach SYSOUT

Eingang

```
DCL SNAP ENTRY ( ) OPTIONS (LIBRARY) [EXTERNAL] [CONSTANT];
```

Parameter

Keine

Wirkung

Durch den Aufruf CALL SNAP wird eine Liste der zur Zeit aktiven Prozeduren und deren Aufruffolge ausgegeben. Dabei wird die zeitlich älteste Prozedur (die Haupt-Prozedur) als letzte aufgelistet. Für jede aufgerufene und noch aktive Prozedur wird eine Zeile ausgegeben.

Der Aufbau der Ausgabe ist im Abschnitt 9.7 ausführlich erläutert.

```
VERSCHA:              /* BEISPIEL FUER AUFRUFVERSCHACHTELUNG */
          PROCEDURE OPTIONS(MAIN);

          DCL  SNAP ENTRY() OPTIONS(LIBRARY);

          PUT  SKIP LIST ('    SNAP');

          ON ZERODIVIDE BEGIN;
                        PUT SKIP LIST ('~~~ZERO');
                        CALL FEHLER;
                        GOTO ENDE;
                        END;
          BEGIN;
          PUT SKIP LIST ('~~~ BEG');
          DCL (X,Y,Z) INIT(0),
          X = Y / Z;
          END;

          FEHLER: PROC;
                  PUT SKIP LIST ('    FEHLER');

                  CALL SNAP;

                  END;
```

SYSLST

```
... SNAP
... BEG
... ZERO
... FEHLER
```

SYSOUT

```
****************************** SNAP *************************


START OF PRINTING OF NESTED SUBROUTINES

CALLED FROM     TYPE      ON ADDRESS      SOURCE-REFERENCE

  SNAP          SYSTEM      0020D2
  FEHLER        ENTRY       000406                25     1
  ZEROIDE       ON          000300                13     1
  ER$INTR       SYSTEM      0107EA
  ER$PUB        SYSTEM      010EA0
  SR$STXT       SYSTEM      00FD04
  ##00004       BEGIN       0003FA                19     1
  BSNAP         ENTRY       000244                11     1
```

Bild 11-11 Beispiel für Aufrufverschachtelung (SNAP)

TRACE Ablaufverfolgung einschalten

```
TRACE (a)
```

Einschalten der Ablaufverfolgung

Eingang

```
DCL TRACE ENTRY(CHAR(*)) OPTIONS (LIBRARY)
                         [EXTERNAL] [CONSTANT]
```

Parameter

Zeichenfolge die folgende Buchstaben mit der angegebenen Bedeutung enthalten kann:

P Ablaufverfolgung für Prozeduraufruf (PROCEDURE)
C Ablaufverfolgung für CALL-Anweisungen
R Ablaufverfolgung für Rückkehr von Prozeduraufrufen (RETURN)
G Ablaufverfolgung für Sprünge (GOTO)
L Ablaufverfolgung für Marken (LABEL)
T zusätzliche Ausgabe der Ablaufverfolgung auf der Datenstation

Andere Buchstaben werden ignoriert. Der leere String (") entspricht der Angabe 'PCRGL'.

Wirkung

Voraussetzung für die Ablaufverfolgung ist, daß bei der Übersetzung der Prozedur durch die Übersetzer-Steuerung

```
*COMOPT DEBUG=Angabe
```

die gewünschte Ablaufverfolgung eingearbeitet ist. Die gewünschte Ablaufverfolgung kann eingeschaltet werden entweder durch die Objekt-Steuerung

```
*RUNOPT TRACE=Angabe
```

(sie beginnt dann am Anfang der Procedure) oder dynamisch durch den Aufruf

```
CALL TRACE (Ausdruck);
```

(sie beginnt dann unmittelbar nach dem Aufruf). Durch den Aufruf von NOTRACE kann die gewünschte Ablaufverfolgung wieder dynamisch abgeschaltet werden.

Die Voreinstellung des PL/I-Systems ist so eingestellt, daß keine Ablaufverfolgung eingearbeitet und auch keine eingeschaltet wird (siehe Kapitel 9).

12 Gemeinsam benutzbare Programme

12.1 Voraussetzungen

Für Programme mit gemeinsam benutzbaren Moduln gelten folgende Voraussetzungen:

- Die gemeinsam zu benutzenden Moduln müssen dynamisch geladen werden.
- Die gemeinsam benutzbaren Moduln dürfen nur lesend benutzt werden.
- Es darf keine Adreßbezüge von gemeinsam benutzbaren Moduln zu nicht gemeinsam benutzbaren Moduln geben.
- Veränderliche Daten müssen in dynamisch angeforderten Speicherbereichen stehen. Diese Voraussetzung ist erforderlich, wenn alle Moduln im Klasse-4-Speicher liegen sollen. Dies ist bei der derzeitigen Realisierung für gemeinsame benutzbare PLI1-Programme immer erforderlich.

PL/I-Programme erfüllen diese Voraussetzungen, wenn sie keine STATIC-Variable, CONTROLLED-Variable und keine Ein-Ausgabe-Anweisungen enthalten.
Die Moduln ITP#RTS# und ITP#IOS# des "shareable" PL/I-Laufzeitsystems genügen den Voraussetzungen für gemeinsam benutzbare Moduln.
Wenn Assembler-Moduln verwendet werden, unterliegen sie ebenfalls den obigen Voraussetzungen.

12.2 PL/I-Programme

Im folgenden werden Methoden beschrieben, die es ermöglichen, aus PL/I-Programmen, die den obigen Voraussetzungen widersprechen, gemeinsam verwendbare Programme zu erzeugen. Zu diesem Zweck sind in PLI1 Spracherweiterungen vorhanden.

12.2.1 STATIC-Variable

Durch Spracherweiterungen kann man STATIC-Variable aus externen Prozeduren entfernen, wenn man die Übersicht über alle STATIC-Variablen hat. STATIC-Variable werden über die im PLI1-System vordefinierte (eingebaute) Absolut-Zeiger-Matrix PLI1GLOBAL (0 : 127) auf BASED-Variable abgebildet.

Beispiel

```
DCL Name STATIC EXTERNAL INIT (Wert);
```

ist zu ersetzen durch

```
DCL Name BASED (PLI1GLOBAL(n)) INIT (Wert);
```

wobei n eine ganze Zahl von 0 bis 127 ist. Für jeden Namen wird eine eigene Zahl benötigt.

Am Anfang des Programms ist die Anweisung

```
ALLOCATE Name;
```

einzufügen.

Hat man mehrere STATIC-Variable zu entfernen, so empfiehlt es sich, diese in einer Struktur zusammenzufassen.
Für INTERNAL STATIC-Variable mit INITIAL-Attribut, auf die nur lesend zugegriffen wird, sollte statt der obigen Vorgehensweise in der Vereinbarung

```
STATIC (CONSTANT)
```

geschrieben werden, wenn sie nicht ohnehin vom Übersetzer als Konstante erkannt wird (siehe dazu Kapitel 8).

12.2.2 Ein-Ausgabe-Anweisungen

Bei der Vereinbarung einer Datei legt der Übersetzer für die Datei-Konstante eine STATIC-Variable an.

Eine PL/I-Spracherweiterung ermöglicht es, das Entstehen dieser STATIC-Variablen zu verhindern. Dazu ist die Datei-Konstante um die Angabe

```
ENVIRONMENT (PLI1GLOBAL (n))
```

zu erweitern. Hierbei ist zu beachten, daß bei mehreren externen Prozeduren der gleiche Wert n nur für gleiche Datei-Konstanten und auch nicht anderweitig verwendet werden darf.

Bei den Anweisungen der Strom-Ein-Ausgabe ohne Datei-Angabe wird implizit die Datei-Konstante SYSIN für Eingabe und SYSPRINT für Ausgabe vereinbart. Diese Datei-Konstanten müssen ebenfalls explizit mit der obigen ENVIRONMENT-Angabe vereinbart werden; analoges gilt für die Datei SYSOUT.

Die einzige Ein-Ausgabe-Anweisung, die ohne spezielle Vorkehrungen verwendet werden kann, ist DISPLAY mit/ohne REPLY.

12.2.3 CONTROLLED-Variable

Für CONTROLLED-Variable erzeugt der Übersetzer implizit STATIC-Variable. Eine PL/I-Spracherweiterung ermöglicht es, das Entstehen dieser Variablen zu verhindern. Dazu ist bei der Vereinbarung die Angabe

```
CONTROLLED (PLI1GLOBAL (n))
```

anzugeben. Hierbei ist zu beachten, daß n eindeutig dem Namen der CTL-Variable zuordenbar sein muß.

12.3 Eintragen in den Klasse-4-Speicher

Sobald alle Moduln eines Programms gemeinsam benutzbar gemacht worden sind und übersetzt wurden, wird mit dem Modulbinder ein Großmodul erzeugt, wobei der Verbindungsmodul ITP#AOD# zum dynamischen Laufzeitsystem von PLI1 angebunden wird. Der Großmodul wird mit Hilfe des LMR [3] als "READ ONLY" erklärt und mit dem SHARE-Kommando dann in die Share-Tabelle des Systems eingetragen (siehe BS2000 Systemüberwachung).
Die Moduln ITP#RTS# und ITP#IOS# des Laufzeitsystems sollten ebenfalls in die Share-Tabelle eingetragen worden sein.

Für den Start des Programms wird anschließend der dynamische Bindelader verwendet:

```
/EXEC (Modulname [,Bibname]) [,weitere Angaben]
```

Achtung bei Programm BS2SRT aus Kapitel 11:
Die SORT-Routine ist nicht gemeinsam benutzbar.

13 PLI1-ASSEMBLER-Makro-Schnittstelle

13.1 Allgemeines

Beim Anschluß von ASSEMBLER-Programmen an PL/I-Programme und umgekehrt sind im ASSEMBLER-Programm die PLI1-Konventionen zu beachten (siehe Kapitel 7). Da dieses einen erheblichen Programmieraufwand erfordern würde, wird ein Satz von Definitions- und Aktionsmakros angeboten, der diesen Aufwand im Normalfall auf das Einfügen einiger Makros reduziert. Insbesondere sollte die Umstellung bestehender ASSEMBLER-Unterprogramme auf die Aufrufbarkeit durch PL/I-Programme nur das Entfernen einiger ASSEMBLER-Befehle und das Einfügen zweier Makroaufrufe erfordern.

Diese Markos können nur bei PL/1-Moduln verwendet werden, die mit "*COMOPT OPTIONS=NOXS" übersetzt wurden.

13.1.1 Tabelle der Makros

(A = Aktionsmakro, D = Definitionsmakro)

```
P$CALL      -A-    Aufruf eines PL/I-Unterprogramms
P$ENTRY     -A-    Erzeugen eines von PL/I aufrufbaren Eingangs
P$ENVIRM    -A-    Einrichten der PLI1-Umgebung
P$ERROR     -A-    Setzen der ERROR-Bedingung
P$LINK      -A-    Nachladen eines PL/I-Unterprogramms
P$PRV       -D-    PLI1-Pseudoregistervektor (DSECT oder Vorbesetzung)
P$REGEQU    -D-    EQU-Anweisungen für Registernotation
P$RETURN    -A-    Rückkehr ins rufende PL/I-Programm
P$STACK     -D-    DSECT für Register- und AUTOMATIC-Speicher
P$STOP      -A-    Beenden des gesamten Programmlaufs
```

13.1.2 Allgemeine Hinweise

- Alle Makros sind mehrfach benutzbar ("reentrant", "shareable") programmiert, d.h. ASSEMBLER-Programme, die mehrfach benutzbar sind, bleiben dies auch nach Einfügen von PLI1-Makros.
- Die Makros sind so konzipiert, daß im Normalfall keine Steuerparameter anzugeben sind, d.h. der jeweils häufigste Anwendungsfall wird voreingestellt.
- Die Makros benutzen zur Koordination die Globalvariablen &ENVIRM#, &STACK#, &MAXPAR#, &PRV#, &REG#.
- Alle ASSEMBLER-Programme, die mit PL/I-Programmen zusammenarbeiten, sollten die PL/I-Registerkonventionen beachten, d.h. Register R12, R13, sollten außerhalb der Makros nicht verändert werden. Falls dies nicht gewährleistet ist, muß jeweils vor dem Aufruf der Aktionsmakros der PLI1-spezifische Inhalt von R12, R13 wiederhergestellt werden (Ausnahme: Makros P$ENVIRM, P$ENTRY).
- Bei eigener STXIT-Behandlung im ASSEMBLER-Programm ist zu beachten, daß beim Aufruf eines PLI1-Unterprogramms, sowie bei der Rückkehr ins rufende PL/I-Programm die PLI1-STXIT-Behandlung reaktiviert wird (Angabe STXIT=YES in den Makros P$CALL und P$RETURN).
- Mit Hilfe der Makros P$ENTRY oder P$ENVIRM kann das ASSEMBLER-Programm von der PLI1-Speicherverwaltung dynamischen Speicher anfordern. Programme, die selbst Speicher anfordern (REQM-Makro), sollten umgestellt werden, um eine Zerstückelung des virtuellen Adreßraums zu vermeiden.
- Beispiele siehe Anhang A.6.

13.2 Makros

P$CALL

Aufruf einer PL/I-Prozedur aus einem ASSEMBLER-Programm mit wahlweiser Übergabe einer Parameterliste an das PL/I-Programm

`Name P$CALL PL/I-Prozedur,Param PARNUM=0|leer|Anzahl,STXIT=N[O]`	`(Param1, Param2,...), Y[ES]`

Regeln:

- Der Makro darf nur hinter den Makros P$ENVIRM oder P$ENTRY aufgerufen werden, Überprüfung durch Globalvariable &ENVIRM#.
- Ein Basis-Register ist nicht notwendig, intern wird R10 verwendet.
- Die Register R0, R1, R2, R3, R4, R14 werden verändert und nicht restauriert, R10 und R15 werden verwendet und restauriert.
- R12, R13 müssen die PLI1-spezifischen Werte enthalten.

Beschreibung der Parameter:

Name — Wird nicht verwendet

PL/I-Prozedur — Name des aufzurufenden PL/I-Programms
Falls nicht angegeben, muß die V-Konstante der zu rufenden PL/I-Prozedur über das Parameter-Register R1 übergeben werden, d.h. R1 enthält die Adresse der V-Konstante.
Sonderfall: Siehe nächster Parameter

Param | (Param1, Param2,...)

Name(n) des/der an das PL/I-Programm zu übergebenden Parameter.
Falls nicht angegeben, wird entweder kein Parameter übergeben oder eine Parameterliste vom ASSEMBLER-Typ.

Eine ASSEMBLER-Parameterliste hat folgenden Aufbau:

```
R1            Parameter-Register
A (Param1)    Adresse erster Parameter
A (Param2)       "    zweiter    "
                          .
                          .
                          .
X'80' A (Param n) Adresse letzter Parameter,
                  gekennz.
            durch X'80' ('80'B4).
```

Sonderfall:

Falls der Parameter "PL/I-Prozedur" nicht angegeben ist, muß als erster Eintrag der Assembler-Parameterliste die Adresse der V-Konstante der zu rufenden PL/I-Prozedur angegeben werden.

```
PARNUM = 0 | leer | Anzahl
```

– Voreinstellung 0 bedeutet, daß keine Parameter an das PL/I-Programm übergeben werden sollen. Ausnahme: Wenn eine explizite Parameterliste ("Param") angegeben ist, wird 0 wie die leere Eingabe behandelt.

– Die leere Eingabe bedeutet, daß keine Überprüfung der aktuellen Parameteranzahl stattfinden soll.

– Ist "Anzahl" angegeben, wird in jedem Fall geprüft, ob die angegebene Zahl mit der aktuellen Parameteranzahl übereinstimmt, d.h. falls "Param" angegeben ist, wird die Zahl der Einträge in der Unterliste mit "Anzahl" verglichen und bei Nichtübereinstimmung die Makrogenerierung mit MNOTE abgebrochen. Falls die Übergabe über Parameter-Register R1 erfolgt, kann die Überprüfung nur zur Laufzeit auf Grund der Angabe X'80' ('80'B4) im letzten Parameter erfolgen. Bei Nichtübereinstimmung wird die ERROR-Bedingung mit ONCODE=5010 gesetzt. Der Parameter "PL/I-Prozedur" wird, falls angegeben, nicht mitgezählt.

```
STXIT = Y[ES] |N[O]
```

– Voreinstellung NO bedeutet keine Aktion.

– YES bedeutet, daß vor Aufruf des PL/I-Programms die PLI1-spezifischen STXIT-Routinen aktiviert werden. Dies ist ratsam bzw. notwendig, falls im rufenden ASSEMBLER-Programm ein STXIT-Makro verwendet wurde.

Hinweis

- Die Maximalzahl der an das PL/I-Programm übergebbaren Parameter kann in den Makros P$ENTRY, P$ENVIRM festgelegt werden, bei zur Übersetzzeit erkennbaren Fehlern (Parameterzahl größer als Globalvariable &MAXPAR) wird die Makro-Generierung mit MNOTE abgebrochen. Bei zur Laufzeit erkannten Fehlern wird die ERROR-Bedingung P$CALL mit ONCODE=5010 gesetzt.
- Die aufgerufene PL/I-Prozedur kann eine Funktion sein, es sind dabei für die Funktionswertübergabe die PLI1-Rückgabe Konventionen, wie im Abschnitt 7.1.4 beschrieben, zu beachten, insbesondere wird bei der Rückgabe einiger Datentypen das Register R1 verwendet.
- Die an das PL/I-Programm übergebenen Parameter dürfen keine Deskriptoren benötigen, d.h. Parameter mit *-Angaben sind nicht zulässig. Als Alternative kann die Verwendung von VARYING-Zeichenketten oder von BASED-Variablen mit variabler Dimensionierung angesehen werden.

Beispiel

```
/* ALTERNATES FOR CHAR(*)PARM*/
DCL A CHAR(100) VARYING PARAMETER;
DCL A CHAR(N)   BASED(P),
    P POINTER   PARAMETER,
    N BIN FIXED PARAMETER;
```

P$ENTRY

Erzeugung eines Prozedur-Eingangs und Erzeugung des Prozedurkopfes, der bei einem von PL/I aufgerufenen Eingangspunkt erwartet wird.

```
Name  P$ENTRY  LENGTH = Bytes   , MAXPAR=  Anzahl
```

Regeln:

- Der mit dem Makro P$ENTRY generierte Eingang darf nur aufgerufen werden, wenn die PLI1-Umgebung aktiviert ist.
- Der Makro kann mehrfach in einem Programm eingefügt werden, es wird dabei jeweils ein externer Eingangspunkt erzeugt.

Beschreibung der Parameter:

Name
: Falls der Makro unmittelbar der START-Anweisung folgt, sollte die Angabe "Name" entfallen, - falls "Name" angegeben ist wird zusätzlich zum Prozedurkopf eine ENTRY-Anweisung generiert.

LENGTH = Bytes
: Dem ASSEMBLER-Unterprogramm wird von der PLI1-Speicherverwaltung dynamischer Benutzerspeicher zur Verfügung gestellt. Ist "Bytes" = 0 (Voreinstellung) wird kein Benutzer-Speicher angelegt (siehe auch Parameter MAXPAR).

MAXPAR = leer | Anzahl
: Die Voreinstellung "leer" bedeutet, daß die Voreinstellung des Makros P$STACK verwendet wird.
Falls "Anzahl" >4 wird ein Bereich von (Anzahl minus 4) Ganzworten für die vom Makro P$CALL benutzte Parameterliste hinter der Register-Savearea reserviert. Details siehe Makro P$STACK.
Falls "Anzahl" ≤4: Die Parameterübergabe durch Makro P$CALL erfolgt über die Register R1,..., R4; zusätzlicher Speicher für die Parameterliste ist nicht notwendig.

Hinweis

- Die Globalvariable &ENVIRM# wird benutzt.
- Der durch den Makro spezifizierte Eingang kann auf der PL/I-Seite mit folgenden OPTIONS-Angaben vereinbart werden:

  ASSEMBLER, VARIABLE, PLI1 (voreingestellt).
  Nicht verwendet werden sollten: LIBRARY, COBOL, FORTRAN.
- Das Attribut RETURNS ist zulässig, falls nicht OPTIONS (ASSEMBLER) angegeben wurde, die Rückgabe des Funktionswerts muß nach PLI1-Konvention (siehe Benutzer-Handbuch Abschnitt 7.1.4) erfolgen. Falls die Rückgabe über Register R1 erfolgt, muß beim Makro P$RETURN der Parameter R1RETRN=YES angegeben werden.

P$ENVIRM

Einrichten der PLI1-Umgebung, Aktivieren der PLI1-STXIT-Behandlung und der Speicherverwaltung

```
Name P$ENVIRM  MAXPAR =  Anzahl  ,  LENGTH=Bytes, BASEREG = RXX
```

Regeln:

- Der Makro darf je Programm nur einmal aufgerufen werden, er darf nicht hinter P$ENTRY aufgerufen werden, sonst MNOTE.
- Registerverwendung siehe Parameter BASEREG=

Beschreibung der Parameter:

Name — Wird nicht verwendet

MAXPAR= leer | Anzahl
"leer" bedeutet, daß die Voreinstellung des Makros P$STACK verwendet wird.
"Anzahl" gibt die Maximalzahl von Parametern an, die dem Makro P$CALL mitgegeben werden sollen, die Angabe beeinflußt die Größe des dynamischen Speicherbereichs, siehe nächster Parameter.

LENGTH = Bytes
Voreinstellung 0 bedeutet, daß aus der Register-Savearea und ggf. dem Parameterbereich kein dynamischer Speicher angefordert wird.
"Bytes" gibt an, wieviel dynamischer Speicher Speicher dem ASSEMBLER-Programm von der PLI1-Speicherverwaltung zur Verfügung gestellt werden soll (notwendig bei "shareable" Programmierung).

BASEREG = RXX | leer
"leer" bedeutet, daß beim Aufruf von P$ENVIRM kein Basisregister spezifiziert ist, R10 wird angenommen und bleibt über das Ende der Makrogenerierung hinaus gültig.
"RXX" (XX=1,2,...,(11),14,15) kann angegeben werden um dem Makro mitzuteilen, daß zum Zeitpunkt des Makroaufrufs ein bestimmtes Basisregister gültig ist. Dieses Register ist nach dem Makro-Aufruf weiterhin Basisregister mit der vorher gültigen Basisadresse.
Innerhalb des Makros wird immer Register R10 als Basisregister verwendet.

R12, R13 sind nicht zulässig, da diese Register nach Ausführung des Makros den PLI1-spezifischen Wert behalten müssen.
R11 wird nicht empfohlen, da dieses Register das einzige ist, das vom Makro P$ENVIRM nicht verändert wird, es kann benutzt werden um irgend ein anderes Register zu speichern (notwendig bei "shareable" Programmierung).

Hinweis

- Der Makro verwendet alle Register außer R11, wegen seiner Reentrant-Eigenschaft können sie nicht restauriert werden. Siehe auch Parameter BASEREG.
- Bei versehentlichem Mehrfachaufruf des Makros kann keine Warnung erfolgen, je nach Einstellung der Laufzeitoption STORAGE wird durch jeden Aufruf eine bestimmte Menge virtueller Speicher blockiert, oder es tritt die STORAGE-Bedingung auf.

P$ERROR

Setzen der PL/I-ERROR-Bedingung analog SIGNAL ERROR; jedoch mit Angabe eines bestimmten ONCODE Werts und ggf. einer Meldungsnummer

```
Name      P$ERROR  ONCODE = Code   ,MSG=Subcode
```

Regeln:

- Der Makro darf nur nach den Makros P$ENTRY oder P$ENVIRM aufgerufen werden, Überprüfung durch Globalvariable &ENVIRM#.
- Ein Basis-Register ist nicht notwendig, intern wird R10 verwendet, keine Register werden restauriert.
- R12, R13 müssen die PLI1-spezifischen Werte enthalten.

Beschreibung der Parameter:

Name	Wird nicht verwendet
ONCODE =	Code,MSG=Subcode Voreinstellung ONCODE=5000, MSG=00 "Code" gibt den PL/I-ONCODE an, der von der PLI1-Fehlerbehandlung verwendet werden soll, um einen speziellen Fehlertext auszugeben. Die Angabe MSG=Subcode ist nur notwendig, wenn mehrere Fehlertexte zu einem bestimmten ONCODE existieren. Die Texte müssen in den Dateien $TSOS.PLI1.TEXT.E oder D enthalten sein bzw. vom Benutzer dort hineingebracht werden.
MSG = Subcode	Siehe Parameter ONCODE Falls eigene Fehlermeldungen und ONCODE-Werte definiert werden sollen, muß eine entsprechende Erweiterung der Textdateien vorgenommen werden. siehe Parameter ONCODE=, MSG=. Der zum Einfügen der Texte in die Textdateien zu verwendende ISAM-Schlüssel ist achtstellig 015CCCMM, wobei 5CCC der ONCODE mit führenden Nullen und MM die zusätzliche Meldungsnummer ist.

P$LINK

Nachladen eines PL/I-Unterprogramms (DLL)

```
Name  P$LINK  PL/I-Name  ,Adr.-Konst   ,LIBNAM=Bibl.-Name
```

Regeln:

- Der Makro sollte nur benutzt werden, wenn die PLI1-Umgebung (R12, R13) eingerichtet ist, weil bei Fehlern im LINK-Makro die PLI1-Fehlerbehandlung über Makro P$ERROR mit ONCODE=5015 aufgerufen wird.
- Ein Basis-Register R1 wird benötigt.
- Register R0, R1 werden verändert und nicht restauriert, R15 wird benutzt und restauriert.

Beschreibung der Parameter:

Name — Wird nicht benutzt

PL/I-Name — Name des nachzuladenden PL/I-Unterprogramms.
Falls nicht angegeben, wird in R1 die Adresse eines LINK-Parameterblocks erwartet, wie er z.B. durch den Makroaufruf LINK MF=L,... generiert wird. In diesem Parameterblock muß mindestens das Namensfeld besetzt sein.

Adr.-Konst — Name einer Adreßkonstante, der die Ladeadresse des nachgeladenen PL/I-Unterprogramms zugewiesen wird.
Falls nicht angegeben, bleibt die Ladeadresse in R1.

LIBNAM= Bibl.-Name
Angabe einer speziellen Ladebibliothek
Voreinstellung NONE bedeutet, daß nur die $TASKLIB-Bibliothek verwendet falls nicht in dem über Register R1 adressierten Parameterblock bereits eine Bibliothek angegeben ist.

Hinweis

- Das nachgeladene Programm wird nicht gestartet (INHIBIT=YES), dies muß mittels Makro P$CALL erfolgen.
- Die Bibliothek TASKLIB (bzw. $TASKLIB) kann mit Hilfe des Kommandos /SYSFILE TASKLIB=Bibliothek einer beliebigen LMR- oder LMS-Bibliothek zugewiesen werden.
- Der Makro sollte nur in Zusammenhang mit dem nachladbaren ("shareable") Laufzeitsystem (LZS) verwendet werden. Im Falle des nichtladbaren LZS (statisches LZS, Steuermodul P$ANFOS#) werden im Pseudo-Register-Vektor (PRV) beim Start des Programms alle benötigten LZS-Moduln adressierbar gemacht, sollte das nachgeladene PL/I-Unterprogramm zusätzliche Moduln benötigen, so sind die entsprechenden Adressen im PRV nicht versorgt, was beim Aufruf zu Adressierungsfehler (ONCODE=8095) führt.

  Abhilfe: Falls auf die Anwendung des statischen LZS nicht verzichtet werden kann, müssen alle betreffenden Moduln explizit ins rufende Programm eingebunden werden.

P$PRV

Generierung einer DSECT oder einer Initialisierungskonstante für den PLI1-Pseudo-Register-Vektor (PRV).

```
&Name    P$PRV  TYPE=  D[SECT] |C [ONST]
```

Regeln:

- Der Makro wird nur einmal je Programmsystem aufgelöst, Steuerung über Globalvariable &PRV
- Die vom Makro generierte DSECT wird in allen P$-Aktionsmakros verwendet, der Makro wird deshalb von diesen Makros aufgerufen.
- Das mit der PRV-DSECT verbundene Basisregister R12 wird nur im Makro P$ENVIRM oder in einem PL/I-Hauptprogramm gesetzt.

Beschreibung der Parameter:

Name
: Name der generierten DSECT bzw. der 4 K-Konstante
Falls nicht angegeben, wird bei TYPE=DSECT voreingestellt.

TYPE = D[SECT]|C[ONST]
: Voreinstellung DSECT
Die Angabe CONST ist im Normalfall für den Benutzer nicht sinnvoll.

Hinweis

- Der Pseudo-Register-Vektor wird vom ASSEMBLER-Benutzerprogramm normalerweise nicht außerhalb der PLI1-Makros angesprochen.

  Ausnahme: Der PRV-Eintrag PLI1GLOBAL (128 Zeiger) kann bei "shareable" Programmierung benutzt werden, um die vordefinierte Zeigermatrix PLI1GLOBAL(0:127) der beteiligten PL/I-Programme anzusprechen als Ersatz für die in diesem Fall nicht erlaubten EXTERNAL- bzw. COMMON-Variablen.
- Die Auflistung der Makro-Auflösung wird unterdrückt. Die davor gültige PRINT-Anweisung bleibt erhalten.

P$REGEQU

Generieren von EQU-Anweisungen für PLI1-Registernotation (RXX)

```
P$REGEQU
```

Regeln:

- Der Makro wird je Programm nur einmal generiert, Steuerung über Globalvariable &REG#.
- Der Makro wird von allen PLI1-Makros verwendet.

Parameter:

keine

Hinweis

Die Generierung des Makros in bestehenden ASSEMBLER-Programmen kann zu M-Flags führen. Falls das Entfernen der Namensgleichheiten Probleme macht, kann die Generierung des Makros durch Setzen von $REG# SETA 1 verhindert werden.

P$RETURN

Rückkehr aus einem ASSEMBLER-Unterprogramm ins rufende PL/I-Programm.

```
Name  P$RETURN      R1RETURN = N[O]|Y[ES] ,
                    STXIT    = N[O]|Y[ES]
```

Regeln:

- Dieser Makro sollte nur in Verbindung mit dem Makro P$ENTRY verwendet werden, d.h. ein ASSEMBLER-Programm darf nur mit P$RETURN verlassen werden, wenn an seinem Eingangspunkt P$ENTRY aufgerufen wurde.
- kein Basis-Register ist notwendig.
- Das Register R13 muß auf demselben Stand stehen, wie es im Makro P$ENTRY gesetzt wurde. Bei Verletzung dieser Regel greift das Programm, in das zurückgekehrt wird, auf falsche Generationen dynamischen Speichers zurück.
- Die beim Aufruf vorhandenen Register-Stände werden wiederhergestellt. Ausnahme: bei Register R1, Parameter R1RETURN = YES

Beschreibung der Parameter:

R1RETURN = N[O] |Y[ES]

Normeinstellung NO bedeutet, daß alle Register auf den Stand beim Aufruf (MakroP$ENTRY) zurückgesetzt werden.
YES muß angegeben werden, wenn die ASSEMBLER-Routine einen Funktionswert über Register R1 zurückgibt. Dies ist nur möglich bei ASSEMBLER-Programmen nach PLI1-Konvention (siehe entsprechende Beschreibung im Abschnitt 7.1.4).

STXIT = N[O] |Y[ES]

Voreinstellung NO bedeutet keine Aktion
YES ist anzugeben, wenn in der ASSEMBLER-Routine ein STXIT-Makro verwendet wird, es wird dann vor Rückkehr ins PL/I-Programm der PLI1-STXIT wiederhergestellt.

Hinweis

- Die PLI1-STXIT-Angabe betrifft alle ERROR-Bedingungen mit ONCODE = 8089 sowie die Benutzbarkeit des /INTR-Kommandos (ATTENTION-Condition).
- Das Register R12 sollte in der ASSEMBLER-Routine nach Möglichkeit nicht benutzt werden, dieser Makro stellt aber den von PLI1 erwarteten Wert wieder her. Voraussetzung ist, daß der Wert von R13 korrekt ist.
- Der STXIT-SVC, der bei STXIT=YES durchgeführt wird, benötigt einige tausend Befehle im Betriebssystem (P2-, P3-Zeit). Er sollte daher, um den allgemeinen Systemdurchsatz nicht ungünstig zu beeinflussen, nicht zu häufig benutzt werden, daher die Voreinstellung STXIT=NO.

P$STACK

Definition einer DSECT zur Adressierung der einzelnen Felder eines dynamischen Speicherbereichs.

```
Name    P$STACK    MAXPAR = Anzahl
```

Regeln:

- Der Makro kann nur einmal je Programm generiert werden, Koordination erfolgt über die Globalvariable &STACK
- Die entstehende DESECT wird mit Basisregister R13 adressiert.
- Die Weitergabe des Parameters MAXPAR an die Makros P$ENTRY und P$ENVIRM erfolgt über die Globalvariable &MAXPAR

Beschreibung der Parameter:

Name
: Name der generierten DSECT
Falls nicht angegeben wird P$STACK angenommen

MAXPAR = Anzahl
: Maximale Anzahl der Parameter, die über P$CALL an ein aufzurufendes PL/I-Unterprogramm weitergegeben werden sollen.
Voreinstellung 64, eine größere Angabe ist nicht zulässig.
Es wird im dynamischen Speicher ein Feld von (Anzahl minus 4) Ganzworten für die PLI1-Parameterliste angelegt, die generierten Namen sind PLIPAR4, PLIPAR5, ..., PLIPAR64.

Hinweis

- Dieser Makro wird in der Regel nicht vom Benutzer, sondern in den Makros P$ENTRY, P$ENVIRM aufgerufen.
- Der Benutzer sollte ihn nur dann selbst aufrufen, wenn er die Voreinstellungen für "Name" und "Anzahl" ändern will. In diesem Fall muß der Makro vor P$ENTRY, P$ENVIRM oder anderen Aktionsmakros aufgerufen werden. Eine Angabe "Anzahl" ≤ 4 ist nur sinnvoll, wenn eine entsprechende Prüfung erfolgen soll.

P$STOP

Beenden des Programms analog zur PL/I-Anweisung STOP;

```
Name  P$STOP
```

Regeln:

– Der Makro darf nur nach P$ENTRY oder P$ENVIRM aufgerufen werden, Prüfung über Globalvariable &ENVIRM#.

Beschreibung der Parameter:

Name Wird nicht verwendet.

A Anhang

A.1 Liste der Warnungen und Fehlermeldungen des Compilers

Eine vollständige Liste der Meldungen, welche der Compiler aufgrund syntaktisch oder semantisch falscher Anweisungen oder wegen Implementierungsbeschränkung ausgeben kann, befindet sich in der Textdatei

```
PLI1.TEXT.D  (deutsch) bzw.
PLI1.TEXT.E  (englisch).
```

Die Texte sind weitgehend selbsterklärend.

Da alle Diagnosemeldungen um eine eindeutige Referenz auf die Quellzeile ergänzt werden, wird das Beseitigen von Fehlern in der Quelle in der Regel einfach sein.

Bei einigen Fehlern wird zusätzlich ein Hinweis gegeben, wie man das Problem beseitigen kann.

In nahezu allen Fällen wird nach Feststellung eines Regelverstoßes und Ausgabe der Meldung mit der Übersetzung normal fortgefahren. Falls der Compiler Annahmen trifft, um weiterarbeiten zu können, so wird auch dies in der Meldung vermerkt.

Wohin die Ausgabe der Meldungen erfolgt, kann mit den Steueranweisungen DIAGNOST und MESSAGE festgelegt werden (siehe Abschnitt 3.5). Die Voreinstellung ist für alle Meldungen SYSLST.

Die Datensätze in den Textdateien haben folgenden Aufbau:

Nr. Gewicht Text

wobei gilt:

Nr.: Die ersten 8 Stellen sind die laufende Nummer des Fehlers.

Gewicht: Die folgenden 3 Stellen haben folgende Bedeutung:

C01	Warnung
C02	leichter Fehler
C03	schwerer Fehler; Bindemodul wird nicht erzeugt.
C04	sehr schwerer Fehler; sofortiger Abbruch der Übersetzung.

Text: Ist im Fehlertext das Zeichen @ enthalten, so wird hierfür der jeweils aktuelle Wert eingesetzt.

Will man die Erzeugung von Bindemoduln von der Anzahl der aufgetretenen Meldungen bestimmten Gewichts abhängig machen, kann man die Steueranweisung OBJECT verwenden (siehe Abschnitt 3.6.1).

Will man die Ausgabe von Meldungen bestimmten Gewichts unterdrücken, kann hierfür die Steueranweisung DIAGNOST eingesetzt werden (siehe Abschnitt 3.5.2).

A.2 Liste der Fehlermeldungen aus Objektprogrammen (ONCODE-Werte)

Treten beim Ablauf eines PL/I-Programms Fehler auf, und es existiert keine der Fehlerklasse zugeordnete ON-Einheit, so wird standardmäßig einer der nachfolgenden Fehlertexte ausgegeben.

Jedem Fehlertext ist ein ONCODE-Wert (Fehlernummer) zugeordnet, der in einer ON-Einheit mit der eingebauten Funktion ONCODE abgerufen werden kann. Nähere Angaben hierzu findet man im Kapitel 11 der Sprachbeschreibung [1].

Bei Fehlern, die vom Datenverwaltungs-System (DVS) entdeckt werden, wird der zugehörige Fehlerschlüssel als Teil eines erklärenden Textes ausgegeben. Die Bedeutung dieser Fehlerschlüssel ist im Manual [6] zu finden.

In der nachfolgenden Liste der ONCODE-Werte und der zugehörigen Meldungen gilt folgendes:

- Die Liste ist nach aufstehenden ONCODE-Werten geordnet.
- Eine Überschrift gibt jeweils an, bei welcher Bedingung die nachfolgenden ONCODES auftreten können. Dabei kann eine Bedingung mehrmals als Überschrift auftauchen.
- Ist mehr als einmal der gleiche ONCODE-Wert mit unterschiedlichen Meldungen vorhanden, so stellt jede Meldung eine Alternative dar. Durch den Aufruf CALL ERROUT wird jeweils die gültige Alternative ausgegeben.
- An Stelle des in der Auflistung im Text angegebenen Zeichens @ wird bei der Angabe im Protokoll jeweils der aktuelle Wert eingesetzt.
- Die folgenden Fehlertexte sind in der ISAM-Datei PLI1.TEXT.D mit dem Schlüssel ’01ccccmm’ abgespeichert, wobei c der ONCODE mit führenden Nullen und m eine Zusatznummer bei gleichem ONCODE ist.

ERROR-Bedingung (siehe auch ONCODE = 9 und 1000)

```
  3    KEINE OTHERWISE-KLAUSEL VORHANDEN UND KEINE WHEN-KLAUSEL
       ERFUELLT

  3    DATENTYP DES RETURN-AUSDRUCKS UNVERTRAEGLICH MIT RETURNS
       ATTRIBUT

  3    RUECKKEHR AUS FUNKTION OHNE FUNKTIONSWERT
```

FINISH-Bedingung

```
  4    PROGRAMM-ENDE WURDE ERREICHT

  4    PROGRAMM-ENDE WURDE SIGNALISIERT
```

ERROR-Bedingung

```
  9    ERROR-CONDITION WURDE SIGNALISIERT
```

NAME-Bedingung

```
 10    GET FILE DATA: NAME SYNTAKTISCH FALSCH
       (ONFILE='@',ONFIELD='@')

 10    GET FILE DATA: NAME LAENGER ALS 256 ZEICHEN
       (ONFILE='@',ONFIELD='@')

 10    GET FILE DATA: NAME NICHT IN VARIABLENLISTE ENTHALTEN
       (ONFILE='@',ONFIELD='@')

 10    GET FILE DATA: ANZAHL DER INDIZES STIMMT NICHT MIT DER
       DEKLARATION UEBEREIN (ONFILE='@',ONFIELD='@')

 10    GET FILE DATA: INDEX LIEGT AUSSERHALB DER DEKLARIERTEN
       GRENZEN (ONFILE='@',ONFIELD='@')

 10    GET FILE DATA: UNZULAESSIGER DATENTYP
       (ONFILE='@',ONFIELD='@')

 10    GET FILE DATA: MEHR ALS 64 QUALIFIKATOREN IM NAMEN
       (ONFILE='@',ONFIELD='@')

 10    GET FILE DATA: NAME NICHT GUELTIG (ONFILE='@',ONFIELD='@')

 10    GET FILE DATA: TEILQUALIFIZIERTER NAME IST IM GLEICHEN BLOCK
       MEHRDEUTIG (ONFILE='@',ONFIELD='@')

 10    NAMENSFEHLER BEI GET DATA VON DATEI @ WURDE SIGNALISIERT
```

RECORD-Bedingung (siehe auch ONCODE = 3 und 1000)

```
20    SATZLAENGEN-FEHLER BEI DATEI @ WURDE SIGNALISIERT

21    VARIABLENLAENGE IST KLEINER ALS SATZLAENGE. (ONFILE='@')

21    SATZLAENGE GROESSER ALS AKTUELLE LINESIZE BEI SYSDTA
      (ONFILE='@')

22    VARIABLENLAENGE IST GROESSER ALS SATZLAENGE. (ONFILE='@')

23    VARIABLENLAENGE IST ZU KLEIN FUER SATZSCHLUESSEL.
      (ONFILE='@')

23    VARIABLENLAENGE IST NULL. (ONFILE='@')
```

TRANSMIT-Bedingung

```
40    EIN-/AUSGABE-UEBERTRAGUNGSFEHLER BEI DATEI @ WURDE
      SIGNALISIERT

41    UNKORRIGIERBARER UEBERTRAGUNGSFEHLER BEI DER AUSGABE
      (ONFILE='@')

41    UNKORRIGIERBARER FEHLER BEI DER AUSGABE.
      (ONFILE='@',DVS-FEHLER:@)

41    RECSIZE GROESSER ALS (BLKSIZE - PAD) ODER
      FALSCHE REIHENFOLGE DER ZUGRIFFE AUF SHARED UPDATE DATEIEN
      (ONFILE='@',DVS-FEHLER:@)

41    NICHT GENUEGEND FREIER SPEICHER FUER SEKUNDAERZUWEISUNG
      VORHANDEN (ONFILE='@',DVS-FEHLER:@)

42    UNKORRIGIERBARER UEBERTRAGUNGSFEHLER BEI DER EINGABE
      (ONFILE='@')

42    UNKORRIGIERBARER FEHLER BEI DER EINGABE.
      (ONFILE='@',DVS-FEHLER:@)
```

KEY-Bedingung

```
50    SATZSCHLUESSEL-FEHLER FUER DATEI @ WURDE SIGNALISIERT

51    DER SPEZIFIZIERTE SATZSCHLUESSEL IST NICHT AUFFINDBAR.
      (ONFILE='@',ONKEY='@',DVS-FEHLER:@)

52    DER SPEZIFIZIERTE SATZSCHLUESSEL IST SCHON VORHANDEN.
      (ONFILE='@',ONKEY='@',DVS-FEHLER:@)

53    DER AKTUELLE SATZSCHLUESSEL IST NICHT GROESSER ALS DER
      VORANGEGANGENE. (ONFILE='@',ONKEY='@')

53    DER AKTUELLE SATZSCHLUESSEL IST KLEINER ALS DER
      VORANGEGANGENE. (ONFILE='@',ONKEY='@')

53    DER AKTUELLE SATZSCHLUESSEL IST NICHT GROESSER ALS DER
      VORANGEGANGENE. (ONFILE='@',ONKEY='@',DVS-FEHLER:'@')

54    DER SPEZIFIZIERTE SATZSCHLUESSEL IST NICHT KONVERTIERBAR.
      (ONFILE='@',ONKEY='@')

55    DER SPEZIFIZIERTE SATZSCHLUESSEL IST UNZULAESSIG.
      (ONFILE='@',ONKEY='@')

56    DER SPEZIFIZIERTE SATZSCHLUESSEL IST AUSSERHALB DER
      REGIONALDATEI. (ONFILE='@',ONKEY='@')

57    FUER DEN SPEZIFIZIERTEN SATZ IST KEIN SPEICHERRAUM
      VORHANDEN. (ONFILE='@',ONKEY='@',DVS-FEHLER:@)

57    FUER DEN SPEZIFIZIERTEN SATZ IST KEIN SPEICHERRAUM
      VORHANDEN. (ONFILE='@',ONKEY='@')
```

ENDFILE-Bedingung

```
70    DATEIENDE IST ERREICHT. (ONFILE='@')

70    VERSUCH, BEI STREAM INPUT NACH DATEIENDE EIN WEITERES
      DATENELEMENT ZU LESEN (ONFILE='@')

70    DATEIENDE IST ERREICHT. (ONFILE='@',DVS-FEHLER:@)

70    DATEI-ENDE FUER DATEI @ WURDE SIGNALISIERT

71    BANDDATEIENDE : DOPPELTE BANDMARKE ERREICHT   (ONFILE='@')
```

UNDEFINEDFILE-Bedingung

```
80    FEHLER IN DEN KENNDATEN DER DATEI @ WURDE SIGNALISIERT

81    WIDERSPRUECHLICHE DECLARE- UND OPEN-ATTRIBUTE.  (ONFILE='@')

81    WIDERSPRUECHLICHE ATTRIBUTE INPUT UND OUTPUT IN DECLARE UND
      OPEN.  (ONFILE='@')

81    WIDERSPRUECHLICHE ATTRIBUTE INPUT UND UPDATE IN DECLARE UND
      OPEN.  (ONFILE='@')

81    WIDERSPRUECHLICHE ATTRIBUTE OUTPUT UND UPDATE IN DECLARE UND
      OPEN. (ONFILE='@')

81    WIDERSPRUECHLICHE ATTRIBUTE RECORD UND STREAM IN DECLARE UND
      OPEN. (ONFILE='@')

81    WIDERSPRUECHLICHE ATTRIBUTE SEQUENTIAL UND DIRECT IN DECLARE
      UND OPEN. (ONFILE='@')

81    WIDERSPRUECHLICHE ATTRIBUTE BACKWARDS UND STREAM IN DECLARE
      UND OPEN (ONFILE='@')

81    UNZULAESSIGE ATTRIBUTE VON DECLARE BZW OPEN BEI TRANSIENT.
      (ONFILE='@')

82    CONSECUTIVE UND DIRECT BZW KEYED SIND WIDERSPRUECHLICH.
      (ONFILE='@')

82    INDEXED ODER REGIONAL WIDERSPRUECHLICH ZU PRINT, STREAM,
      TRANSIENT ODER BACKWARDS. (ONFILE='@')

82    OUTPUT OHNE KEYED BEI INDEXED ODER REGIONAL NICHT ZULAESSIG
      (ONFILE='@')

82    DIRECT OUTPUT UND INDEXED SIND WIDERSPRUECHLICH.
      (ONFILE='@')

82    UNZULAESSIGE ENVIRONMENT-ANGABE BEI SYSTEM-DATEI.
      (ONFILE='@')

82    DATEIEN MIT NAMEN *DUMMY SIND FUER
      REGIONAL(1/3)-ORGANISATION VERBOTEN (ONFILE='@')

82    WIDERSPRUCH ZWISCHEN FILE-ATTRIBUTEN UND PHYSIKALISCHER
      ORGANISATION : UPDATE NUR BEI DIREKTZUGRIFFS-TRAEGERN
      ERLAUBT  (ONFILE='@')

82    OUTPUT BEI BTAM NICHT ERLAUBT. (ONFILE='@')

82    WIDERSPRUCH ZWISCHEN FILE-ATTRIBUTEN UND PHYSIKALISCHER
      ORGANISATION : INDEXED NUR AUF DIREKTZUGRIFFS-TRAEGERN
      ERLAUBT (ONFILE='@')

82    WIDERSPRUCH ZWISCHEN FILE-ATTRIBUTEN UND -ORGANISATION :
      TRAEGERANGABEN FALSCH ODER WIDERSPRECHEND (ONFILE='@' ,
      DVS-FEHLER : @ )
```

```
83    DATEI-KENNDATEN UNVOLLSTAENDIG : RECFORM FEHLT (ONFILE='@')

83    DATEI-KENNDATEN UNVOLLSTAENDIG:RECSIZE FUER REGIONAL(1)-FILE
      FEHLT (ONFILE='@')

83    DATEI-KENNDATEN UNVOLLSTAENDIG:RECSIZE FUER REGIONAL(3)-FILE
      FEHLT (ONFILE='@')

83    DATEI-KENNDATEN UNVOLLSTAENDIG: KEYLEN FUER REGIONAL(3)-FILE
      FEHLT (ONFILE='@')

83    DATEI-KENNDATEN UNVOLLSTAENDIG : RECSIZE ODER BLKSIZE FUER
      MAGNETBAND-DATEI FEHLEN (ONFILE='@')

83    DATEI-KENNDATEN UNVOLLSTAENDIG : RECSIZE FUER DATEI MIT
      RECFORM=F FEHLT (ONFILE='@')

84    KEIN GUELTIGES FILE-KOMMANDO GEGEBEN (ONFILE='@' ,
      DVS-FEHLER : @ )

85    REGIONAL-DATEI KANN NICHT FORMATIERT WERDEN. (ONFILE='@')

86    WIDERSPRUCH ZWISCHEN ENVIRONMENT-OPTION UND DATEIKENNDATEN:
      DER WERT VON LINESIZE IST MIT DEM VON BLKSIZE|RECSIZE|KEYLEN
      UND RECFORM UNVERTRAEGLICH (ONFILE='@')

86    LINESIZE =< 0. (ONFILE='@')

86    WIDERSPRUCH ZWISCHEN LINESIZE/PAGESIZE UND GERAETEKENNDATEN
      BEI BLOCKMODUS (ONFILE='@')

86    BLOCKMODUS AUF DIESEM GERAET NICHT MOEGLICH

87    WIDERSPRUCH ZWISCHEN DATEI-KENNDATEN (ONFILE='@' ,
      DVS-FEHLER : @ )

87    REGIONAL(1)-FILES BENOETIGEN FCBTYPE=PAM IM FILE-KOMMANDO
      (ONFILE='@')

87    DATEI NICHT REGIONAL(1)-ORGANISIERT  (ONFILE='@')

87    WIDERSPRUCH ZWISCHEN FILE-ATTRIBUTEN UND DATEI-KENNDATEN:
      REGIONSIZE > SPACE BEI REGIONAL(1)-FILE  (ONFILE='@')

87    REGIONAL(3)-FILES BENOETIGEN FCBTYPE=PAM IM FILE-KOMMANDO
      (ONFILE='@')

87    WIDERSPRUCH ZWISCHEN FILE-ATTRIBUTEN UND DATEI-KENNDATEN:
      REGIONSIZE > SPACE BEI REGIONAL(3)-FILE  (ONFILE='@')

87    WIDERSPRUCH ZWISCHEN FILE-ATTRIBUTEN UND DATEI-KENNDATEN:
      RECSIZE > REGIONSIZE BEI REGIONAL(3)-FILE  (ONFILE='@')

87    FEHLERHAFTE REGIONSIZE(=BLKSIZE) BEI REGIONAL(3)-FILE:
      BLKSIZE MUSS VON DER FORM STD|(STD,N) MIT N>0 SEIN
      (ONFILE='@')
```

```
87    WIDERSPRUCH ZWISCHEN DATEI-KENNDATEN: RECFORM, RECSIZE,
      BLKSIZE SIND FALSCH ODER UNVERTRAEGLICH ODER GAR NICHT
      ANGEGEBEN (ONFILE='@' , DVS-FEHLER : @ )

87    WIDERSPRUCH ZWISCHEN DATEI-KENNDATEN: KEYANGABEN FEHLERHAFT
      ODER MIT RECFORM|RECSIZE UNVERTRAEGLICH (ONFILE='@' ,
      DVS-FEHLER: @)

87    WIDERSPRUCH ZWISCHEN DATEI-KENNDATEN: KEYANGABEN FEHLERHAFT
      ODER MIT RECFORM|RECSIZE UNVERTRAEGLICH (ONFILE='@' ,
      DVS-FEHLER : @ )

87    WIDERSPRUCH ZWISCHEN FILE-ATTRIBUTEN UND DATEI-KENNDATEN :
      BEI OUTPUT-FILES MUSS VERFALLSDATUM ERREICHT SEIN | DARF
      NICHT STATE = FOREIGN SEIN (ONFILE='@' , DVS-FEHLER : @ )

87    FALSCH: RECSIZE 0 BEI F-FORMAT. (ONFILE=@)

87    RECSIZE GROESSER ALS BEI WERT VON BLKSIZE
      ZULAESSIG.(ONFILE='@')

87    WIDERSPRUCH ZWISCHEN DATEIKENNDATEN : DATEIEN AUF
      DIREKTZUGRIFFSTRAEGERN  ERFORDERN
      BLKSIZE=(STD,N),1=<N=<16.(ONFILE='@')

87    SCALARVARYING UND UNVERTRAEGLICHE KEYPOS ODER PRINT-DATEI
      ODER VORSCHUB IM FILE-KOMMANDO SPEZIFIZIERT. (ONFILE='@')

87    WIDERSPRUCH ZWISCHEN ENVIRONMENT-OPTION UND DATEI-KENNDATEN:
      INDEXED-ORGANISIERTE FILES BENOETIGEN FCBTYPE=ISAM IM
      FILE-KOMMANDO (ONFILE='@')

87    WIDERSPRUCH ZWISCHEN DATEI-KENNDATEN:  BEI FCBTYPE=ISAM IM
      FILE-KOMMANDO MUSS RECFORM=F/V SEIN  (ONFILE='@')

87    WIDERSPRUCH ZWISCHEN ENVIRONMENT-OPTION UND DATEIKENNDATEN:
      UNZULAESSIGE KEY-ANGABEN FUER CONSECUTIVE-ORGANISIERTE FILES
      (ONFILE=@)

87    WIDERSPRUCH ZWISCHEN DEM WERT VON KEYLEN ODER KEYPOS IM
      FILE- KOMMANDO/ENVIRONMENT-OPTIONS UND DEM KATALOGEINTRAG
      (ONFILE='@')

87    WIDERSPRUCH IN ENVIRONMENT-OPTIONS : FUER INDEXED
      ORGANISIERTE DATEIEN DARF NICHT RECFORM=U ANGEGEBEN WERDEN
      (ONFILE='@')

87    WIDERSPRUCH IN ENVIRONMENT-OPTIONS : BEI KEYLOC(0) MUSS AUCH
      RECFORM IN DEN ENVIRONMENT-OPTIONS ANGEGEBEN SEIN
      (ONFILE='@')

87    WIDERSPRUCH ZWISCHEN FILE-ATTRIBUTEN UND DATEIKENNDATEN: BEI
      STREAM FILES DARF NICHT FCBTYPE=PAM IM FILE-KOMMANDO
      ANGEGEBEN WERDEN (ONFILE='@')

87    WIDERSPRUCH ZWISCHEN FILE-ATTRIBUTEN UND DATEIKENNDATEN: BEI
      PRINT FILES DARF NICHT FCBTYPE=PAM|ISAM ANGEGEBEN WERDEN
      (ONFILE='@')
```

```
87    DATEIKENNDATEN SIND WIDERSPRUECHLICH: BEI PRINT-DATEIEN
      MUESSEN DIE WERTE VON RECFORM UND RECSIZE|BLKSIZE WENIGSTENS
      EIN DATENZEICHEN PRO SATZ ZULASSEN (ONFILE='@')

87    WIDERSPRUCH ZWISCHEN FILE-ATTRIBUTEN UND DATEIKENNDATEN :
      BACKWARDS BENOETIGT FCBTYPE=SAM IM FILE-KOMMANDO
      (ONFILE='@')

88    WIDERSPRUCH ZWISCHEN FILE-ORGANISATION UND DATEI-KENNDATEN:
      REGIONAL(1)-ORGANISIERTE FILES BENOETIGEN RECFORM=F IM
      FILE-KDO  (ONFILE='@')

88    WIDERSPRUCH ZWISCHEN FILE-ORGANISATION UND DATEI-KENNDATEN:
      REGIONAL(3)-ORGANISIERTE FILES BENOETIGEN BEI FCBTYPE=PAM
      RECFORM=F|V IM FILE-KOMMANDO (ONFILE='@')

88    WIDERSPRUCH ZWISCHEN CATALOG UND ENVIRONMENT: VORSCHUB.
      (ONFILE='@')

88    WIDERSPRUCH ZWISCHEN CATALOG UND ENVIRONMENT: RECSIZE.
      (ONFILE='@')

88    WIDERSPRUCH ZWISCHEN CATALOG UND ENVIRONMENT: RECFORM.
      (ONFILE='@')

88    WIDERSPRUCH ZWISCHEN CATALOG UND ENVIRONMENT: KEYLEN.
      (ONFILE='@')

88    WIDERSPRUCH ZWISCHEN CATALOG UND ENVIRONMENT: KEYLOC.
      (ONFILE='@')

89    PASSWORT UNGUELTIG ODER NICHT ANGEGEBEN  (ONFILE='@')
```

ENDPAGE-Bedingung

```
90    LINENO ERREICHT PAGESIZE + 1 (ONFILE='@')

90    SEITENUEBERLAUF FUER DATEI @ WURDE SIGNALISIERT
```

UNDEFINEDFILE-Bedingung

```
 93    NICHT IDENTIFIZIERBARER DVS-FEHLER (ONFILE='@' , DVS-FEHLER:
       @ )

 93    BEIM OEFFNEN EINER DATEI MELDETE DAS DVS EINEN FUER DAS
       E/A-SYSTEM UNSPEZIFISCHEN FEHLER (EVTL. BLKSIZE|SPACE
       FALSCH|UNGENUEGEND) (ONFILE='@',DVS-FEHLER: @)

 93    NICHT GENUEGEND KLASSE-5-SPEICHER VORHANDEN
       (ONFILE='@',DVS-FEHLER:@)

 93    FCBTYPE IM FILE-KOMMANDO STIMMT NICHT MIT DEM IM
       KATALOGEINTRAG UEBEREIN (ONFILE='@')

 94    VERSUCH, LEEREN ODER NICHT EXISTIERENDEN FILE INPUT ZU
       EROEFFNEN (ONFILE='@' , DVS-FEHLER: @ )

 94    INPUT VERSUCHT AUF NOCH NICHT KATALOGISIERTE DATEI.
       (ONFILE='@')

 95    VERSUCH , EINEM FILE , DER MIT 'CLOSE LEAVE' GESCHLOSSEN
       WURDE , BEIM EROEFFNEN EINE ANDERE DATEI ZUZUWEISEN
       (ONFILE='@')

 96    BACKWARDS/OUTPUT NACH CLOSE LEAVE BEI LABEL=NSTD NICHT
       ZULAESSIG  (ONFILE='@')

 98    NUR EINE TRANSIENT-DATEI PRO JOB MOEGLICH. (ONFILE='@')

 99    VERSUCH, EINE BLOCKIERTE ODER NICHT MEHRBENUTZBARE DATEI ZU
       EROEFFNEN (ONFILE='@' , DVS-FEHLER: @ )

110    KEIN SPACE FUER REGIONAL-DATEI. (ONFILE='@')

110    WERT DES SPACE-PARAMETERS IM FILE-KOMMANDO ZU KLEIN ODER IM
       WIDERSPRUCH ZUM WERT VON BLKSIZE.(ONFILE='@')

110    KEIN SPACE FUER ISAM-DATEI. (ONFILE='@')
```

STRINGSIZE-Bedingung

```
150    VERLUST GUELTIGER ZEICHEN BEI STRING-ZUWEISUNG

150    @: ZEICHENVERLUST BEI ZUWEISUNG.@ ZEICHEN SOLLEN UEBERTRAGEN
       WERDEN, DAS ZIEL UMFASST@ ZEICHEN.

150    @: INFORMATIONSVERLUST BEI ZUWEISUNG.@ BITS SOLLEN
       UEBERTRAGEN WERDEN, DAS ZIEL UMFASST@ BITS.

150    VERLUST GUELTIGER ZEICHEN BEI DISPLAY-ANWEISUNG

150    VERLUST GUELTIGER ZEICHEN BEI STRING-ZUWEISUNG WURDE
       SIGNALISIERT
```

OVERFLOW-Bedingung

```
300    GLEITKOMMA-UEBERLAUF AUFGETRETEN

300    @ EINFACH GENAU: UEBERGELAUFENES ERGEBNIS

300    @ EINFACH GENAUE POTENZIERUNG: UEBERGELAUFENES ERGEBNIS

300    @ EINFACH GENAU: SINGULAERE STELLE BEI (2*N+1)*PI/2

300    @ DOPPELT GENAU: UEBERGELAUFENES ERGEBNIS

300    @ DOPPELT GENAUE POTENZIERUNG: UEBERGELAUFENES ERGEBNIS

300    @ DOPPELT GENAU: SINGULAERE STELLE BEI (2*N+1)*PI/2

300    @ ERWEITERT GENAU: UEBERGELAUFENES ERGEBNIS

300    @ ERWEITERT GENAUE POTENZIERUNG: UEBERGELAUFENES ERGEBNIS

300    @ ERWEITERT GENAU: SINGULAERE STELLE BEI (2*N+1)*PI/2

300    @ KOMPLEX EINFACH GENAU: ABS(REAL(X)) > 174.673

300    @ KOMPLEX EINFACH GENAU: ABS(IMAG(X)) > 174.673

300    @ KOMPLEX EINFACH GENAU: ABS(2 * IMAG(X)) > 174.673

300    @ KOMPLEX EINFACH GENAU: ABS(2 * REAL(X)) > 174.673

300    @ KOMPLEXE EINFACH GENAUE POTENZIERUNG: UEBERLAUF DES
       ZWISCHENWERTES ABS(REAL(D2 * LOG(C2))) > 174.673

300    @ KOMPLEX EINFACH GENAU: SINGULAERE STELLE (2*N+1)*PI/2 + 0I

300    @ KOMPLEX EINFACH GENAU: SINGULAERE STELLE  0 +
       ((2*N+1)*PI/2)I

300    @ KOMPLEX DOPPELT GENAU: ABS(REAL(X)) > 174.673

300    @ KOMPLEX DOPPELT GENAU: ABS(IMAG(X)) > 174.673

300    @ KOMPLEX DOPPELT GENAU: ABS(2 * IMAG(X)) > 174.673

300    @ KOMPLEX DOPPELT GENAU: ABS(2 * REAL(X)) > 174.673

300    @ KOMPLEXE DOPPELT GENAUE POTENZIERUNG: UEBERLAUF DES
       ZWISCHENWERTES ABS(REAL(D4 * LOG(C4))) > 174.673

300    @ KOMPLEX DOPPELT GENAU: SINGULAERE STELLE (2*N+1)*PI/2 + 0I

300    @ KOMPLEX DOPPELT GENAU: SINGULAERE STELLE  0 +
       ((2*N+1)*PI/2)I

300    @ KOMPLEX ERWEITERT GENAU: ABS(REAL(X)) > 174.673

300    @ KOMPLEX ERWEITERT GENAU: ABS(IMAG(X)) > 174.673
```

```
300    @ KOMPLEX ERWEITERT GENAU: ABS(2 * IMAG(X)) > 174.673

300    @ KOMPLEX ERWEITERT GENAU: ABS(2 * REAL(X)) > 174.673

300    @ KOMPLEXE ERWEITERT GENAUE POTENZIERUNG: UEBERLAUF DES
       ZWISCHENWERTES ABS(REAL(D8 * LOG(C8))) > 174.673

300    @ KOMPLEX ERWEITERT GENAU: SINGULAERE STELLE (2*N+1)*PI/2 +
       0I

300    @ KOMPLEX ERWEITERT GENAU: SINGULAERE STELLE  0 +
       ((2*N+1)*PI/2)I

300    @ KOMPLEXE ERWEITERT GENAUE GANZZAHLIGE POTENZIERUNG:
       EXPONENTENUEBERLAUF

300    @ KOMPLEX ERWEITERT GENAU: EXPONENTENUEBERLAUF

300    ROUND(X,K): EXPONENTENUEBERLAUF

300    GLEITKOMMA-UEBERLAUF WURDE SIGNALISIERT
```

FIXEDOVERFLOW-Bedingung

```
310    FESTKOMMA-UEBERLAUF AUFGETRETEN (BINAER ODER DEZIMAL)

310    @ KOMPLEX DEZIMAL: UEBERGELAUFENES ERGEBNIS

310    @ COMPLEX BIN FIXED: HALBWORT-ERGEBNIS UEBERGELAUFEN

310    FESTKOMMA-UEBERLAUF WURDE SIGNALISIERT
```

ZERODIVIDE-Bedingung

```
320    DIVISION DURCH NULL WURDE VERSUCHT

320    @ ERWEITERT GENAU: DIVISION DURCH NULL

320    @ COMPLEX BIN FIXED: DIVISION DURCH NULL

320    @ ERWEITERT GENAU: ZWEITES ARGUMENT GLEICH NULL

320    @ KOMPLEX ERWEITERT GENAU: DIVISION DURCH NULL

320    @ KOMPLEX EINFACH GENAU: DIVISION DURCH NULL

320    @ KOMPLEX DOPPELT GENAU: DIVISION DURCH NULL

320    DIVISION DURCH NULL WURDE SIGNALISIERT
```

UNDERFLOW-Bedingung

```
330    GLEITKOMMA-UNTERLAUF AUFGETRETEN

330    @ KOMPLEXE ERWEITERT GENAUE GANZZAHLIGE POTENZIERUNG:
       EXPONENTENUNTERLAUF

330    @ KOMPLEX ERWEITERT GENAU: EXPONENTENUNTERLAUF

330    GLEITKOMMA-UNTERLAUF WURDE SIGNALISIERT
```

SIZE-Bedingung

```
340    VERLUST FUEHRENDER GUELTIGER ZIFFERN BEI ZUWEISUNG

340    VERLUST FUEHRENDER GUELTIGER ZIFFERN BEI KONVERTIERUNG

340    @ KOMPLEX DEZIMAL: VERLUST FUEHRENDER ZIFFERN, DA
       GENAUIGKEIT P DER ERGEBNISVARIABLEN ZU GERING

340    @ COMPLEX BIN FIXED: ERGEBNIS BELEGT MEHR STELLEN ALS
       DEKLARIERT, STEHT JEDOCH KORREKT IM
       ERGEBNIS-MASCHINEN(HALB)WORT.

340    VERLUST FUEHRENDER GUELTIGER ZIFFERN WURDE SIGNALISIERT

341    VERLUST FUEHRENDER GUELTIGER ZIFFERN BEI KONVERTIERUNG
       WAEHREND EINES EIN-/AUSGABE VORGANGS
```

STRINGRANGE-Bedingung

```
350    SUBSTRING UEBERSCHREITET STRINGGRENZE

350    FEHLER BEI SUBSTRING WURDE SIGNALISIERT
```

AREA-Bedingung

```
360    SPEICHERMANGEL BEI ALLOCATE-ANWEISUNG IN EINEM BENANNTEN
       AREA

361    SPEICHERMANGEL BEI ASSIGN-ANWEISUNG IN EINEM BENANNTEN AREA

362    SPEICHERMANGEL IN EINEM BENANNTEN AREA WURDE SIGNALISIERT
```

ATTENTION-Bedingung

```
400    UNTERBRECHUNG DURCH '/INTR @' KOMMANDO VOM TERMINAL

400    UNTERBRECHUNG DURCH 'SIGNAL ATTENTION' STATEMENT
```

CONDITION-Bedingung

```
500    @-CONDITION WURDE SIGNALISIERT
```

SUBSCRIPTRANGE-Bedingung

```
520    INDEX AUSSERHALB DER VEREINBARTEN FELDGRENZEN

520    INDEX-FEHLER WURDE SIGNALISIERT

521    ISUB-INDEX AUSSERHALB DER VEREINBARTEN FELDGRENZEN
```

STORAGE-Bedingung

```
530    MAXIMALE GROESSE DES STANDARD-AREA LT. STORAGE OPTION
       UEBERSCHRITTEN

531    SPEICHERMANGEL BEI DER STANDARD-AREA-VERWALTUNG

540    SPEICHERMANGEL WURDE SIGNALISIERT

550    SEGMENTNUMMERN-ENGPASS BEI DER STACK-VERWALTUNG

551    SPEICHERMANGEL BEI DER STACK-VERWALTUNG

553    SPEICHERFORDERUNG IM STACK UEBERSCHREITET ADDRESSENRAUM
```

CONVERSION-Bedingung

```
600    FEHLER BEI KONVERTIERUNG IN EINEN CHARACTER STRING BEI GET
       STRING ANWEISUNG (ONSOURCE='@',ONCHARPOS=@)

600    KONVERTIERUNGSFEHLER WURDE SIGNALISIERT

601    FEHLER BEI KONVERTIERUNG IN EINEN CHARACTER STRING BEI GET
       FILE ANWEISUNG (ONFILE='@',ONSOURCE='@',ONCHARPOS=@)

601    FEHLER BEI KONVERTIERUNG IN EINEN CHARACTER STRING BEI GET
       STRING ANWEISUNG (ONSOURCE='@',ONCHARPOS=@)

602    FEHLER BEI KONVERTIERUNG IN EINEN CHARACTER STRING MIT GET
       FILE ANWEISUNG NACH EINEM UEBERTRAGUNGSFEHLER
       (ONFILE='@',ONSOURCE='@',ONCHARPOS=@)

603    FEHLER BEI KONVERTIERUNG VON F-FORMAT BEI EINGABE MIT GET
       STRING ANWEISUNG (ONSOURCE='@',ONCHARPOS=@)

604    FEHLER BEI KONVERTIERUNG VON F-FORMAT BEI EINGABE MIT GET
       FILE ANWEISUNG (ONFILE='@',ONSOURCE='@',ONCHARPOS=@)

605    FEHLER BEI KONVERTIERUNG VON F-FORMAT BEI EINGABE MIT GET
       FILE ANWEISUNG NACH EINEM UEBERTRAGUNGSFEHLER
       (ONFILE='@',ONSOURCE='@',ONCHARPOS=@)
```

```
606    FEHLER BEI KONVERTIERUNG VON E-FORMAT BEI EINGABE MIT GET
       STRING ANWEISUNG (ONSOURCE='@',ONCHARPOS=@)

607    FEHLER BEI KONVERTIERUNG VON E-FORMAT BEI EINGABE MIT GET
       FILE ANWEISUNG (ONFILE='@',ONSOURCE='@',ONCHARPOS=@)

608    FEHLER BEI KONVERTIERUNG VON E-FORMAT BEI EINGABE MIT GET
       FILE ANWEISUNG NACH EINEM UEBERTRAGUNGSFEHLER
       (ONFILE='@',ONSOURCE='@',ONCHARPOS=@)

609    FEHLER BEI KONVERTIERUNG VON B-FORMAT BEI EINGABE MIT GET
       STRING ANWEISUNG (ONSOURCE='@',ONCHARPOS=@)

610    FEHLER BEI KONVERTIERUNG VON B-FORMAT BEI EINGABE MIT GET
       FILE ANWEISUNG (ONFILE='@',ONSOURCE='@',ONCHARPOS=@)

611    FEHLER BEI KONVERTIERUNG VON B-FORMAT BEI EINGABE MIT GET
       FILE ANWEISUNG NACH EINEM UEBERTRAGUNGSFEHLER
       (ONFILE='@',ONSOURCE='@',ONCHARPOS=@)

612    KONVERTIERUNG VON CHARAKTERSTRING NACH ARITHMETISCH
       (ONSOURCE='@',ONCHARPOS=@)

612    FEHLER BEI KONVERTIERUNG VON CHARACTER NACH ARITHMETISCH BEI
       EIN- ODER AUSGABE MIT GET ODER PUT STRING ANWEISUNG
       (ONSOURCE='@',ONCHARPOS=@)

613    FEHLER BEI KONVERTIERUNG VON CHARACTER NACH ARITHMETISCH BEI
       EIN- ODER AUSGABE MIT GET ODER PUT FILE ANWEISUNG
       (ONFILE='@',ONSOURCE='@',ONCHARPOS=@)

614    FEHLER BEI KONVERTIERUNG VON CHARACTER NACH ARITHMETISCH BEI
       EINGABE MIT GET FILE ANWEISUNG NACH EINEM
       UEBERTRAGUNGSFEHLER (ONFILE='@',ONSOURCE='@',ONCHARPOS=@)

615    KONVERTIERUNG VON CHARAKTERSTRING NACH BITSTRING
       (ONSOURCE='@',ONCHARPOS'=@)

615    FEHLER BEI KONVERTIERUNG VON CHARACTER NACH BIT BEI EIN-
       ODER AUSGABE MIT GET ODER PUT STRING ANWEISUNG
       (ONSOURCE='@',ONCHARPOS'=@)

616    FEHLER BEI KONVERTIERUNG VON CHARACTER NACH BIT BEI EIN-
       ODER AUSGABE MIT GET ODER PUT FILE ANWEISUNG
       (ONFILE='@',ONSOURCE='@',ONCHARPOS=@)

617    FEHLER BEI KONVERTIERUNG VON CHARACTER NACH BIT BEI EINGABE
       MIT GET FILE ANWEISUNG NACH EINEM UEBERTRAGUNGSFEHLER
       (ONFILE='@',ONSOURCE='@',ONCHARPOS=@)

618    KONVERTIERUNG VON CHARAKTERSTRING NACH PICTURE
       (ONSOURCE='@',ONCHARPOS=@)

618    FEHLER BEI KONVERTIERUNG VON CHARACTER NACH PICTURE
       CHARACTER STRING BEI EIN- ODER AUSGABE MIT GET ODER PUT
       STRING ANWEISUNG (ONSOURCE='@',ONCHARPOS=@)
```

```
619    FEHLER BEI KONVERTIERUNG VON CHARACTER NACH PICTURE
       CHARACTER STRING BEI EIN- ODER AUSGABE MIT GET ODER PUT FILE
       ANWEISUNG (ONFILE='@',ONSOURCE='@',ONCHARPOS=@)

620    FEHLER BEI KONVERTIERUNG VON CHARACTER NACH PICTURE
       CHARACTER STRING BEI EINGABE MIT GET FILE ANWEISUNG NACH
       EINEM UEBERTRAGUNGSFEHLER
       (ONFILE='@',ONSOURCE='@',ONCHARPOS=@)

621    FEHLER BEI KONVERTIERUNG VON P-FORMAT (ARITH.) BEI EINGABE
       MIT GET STRING ANWEISUNG (ONSOURCE='@',ONCHARPOS=@)

622    FEHLER BEI KONVERTIERUNG VON P-FORMAT (ARITH.) BEI EINGABE
       MIT GET FILE ANWEISUNG (ONFILE='@',ONSOURCE='@',ONCHARPOS=@)

623    FEHLER BEI KONVERTIERUNG VON P-FORMAT (ARITH.) BEI EINGABE

       MIT GET FILE ANWEISUNG NACH EINEM UEBERTRAGUNGSFEHLER
       (ONFILE='@',ONSOURCE='@',ONCHARPOS=@)

624    FEHLER BEI KONVERTIERUNG VON P-FORMAT (CHAR.) BEI EINGABE
       MIT GET STRING ANWEISUNG (ONSOURCE='@',ONCHARPOS=@)

625    FEHLER BEI KONVERTIERUNG VON P-FORMAT (CHAR.) BEI EINGABE
       MIT GET FILE ANWEISUNG (ONFILE='@',ONSOURCE='@',ONCHARPOS=@)

626    FEHLER BEI KONVERTIERUNG VON P-FORMAT (CHAR.) BEI EINGABE
       MIT GET FILE ANWEISUNG NACH EINEM UEBERTRAGUNGSFEHLER
       (ONFILE='@',ONSOURCE='@',ONCHARPOS=@)
```

ERROR-Bedingung (siehe auch ONCODE = 3 und 9)

```
1002   GET/PUT STRING UEBERSCHREITET STRINGLAENGE

1003   FEHLER BEI CLOSE NACH VORANGEGANGENER KEY-CONDITION
       (ONFILE='@')

1004   PAGE/LINE NUR BEI PRINT-DATEIEN ERLAUBT (ONFILE='@')

1004   PAGE/LINE/SKIP BEI PUT STRING NICHT ERLAUBT

1004   SKIP/COLUMN BEI GET STRING NICHT ERLAUBT

1007   VORANGEGANGENE READ-ANWEISUNG VOR REWRITE- ODER
       DELETE-ANWEISUNG  FEHLT (ONFILE='@')

1008   FEHLER BEI GET STRING DATA (ONFIELD='@')

1009   E/A-ANWEISUNG UND FILE-ATTRIBUTE SIND UNVERTRAEGLICH
       (ONFILE='@')

1011   E/A-FEHLER

1016   ERFOLGLOSES IMPLIZITES OPEN (ONFILE='@')

1017   VERSUCH , MIT EINEM OPEN/READ HINTER DAS BANDDATEIENDE ZU
       POSITIONIEREN   (ONFILE='@')
```

```
1018    UNERWARTETES DATEIENDE BEI STREAM INPUT (GELESENES
        DATENELEMENT UNVOLLSTAENDIG) (ONFILE='@')

1101    FEHLER IM EIN/AUSGABESYSTEM. (ONFILE='@')

1102    FEHLER BEIM BEARBEITEN VON REGIONALDATEIEN. (ONFILE='@')

1400    FREIGABE-VERSUCH FUER NICHT ZUGEWIESENEN SPEICHER IM
        STANDARD-AREA

1401    FREIGABE-VERSUCH IM STANDARD-AREA NICHT AUF
        DOPPELWORT-GRENZE

1410    NACH AREA-CONDITION RUECKKEHR AUS ON-UNIT OHNE
        SPEICHERFREIGABE

1411    FREIGABE-VERSUCH FUER NICHT ZUGEWIESENEN SPEICHER IN EINEM
        BENANNTEN AREA

1412    FREIGABE-VERSUCH IN LUECKE IN EINEM BENANNTEN AREA

1413    WIDERSPRUECHLICHE VERWALTUNGSINFORMATION EINES BENANNTEN
        AREA

1414    FREIGABE-VERSUCH IN EINEM BENANNTEN AREA NICHT AUF
        DOPPELWORTGRENZE

1500    @ EINFACH GENAU: ARGUMENT IST NEGATIV

1501    @ DOPPELT GENAU: ARGUMENT IST NEGATIV

1502    @ ERWEITERT GENAU: ARGUMENT IST NEGATIV

1503    @ ERWEITERT GENAU: ARGUMENT IST NICHT POSITIV

1503    @ ERWEITERT GENAUE POTENZIERUNG: BASIS IST NEGATIV

1504    @ EINFACH GENAU: ARGUMENT IST NICHT POSITIV

1504    @ EINFACH GENAUE POTENZIERUNG: BASIS IST NEGATIV

1505    @ DOPPELT GENAU: ARGUMENT IST NICHT POSITIV

1505    @ DOPPELT GENAUE POTENZIERUNG: BASIS IST NEGATIV

1506    @ EINFACH GENAU: ABSOLUTBETRAG DES ARGUMENTES IST GROESSER
        (2**18) * PI

1506    @ EINFACH GENAU: ABSOLUTBETRAG DES ARGUMENTES IST GROESSER
        (2**18) * 180

1506    @ KOMPLEX EINFACH GENAU: ABS(IMAG(X)) > 2**18 * PI

1506    @ KOMPLEX EINFACH GENAU: ABS(REAL(X)) > 2**18 * PI

1506    @ KOMPLEX EINFACH GENAU: ABS(2   REAL(X)) > 2**18 * PI
```

```
1506    @ KOMPLEX EINFACH GENAU: ABS(2   IMAG(X)) > 2**18 * PI

1506    @ KOMPLEXE EINFACH GENAUE POTENZIERUNG: UEBERLAUF DES
        ZWISCHENWERTES ABS(IMAG(D2   LOG(C2))) > 2**18 * PI

1507    @ DOPPELT GENAU: ABSOLUTBETRAG DES ARGUMENTES IST GROESSER
        (2**50) * PI

1507    @ DOPPELT GENAU: ABSOLUTBETRAG DES ARGUMENTES IST GROESSER
        (2**50) * 180

1507    @ KOMPLEX DOPPELT GENAU: ABS(IMAG(X)) > 2**50 * PI

1507    @ KOMPLEX DOPPELT GENAU: ABS(REAL(X)) > 2**50 * PI

1507    @ KOMPLEX DOPPELT GENAU: ABS(2   REAL(X)) > 2**50 * PI

1507    @ KOMPLEX DOPPELT GENAU: ABS(2   IMAG(X)) > 2**50 * PI

1507    @ KOMPLEXE DOPPELT GENAUE POTENZIERUNG: UEBERLAUF DES
        ZWISCHENWERTES ABS(IMAG(D4 * LOG(C4))) > 2**50 * PI

1508    @ EINFACH GENAU: ABSOLUTBETRAG DES ARGUMENTES IST GROESSER
        (2**18) * PI

1508    @ EINFACH GENAU: ABSOLUTBETRAG DES ARGUMENTES IST GROESSER
        (2**18) * 180

1509    @ DOPPELT GENAU: ABSOLUTBETRAG DES ARGUMENTES IST GROESSER
        (2**50) * PI

1509    @ DOPPELT GENAU: ABSOLUTBETRAG DES ARGUMENTES IST GROESSER
        (2**50) * 180

1510    @ EINFACH GENAU: BEIDE ARGUMENTE SIND GLEICH NULL

1510    @ KOMPLEX EINFACH GENAU: ARGUMENT GLEICH 0+0I

1511    @ DOPPELT GENAU: BEIDE ARGUMENTE SIND GLEICH NULL

1511    @ KOMPLEX DOPPELT GENAU: ARGUMENT GLEICH 0+0I

1514    @ EINFACH GENAU: ABSOLUTBETRAG DES ARGUMENTES IST >= 1

1515    @ DOPPELT GENAU: ABSOLUTBETRAG DES ARGUMENTES IST >= 1

1516    @ ERWEITERT GENAU: ABSOLUTBETRAG DES ARGUMENTES IST >= 1

1517    @ ERWEITERT GENAU: ABSOLUTBETRAG DES ARGUMENTES IST GROESSER
        (2**106) * PI

1517    @ ERWEITERT GENAU: ABSOLUTBETRAG DES ARGUMENTES IST GROESSER
        (2**106) * 180

1517    @ KOMPLEX ERWEITERT GENAU: ABS(IMAG(X)) > 2**100

1517    @ KOMPLEX ERWEITERT GENAU: ABS(REAL(X)) > 2**100
```

```
1517    @ KOMPLEX ERWEITERT GENAU: ABS(2 * REAL(X)) > 2**100

1517    @ KOMPLEX ERWEITERT GENAU: ABS(2 * IMAG(X)) > 2**100

1517    @ KOMPLEXE ERWEITERT GENAUE POTENZIERUNG: UEBERLAUF DES
        ZWISCHENWERTES ABS(IMAG(D8 * LOG(C8))) > 2**100

1518    @ EINFACH GENAU: ABSOLUTBETRAG DES ARGUMENTES IST GROESSER 1

1519    @ DOPPELT GENAU: ABSOLUTBETRAG DES ARGUMENTES IST GROESSER 1

1520    @ ERWEITERT GENAU: ABSOLUTBETRAG DES ARGUMENTES IST GROESSER
        1

1521    @ ERWEITERT GENAU: BEIDE ARGUMENTE SIND GLEICH NULL

1521    @ KOMPLEX ERWEITERT GENAU: ARGUMENT GLEICH 0+0I

1522    @ ERWEITERT GENAU: ABSOLUTBETRAG DES ARGUMENTES IST GROESSER
        (2**106) * PI

1522    @ ERWEITERT GENAU: ABSOLUTBETRAG DES ARGUMENTES IST GROESSER
        (2**106) * 180

1549    GANZZAHLIGE POTENZIERUNG: BASIS GLEICH NULL, EXPONENT NICHT
        POSITIV

1550    @ EINFACH GENAUE GANZZAHLIGE POTENZIERUNG: BASIS GLEICH
        NULL, EXPONENT NICHT POSITIV

1551    @ DOPPELT GENAUE GANZZAHLIGE POTENZIERUNG: BASIS GLEICH
        NULL, EXPONENT NICHT POSITIV

1552    @ EINFACH GENAUE POTENZIERUNG: BASIS GLEICH NULL, EXPONENT
        NICHT POSITIV

1553    @ DOPPELT GENAUE POTENZIERUNG: BASIS GLEICH NULL, EXPONENT
        NICHT POSITIV

1554    @ KOMPLEXE EINFACH GENAUE GANZZAHLIGE POTENZIERUNG: BASIS
        GLEICH 0+0I UND EXPONENT NICHT POSITIV

1555    @ KOMPLEXE DOPPELT GENAUE GANZZAHLIGE POTENZIERUNG: BASIS
        GLEICH 0+0I UND EXPONENT NICHT POSITIV

1556    @ KOMPLEXE EINFACH GENAUE POTENZIERUNG: BASIS GLEICH 0+0I
        UND EXPONENT NICHT POSITIV REELL

1557    @ KOMPLEXE DOPPELT GENAUE POTENZIERUNG: BASIS GLEICH 0+0I
        UND EXPONENT NICHT POSITIV REELL

1558    @ KOMPLEX EINFACH GENAU: ARGUMENT GLEICH 1I BZW. -1I

1558    @ KOMPLEX EINFACH GENAU: ARGUMENT GLEICH 1 BZW. -1

1559    @ KOMPLEX DOPPELT GENAU: ARGUMENT GLEICH 1I BZW. -1I
```

1559 @ KOMPLEX DOPPELT GENAU: ARGUMENT GLEICH 1 BZW. -1

1560 @ ERWEITERT GENAUE GANZZAHLIGE POTENZIERUNG: BASIS GLEICH NULL, EXPONENT NICHT POSITIV

1561 @ ERWEITERT GENAUE POTENZIERUNG: BASIS GLEICH NULL, EXPONENT NICHT POSITIV

1562 @ KOMPLEXE ERWEITERT GENAUE GANZZAHLIGE POTENZIERUNG: BASIS GLEICH 0+0I UND EXPONENT NICHT POSITIV

1563 @ KOMPLEXE ERWEITERT GENAUE POTENZIERUNG: BASIS GLEICH 0+0I UND EXPONENT NICHT POSITIV REELL

1564 @ KOMPLEX ERWEITERT GENAU: ARGUMENT GLEICH 1I BZW. -1I

1564 @ KOMPLEX ERWEITERT GENAU: ARGUMENT GLEICH 1 BZW. -1

1570 ROUND(X,K): K IST NICHT POSITIV

1599 @: FEHLER IN DEN MATHEMATISCHEN ROUTINEN

1600 DSCEXT: N O T H A L T BEIM AUFRUF EINER UDS-FUMKTION. PARAMETER 'BENUTZERINFORMATION' NICHT VORHANDEN BZW. ZERSTOERT.

1610 FORPL: FEHLER IM PLI1-LAUFZEITSYSTEM - PLI1 ABSCHLUSSPROZEDUR BEI FOR1 NICHT ANMELDBAR

1611 PLFOR: FEHLER IM PLI1-LAUFZEITSYSTEM - FOR1 ABSCHLUSSPROZEDUR BEI PLI1 NICHT ANMELDBAR

1612 SPRACHTRANSFER PLI1 NACH FOR1/ASS: DIE DEKLARATION ENTHAELT UNZULAESSIGEN TYP DES RETURNS-ATTRIBUTES

1613 DAS PLI1 LAUFZEITSYSTEM LAESST DIE ANMELDUNG EINER ABSCHLUSSPROZEDUR NICHT ZU

1614 DAS FOR1 LAUFZEITSYSTEM LAESST DIE ANMELDUNG EINER ABSCHLUSSPROZEDUR NICHT ZU

3000 DIE FORMAT-SPEZIFIKATION E(W,D,S) BEDINGT ZIFFERNVERLUST

3000 DIE FELDWEITE W BEI F-FORMAT IST ZU KLEIN

3001 VERSCHACHTELUNGSTIEFE VON FORMATLISTEN GROESSER 10

3002 ZU WENIGE ODER ZU VIELE PARAMETER IN C-FORMAT ELEMENT

3006 UNGUELTIGE ZUWEISUNG AN EINEN PICTURED CHARACTER STRING

3010 ILLEGALES FORMATELEMENT BEI EIN- ODER AUSGABE

3011 ZIELTYP ILLEGAL BEI EINGABE

3012 QUELLTYP ILLEGAL BEI AUSGABE

3013	BITSTRING WURDE MIT UNGUELTIGEM RADIXFAKTOR ANGEGEBEN
3790	RUECKSPRUNG AUS ON-UNIT NACH SUBSCRIPTRANGE-CONDITION
3791	RUECKSPRUNG AUS ON-UNIT NACH STORAGE-CONDITION
3792	RUECKSPRUNG AUS ON-UNIT NACH STRINGRANGE-CONDITION
3799	RUECKSPRUNG AUS ON-UNIT NACH CONVERSION-CONDITION OHNE KORREKTUR DES FEHLERHAFTEN STRINGS
3802	@(A,N): UNZULAESSIGE DIMENSIONSANGABE, N NICHT POSITIV
3802	@(A,N): UNZULAESSIGE DIMENSIONSANGABE, N GROESSER ALS DIMENSION VON A
3804	NAME LAENGER ALS ERLAUBTER MAXIMALWERT (NAME='@')
4001	ZUWEISUNG AN NICHT ALLOKIERTE CONTROLLED VARIABLE BEI GET FILE DATA (ONFILE='@')
4001	ZUWEISUNG AN NICHT ALLOKIERTE CONTROLLED VARIABLE BEI GET STRING DATA
5000	P$ERROR WURDE OHNE ONCODE AUFGERUFEN
5010	P$CALL: X'80' FEHLT IM LETZTEN PARAMETER ODER ZU VIELE PARAMETER
5015	P$LINK: MODUL WURDE NICHT GEBUNDEN, SIEHE DLL-MELDUNG
7000	DIE DATEI @ KANN NICHT ORDNUNGSGEMAESS GESCHLOSSEN WERDEN.
8081	RECHENZEIT (CPU) IST ABGELAUFEN (SIEHE MAKRO SETIC)
8082	REALZEITGEBER IST ABGELAUFEN (SIEHE MAKRO SETIC)
8091	NICHT DECODIERBARER OPERATIONSCODE
8092	PRIVILEGIERTE OPERATION
8094	SPEICHERSCHUTZ-VERGEHEN
8095	ADRESSIERUNGS-FEHLER
8097	DATENFEHLER IM DEZIMAL-FORMAT
8098	FEHLER BEI AUSRICHTUNG AN BYTEGRENZE
8099	PROGRAMMLAUFZEIT UEBERSCHRITTEN

A.3 Implementierungsbeschränkungen

Für die Verwendung von arithmetischen Werten, für Folgen und Masken, für die Länge von Namen, für Ein-/Ausgabevorgänge und für Aufzählungen sind durch PLI1 Begrenzungen festgelegt, die nachfolgend aufgelistet sind. Beim Überschreiten dieser Grenzwerte erhält man vom Compiler und in einigen Fällen vom Laufzeitsystem eine entsprechende Fehlermeldung.

1. Grenzen für arithmetische Werte

Sprachelement	Begrenzung
Maximale Anzahl der Ziffern (einschl. der führenden Nullen) einer Festpunktkonstante)	56 für BINARY 16 für DECIMAL (es können mehr geschrieben werden, doch beeinflußen diese Ziffern nur den Skalenfaktor)
Maximale Anzahl der Ziffern im Exponenten einer Gleitpunktkonstanten	3 für BINARY 2 für DECIMAL
Maximale Anzahl der Ziffern in der Mantisse einer Gleitpunktkonstanten	112 für BINARY 33 für DECIMAL
Wertebereich einer Festpunktkonstanten x	$-2^{31} + 1 \leq x \leq 2^{31}-1$ für BINARY $-(10^{15} - 1) \leq x \leq 10^{15} - 1$ für DECIMAL
Wertebereich einer Gleitpunktkonstanten x	$2^{-260} < \lvert x \rvert < 2^{252}$ angenähert und x = 0 für BINARY $10^{-78} < \lvert x \rvert < 10^{75}$ und x = 0 für DECIMAL
Genauigkeitsbereich p eines arithmetischen Festpunktdatenelementes	$1 \leq p \leq 31$ für BINARY $1 \leq p \leq 15$ für DECIMAL
Genauigkeitsbereich p eines arithmetischen Gleitpunktdatenelementes	$1 \leq p \leq 109$ für BINARY $1 \leq p \leq 33$ für DECIMAL
Bereich der Skalierung k für ein arithmetisches Festpunktdatenelement	$-128 \leq k \leq 127$

2. Grenzen für Folgen und Masken

Sprachelement	Begrenzung
Maximallänge einer Bitfolge- oder Zeichenfolge-Konstanten	32767 Bits bzw. Zeichen. 509 Bits bzw. 510 Zeichen, wenn konvertiert werden muß. (Beide Angaben gelten nach Ausführung des Replikators)
Bereich der Länge l einer vereinbarten Zeichenfolge	1 ≤ l ≤ 32767 Zeichen
Bereich der Länge l einer vereinbarten Bitfolge	1 ≤ l ≤ 32767 Bits
Wertebereich des Replikators r für Folgen	0 ≤ r ≤ 32767
Bereich der Länge l einer alphanumerischen Maske	1 ≤ l ≤ 511 Zeichen
Bereich der Länge l einer numerischen Maske	1 ≤ l ≤ 255 Zeichen
Maximale Anzahl von Ziffern-Maskenzeichen in numerischen Masken	p ≤ 15 Festpunkt p ≤ 16 Gleitpunkt, 2 für Exponent
Bereich für Skalierung k in Masken	- 128 ≤ k ≤ 127

3. Grenzen für Namen

Sprachelement	Begrenzung
Maximale Anzahl von Zeichen in einem Bezeichner	255 (es können mehr angegeben werden, die jedoch ignoriert werden)
Maximale Anzahl von Zeichen in einem Bezeichner mit Attribut EXTERNAL	7 (bei längeren Namen werden die ersten 4 und letzten 3 Zeichen verwendet. Zeichen_wird durch $ ersetzt).
Maximale Anzahl von Zeichen in einem Bezeichner mit Attribut FILE CONSTANT	31 für eingebaute Funktion ONFILE, Rest wird abgeschnitten, 8 für LINK-Namen, Rest wird abgeschnitten. Zeichen_wird durch $ ersetzt.
Maximale Anzahl von Zeichen für Bezeichner in einem Datenstrom (bei PUT/GET DATA)	255 einschließlich maximal 64 Qualifikatoren
Maximale Anzahl von Zeichen für Namen von %INCLUDE-Texten bzw. Bibliotheken	8 (bei längeren Namen werden die ersten 5 und letzten 3 Zeichen verwendet)

4. Grenzen für Matrizen und Bereiche (AREA)

Sprachelement	Begrenzung
Maximale Anzahl der Dimensionen einer Matrix	255
Maximale Länge eines Bereichs (AREA)	16 777 191 Zeichen

5. Grenzen für Auflistungen, Verschachtelungen

Sprachelement	Begrenzung
Maximale Anzahl von %INCLUDE-Texten	255
Verschachtelungstiefe für %INCLUDE-Texte	5
Größte logische Schachtelungstiefe für Klammern aller Art (Blöcke,DO-Anweisungen, Ausdrücke, Faktorisierung bei DECLARE usw.)	nur begrenzt durch Stapelspeicher, siehe Steueranweisung STORAGE im Kapitel 8
Maximale Anzahl von Basen in der Qualifikationskette einer BASED-Variablen	128
Maximale Anzahl von Werten in einer INITIAL-Liste	1024 für AUTOMATIC
Maximale Anzahl von Bezügen links vom Zuweisungszeichen	128
Maximale Anzahl von Parametern	64

6. Grenzen für Ein-/Ausgabevorgänge

Sprachelement	Begrenzung
Maximale Anzahl von Daten-Elementen bei E/A-Listen	128
Verschachtelungstiefe für Formatlisten	10
Maximale Angabe n bei SKIP(n),LINE(n), COLUMN(n)	2^{31}
Maximale Länge des Ausdrucks bei GET STRING	32767 Zeichen
Maximale Zeichenzahl für Werte bei GET	32767 (einschließlich folgender Leerzeichen bei GET LIST)
Maximale Zeichenzahl für einen Wert bei PUT	65536 (nach Konvertierung in externe Darstellung)
Länge der Namen für PUT DATA	256 (ohne Gleichheitszeichen)
Länge der Namen für GET DATA	256 (mit Gleichheitszeichen)
Länge der Zeichenfolge bei DISPLAY	2044 Zeichen
Bereich der Satzlänge r bei satzorientierter Übertragung in Zeichen	$1 \leq r \leq 32767$ (minimal 12 für Banddateien, maximal 2048 für CONSECUTIVE auf PAM und 32763 für ISAM) Für Plattendateien ohne PAM-Key (BLKCTRL=DATA) gilt CONSECUTIVE auf SAM maximal 32752, CONSECUTIVE oder INDEXED auf ISAM maximal 32492
Bereich für Schlüssellänge k in Zeichen	$1 \leq k \leq 255$
Bereich für Schlüsselposition c	$1 \leq c \leq r - k$ und 0 in Verbindung mit KEYLOC
Zusatzregeln für Satzlänge r bei:	
RECFORM = F (außer ISAM)	r ≤ Angabe bei BLKSIZE
RECFORM = F (bei ISAM)	r ≤ Angabe bei BLKSIZE - 4
RECFORM = V (außer REGIONAL(i))	r ≤ Angabe bei BLKSIZE - 4
RECFORM = V (bei REGIONAL(3))	r ≤ Angabe bei BLKSIZE - 8
RECFORM = F/V, FCBTYPE=SAM/ISAM,BLKCTRL=DATA	r ≤ Angabe bei BLKSIZE - 16
ENVIRONMENT (INDEXED)	$r \geq c + k$ (c,k: siehe oben)

A.4 Unterschiede zu PL/I-D

Der Sprachumfang des Compilers PLI1 umfaßt weitgehend aufwärtskompatibel den Sprachumfang von PL/I-D.

Bei Programmumstellungen sollten jedoch folgende Punkte beachtet werden:

1. Unterschiede im Sprachumfang

- Bei GET STRING kann bei PLI1 nicht gleichzeitig COPY angegeben werden.
- Wird eine interne Prozedur zusätzlich durch DECLARE vereinbart, so ignoriert PLI1 die DCL-Anweisung. Von PL/I-D werden u.U. Parameteranpassungen durchgeführt.
- Unterschiedliche Regeln für die Aggregat-Zuweisung; z.B. wird bei DCL A(10,20); A = A/A (1,1); von PLI1 der Wert von A(1,1) vor der Ausdrucksberechnung in einer Hilfsvariablen fixiert.
- Die Zahl der Maskenzeichen für alphanumerische Masken (PICTURE) darf bei PLI1 höchstens 511 sein.
- Das Schlüsselwort RETURNS in der PROCEDURE- bzw. ENTRY-Anweisung darf nicht weggelassen werden (PL/1-D gestattet z.B. die Form A: PROC PTR;).
- Funktions-Prozeduren ohne Argument (insbesondere auch die BUILTIN-Funktionen DATE, TIME, usw.) müssen mit leerer Klammerangabe aufgerufen werden, falls sie nicht explizit vereinbart worden sind, z.B.

```
A = DATE();     oder     DCL DATE BUILTIN;
                         A = DATE;
```

- Die %INCLUDE-Anweisung erlaubt nur das Format "%INCLUDE Bibliothek (Element);" nicht jedoch das Format "%INCLUDE Bibliothek.Element;".

2. Unterschiede im Betriebsverhalten

- Die Steuerung von Compiler und Objektprogramm sind verschiedenartig.
- Die Implementierungseinschränkungen weichen voneinander ab.
- Die Tabulatoreinstellungen für PUT LIST sind unterschiedlich.
- Die DISPLAY-Anweisung arbeitet wahlweise auf der Dialogstation, auf der Operateurkonsole oder auf den Systemdateien SYSOUT bzw. SYSDTA.
- Eine BUFFERS-Angabe bei ENVIRONMENT ist wirkungslos.
- Die Voreinstellung für das Satzformat bei der STREAM-Ein-/Ausgabe ist bei PL/I-D F, bei PLI1 V. Falls wegen besserer Hantierbarkeit der Dateien im BS2000 von fester auf variable Satzlänge umgestellt wird, sind die Besonderheiten der STREAM-Ein-/Ausgabe mit variabler Satzlänge zu beachten. Insbesondere gilt dies für formatgesteuerte Eingabe.
- PLI1 besetzt Variable nicht vor.

3. Unterschiede an der Prozedurschnittstelle

- Von PL/I-D übersetzte Prozeduren können nicht an von PLI1 übersetzte Programme angebunden werden.
- Interne Prozeduren können nicht explizit vereinbart werden, d.h. entsprechende DCL-Anweisungen werden ignoriert. Unterschiede können entstehen, wenn durch PLI1 Konvertierungen (auf Grund der Vereinbarung der Parameter im internen Unterprogramm) erzeugt bzw. nicht erzeugt werden.
- Für PL/I-D-Programme gefertigte Assemblerprozeduren sind für einfache Fälle lauffähig. Unterschiede, die zu Fehlern führen können, bestehen in folgenden Punkten:
  - Die Parameterübergabe ist für mehr als 4 Parameter unterschiedlich, ebenso die Rückgabe von Funktionswerten bei speziellen Datentypen.
  - Die über R13 und R15 erreichbare Rückverfolgungsinformation ist anders aufgebaut, d.h. Bedingungsbehandlung usw. können nicht angestoßen werden.
  - Der Pseudoregister-Vektor (PRV) ist anders aufgebaut.
  - Die Bildungsregel für abgekürzte Bezeichner mit Attribut EXTERNAL ist unterschiedlich.
  - Die Verwaltungsinformation für Bereiche (AREA) ist anders aufgebaut und bei PLI1 länger.
  - Die Darstellungen für den Null-Zeiger unterscheiden sich.
  - Es ist generell zweckmäßig, größere Assembler-Unterprogramme auf die Standard-Assembler-Konventionen (OPTIONS (ASSEMBLER)) umzustellen (siehe Kapitel 7).

A.5 Beispiele für Sortieren

Beispiel 1

Bearbeitung einer F-Datei unter Verwendung je eines Benutzerausgangs zur Eingabe- und Ausgabesatzbearbeitung

```
/*  BEISPIEL FUER DIE ANWENDUNG DES PL1-SORT   */
/*  AUFGABE:                                   */
/*  SAETZE EINER F-DATEI SIND NACH FOLGENDEN   */
/*  KRITERIEN ZU AENDERN UND DANN ALPHABETISCH */
/*  ZU SORTIEREN:                              */
/*  1.DER INHALT EINES FELDES IST NACH VOR-    */
/*    SCHRIFT ZU AENDERN.                      */
/*  2.DOPPELT VORHANDENE SAETZE KOMMEN NUR     */
/*    EINMAL IN DIE SORTIERDATEI               */
/*  3.EIN BESTIMMTER SATZ IST ZU LOESCHEN      */
/*  4.EIN SATZ IST EINZUFUEGEN UND RICHTIG     */
/*    EINZUORDNEN.                             */
PL1SRTA:PROC OPTIONS(MAIN);
DCL RETCODE       FIXED BIN(31) INIT(0);
DCL M             FIXED BIN(15) EXTERNAL;
DCL N             FIXED BIN(15) EXTERNAL;
DCL BS2SRTD       ENTRY (CHAR(*),CHAR(*),FIXED BIN(31),ENTRY,ENTRY);
DCL PL1E21        ENTRY EXTERNAL;
DCL PL1E23        ENTRY EXTERNAL;
DCL SYSPRINT      FILE CONSTANT;
/*
  M = 0;
  N = 0;
/*
D:    CALL BS2SRTD(
' SORT FIELDS=(1,15,A,CH,23,4,N,CH),FORMAT=CH,OPT=REC,SIZE=15',
' RECORD LENGTH=(80,80,80),TYPE=F',
RETCODE,PL1E21,PL1E23);
/*
EXIT:      PUT SKIP(2) LIST('RETCODEHP = ',RETCODE);
/*
END;

PL1E21: PROC (SATZEIN,RETCODE) RETURNS(CHAR(*));
  DCL RETCODE     FIXED BIN(31) PARM;
  DCL SATZEIN     CHAR(80) PARM;
  DCL M           FIXED BIN(15) EXTERNAL;
  DCL N           FIXED BIN(15) EXTERNAL;
  DCL EINFUEGESATZ CHAR(80) INIT('EINFUEGESATZ FUER TEST VON PL1E21');
  DCL SYSPRINT    FILE CONSTANT;
/*                                                          */
    IF RETCODE = 8 THEN GOTO EXIT;
/*                                                          */
    M = M + 1;
    PUT SKIP LIST('SATZ-NR       = ',M);
```

```
/*                                                                  */
   IF SUBSTR(SATZEIN,17,2) = '99'
         THEN DO;
                SUBSTR(SATZEIN,17,2) = '77';
                RETURN(SATZEIN);
              END;
/*                                                                  */
   IF SUBSTR(SATZEIN,18,12) = 'SORTIERDATEI'
          THEN DO;
                RETCODE = 4;
                RETURN(SATZEIN);
      END;
 /*                                                                  */
  RETURN(SATZEIN);
/*                                                                  */
  EXIT: IF N = 0
          THEN DO;
                RETCODE = 12; N = 1;
                RETURN(EINFUEGESATZ);
              END;
  PUT SKIP(3) LIST('******************** DATEIENDE ************');
  N=0;M=0;
  RETURN(' ');
 END;

PL1E23: PROC (SATZEIN,RETCODE) RETURNS(CHAR(*));
  DCL RETCODE         FIXED BIN(31) PARM;
  DCL SATZEIN         CHAR(80) PARM;
  DCL N               FIXED BIN(15) EXTERNAL;
  DCL SYSPRINT        FILE CONSTANT;
/*                                                                  */
    IF RETCODE = 8 THEN GOTO EXIT;
/*                                                                  */
    N = N + 1;
    PUT SKIP LIST('SATZ-NR      = ',N);
/*                                                                  */
    IF RETCODE = 4
       THEN DO;
              PUT SKIP LIST('GELOESCHTER EINGABESATZ = ',SATZEIN);
              RETURN(SATZEIN);
            END;
/*                                                                  */
RETURN(SATZEIN);
/*                                                                  */
EXIT: PUT SKIP(3) LIST('******************** DATEIENDE ************');
      RETURN(' ');
END;
```

Beispiel 2

Bearbeitung einer V-Datei unter Verwendung je eines Benutzerausgangs zur Eingabe- und Ausgabesatzbearbeitung

```
/*  BEISPIEL FUER DIE ANWENDUNG DES PL1-SORT     */
/*  AUFGABE:                                     */
/*  SAETZE EINER V-DATEI SIND NACH FOLGENDEN     */
/*  KRITERIEN ZU AENDERN UND DANN ALPHABETISCH   */
/*  ZU SORTIEREN:                                */
/*  1.WENN EIN BESTIMMTES FELD EINEN VORGEGE-    */
/*    BENEN INHALT BESITZT, IST DIESER NACH      */
/*    VORSCHRIFT ZU AENDERN.                     */
/*  2.DOPPELT VORHANDENE SAETZE KOMMEN NUR       */
/*    EINMAL IN DIE SORTIERDATEI                 */
/*  3.ALS LETZTER SATZ DER SORTIERDATEI IST EIN  */
/*    KENNZEICHENSATZ EINZUFUEHREN.              */
PL1SORT:PROC OPTIONS(MAIN);
DCL RETCODE       FIXED BIN(31) INIT(0);
DCL M             FIXED BIN(15) EXTERNAL;
DCL N             FIXED BIN(15) EXTERNAL;
DCL BS2SRTD       ENTRY (CHAR(*),CHAR(*),FIXED BIN(31),ENTRY,ENTRY);
DCL PL1E21V       ENTRY EXTERNAL;
DCL PL1E23V       ENTRY EXTERNAL;
DCL SYSPRINT      FILE CONSTANT;
/*
  M = 0;
  N = 0;
/*
D:    CALL BS2SRTD(
' SORT FIELDS=(13,15,A,CH,23,4,N,CH),FORMAT=CH,OPT=REC,SIZE=15',
' RECORD LENGTH=(80,80,80),TYPE=F',
RETCODE,PL1E21V,PL1E23V);
/*
EXIT:      PUT SKIP(2) LIST('RETCODEHP = ',RETCODE);
/*
END;

PL1E21V: PROC (SATZEIN,RETCODE) RETURNS(CHAR(*));
  DCL RETCODE             FIXED BIN(31) PARM;
  DCL SATZEIN             CHAR(76) PARM;
    IF RETCODE = 8 THEN GOTO EXIT;
    BEGIN;
  DCL 1  VSATZ            BASED(P),
       2 SATZLNG1         FIXED BIN(15),
       2 FILLER1          CHAR(2),
       2 STRINGEIN        CHAR(1 REFER(SATZLNG1));
  DCL    P PTR;
  DCL 1  ARBEITSSATZ,
       2 SATZLNG2         FIXED BIN(15),
       2 FILLER2          CHAR(2),
       2 ARBEITSSTRING    CHAR(SATZLNG1 - 4);
  DCL  UEBERGABESATZ      CHAR(SATZLNG2) BASED(ADDR(ARBEITSSATZ));
  DCL  N                  FIXED BIN(15) EXTERNAL;
  DCL SYSPRINT            FILE CONSTANT;
/*                                                              */
    P = ADDR(SATZEIN);
    IF P = NULL() THEN GOTO EXIT;
```

```
/*                                                                    */
SATZLNG2 = SATZLNG1;FILLER2 = '  ';ARBEITSSTRING = STRINGEIN;N = N+1;
PUT SKIP LIST('SATZNR-E21 = ',N);
/*                                                                    */
    IF SUBSTR(ARBEITSSTRING,27,12) = 'MAINT.-PROG.'
       THEN DO;
             SUBSTR(ARBEITSSTRING,27,12) = 'DIENSTPROG.';
             RETURN(UEBERGABESATZ);
            END;
    END;
/*                                                                    */
RETURN(UEBERGABESATZ);
/*                                                                    */
EXIT: PUT SKIP(4) LIST('************  DATEIENDE  ***************');
      N = 0;
      RETURN('  ');
/*                                                                    */
END;

PL1E23V: PROC (SATZEIN,RETCODE) RETURNS(CHAR(*));
  DCL RETCODE                FIXED BIN(15) PARM;
  DCL SATZEIN                CHAR(76) PARM;
    IF RETCODE = 8 THEN GOTO EXIT;
    BEGIN;
  DCL 1  VSATZ               BASED(ADDR(SATZEIN)),
       2 SATZLNG1            FIXED BIN(15),
       2 FILLER1             CHAR(2),
       2 STRINGEIN           CHAR(1 REFER(SATZLNG1));
  DCL 1  VSATZH              BASED(ADDR(SATZEIN)),
       2 SATZLNGH            FIXED BIN(15),
       2 FILLERH             CHAR(2),
       2 AUSSTRING           CHAR(SATZLNG1-4);
  DCL 1  ARBEITSSATZ,
       2 SATZLNG2            FIXED BIN(15),
       2 FILLER2             CHAR(2),
       2 ARBEITSTRING        CHAR(SATZLNG1-4);
  DCL UEBERGABESATZ          CHAR(SATZLNG2) BASED(ADDR(ARBEITSSATZ));
  DCL M                      FIXED BIN(15) EXTERNAL;
  DCL N                      FIXED BIN(15) EXTERNAL;
  DCL SYSPRINT               FILE CONSTANT;
/*                                                                    */
SATZLNG2 = SATZLNG1;FILLER2 = '  ';ARBEITSSTRING = STRINGEIN;N = N+1;
PUT SKIP LIST('SATZNR-E23 = ',N);
/*                                                                    */
  IF RETCODE = 4
     THEN DO;
            PUT SKIP LIST('DOPPELTER EINGABESTZ = ',VSATZH);
            RETURN(UEBERGABESATZ);
  END;
/*                                                                    */
  RETURN(UEBERGABESATZ);
/*                                                                    */
      IF M = 0
         THEN DO;
                M = 1;
                RETCODE = 12;
                SATZLNG2 = 76;
                ARBEITSSTRING = '99999999' || '9999-DATEIENDE'||
```

```
                  '*****************************************';
                  PUT SKIP LIST(EINGABESATZ = ',
                   '99999999'   '9999-DATEIENDE'
                  '*****************************************');
                  RETURN(UEBERGABESATZ);
                END;
    END;
         PUT SKIP(4) LIST('*************   DATEIENDE ********');
         RETURN('  ');
END;
```

Beispiel 3

Bearbeitung einer V-Datei unter Verwendung je eines Benutzerausgangs zur Eingabe- und Ausgabesatzbearbeitung.

```
PL1SRTA:PROC OPTIONS(MAIN);
DCL RETCODE    FIXED BIN(31) INIT(0);
DCL M          FIXED BIN(15) EXTERNAL;
DCL N          FIXED BIN(15) EXTERNAL;
DCL BS2SRT     ENTRY OPTIONS(VARIABLE);
DCL PL1E21V    ENTRY EXTERNAL;
DCL PL1E23V    ENTRY EXTERNAL;
DCL SYSPRINT   FILE CONSTANT;
     DCL SPRUNG BIN FIXED(31);
/*                                                                  */
  M = 0;
  N = 0;
/*                                                                  */
/*           ************************************                   */
/*     SORT-AUFRUF MIT TYPE=V                                       */
/*     EINGABEDATEI IST ISAM-DATEI                                  */
/*     LAENGE IN DEN ERSTEN 4 BYTES, ISAM SCHLUESSEL 8 BYTES        */
/*     SORTIEREN AB 13.BYTE                                         */
/*           ************************************                   */
/*     BENUTZEREINGANG PL1E21V:                                     */
/*     'MAINT.-PROG.' WIRD IN 'DIENSTPROGR.'GEAENDERT               */
/*     BENUTZEREINGANG PL1E23V:                                     */
/*     EIN SATZ WIRD EINGEFUEGT                                     */
/*                                                                  */
/*     13 BIS 22 AUFSTEIGEND NACH CHARACTER SORTIEREN               */
/*     MIT PL1E21V UND PL1E23V                                      */
/*     AUSGABEDATEI IST ISAM-DATEI                                  */
/*           ************************************                   */
DV:  CALL BS2SRT(
' SORT FIELDS=(13,10,A,B)',
' RECORD LENGTH=(92,92,92),TYPE=V'
RETCODE,PL1E21V,PL1E23V);
/*                                                                   */
/*           ************************************                    */
       PUT SKIP(2) LIST('RETCODEHP = ',RETCODE);
/*                                                                   */
END;
```

Beispiel 4

Erstellen einer Statistik-Abgangsdatei geordnet nach Kundennummern

```
/* BEISPIEL FUER DIE ANWENDUNG DES                                    */
/* PL1-SORT                                                           */
/* AUFGABE:                                                           */
/* 1. SORTIEREN DER AUSGABESAETZE NACH                                */
/*    KUNDENNUMMERN.                                                  */
/* 2. AUSWAHL DER ABGANGSSAETZE AUS DER                               */
/*    BEWEGUNGSDATEI.                                                 */
/* 3. BILDEN EINER SUMMENDATEI, IN DER JEDER                          */
/*    KUNDE ENTHALTEN IST, AN DEN IRGEND EIN                          */
/*    LAGERARTIKEL GELIEFERT WURDE.                                   */
  SORTP:PROC OPTIONS(MAIN);
  DCL BS2SRT ENTRY OPTIONS(VARIABLE);
  DCL RET BIN FIXED(31) INIT(0);
  CALL BS2SRT(' SORT FIELDS=(13,7,A,CH)',
  ' INCLUDE COND=(1,1,CH,EQ,C''I'')',
  ' SUM FIELDS=((40,3,ZD),(43,8,ZD,2))',
   RET,0,0);
   END;
```

A.6 Zusatz-Information zu Informations-Meldungen

Zu den Informations-Meldungen, wie sie im Abschnitt 3.9 beschrieben sind, sind nachfolgend Texte aufgelistet, die weitere Informationen zu

1. Informations-Meldung 500
2. Informations-Meldung 503
3. Informations-Meldung 504

geben.

1. Zu Informations-Meldung 500

Die Informations-Meldung 500 besagt, daß eine Out-Line-Folge mit der Nummer n generiert wurde. In der nachfolgenden Liste ist links die Nummer n in aufsteigender Reihenfolge angegeben. Es folgt ein erläuternder Text, der die Leistung der generierten Out-Line-Folge beschreibt und danach der Name des Laufzeitmoduls, in dem diese Out-Line-Folge realisiert ist. Der Name des Moduls ist z.B. im Binder-Protokoll zu finden (siehe Kapitel 4).

Nr	Leistung	Modul
19	Normales Programmende, STOP	ITPRAHM#
102	ALLOCATE im Standard-Area	ITPSTVW#
103	FREE im Standard-Area	ITPSTVW#
104	ALLOCATE in benanntem Area	ITPSTVW#
105	FREE in benanntem Area	ITPSTVW#
108	Stackverlängerung im Block-Prolog	ITPSTVW#
110	Stackverl. für variabel lange AUTO und tempor. Variable	ITPSTVW#
114	Stackverlängerung für RETURNS mit *-Angabe	ITPSTVW#
200	Ablaufverfolgung (Trace) einer RETURN-Anweisung	ITPTHTR#
201	Ablaufverfolgung (Trace) einer GOTO-Anweisung	ITPTHTR#
202	Ablaufverfolgung (Trace): GOTO mit indiz. LABEL-Größe	ITPTHTR#
203	Ablaufverfolgung (Trace) einer CALL-Anweisung	ITPTHTR#
204	Ablaufverfolgung (Trace): Durchlauf einer Programm-Marke	ITPTHTR#
205	Testhilfe Kontrollpunkt/Haltepunkt	ITPTHBK#
226	Abprüfung/Meldung verschiedener Bedingungen/Conditions	ITPCDHD#
228	Abprüfung/Meldung der CONVERSION-Bedingung	ITPCDHD#
229	SIGNAL-Anweisung	ITPCDHD#
248	ON-Anweisung	ITPCDHD#
249	REVERT-Anweisung	ITPCDHD#
256	Zuweisung von BIT-Folgen, Ziel NONVARYING	ITPBIT##
257	Zuweisung von BIT-Folgen, Ziel VARYING	ITPBIT##
258	NOT-Operation auf BIT-Folge, Ziel NONVARYING	ITPBIT##
259	NOT-Operation auf BIT-Folge, Ziel VARYING	ITPBIT##
260	Zuweisung von BIT/CHAR-Folgen, bei (mögl.) Überlappung	ITPBIT##
261	AND-Operation mit BIT-Folgen, Ziel NONVARYING	ITPBIT##
262	AND-Operation mit BIT-Folgen, Ziel VARYING	ITPBIT##
263	OR-Operation mit BIT-Folgen, Ziel NONVARYING	ITPBIT##
264	OR-Operation mit BIT-Folgen, Ziel VARYING	ITPBIT##
265	XOR-Operation mit BIT-Folgen, Ziel NONVARYING	ITPBIT##

Nr	Leistung	Modul
266	XOR-Operation mit BIT-Folgen, Ziel VARYING	ITPBIT##
267	Vergleich von BIT-Folgen	ITPBIT##
268	Erstaufruf f. Verkettung von BIT-Folgen, Ziel NONVARYING	ITPBIT##
269	Erstaufruf für Verkettung von BIT-Folgen, Ziel VARYING	ITPBIT##
270	Folgeaufruf für Verkettung von BIT-Folgen	ITPBIT##
271	Schlussaufruf für Verkettung von BIT-Folgen, Spezialfall	ITPBIT##
272	Schlussaufruf für Verkettung von BIT-Folgen, Normalfall	ITPBIT##
273	Prüfe BIT-Folge, ob alle Bits gleich '0'B, '1'B, sonst	ITPBIT##
275	Daten-Konvertierung	ITPKONV#
276	Eingebaute Funktion BOOL, Ziel NONVARYING	ITPSBOB#
277	Eingebaute Funktion BOOL, Ziel VARYING	ITPSBOB#
278	Eingebaute Funktion INDEX für BIT-Folgen	ITPSIXB#
279	Eingebaute Funktion INDEX für CHAR-Folgen	ITPSIXC#
280	Eingebaute Funktion SEARCH	ITPSSVC#
283	Eingebaute Funktion VERIFY	ITPSSVC#
285	Ablaufverfolgung (Trace) einer ENTRY/PROC-Anweisung	ITPTHPT#
286	Eingebaute Funktion SQRT, Argument FLOAT einfach	ITPRRE##
287	Eingebaute Funktion SQRT, Argument FLOAT doppelt	ITPRRD##
288	Eingebaute Funktion SQRT, Argument FLOAT erweitert	ITPRRW##
289	Eingebaute Funktion SQRT, Argument CPLX FLOAT einfach	ITPRCE##
290	Eingebaute Funktion SQRT, Argument CPLX FLOAT doppelt	ITPRCD##
291	Eingebaute Funktion SQRT, Argument CPLX FLOAT erweitert	ITPRCW##
292	Eingebaute Funktion SIN, Argument FLOAT einfach	ITPRRE##
293	Eingebaute Funktion SIN, Argument FLOAT doppelt	ITPRRD##
294	Eingebaute Funktion SIN, Argument FLOAT erweitert	ITPRRW##
295	Eingebaute Funktion SIN, Argument CPLX FLOAT einfach	ITPRCE##
296	Eingebaute Funktion SIN, Argument CPLX FLOAT doppelt	ITPRCD##
297	Eingebaute Funktion SIN, Argument CPLX FLOAT erweitert	ITPRCW##
298	Eingebaute Funktion SIND, Argument FLOAT einfach	ITPRRE##
299	Eingebaute Funktion SIND, Argument FLOAT doppelt	ITPRRD##
300	Eingebaute Funktion SIND, Argument FLOAT erweitert	ITPRRW##
304	Eingebaute Funktion COS, Argument FLOAT einfach	ITPRRE##
305	Eingebaute Funktion COS, Argument FLOAT doppelt	ITPRRD##
306	Eingebaute Funktion COS, Argument FLOAT erweitert	ITPRRW##
307	Eingebaute Funktion COS, Argument CPLX FLOAT einfach	ITPRCE##
308	Eingebaute Funktion COS, Argument CPLX FLOAT doppelt	ITPRCD##
309	Eingebaute Funktion COS, Argument CPLX FLOAT erweitert	ITPRCW##
310	Eingebaute Funktion COSD, Argument FLOAT einfach	ITPRRE##
311	Eingebaute Funktion COSD, Argument FLOAT doppelt	ITPRRD##
312	Eingebaute Funktion COSD, Argument FLOAT erweitert	ITPRRW##
316	Eingebaute Funktion TAN, Argument FLOAT einfach	ITPRRE##
317	Eingebaute Funktion TAN, Argument FLOAT doppelt	ITPRRD##
318	Eingebaute Funktion TAN, Argument FLOAT erweitert	ITPRRW##
319	Eingebaute Funktion TAN, Argument CPLX FLOAT einfach	ITPRCE##
320	Eingebaute Funktion TAN, Argument CPLX FLOAT doppelt	ITPRCD##
321	Eingebaute Funktion TAN, Argument CPLX FLOAT erweitert	ITPRCW##
322	Eingebaute Funktion TAND, Argument FLOAT einfach	ITPRRE##
323	Eingebaute Funktion TAND, Argument FLOAT doppelt	ITPRRD##
324	Eingebaute Funktion TAND, Argument FLOAT erweitert	ITPRRW##
328	Eingebaute Funktion ASIN, Argument FLOAT einfach	ITPRRE##
329	Eingebaute Funktion ASIN, Argument FLOAT doppelt	ITPRRD##
330	Eingebaute Funktion ASIN, Argument FLOAT erweitert	ITPRRW##
334	Eingebaute Funktion ASIND, Argument FLOAT einfach	ITPRRE##
335	Eingebaute Funktion ASIND, Argument FLOAT doppelt	ITPRRD##

Nr	Leistung	Modul
336	Eingebaute Funktion ASIND, Argument FLOAT erweitert	ITPRRW##
340	Eingebaute Funktion ACOS, Argument FLOAT einfach	ITPRRE##
341	Eingebaute Funktion ACOS, Argument FLOAT doppelt	ITPRRD##
342	Eingebaute Funktion ACOS, Argument FLOAT erweitert	ITPRRW##
346	Eingebaute Funktion ACOSD, Argument FLOAT einfach	ITPRRE##
347	Eingebaute Funktion ACOSD, Argument FLOAT doppelt	ITPRRD##
348	Eingebaute Funktion ACOSD, Argument FLOAT erweitert	ITPRRW##
352	Eingebaute Funktion ATAN, Argument FLOAT einfach	ITPRRE##
353	Eingebaute Funktion ATAN, Argument FLOAT doppelt	ITPRRD##
354	Eingebaute Funktion ATAN, Argument FLOAT erweitert	ITPRRW##
355	Eingebaute Funktion ATAN, Argument CPLX FLOAT einfach	ITPRCE##
356	Eingebaute Funktiom ATAN, Argument CPLX FLOAT doppelt	ITPRCD##
357	Eingebaute Funktion ATAN, Argument CPLX FLOAT erweitert	ITPRCW##
358	Eingebaute Funktion ATAND, Argument FLOAT einfach	ITPRRE##
359	Eingebaute Funktion ATAND, Argument FLOAT doppelt	ITPRRD##
360	Eingebaute Funktion ATAND, Argument FLOAT erweitert	ITPRRW##
364	Eingebaute Funktion LOG2, Argument FLOAT einfach	ITPRRE##
365	Eingebaute Funktion LOG2, Argument FLOAT doppelt	ITPRRD##
366	Eingebaute Funktion LOG2, Argument FLOAT erweitert	ITPRRW##
370	Eingebaute Funktion LOG, Argument FLOAT einfach	ITPRRE##
371	Eingebaute Funktion LOG, Argument FLOAT doppelt	ITPRRD##
372	Eingebaute Funktion LOG, Argument FLOAT erweitert	ITPRRW##
373	Eingebaute Funktion LOG, Argument CPLX FLOAT einfach	ITPRCE##
374	Eingebaute Funktion LOG, Argument CPLX FLOAT doppelt	ITPRCD##
375	Eingebaute Funktion LOG, Argument CPLC FLOAT erweitert	ITPRCW##
376	Eingebaute Funktion LOG10, Argument FLOAT einfach	ITPRRE##
377	Eingebaute Funktion LOG10, Argument FLOAT doppelt	ITPRRD##
378	Eingebaute Funktion LOG10, Argument FLOAT erweitert	ITPRRW##
382	Eingebaute Funktion EXP, Argument FLOAT einfach	ITPRRE##
383	Eingebaute Funktion EXP, Argument FLOAT doppelt	ITPRRD##
384	Eingebaute Funktion EXP, Argument FLOAT erweitert	ITPRRW##
385	Eingebaute Funktion EXP, Argument CPLX FLOAT einfach	ITPRCE##
386	Eingebaute Funktion EXP, Argument CPLX FLOAT doppelt	ITPRCD##
387	Eingebaute Funktion EXP, Argument CPLX FLOAT erweitert	ITPRCW##
388	Eingebaute Funktion ATANH, Argument FLOAT einfach	ITPRRE##
389	Eingebaute Funktion ATANH, Argument FLOAT doppelt	ITPRRD##
390	Eingebaute Funktion ATANH, Argument FLOAT erweitert	ITPRRW##
391	Eingebaute Funktion ATANH, Argument CPLX FLOAT einfach	ITPRCE##
392	Eingebsute Funktion ATANH, Argument CPLX FLOAT doppelt	ITPRCD##
393	Eingebaute Funktion ATANH, Argument CPLX FLOAT erweitert	ITPRCW##
394	Eingebaute Funktion COSH, Argument FLOAT einfach	ITPRRE##
395	Eingebaute Funktion COSH, Argument FLOAT doppelt	ITPRRD##
396	Eingebaute Funktion COSH, Argument FLOAT erweitert	ITPRRW##
397	Eingebaute Funktion COSH, Argument CPLX FLOAT einfach	ITPRCE##
398	Eingebaute Funktion COSH, Argument CPLX FLOAT doppelt	ITPRCD##
399	Eingebaute Funktion COSH, Argument CPLX FLOAT erweitert	ITPRCW##
400	Eingebaute Funktion ERF, Argument FLOAT einfach	ITPRRE##
401	Eingebaute Funktion ERF, Argument FLOAT doppelt	ITPRRD##
402	Eingebaute Funktion ERF, Argument FLOAT erweitert	ITPRRW##
406	Eingebaute Funktion ERFC, Argument FLOAT einfach	ITPRRE##
407	Eingebaute Funktion ERFC, Argument FLOAT doppelt	ITPRRD##
408	Eingebaute Funktion ERFC, Argument FLOAT erweitert	ITPRRW##
412	Eingebaute Funktion SINH, Argument FLOAT einfach	ITPRRE##
413	Eingebaute Funktion SINH, Argument FLOAT doppelt	ITPRRD##

Nr	Leistung	Modul
414	Eingebaute Funktion SINH, Argument FLOAT erweitert	ITPRRW##
415	Eingebaute Funktion SINH, Argument CPLX FLOAT einfach	ITPRCE##
416	Eingebaute Funktion SINH, Argument CPLX FLOAT doppelt	ITPRCD##
417	Eingebaute Funktion SINH, Argument CPLX FLOAT erweitert	ITPRCW##
418	Eingebaute Funktion TANH, Argument FLOAT einfach	ITPRRE##
419	Eingebaute Funktion TANH, Argument FLOAT doppelt	ITPRRD##
420	Eingebaute Funktion TANH, Argument FLOAT erweitert	ITPRRW##
421	Eingebaute Funktion TANH, Argument CPLX FLOAT einfach	ITPRCE##
423	Eingebaute Funktion TANH, Argument CPLX FLOAT erweitert	ITPRCW##
424	Eingebaute Funktion ATAN2, Argument FLOAT einfach	ITPRRE##
425	Eingebaute Funktion ATAN2, Argument FLOAT doppelt	ITPRRD##
426	Eingebaute Funktion ATAN2, Argument FLOAT erweitert	ITPRRW##
430	Eingebaute Funktion ATAND2, Argument FLOAT einfach	ITPRRE##
431	Eingebaute Funktion ATAND2, Argument FLOAT doppelt	ITPRRD##
432	Eingebaute Funktion ATAND2, Argument FLOAT erweitert	ITPRRW##
436	Potenzierung Ganzzahl kurz ** Ganzzahl kurz	ITPRND##
437	Potenzierung Ganzzahl kurz ** Ganzzahl lang	ITPRND##
438	Potenzierung Ganzzahl lang ** Ganzzahl kurz	ITPRND##
439	Potenzierung Ganzzahl lang ** Ganzzahl lang	ITPRND##
440	Potenzierung FLOAT einfach ** Ganzzahl kurz	ITPRRE##
441	Potenzierung FLOAT doppelt ** Ganzzahl kurz	ITPRRD##
442	Potenzierung FLOAT erweitert ** Ganzzahl kurz	ITPRRW##
443	Potenzierung CPLX FLOAT einfach ** Ganzzahl kurz	ITPRCE##
444	Potenzierung CPLX FLOAT doppelt ** Ganzzahl kurz	ITPRCD##
445	Potenzierung CPLX FLOAT erweitert ** Ganzzahl kurz	ITPRCW##
446	Potenzierung FLOAT einfach ** Ganzzahl lang	ITPRRE##
447	Potenzierung FLOAT doppelt ** Ganzzahl lang	ITPRRD##
448	Potenzierung FLOAT erweitert ** Ganzzahl lang	ITPRRW##
449	Potenzierung CPLX FLOAT einfach ** Ganzzahl lang	ITPRCE##
450	Potenzierung CPLX FLOAT doppelt ** Ganzzahl lang	ITPRCD##
451	Potenzierung CPLX FLOAT erweitert ** Ganzzahl lang	ITPRCW##
452	Potenzierung FLOAT einfach ** FLOAT einfach	ITPRRE##
453	Potenzierung FLOAT doppelt ** FLOAT doppelt	ITPRRD##
454	Potenzierung FLOAT erweitert ** FLOAT erweitert	ITPRRW##
455	Potenzierung CPLX FLOAT einfach ** CPLX FLOAT einfach	ITPRCE##
456	Potenzierung CPLX FLOAT doppelt ** CPLX FLOAT doppelt	ITPRCD##
457	Potenzierung CPLX FLOAT erweitert ** CPLX FLOAT erweit.	ITPRCW##
458	Division von Operanden vom Typ FLOAT erweitert	ITPRRW##
459	Eingebaute Funktion MOD, Argument FLOAT erweitert	ITPRRW##
460	Eingeb. Funktion COPY für BIT-Folgen, Ziel NONVARYING	ITPSCRB#
461	Eingeb. Funktion COPY für BIT-Folgen, Ziel VARYING	ITPSCRB#
462	Eingeb. Funktion REVERSE f. BIT-Folgen, Ziel NONVARYING	ITPSCRB#
463	Eingeb. Funktion REVERSE f. BIT-Folgen, Ziel VARYING	ITPSCRB#
464	Eingeb. Funktion COPY für CHAR-Folgen, Ziel NONVARYING	ITPSCRC#
465	Eingeb. Funktion COPY für CHAR-Folgen, Ziel VARYING	ITPSCRC#
466	Eingeb. Funktion REVERSE f. CHAR-Folgen, Ziel NONVARYING	ITPSCRC#
467	Eingeb. Funktion REVERSE f. CHAR-Folgen, Ziel VARYING	ITPSCRC#
468	Eingeb. Funktion TRANSLATE, 3 Argumente, Ziel NONVARYING	ITPSTRC#
469	Eingeb. Funktion TRANSLATE, 3 Argumente, Ziel VARYING	ITPSTRC#
470	Eingeb. Funktion TRANSLATE, 2 Argumente, Ziel NONVARYING	ITPSTRC#
471	Eingeb. Funktion TRANSLATE, 2 Argumente, Ziel VARYING	ITPSTRC#
485	Abprüfung/Meldung der SUBSCRIPTRANGE-Bedingung	ITPCOND#
486	Abprüfung/Meldung der STRINGRANGE-Bedingung	ITPCOND#

Nr	Leistung	Modul
487	Eingebaute Funktion ROUND (ISO-Fall), Argument DEC FLOAT	ITPRND##
488	Eingebaute Funktion ROUND (ISO-Fall), Argument BIN FLOAT	ITPRND##
489	Eingeb. Fkt. ROUND (ISO-Fall), Argument CPLX BIN FLOAT	ITPRND##
490	Eingeb. Fkt. ROUND (ISO-Fall), Argument CPLX DEC FLOAT	ITPRND##
491	Eingeb. Fkt. ROUND (NOISO-Fall), Argument CPLX BIN FLOAT	ITPRND##
492	Eingeb. Fkt. ROUND (NOISO-Fall), Argument CPLX DEC FLOAT	ITPRND##
493	Eingebaute Funktion ABS, Argument CPLX BIN FLOAT einfach	ITPRCE##
494	Eingebaute Funktion ABS, Argument CPLX BIN FLOAT doppelt	ITPRCD##
495	Eingebaute Funktion ABS, Argument CPLX BIN FLOAT erweit.	ITPRCW##
496	Eingebaute Funktion ABS, Argument CPLX BIN FIXED kurz	ITPACB##
497	Eingebaute Funktion ABS, Argument CPLX BIN FIXED lang	ITPACB##
498	Vergleich von Operanden vom Typ CPLX BIN FIXED	ITPACB##
499	Addition von Operanden vom Typ CPLX BIN FIXED	ITPACB##
500	Subtraktion von Operanden vom Typ CPLX BIN FIXED	ITPACB##
501	Multiplikation von Operanden vom Typ CPLX BIN FIXED	ITPACB##
502	Division von Operanden vom Typ CPLX BIN FIXED	ITPACB##
503	Eingebaute Funktion ROUND, Argument CPLX BIN FIXED	ITPACB##
504	Eingebaute Funktion ABS, Argument CPLX DEC FIXED	ITPACD##
505	Vergleich von Operanden vom Typ CPLX DEC FIXED	ITPACD##
506	Addition von Operanden vom Typ CPLX DEC FIXED	ITPACD##
507	Subtraktion von Operanden vom Typ CPLX DEC FIXED	ITPACD##
508	Multiplikation von Operanden vom Typ CPLX DEC FIXED	ITPACD##
509	Division von Operanden vom Typ CPLX DEC FIXED	ITPACD##
510	Eingebaute Funktion ROUND, Argument CPLX DEC FIXED	ITPACD##
511	Vergleich von Operanden vom Typ CPLX FLOAT	ITPACF##
512	Addition von Operanden vom Typ CPLX FLOAT	ITPACF##
513	Subtraktion von Operanden vom Typ CPLX FLOAT	ITPACF##
514	Multiplikation von Operanden vom Typ CPLX FLOAT	ITPACF##
515	Division von Operanden vom Typ CPLX FLOAT	ITPACF##
516	Eingeb. Funktion REPEAT f. BIT-Folgen, Ziel NONVARYING	ITPSCRB#
517	Eingeb. Funktion REPEAT f. BIT-Folgen, Ziel VARYING	ITPSCRB#
518	Eingeb. Funktion REPEAT f. CHAR-Folgen, Ziel NONVARYING	ITPSCRC#
519	Eingeb. Funktion REPEAT f. CHAR-Folgen, Ziel VARYING	ITPSCRC#
520	Aufruf eines Eingangs vom Typ OPTIONS(FORTRAN)	ITPLXFV#
521	Aufruf eines Eingangs vom Typ OPTIONS(FORTRAN INTER)	ITPLXFV#
522	Aufruf eines Eingangs vom Typ OPTIONS(COBOL)	ITPLXFV#
770	Endeaufruf fuer GET-Anweisung	ITPGET##
771	Endeaufruf fuer PUT-Anweisung	ITPPUT##
772	PUT DATA Anweisung mit Liste, Folgeaufr. f. 1 Datenelem.	ITPPVD##
773	GET LIST Anweisung, Folgeaufruf für 1 Datenelement	ITPGVL##
774	GET EDIT Anweisung, Folgeaufruf für 1 Datenelement	ITPGVE##
775	PUT LIST Anweisung, Folgeaufruf für 1 Datenelement	ITPPVL##
776	PUT EDIT Anweisung, Folgeaufruf für 1 Datenelement	ITPPVE##
777	Satz-orientierte Ein/Ausgabe; Transport-Anweisung	ITPIORC#
778	OPEN Anweisung	ITPOPEN#
779	CLOSE Anweisung	ITPOPEN#
780	GET DATA Anweisung oder Anfangsaufruf für GET-Anweisung	ITPGET##
781	PUT DATA ohne Liste oder Anfangsaufruf für PUT-Anweisung	ITPPUT##
831	Ablaufverfolgung f. ENTRY/PROC in optimierter Prozedur	ITPTHPT#

2. Zu Informations-Meldung 503

Die Informations-Meldung 503 besagt, daß eine Out-Line-Folge mit dem Namen e generiert wurde. In der nachfolgenden Liste ist links der Name e in alphabetischer Reihenfolge angegeben. Es folgt ein erläuternder Text, der die Leistung der generierten Out-Line-Folge beschreibt, und danach der Name des Laufzeitmoduls, in dem diese Out-Line-Folge realisiert ist. Der Name des Moduls ist z.B. im Binder-Protokoll zu finden (siehe Kapitel 4).

Eingang	Leistung	Modul
AFTERB##	Eingebaute Funktion AFTER, Argument BIT-Folge	ITPSAFB#
AFTERC##	Eingebaute Funktion AFTER, Argument CHAR-Folge	ITPSAFC#
ALL#####	Eingebaute Funktion ALL	ITPBALL#
ANY#####	Eingebaute Funktion ANY	ITPBANY#
BEFOREB#	Eingebaute Funktion BEFORE, Argument BIT-Folge	ITPSAFB#
BEFOREC#	Eingebsute Funktion BEFORE, Argument CHAR-Folge	ITPSAFC#
BOUND###	Eingebaute Funktionen DIM, HBOUND, LBOUND	ITPBBND#
COUNT###	Eingebaute Funktion COUNT	ITPOPEN#
DATAFLD#	Eingebaute Funktion DATAFIELD	ITPCDHD#
DATE####	Eingebaute Funktion DATE	ITPBDAT#
DECATB##	Eingebaute Funktion DECAT, Argument BIT-Folge	ITPSAFB#
DECATC##	Eingebaute Funktion DECAT, Argument CHAR-Folge	ITPSAFC#
DISPPLY#	DISPLAY-Anweisung ohne REPLY	ITPIODI#
DISPRLY#	DISPLAY-Anweisung mit REPLY	ITPIODI#
EVERY###	Eingebaute Funktion EVERY	ITPBEVR#
GETLENO#	Eingebaute Funktion LINENO	ITPOPEN#
GETPANO#	Eingebaute Funktion PAGENO	ITPOPEN#
ON$CHAR#	Eingebaute Funktion ONCHAR	ITPCDHD#
ON$CNT##	Eingebaute Funktion ONCOUNT	ITPCDHD#
ON$CODE#	Eingebaute Funktion ONCODE	ITPCDHD#
ON$FILE#	Eingebaute Funktion ONFILE	ITPCDHD#
ON$FLD##	Eingebaute Funktion ONFIELD	ITPCDHD#
ON$INTR#	Eingebaute Funktion ONINTR	ITPCDHD#
ON$KEY##	Eingebaute Funktion ONKEY	ITPCDHD#
ON$LOC##	Eingebaute Funktion ONLOC	ITPCDHD#
ON$SRCE#	Eingebaute Funktion ONSOURCE	ITPCDHD#
POLY####	Eingebaute Funktion POLY	ITPBPLY#
PV$CHAR#	Pseudovariable ONCHAR	ITPCDHD#
PV$SRCE#	Pseudovariable ONSOURCE	ITPCDHD#
SAMEKEY#	Eingebaute Funktion SAMEKEY	ITPBSKY#
SETPENO#	Pseudovariable PAGENO	ITPOPEN#
SOME####	Eingebaute Funktion SOME	ITPBSOM#
ST$NMAS#	Zuweisung benannter AREA’s	ITPSTVW#
TIME####	Eingebaute Funktion TIME	ITPBDAT#
VALID###	Eingebaute Funktion VALID	ITPKONV#

3. Zu Informations-Meldung 504

Die Informations-Meldung 504 besagt, daß eine Out-Line-Folge für eine Konvertierung vom Typ t generiert wurde. In der nachfolgenden Liste ist links der Typ t in aufsteigender Reihenfolge angegeben. Es folgt ein erläuternder Text, der die Leistung der generierten Out-Line-Folge beschreibt.

In der Liste werden folgende Abkürzungen verwendet:

⟶	Konvertierrichtung
BIT	BIT-Folgen ALIGNED oder UNALIGNED
CHARACTER	CHARACTER-Folgen ALIGNED oder UNALIGNED
FLOAT	sofern nicht näher spezifiziert: DEC FLOAT oder BIN FLOAT, einfach oder doppelt genau
PIC(...)	PICTURE vom Typ alphanumerisch, numerisch bzw. mit Angabe des numerischen Typs
numerisch	arithmetische Datentypen DECIMAL bzw. BINARY, FLOAT bzw. FIXED

Die Konvertierung von/nach COMPLEX wird, soweit nicht anders vermerkt, im allgemeinen für Real- und Imaginärteil getrennt ausgeführt, es erfolgen also zwei Aufrufe.

Die Konvertierungen von/nach FLOAT/PIC (float) mit erweiterter Genauigkeit (4-fach) einerseits bzw. mit einfacher und doppelter Genauigkeit andererseits erfolgen über getrennte Schlüssel.

Typ	Konvertierleistung
1	BIT → BIT
2	BIT → CHARACTER
3	BIT → BINARY FIXED
4	BIT → DECIMAL FIXED
5	BIT → FLOAT
6	BIT → PICTURE(decimal fixed)
7	BIT → PICTURE(decimal float)
8	BIT → PICTURE(alphanumerisch)
9	CHARACTER oder PICTURE(alphanumerisch) → CHARACTER oder PICTURE(alphanumerisch)
10	CHARACTER oder PICTURE(alphanumerisch) → BIT
11	CHARACTER oder PICTURE(alphanumerisch) → numerisch oder PICTURE(numerisch), reell oder complex
12	CHARACTER oder PICTURE(alphanumerisch) → PICTURE(alphanumerisch)
13	numerisch oder PICTURE(numerisch), reell oder complex → CHARACTER oder PICTURE(alphanumerisch)
17	BINARY FIXED → DECIMAL FIXED
18	BINARY FIXED → FLOAT
19	BINARY FIXED → BIT
20	BINARY FIXED → PICTURE(decimal fixed)
21	BINARY FIXED → PICTURE(decimal float)
24	FLOAT → BINARY FIXED
25	FLOAT → DECIMAL FIXED
26	FLOAT → PICTURE(decimal fixed)
27	FLOAT → PICTURE(decimal float)
28	FLOAT → BIT
29	FLOAT → CHARACTER gemaess E-Format
31	DECIMAL FIXED → DECIMAL FIXED
32	DECIMAL FIXED → BINARY FIXED
33	DECIMAL FIXED → FLOAT
34	DECIMAL FIXED → BIT
35	DECIMAL FIXED → PICTURE(decimal fixed)
36	DECIMAL FIXED → PICTURE(decimal float)
48	PICTURE(numerisch) → PICTURE(decimal fixed)
49	PICTURE(numerisch) → BINARY FIXED
50	PICTURE(numerisch) → DECIMAL FIXED
51	PICTURE(numerisch) → FLOAT
52	PICTURE(numerisch) → PICTURE(decimal float)
53	PICTURE(numerisch) → BIT
67	numerisch, BIT, CHARACTER oder PICTURE → FLOAT erweitert genau
69	FLOAT erweitert genau → CHARACTER gemaess E-Format
70	FLOAT erweitert genau → numerisch, BIT, CHARACTER oder PICTURE
73	FLOAT → FLOAT
74	BINARY FIXED → BINARY FIXED
75	BINARY FIXED → BIT
76	FLOAT → BIT
77	FLOAT → CHARACTER gemaess E-Format

A.7 Laufzeit-Moduln

Nachstehend sind in alphabetischer Reihenfolge die Namen der Moduln des statischen Laufzeitsystems aufgelistet. Dahinter ist erläutert, welche Leistung der Modul erbringt. Die Moduln werden immer dann eingebunden, wenn die beschriebene Leistung im Benutzer-Programm verwendet wird.

In den mit "***" eingeleiteten Zeilen sind jene Moduln des Laufzeit-Systems angegeben, die auf den davor beschriebenen Modul Bezug nehmen, deren Einbinden in ein Programm also auch zwingend das Einbinden dieses Moduls nach sich zieht.

Modul	Leistung / aufgerufen von Modul (***)
ITP#AOS#	Anfangsbehandlung für PLI1-Objektprogramme bei Verwendung des statischen Laufzeit-Systems
ITPACB##	Out-line Folgen für Addition, Subtraktion, Multiplikation, Division, Vergleich, Absolutbetrag und Rundung von CPLX BIN FIXED Operanden
ITPACD##	Out-line Folgen für Addition, Subtraktion, Multiplikation, Division, Vergleich, Absolutbetrag und Rundung von CPLX DEC FIXED Operanden
ITPACF##	Out-line Folgen für Addition, Subtraktion, Multiplikation, Division und Vergleich von CPLX FLOAT
ITPBALL#	Eingebaute Funktion ALL
ITPBANY#	Eingebaute Funktion ANY
ITPBBND#	Eingebaute Funktionen DIM, HBOUND, LBOUND
ITPBDAT#	Eingebaute Funktionen DATE, TIME
ITPBEVR#	Eingebaute Funktion EVERY
ITPBIT##	Verschiedene Operationen mit BIT-Folgen (AND, OR, XOR, NOT, Zuweisung, Verkettung, Vergleich, Test auf alle Bits = '0'B oder '1'B), Zuweisung von CHAR-Folgen bei mögl. Überlappung
***	ITPBALL#, ITPBANY#, ITPBEVR#, ITPBSOM#, ITP#AOS#
ITPBITN#	Auffüllen eines Speicherbereiches mit binären Nullen
***	ITPSBOB#, ITPSCRB#
ITPBPLY#	Eingebaute Funktion POLY
ITPBSKY#	Eingebaute Funktion SAMEKEY
ITPBSOM#	Eingebaute Funktion SOME
ITPCDHD#	Bedingungs(Condition)-Behandlung, Meldung verschiedener Bedingungen, ON-, REVERT- und SIGNAL-Anweisung, eingebaute Funktionen/Pseudovariable DATAFIELD, ONCHAR, ONCODE, ONCOUNT, ONFIELD, ONFILE, ONINTR, ONKEY, ONLOC, ONSOURCE, Service-Routine ERROUT
***	ITP#AOS#, ITPGVD##, ITPGVE##, ITPOPEN#, ITPPVD##, ITPPVE##
ITPCOND#	Meldung der STRINGRANGE- und SUBSCRIPTRANGE-Bedingung
***	ITP#AOS#
ITPDASI#	Kern für eingebaute Funktion ASIN/ACOS, doppelt genau
***	ITPRRD##
ITPDATH#	Kern für eingebaute Funktion ATANH, doppelt genau
***	ITPYATA#, ITPRRD##
ITPDAT2#	Kern für eingebaute Funktion ATAN/ATAN2, doppelt genau
***	ITPYATA#, ITPYLOG#, ITPRRD##
ITPDERF#	Kern für eingebaute Funktion ERF/ERFC, doppelt genau
***	ITPRRD##

```
Modul     Leistung / aufgerufen von Modul (***)

ITPDEXP#  Kern für eingebaute Funktion EXP, doppelt genau
     ***  ITPDERF#, ITPDPWR#, ITPDSIH#, ITPDTAH#, ITPYEXP#,
     ***  ITPYSIN#, ITPRRD##
ITPDLOG#  Kern für eingebaute Funktion LOG/LOG10/LOG2, doppel genau
     ***  ITPDATH#, ITPDPWR#, ITPYLOG#, ITPYPWC#, ITPRRD##
ITPDPWI#  Kern für Potenzierung FLOAT doppelt genau    Ganzzahl
     ***  ITPRRD##
ITPDPWR#  Kern für Potenzierung FLOAT doppelt ** FLOAT doppelt genau
     ***  ITPRRD##
ITPDSIH#  Kern für eingebaute Funktion SINH/COSH, doppelt genau
     ***  ITPYTAN#, ITPRRD##
ITPDSIN#  Kern für eingebaute Funktion SIN/COS, doppelt genau
     ***  ITPYEXP#, ITPYSIN#, ITPYTAN#, ITPRRD##
ITPDSQR#  Kern für eingebaute Funktion SQRT, doppelt genau
     ***  ITPDASI#, ITPYABS#, ITPYSQR#, ITPRRD##
ITPDTAH#  Kern für eingebaute Funktion TANH, doppelt genau
     ***  ITPRRD##
ITPDTAN#  Kern für eingebaute Funktion TAN/COTAN, doppelt genau
     ***  ITPRRD##
ITPEASI#  Kern für eingebaute Funktion ASIN/ACOS, einfach genau
     ***  ITPRRE##
ITPEATH#  Kern für eingebaute Funktion ATANH, einfach genau
     ***  ITPXATA#, ITPRRE##
ITPEAT2#  Kern für eingebaute Funktion ATAN/ATAN2, einfach genau
     ***  ITPXATA#, ITPXLOG#, ITPRRE##
ITPEERF#  Kern für eingebaute Funktion ERF/ERFC, einfach genau
     ***  ITPRRE##
ITPEEXP#  Kern für eingebaute Funktion EXP, einfach genau
     ***  ITPEERF#, ITPEPWR#, ITPESIH#, ITPETAH#, ITPXEXP#, ITPXSIN#,
     ***  ITPRRE##
ITPELOG#  Kern für eingebaute Funktion LOG/LOG10/LOG2, einfach genau
     ***  ITPEATH#, ITPEPWR#, ITPXLOG#, ITPXPWC#, ITPRRE##
ITPEPWI#  Kern für Potenzierung FLOAT einfach genau    Ganzzahl
     ***  ITPRRE##
ITPEPWR#  Kern für Potenzierung FLOAT einfach ** FLOAT einfach genau
     ***  ITPRRE##
ITPESIH#  Kern für eingebaute Funktion SINH/COSH, einfach genau
     ***  ITPXTAN#, ITPRRE##
ITPESIN#  Kern für eingebaute Funktion SIN/COS, einfach genau
     ***  ITPXEXP#, ITPXSIN#, ITPXTAN#, ITPRRE##
ITPESQR#  Kern für eingebaute Funktion SQRT, einfach genau
     ***  ITPEASI#, ITPXABS#, ITPXSQR#, ITPRRE##
ITPETAH#  Kern für eingebaute Funktion TANH, einfach genau
     ***  ITPRRE##
ITPETAN#  Kern für eingebaute Funktion TAN/COTAN, einfach genau
     ***  ITPRRE##
ITPFL###  Besorgt die Formatbeschreibung für eine Variable bei der Ein/
          Ausgabe im EDIT-Modus
     ***  ITPGVE##,ITPPVE##
ITPGDT##  Besorgt für die GET DATA Anweisung für eine Variable die
          zugehörige Variablen-Adresse und -Beschreibung
     ***  ITPGVD##
ITPGDTX#  Besorgt für die GET DATA Anweisung für eine Variable die
          zugehörige Variablen-Adresse und -Beschreibung im XS-Fall
     ***  ITPGVDX##
```

Modul	Leistung / aufgerufen von Modul (***)
ITPGET##	Anfangs- und Endebehandlung für alle GET-Anweisungen, (Folge) Aufrufe zum Lesen eines Elements (Eingabedatum/Name) bei allen GET-Anweisungen, Steuerung des Lesevorganges (Zeile oder Spalte einstellen, Zeichen überspringen), Ausführung der GET DATA Anweisung mit oder ohne Liste
***	ITPFL###, ITPGVD##, ITPGVE##, ITPGVL##, ITPOPCL#
ITPGVD##	Kern-Routine zur Ausführung der GET DATA Anweisung mit oder ohne Liste
***	ITPGET##
ITPGVDX#	Kern-Routine zur Ausführung der GET DATA Anweisung mit oder ohne Liste (im XS-Fall)
***	ITPGET##
ITPGVE##	Eingabe eines Datenelements einer GET EDIT Anweisung (Folgeaufruf)
ITPGVL##	Eingabe eines Datenelements einer GET LIST Anweisung (Folgeaufruf); auch für ein Element bei GET DATA verwendet
***	ITPGVD##
ITPHXDC#	Service-Routine HEXDEC
***	ITPTHRD#
ITPIODI#	DISPLAY-Anweisung mit/ohne REPLY
ITPIORC#	Transport-Anweisung für Satz-orientierte Ein/Ausgabe; Ausführung von Transport- und Positionierraufträgen auch für Strom-orientierte Ein/Ausgabe; Formatierung von REGIONAL-Dateien
***	ITPGET##, ITPIOSY#, ITPOPCL#, ITPOPEN#, ITPPUT##
ITPIORX#	Transport-Anweisung für Satzorientierte Ein-/Ausgabe usw., entsprechend ITPIORC# im XS-Fall
***	ITPGET##, ITPIOSY#, ITPOPCX#, ITPOPEN#, ITPPUT##
ITPIOSY#	Ausführung von Ein/Ausgabe-Transportaufträgen von/auf BS2000-Systemdateien (SYSDTA, SYSCMD, SYSOUT, SYSLST, Datenstation, Operateurkonsole); Auswerten der SYSFILE-Laufzeitoption und Setzen der Tabulatoren
***	ITPIODI#, ITPGET##, ITPOPEN#, ITPPUT##, ITPIORC#
ITPIPWI#	Kern für Potenzierung ganzzahliger Operanden
***	ITPRND##
ITPKONV#	Konvertierpaket für alle out-line Konvertierungen für Objekt-Routinen, Ein/Ausgabe und mathematische Bibliothek; eingebaute Funktion VALID
***	ITPGVE##, ITPGVL##, ITPPVE##, ITPPVL##, ITPRND##
ITPLXFC#	Common-Bereich für Übergangsroutine ITPLXFN#
ITPLXFN#	Uebergangsroutine zur Umstellung der Programmumgebung bei Aufruf von PLI1-Unterprogrammen aus COBOL- oder FORTRAN-Hauptprogramm
***	ITPLXFC#
ITPLXFV#	Uebergangsroutine zur Umstellung der Programmumgebung bei Aufrufen von COBOL- und FORTRAN-Eingängen mit/ohne Angabe INTER
ITPOPCL#	Kern-Routine zum Eröffnen und Schließen von Dateien
***	ITPOPEN#
ITPOPCX#	Kern-Routine zum Eröffen und Schließen von Dateien im XS-Fall
***	ITPOPEN#
ITPOPEN#	OPEN- und CLOSE-Anweisung; implizites Eröffnen und Schließen von Dateien; eingebaute Funktionen/Pseudovariable COUNT, LINENO, PAGENO
***	ITPGET##, ITPPUT##, ITPIORC#

Modul	Leistung / aufgerufen von Modul (***)
ITPOPRD#	Routine zum Lesen und Analysieren der Objekt-Steueroptionen (RUNOPT)
***	ITP#AOS#
ITPOPWR#	Routine zum Protokollieren der Objekt-Steueroptionen (RUNOPT)
***	ITP#AOS#
ITPPDBA#	Steuerung der Ausgabe für PUT DATA ohne Liste
***	ITPPUT##
ITPPDBX#	Steuerung der Ausgabe für PUT DATA ohne Liste im XS-Fall
***	ITPPUT##
ITPPDSD#	Unter-Routine für Ein/Ausgabe im DATA-Modus
***	ITPGDT##, ITPPVD##
ITPPDSX#	Unter-Routine für Ein-/Ausgabe im DATA-Modus (XS-Fall)
***	ITPGDTX#, ITPPVDX#
ITPPDVA#	Kern-Routine für Ausgabe mittels PUT DATA ohne Liste
***	ITPPDBA#
ITPPDVX#	Kern-Routine für Ausgabe mittels PUT DATA ohne Liste (XS-Fall)
***	ITPPDBX#
ITPPUT##	Anfangs-und Endebehandlung für alle PUT-Anweisungen, (Folge) Aufrufe zur Ausgabe eines Elements (Ausgabedatum/Name) bei allen PUT-Anweisungen, Steuerung des Ausgabevorgangs (Zeile, Seite oder Spalte einstellen), Ausführung der COPY-Option einer GET-Anweisung, Ausführung von PUT DATA ohne Liste
***	ITPFL###, ITPGET##, ITPIOSY#, ITPOPCL#, ITPOPEN#, ITPPVD##,
***	ITPPVE##, ITPPVL##
ITPPVD##	Folgeaufruf für PUT DATA mit Liste; Kern-Routine zur Ausgabe einer Variablen im DATA-Modus
***	ITPPDVA#
ITPPVDX#	Folgeaufruf für PUT DATA mit Liste; Kern-Routine zur Ausgabe einer Variablen im DATA-Modus (XS-Fall)
***	ITPPDVX#
ITPPVE##	Ausgabe eines Datenelements einer PUT EDIT Anweisung (Folgeaufruf)
ITPPVL##	Ausgabe eines Datenelements einer GET LIST Anweisung (Folgeaufruf); auch für ein Element bei PUT DATA verwendet
***	ITPPVD##
ITPRAHM#	Standardrahmen für PLI1-Objektprogramme; Anfangs-, Ende- und Unterbrechungsbehandlung; Fehlerbehandlung für Dienst-routinen, Programmende/STOP-Anweisung; Service-Routine RUNTIME
***	ITP#AOS#, ITPTHBK#, ITPOPEN#, ITPTHRD#, ITPTHTR#
ITPRCD##	Eingebaute Funktionen ABS, ATAN, ATANH, COS, COSH, EXP, LOG, SIN, SINH, SQRT, TAN, TANH für Argumente vom Typ CPLX FLOAT doppelt genau, Potenzierung mit Basis CPLX FLOAT doppelt genau
ITPRCE##	Eingebaute Funktionen ABS, ATAN, ATANH, COS, COSH, EXP, LOG, SIN, SINH, SQRT, TAN, TANH für Argumente vom Typ CPLX FLOAT einfach genau, Potenzierung mit Basis CPLX FLOAT einfach genau
ITPRCW##	Eingebaute Funktionen ABS, ATAN, ATANH, COS, COSH, EXP, LOG, SIN, SINH, SQRT, TAN, TANH für Argumente vom Typ CPLX FLOAT erweitert genau, Potenzierung mit Basis CPLX FLOAT erweitert
ITPRND##	Potenzierung Ganzzahl ** Ganzzahl, eingebaute Funktion ROUND bzw. Rundung für reelle und komplexe FLOAT Operanden

Modul	Leistung / aufgerufen von Modul (***)
ITPRRD##	Eingebaute Funktionen ACOS, ACOSD, ASIN, ASIND, ATAN, ATAN2, ATAND, ATAND2, ATANH, COS, COSD, COSH, ERF, ERFC, EXP, LOG, LOG10, LOG2, SIN, SIND, SINH, SQRT, TAN, TAND, TANH für Argumente vom Typ FLOAT doppelt genau; Potenzierung mit Basis FLOAT doppelt genau
***	ITPBPLY#
ITPRRE##	Eingebaute Funktionen ACOS, ACOSD, ASIN, ASIND, ATAN, ATAN2, ATAND, ATAND2, ATANH, COS, COSD, COSH, ERF, ERFC, EXP, LOG, LOG10, LOG2, SIN, SIND, SINH, SQRT, TAN, TAND, TANH für Argumente vom Typ FLOAT einfach genau; Potenzierung mit Basis FLOAT einfach genau
***	ITPBPLY#
ITPRRW##	Eingebaute Funktionen ACOS, ACOSD, ASIN, ASIND, ATAN, ATAN2, ATAND, ATAND2, ATANH, COS, COSD, COSH, ERF, ERFC, EXP, LOG, LOG10, LOG2, MOD, SIN, SIND, SINH, SQRT, TAN, TAND, TANH für Argumente vom Typ FLOAT erweitert genau; Potenzierung mit Basis FLOAT erweitert genau; Division von Operanden vom Typ FLOAT erweitert genau
***	ITPKONV#, ITPBPLY#
ITPRTAD#	Unter-Routine für Ein/Ausgabe im DATA-Modus
***	ITPGDT##, ITPPDVA#, ITPRTOF#, ITPRTPT#
ITPRTAX#	Unter Routine für Ein-/Ausgabe im DATA-Modus (XS-Fall)
***	ITPGDTX#, ITPPDVX#, ITPRTOX#, ITPRTPX#
ITPRTOF#	Unter-Routine für Ein/Ausgabe im DATA-Modus
***	ITPRTPT#
ITPRTOX#	Unter-Routine für Ein-/Ausgabe im DATA-Modus (XS-Fall)
***	ITPRTPX#
ITPRTPT#	Unter-Routine für Ein/Ausgabe im DATA-Modus
***	ITPRTAD#
ITPRTPX#	Unter-Routine für Ein-/Ausgabe im DATA-Modus (XS-Fall)
***	ITPRTAX#
ITPRTSX#	Unter-Routine für Ein-/Ausgabe im DATA-Modus (XS-Fall)
***	ITPGDTX#
ITPRTSY#	Unter-Routine für Ein/Ausgabe im DATA-Modus
***	ITPGDT##
ITPRTVA#	Unter-Routine für Ein/Ausgabe im DATA-Modus
***	ITPGDT##, ITPPDVA#, ITPRTAD#, ITPPDSD#
ITPRTVX#	Unter-Routine für Ein-/Ausgabe im DATA-Modus (XS-Fall)
***	ITPGDTX#, ITPPDVX#, ITPRTAX#, ITPRTSX#
ITPSAFB#	Eingebaute Funktionen AFTER, BEFORE, DECAT für BIT-Folgen
ITPSAFC#	Eingebaute Funktionen AFTER, BEFORE, DECAT für CHAR-Folgen
ITPSBOB#	Eingebaute Funktion BOOL
ITPSCRB#	Eingebaute Funktionen COPY, REPEAT, REVERSE für BIT-Ketten
ITPSCRC#	Eingebaute Funktionen COPY, REPEAT, REVERSE für CHAR-Ketten
ITPSIXB#	Eingebaute Funktion INDEX für BIT-Ketten
***	ITPSAFB#
ITPSIXC#	Eingebaute Funktion INDEX für CHAR-Ketten
ITPSSVC#	Eingebaute Funktionen SEARCH, VERIFY
ITPSTRC#	EIngebaute Funktion TRANSLATE

Modul	Leistung / aufgerufen von Modul (***)
ITPSTVW#	Speicherverwaltung für Stack (für AUTOMATIC und temporäre Variable), Standardarea (Systemspeicher; für BASED Variable, die nicht in einem benannten AREA angelegt werden, CONTROLLED Variable, Puffer für die Ein/Ausgabe etc.) und benannte AREA's (für BASED Variable); ALLOCATE- und FREE-Anweisung, Zuweisung benannter AREA's
***	ITP#AOS# ITPTHAI# Administration für die Testhilfe AID
***	ITP#AOS#
ITPTHBK#	Behandlung der Testhilfe Kontrollpunkt/Haltepunkt, Analyse und Auswertung der beim Haltepunkt eingegebenen Steueroptionen
ITPTHBO#	Einlesen der am Kontrollpunkt/Haltepunkt eingegebenen Steueroptionen
***	ITPTHBK#
ITPTHLF#	Service-Routinen (Testhilfen) ADUMP, RDUMP, SDUMP
***	ITPTHBK#
ITPTHPT#	Ablaufverfolgung (Trace) für ENTRY/PROC-Anweisungen (PROCTRACE); Service-Routinen PTON, PTOFF
ITPTHRD#	Testhilfen: Rückverfolgung (SNAP), Binärdumps; Feststellen eines Quellbezugs (INCLUDE-File-, Zeilen-, Anweisungs-Nummer); Service-Routine (Testhilfe) SNAP
***	ITP#AOS#, ITPTHBK#, ITPTHLF#
ITPTHTR#	Ablaufverfolgung (Trace) für CALL-, GOTO- und RETURN-Anweisung (CALLTRACE, GOTOTRACE, RETURNTRACE) und Durchlauf einer Programm-Marke (LABELTRACE)
ITPTLIN#	Aufbereitung von Zeilenreferenzen
***	ITPTHRD#, ITPTHTR#, ITPTXST#
ITPTXBS#	Basis-Textausgabe; Ausgabe von Texten auf SYSOUT und/oder SYSLST, Basis-Dienste für Diagnose-Ausgaben
***	ITP#AOS#, ITPTHBK#, ITPTHRD#, ITPTHTR#
ITPTXST#	Standard-Textausgabe; Ausgabe der Texte aus den Meldungs-dateien auf SYSOUT und/oder SYSLST, Texte evtl. mit Vorspann versehen bzw. ergänzen, Auswertung der Angaben bzgl. Sprache und Ausgabegerät aus der RUNOPT-Steuerung
***	ITP#AOS#, ITPOPEN#, ITPTHRD#
ITPWASI#	Kern für eingebaute Funktion ASIN/ACOS, erweitert genau
***	ITPRRW##
ITPWATH#	Kern für eingebaute Funktion ATANH, erweitert genau
***	ITPZATA#, ITPRRW##
ITPWAT2#	Kern für eingebaute Funktion ATAN/ATAN2, erweitert genau
***	ITPZATA#, ITPZLOG#, ITPRRW##
ITPWDIV#	Kern für Division von Operanden vom Typ FLOAT erweitert genau
***	ITPWASI#, ITPWATH#, ITPWAT2#, ITPWERF#, ITPWEXP#, ITPWMOD#,
***	ITPWPWI#, ITPWSIH#, ITPWTAN#, ITPZATA#, KZLMT###, ITPZSQR#,
***	ITPZTAN#, ITPRRW##
ITPWERF#	Kern für eingebaute Funktion ERF/ERFC, erweitert genau
***	ITPRRW##
ITPWEXP#	Kern für eingebaute Funktion EXP/LOG/LOG10/LOG2 und Potenzierung FLOAT erweitert genau ** FLOAT erweitert genau
***	ITPWATH#, ITPWERF#, ITPWSIH#, ITPWTAH#, ITPZEXP#, ITPZLOG#,
***	ITPZPWC#, ITPZSIN#, ITPRRW##
ITPWMOD#	Kern für eingebaute Funktion MOD, erweitert genau
***	ITPRRW##
ITPWPWI#	Kern für Potenzierung FLOAT erweitert genau Ganzzahl
***	ITPRRW##

Modul	Leistung / aufgerufen von Modul (***)
ITPWSIH#	Kern für eingebaute Funktion SINH/COSH, erweitert genau
***	ITPZTAN#, ITPRRW##
ITPWSIN#	Kern für eingebaute Funktion SIN/COS, erweitert genau
***	ITPZEXP#, ITPZSIN#, ITPZTAN#, ITPRRW##
ITPWSQR#	Kern für eingebaute Funktion SQRT, erweitert genau
***	ITPWASI#, ITPZABS#, ITPZSQR#, ITPRRW##
ITPWTAH#	Kern für eingebaute Funktion TANH, erweitert genau
***	ITPRRW##
ITPWTAN#	Kern für eingebaute Funktion TAN/COTAN, erweitert genau
***	ITPRRW##
ITPXABS#	Kern für eingebaute Funktion ABS, cplx einfach genau
***	ITPRCE##
ITPXATA#	Kern für eingebaute Funktion ATAN/ATANH, cplx einfach genau
***	ITPRCE##
ITPXDIV#	Kern für Division von Operanden vom Typ CPLX FLOAT einfach genau
***	ITPACF##
ITPXEXP#	Kern für eingebaute Funktion EXP, cplx einfach genau
***	ITPXPWC#, ITPRCE##
ITPXLOG#	Kern für eingebaute Funktion LOG, clpx einfach genau
***	ITPXPWC#, ITPRCE##
ITPXMLT#	Kern für Multiplikation von Operanden vom Typ CPLX FLOAT einfach genau
***	ITPXPWI#, ITPACF##
ITPXPWC#	Kern für Potenzierung CPLX FLOAT einfach genau ** CPLX FLOAT einfach genau
***	ITPRCE##
ITPXPWI#	Kern für Potenzierung CPLX FLOAT einfach genau ** Ganzzahl
***	ITPRCE##
ITPXSIN#	Kern für eingebaute Funktion SIN/COS/SINH/COSH, cplx einfach genau
***	ITPRCE##
ITPXSQR#	Kern für eingebaute Funktion SQRT, cplx einfach genau
***	ITPRCE##
ITPXTAN#	Kern für eingebaute Funktion TAN/TANH, cplx einfach genau
***	ITPRCE##
ITPYABS#	Kern für eingebaute Funktion ABS, cplx doppelt genau
***	ITPRCD##
ITPYATA#	Kern für eingebaute Funktion ATAN/ATANH, cplx doppelt genau
***	ITPRCD##
ITPYDIV#	Kern für Division von Operanden vom Typ CPLX FLOAT doppelt genau
***	ITPACF##
ITPYEXP#	Kern für eingebaute Funktion EXP, cplx doppelt genau
***	ITPYPWC#, ITPRCD##
ITPYLOG#	Kern für eingebaute Funktion LOG, cplx doppelt genau
***	ITPYPWC#, ITPRCD##
ITPYMLT#	Kern für Multiplikation von Operanden vom Typ CPLX FLOAT doppelt genau
***	ITPYPWI#, ITPACF##

Modul	Leistung / aufgerufen von Modul (***)
ITPYPWC#	Kern für Potenzierung CPLX FLOAT doppelt genau ** CPLX FLOAT doppelt genau
***	ITPRCD##
ITPYPWI#	Kern für Potenzierung CPLX FLOAT doppelt genau ** Ganzzahl
***	ITPRCD##
ITPYSIN#	Kern für eingebaute Funktion SIN/COS/SINH/COSH, cplx doppelt genau
***	ITPRCD##
ITPYSQR#	Kern für eingebaute Funktion SQRT, cplx doppelt genau
***	ITPRCD##
ITPYTAN#	Kern für eingebaute Funktion TAN/TANH, cplx doppelt genau
***	ITPRCD##
ITPZABS#	Kern für eingebaute Funktion ABS, cplx erweitert genau
***	ITPRCW##
ITPZATA#	Kern für eingebaute Funktion ATAN/ATANH, cplx erweitert genau
***	ITPRCW##
ITPZEXP#	Kern für eingebaute Funktion EXP, cplx erweitert genau
***	ITPZPWC#, ITPRCW##
ITPZLOG#	kern für eingebaute Funktion LOG, cplx erweitert genau
***	ITPZPWC#, ITPRCW##
ITPZMLT#	Kern für Multiplikation und Division von Operanden vom Typ CPLX FLOAT erweitert genau
***	ITPZPWC#, ITPZPWI#, ITPACF##
ITPZPWC#	Kern für Potenzierung CPLX FLOAT erweitert genau ** CPLX FLOAT erweitert genau
***	ITPRCW##
ITPZPWI#	Kern für Potenzierung CPLX FLOAT erweitert genau ** Ganzzahl
***	ITPRCW##
ITPZSIN#	Kern für eingebaute Funktion SIN/COS/SINH/COSH, cplx erweitert genau
***	ITPRCW##
ITPZSQR#	Kern für eingebaute Funktion SQRT, cplx erweitert genau
***	ITPRCW##
ITPZTAN#	Kern für eingebaute Funktion TAN/TANH, cplx erweitert genau
***	ITPRCW##
ITP2SRT#	Zwischenroutine zum Aufruf des BS2000-SORT aus PLI1-Programmen und zum wahlweisen Anschluß von Benutzerroutinen für Satz-Ein/Ausgabe bei der Sortierung; Service-Routinen BS2SRTA, BS2SRTB, BS2SRTC, BS2SRTD

A.8 Meldungen des PLI1-Laufzeit-Systems

Diese Meldungen werden ausgegeben, wenn die PL/I-Fehlerbehandlung (Condition-Handling) nicht aufgerufen werden kann, weil z.B. die notwendigen Initialisierungen noch nicht durchgeführt sind (z.B. während der Programmanfangsbehandlung) oder weil die für das Condition-Handling benötigten Routinen sich selbst in Fehlersituationen befinden (z.B. Textausgaberoutinen, Programmunterbrechungsbehandlung). Abhängig von der Fehlerart wird das Programm beendet oder eine diesbezügliche Dialoganfrage gestellt (***CONTINUE (Y/N)).

Format:
***Ennn ERROR IN mmmmmmm:Meldung

wobei

- 'nnn' die Nummer der Meldung ist
- 'mmmmmmm' den verursachenden Modul bezeichnet
- 'Meldung' der Text der Meldung ist.

Eine Liste der Meldungsnummern und -texte mit zusätzlicher Information wird in der folgenden Tabelle gegeben.

Nummer	Meldung	Beschreibung/Ursache	Reaktion
001	INTERRUPT OR WROUT ERROR IN RUNTIME SYSTEM	Fehler in der PLI1-Rahmenroutine	Systemdienst verständigen
002, 003	PROGRAM CANNOT BE INITIATED	Anmeldung Programm-unterbrechungsbehand-lung nicht möglich, z.B. Lademodul zu groß	Speicherengpaß beheben; Lade-modul verkleinern
004	STACK POINTER (R13) DESTROYED-NO COND' HANDLING	Fehlerbehandlung nicht möglich; R13 zerstört; möglicherweise Benutzerfehler	ggf. Programm korrigieren
007	IRREGULAR INTERRUPT	Undefinierte Unter-brechung oder Fehler im Rückverfolger	Systemdienst verständigen
011, 012, 013	STORAGE SHORTAGE	Speichermangel bei Stackinitialisierung oder nach STORAGE-Condition	Angabe bei *RUNOPT STORAGE = STACK vergrößern
014, 015, 016, 017	ILLEGAL VALUE IN STORAGE OPTION IGNORED	Parameter der STORAGE-Option ungültig	STORAGE-Angabe berichtigen

Nummer	Meldung	Beschreibung/Ursache	Reaktion
021	MULTIPLE INITIALISATION	Versuch, PLI1-Umgebung mehrfach zu starten	
022	ILLEGAL PARAMETER	Parameter der STORAGE-Option widersprüchlich	STORAGE-Angabe berichtigen
023	STORAGE SHORTAGE	Speichermangel bei Stackinitialisierung oder nach STORAGE-Condition	Angabe bei *RUNOPT STORAGE = STACK vergrößern
024 025 026 027	ILLEGAL VALUE IN STORAGE OPTION IGNORED	Parameter der STORAGE-Option ungültig	STORAGE-Angabe berichtigen
029	FREE CHAIN DESTROYED IN STANDARD AREA	Lückenkette des Standard-Bereiches zerstört, u.U. Benutzerfehler	u.U. Programm korrigieren
030	LINESIZE TOO LARGE, ADJUSTED TO BUFFERSIZE	Zeilenlänge zu groß bei Dialoggerät	LINESIZE-Dateiparameter berichtigen
031	CONTROL DATA DESTROYED	Verwaltungsinformation zerstört oder Parameter unzulässig	Systemdienst verständigen
032	IRREGULAR INTERRUPT	Undefinierte Unterbrechung oder Fehler im Rückverfolger	Systemdienst verständigen
033	CONTROL DATA DESTROYED	Verwaltungsinformation zerstört oder Parameter unzulässig	Systemdienst verständigen
034	ERROR ON SYSOUT/SYSLST	Fehler in der Textausgabe	Systemdienst verständigen
035	STORAGE SHORTAGE	Speichermangel bei Textausgaberoutinen	Angabe bei *RUNOPT STORAGE = AREA vergrößern
036	PLI1 TEXTFILE NOT AVAILABLE (MISSING, LOCKED, ETC.)	Textdatei fehlerhaft zugeordnet	Textdatei verfügbar machen, z.B. durch /FILE..., LINK = TEXTLINK
037	ERROR ON SYSOUT/SYSLST	Fehler in der Textausgabe	Systemdienst verständigen

Nummer	Meldung	Beschreibung/Ursache	Reaktion
038	FILE NOT USABLE FOR SAVLST	Unverträgliche SAVLST-Datei-Parameter	voreingestellte SAVLST-Datei-Parameter verwenden
039	I/O ERROR on SAVLST	E/A-Fehler bei SAVLST-Datei	Systemdienst verständigen
050 051 052 053	INTERNAL ST$STVW# ERROR	Fehler in der PLI1-Speicherverwaltung	Systemdienst verständigen
055	REQM ERROR (MEMORY SATURATION)	REQM-Makro-Fehler	Systemdienst verständigen
056	RELM ERROR	RELM-Makro-Fehler	Systemdienst verständigen
070	UNRECOVERABLE SYNTAX ERROR IN OPTIONS	Fehler bei der OPTIONS-Eingabe	*RUNOPT-Eingabe korrigieren
071	INPUT TOO LARGE FOR OPTION-Routine	Zu lange OPTIONS-Eingabe	Zeile mit *RUNOPT verkürzen
080	ILLEGAL OPERATION FOR OVERLAPPING BIT STRINGS	Der Modul BIT#OP## des PLI1-Laufzeitsystems wurde mit einem falschen Eingang aufgerufen	Systemdienst verständigen
101	I/O UNABLE TO INITIALIZE TERMINATION	Abschlußprozedur bei OPEN nicht anmeldbar	Systemdienst verständigen
102 103 104 105 106 107 108 109	I/O-System ERROR	DVS-Fehler bei OPEN/CLOSE	Systemdienst verständigen

Nummer	Meldung	Beschreibung/Ursache	Reaktion
110	ATTEMPT TO START A FOR1-PROGRAM OUTSIDE THE MAIN-PROCEDURE	Der Start eines Programms, das FOR1-Prozeduren enthält, erfolgt im PLI1-Laufzeitsystem.	Binden in richtiger Reihenfolge oder mit START-Angabe
111	INCONSISTENT RUNTIME/ I/O-SYSTEM	Vorgebundene Teile des PLI1-Laufzeitsystems (P$RTS###/P$IOS###) passen nicht zum Verbindungsmodul (P$ANFOD#)	passendes Laufzeitsystem besorgen; evtl. neu binden
112	ATTEMPT TO START A PLI1-PROGRAM OUTSIDE THE MAIN-PROCEDURE	Der Start eines Programms, das von fremdsprachigen Prozeduren aufgerufen wird, erfolgt im PLI1-Laufzeitsystem.	Binden in richtiger Reihenfolge

A.9 Beispiele für PLI1-ASSEMBLER-Makros

1. PL/I-Hauptprogramm

```
PLIMAC :PROC  OPTIONS(MAIN);

        PUT SKIP LIST
            ('TEST OF PLI1.MACLIB');
        PUT SKIP LIST
            ('CALL OF ASSEMBLER-SUBROUTINES, THAT ARE CALLING');
        PUT SKIP LIST
            ('PL/I-SUBROUTINES ETC.');
        PUT SKIP;

        DCL  (UP1,UP2,UP3,UP4,UP5)
             ENTRY OPTIONS(ASM);
        DCL  (UP6,UP7) RETURNS (PTR);
        DCL  (UP8,UP9) OPTIONS(PLI1);
        DCL   A CHAR(40) VAR INIT
                ('1234567890  3 CHAR.S ARE NOT PRINTED'),
              B CHAR(40) BASED (P),
              C CHAR(40) VAR INIT('CCCCCCCCCCCC'),
              D BIT (256) INIT ('C0C1C2C3C4C5C6C7C8C9CACBCCCDCECF'B4 ||
                                '808182838485868788898A8B8C8D8E8F'B4),
              P   PTR INIT(ADDR(X)),
              PT6 PTR,
              PT7 PTR,
              X CHAR(100) INIT((100)'X'),
              ERROUT OPTIONS(LIB);

        ON ERROR BEGIN;
              CALL ERROUT;
              GOTO ERR;
        END;

        CALL UP1 (A,B,C,D);
        CALL UP2 (A);
        CALL UP3;
        CALL UP4;
 ERR:   CALL UP5 (PLIUP3,P1,P2,P3,P4,P5,P6);
        PT6 = UP6 ();
        PT7 = UP7 ();
        PUT SKIP EDIT
            ('R12=',UNSPEC(PT6)) (A,B4);
        PUT SKIP EDIT
            ('====',UNSPEC(ADDR(PLI1GLOBAL(1))) | 'FFFFF000'B4)
            (A,B4);

        PUT SKIP EDIT
            ('R13=',UNSPEC(PT7)) (A,B4);
        CALL UP8;
 FIN:
        CALL UP9;
       PUT SKIP LIST('NORMAL END');

       STOP;
```

```
PLIUP1:ENTRY(A1,A2,A3,A4);
       DCL  (A1,
             A2,
             A3 ) CHAR(40) VAR,
             A4   CHAR(32);

       PUT SKIP DATA (A1);
       PUT SKIP DATA (A2);
       PUT SKIP DATA (A3,A4);
       PUT SKIP;
       RETURN;

PLIUP2:ENTRY(B1,B2,B3,B4,B5,B6);
       DCL  (B1,B3,B5(P1,P3,P5)STATIC) CHAR(15) VAR,
            (B2,P2 STATIC) BIN FIXED(31),
            (B4,P4 STATIC) DEC FIXED(5),
            (B6,P6 STATIC) PTR,
             B7 CHAR(15) VAR BASED (B6);
       P1=B1;P2=B2;P3=B3;P4=B4;P5=B5;P6=B6;

       PUT SKIP DATA (B1,B2);
       PUT SKIP DATA (B3,B4);
       PUT SKIP DATA (B5,B7);

       RETURN;

PLIUP3:ENTRY(C1,C2,C3,C4,C5,C6);
       DCL (C1,C3,C5) CHAR(15) VAR,
            C2 BIN FIXED(31),
            C4 DEC FIXED(5),
            C6 PTR,
            C7 CHAR(15) VAR BASED (C6);

       PUT SKIP DATA (C1,C2);
       PUT SKIP DATA (C3,C4);
       PUT SKIP DATA (C5,C7);
       PUT SKIP;

       RETURN;

PLIUP4:ENTRY;

       PUT SKIP LIST ('PLIUP4 ENTERED');
       PUT SKIP;

       RETURN;

PLIUP5:ENTRY;

       PUT SKIP LIST ('PLIUP5 ENTERED');
       PUT SKIP;

       RETURN;

     END PLIMAC;
```

```
UP1         START
            PRINT   NOGEN
SSS         P$STACK
PRV         P$PRV
            P$ENTRY LENGTH=200
*         PARAMS FOR THE FOLLOWING CALL VIA R1 FROM PL/1
            P$CALL  PLIUP1,PARNUM=4
            P$RETURN
UP2         P$ENTRY
            LA      R1,0(,R1)
            WRLST   (1)
            P$RETURN
UP3         P$ENTRY LENGTH=0
            P$CALL  PLIUP2,(P1,P2,P3,P4,P5,P6)
            P$RETURN
P1          DC      H'15',CL40'FIRST PARAMETER'
P2          DC      F'99999'
P3          DC      H'12',CL15'2ND PARAMETER'
P4          DC      P'99999'
P5          DC      H'00',CL15'3RD PARAMETER'
P6          DC      A(P11)
P11         DC      H'17',CL15'WRONG LENGTH STRING'
P12         DC      C'!!!!!'
UP4         P$ENTRY
            P$ERROR   ONCODE =0003,MSG=01
            P$RETURN
UP5         P$ENTRY
            P$CALL    PARNUM=7
            P$RETURN  STXIT=YES
UP6         P$ENTRY
            LR        R1,R12
            P$RETURN  R1RETN=YES,STXIT=YES
UP7         P$ENTRY
            LR        R1,R13
            P$RETURN  R1RETN=YES,STXIT=NO
UP8         P$ENTRY
            P$CALL    PLIUP4
            P$RETURN  STXIT=YES
UP9         P$ENTRY
            BALR      10,0
            USING     *,R10
            LA        R1,=V(PLIUP5)
            P$CALL
            P$RETURN
            END
```

Das PL/I-Programm PLIMAC ruft die Eingänge des ASSEMBLER-Programmes UP1 auf, die über das Makro P$CALL Eingänge des Hauptprogramms rufen.
Dabei entstehen Ausgaben wie folgt:

```
TEST  OF PLI1.MACLIB
CALL OF ASSEMBLER-SUBROUTINES, THAT ARE CALLING
PL/1-SUBROUTINES ETC.

A1='1234567890  3 CHAR.S ARE NOT PRINTED';
A2='XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX';
A3='CCCCCCCCCCCC'   A4=' ABCDEFGHI       abcdefghi ÄÖÜ';
4567890  3 CHAR.S ARE NOT PRINT

B1='FIRST PARAMETER'        B2=           99999;
B3='2ND PARAMETER'  B4=  99999;
B5=''     B7='WRONG LENGTH ST';
*****ERROR-CONDITION, ONCODE=0003 AT OFFSET '027312' IN PROCEDURE WITH ENTRY
UP4
NO WHEN CLAUSE SATISFIED AND NO OTHERWISE CLAUSE SPECIFIED

C1='FIRST PARAMETER'          C2=          99999;
C3='2ND PARAMETER'   C4=   99999;
C5=''     C7='WRONG LENGTH ST';

R12=00028000
====FFFFFDEC
R13=7F02E580
PLIUP4 ENTERED

PLIUP5 ENTERED

NORMAL END

END OF PROGRAM PLIMAC  , RTS 3.2A-600, TIME USED:    0.09 SEC
```

2. ASSEMBLER-Hauptprogramm

```
HP2       START
          PRINT     GEN
          BALR      R11,0
          USING     *,R11
          P$ENVIRM  BASEREG=R11
          P$LINK    PLIUP4,LIBNAM=BIS.OBJ
          ST        R1,PAREND
          LA        R1,PAREND
          P$CALL
          P$LINK    PLIUP2,LIBNAM=BIS.OBJ
          ST        R1,PAREND
          LA        R1,PAREND
          P$CALL    ,(P1,P2,P3,P4,P5,P6)
          P$STOP
P1        DC        H'15',CL40'FIRST PARAMETER'
P2        DC        F'99999'
P3        DC        H'12',CL15'2ND PARAMETER'
P4        DC        P'99999'
P5        DC        H'00',CL15'3RD PARAMETER'
P6        DC        A(P11)
P11       DC        H'17',CL15'WRONG LENGTH STRING'
P12       DC        C'!!!!!'
          END
```

```
 PLIUP2 : PROC (B1,B2,B3,B4,B5,B6) ;
          DCL  (B1,B3,B5) CHAR(15) VAR,
                B2 BIN FIXED(31),
                B4 DEC FIXED(5),
                B6 PTR,
                B7 CHAR(15) VAR BASED (B6);

          PUT SKIP DATA (B1,B2,B3);
          PUT SKIP DATA (B4,B5,B7);
          PUT SKIP;
 END;
```

```
 PLIUP4 : PROC ;
          PUT SKIP LIST
              ('PLIUP4 ENTERED');
          PUT SKIP;

          RETURN;
 END;
```

Das ASSEMBLER-Hauptprogramm HP2 ruft die PL/I-Programme PLIUP2 und PLIUP4, die dynamisch durch das Makro P$LINK angebunden wurden. Dabei entstehen Ausgaben wie folgt:

```
 PLIUP4 ENTERED

 B1='FIRST PARAMETER'         B2=         99999   B3='2ND PARAMETE';
 B4=   99999         B5=''    B7='WRONG LENGTH ST';
```

15 Literatur

[1] **PLI1** (BS2000)
PL/I-Compiler
Beschreibung

Zielgruppe
PL/I-Anwender im BS2000
Inhalt
Grundprinzip der Sprache und Programmbausteine, Datentypen und Attribute, Speicherzuordnung, Anweisungen, Programmaufbau, Ein-/Ausgaben, Ausdrükke, Datenkonvertierung, Formate und Maskenzeichen, Bedingungen, Eingebaute Funktionen, Pseudovariablen, Vorübersetzer und außergewöhnliche Programmereignisse.
Die Beschreibung ist ein Nachschlagewerk, aber auch als Lehrbuch für den PL/I-Anfänger geeignet.

[2] BS2000
Kommandosprache des Organisationsprogramms
Beschreibung

Zielgruppe
BS2000-Anwender (nicht privilegiert)
Inhalt
Alle BS2000-Systemkommandos in lexikalischer Reihenfolge mit Hinweisen und Beispielen.
Folgende Liefereinheiten sind berücksichtigt:
BS2000-GA, MSCF, JV, FT, TIAM
Einsatz
BS2000-Dialogbetrieb, -Prozeduren, -Stapelbetrieb

[3] BS2000
Dienstprogramme
Beschreibung

Zielgruppe
BS2000-Anwender (nicht privilegiert)
Inhalt
Dienstprogramme für den nichtprivilegierten Benutzer des BS2000.
Einsatz
BS2000-Teilnehmerbetrieb

[4] **EDOR** (BS2000)
Beschreibung

Zielgruppe
Datenerfasser, Programmierer
Inhalt
Beschreibung der Anweisungen an den Datenbearbeiter EDOR.
Einsatz
BS2000-Dialogbetrieb

[5] **EDT** (BS2000)
Beschreibung

Zielgruppe
Datenerfasser, Programmierer
Inhalt
Beschreibung der Anweisungen an den Dateibearbeiter EDT, EDT-Prozeduren, Unterprogrammschnittstelle des EDT
Einsatz
BS2000-Dialogbtrieb und -Stapelbetrieb

[6] BS2000
Systemmeldungen
Beschreibung

Zielgruppe
BS2000-Anwender
Inhalt
Standardmeldungen BS2000 - Zentralsystem inklusive SPOOL, RSO, SDF.
Standardmeldungen der Softwareprodukte DCAM, TIAM, RBAM.

[7] BS2000
DVS Plattenverarbeitung
Beschreibung

Zielgruppe
BS2000-Anwender, Assembler-Programmierer (beide nicht privilegiert)
Inhalt
Funktionen des Datenverwaltungssystems im BS2000.
DVS-Kommandos und -Makroaufrufe, Service- und Aktionsmakroaufrufe.
Zugriffsmethoden UPAM, SAM, ISAM und EAM für Plattendateien.
Einsatz
BS2000-Dialogbetrieb, -Stapelbetrieb, -Programmierung.

[8] **COB1** (BS2000)
COBOL-Compiler
Benutzerhandbuch

Zielgruppe
COBOL-Anwender im BS2000
Inhalt
Bedienung des COB1-Compilers und der für die Entwicklung, das Binden, den Ablauf und Test von COBOL-Programmen erforderlichen Software; Aufbau des COB1-Systems und der erzeugten Bindemoduln; Programmierhinweise; Meldungen des Compilers sowie die COB1-Schnittstelle zu UDS.

[10] **SORT** (BS2000)
Beschreibung

Zielgruppe
BS2000-Anwender
Inhalt
Funktionen und Anweisungen für das Sortieren und Mischen von Dateien

[11] **PLI1** (BS2000)
PL/I-Compiler
Taschenbuch

Zielgruppe
PL/I-Anwender im BS2000
Inhalt
Tabellarische Darstellung aller Sprachelemente, der Testhilfen sowie der Steuerungsmöglichkeiten des Compilers und des Objekts.

[12] BS2000
Binder und Lader
Beschreibung

Zielgruppe
BS2000-Anwender
Inhalt
Beschreibung der Anweisungen zum Binden und Laden von Programmen mit TSOSLNK, ELDE und DLL
Einsatz
BS2000-Dialogbetrieb und -Stapelbetrieb

[13] **LMS** (BS2000)
Beschreibung

Zielgruppe
BS2000-Anwender
Inhalt
Beschreibung der Anweisungen zum Erstellen und Verwalten von PLAM-Bibliotheken mit LMS und Speicherung mit Delta-Technik.
Einsatz
BS2000-Dialogbtrieb und -Stapelbetrieb

[14] BS2000
Dialog-Testhilfe
Beschreibung

Zielgruppe
Programmierer
Inhalt
Beschreibung der Kommandos und Makroaufrufe an die Dialogtesthilfe IDA.
Einsatz
BS2000-Dialogbetrieb

[15] BS2000
Jobvariablen
Beschreibung

Zielgruppe
BS2000-Benutzer
Inhalt
Anwendungsmöglichkeiten für Jobvariablen zur Steuerung und Überwachung von Aufträgen und Programmläufen.
Bedingungsabhängige Auftragssteuerung.
Alle erforderlichen Kommandos und Makroaufrufe.
Anwendungsbeispiele.
Einsatz
BS2000-Teilnehmerbetrieb

[16] BS2000
Makroaufrufe an den Ablaufteil
Beschreibung

Zielgruppe
BS2000-Assembler-Programmierer (nicht privilegiert); Systemverwalter
Inhalt
Alle Makroaufrufe an den Ablaufteil in lexikalischer Reihenfolge mit Hinweisen und Beispielen; einschließlich ausgewählter Makroaufrufe für das DVS und für TIAM.
Zusammenstellung der Makroaufrufe nach Anwendungsgebieten. Ausführlicher Lernteil über Ereignissteuerung, Serialisation, Inter-Task-Kommunikation, Contingencies.
Einsatz
BS2000-Anwendungsprogramme

[17] BS2000
DVS Bandverarbeitung
Beschreibung

Zielgruppe
BS2000-Anwender, Assembler-Programmierer (beide nicht privilegiert)
Inhalt
Funktionen des Datenverwaltungssystems im BS2000.
DVS-Kommandos und -Makroaufrufe, Service- und Aktionsmakroaufrufe.
Zugriffsmethoden UPAM, SAM und BTAM für Banddateien.
Einsatz
BS2000-Dialogbetrieb, -Stapelbetrieb, -Programmierung

[18] **AID** (BS2000)
Advanced Interactive Debugger
Testen von PL/I-Programmen
Benutzerhandbuch

Zielgruppe
PL/I-Programmierer
Inhalt
Vorbereitungen für das symbolische Testen von PL/I-Programmen. Beschreibung aller AID-Kommandos, die zum symbolischen Testen zur Verfügung stehen. Beispiel einer AID-Sitzung. Meldungen
Einsatz
Testen von PL/I-Programmen im Dialog- und Stapelbetrieb.

[19] BS2000
Benutzer-Kommandos (SDF-Format)
Benutzerhandbuch

Zielgruppe
BS2000-Anwender
Inhalt
Benutzer-Kommandos des BS2000 in der Syntax der Dialogschnittstelle SDF (System Dialog Facility)
Einsatz
BS2000-Dialogbtrieb und -Stapelbetrieb mit SDF

Bestellen von Handbüchern

Die aufgeführten Handbücher finden Sie mit ihren Bestellnummern im *Druckschriftenverzeichnis Datentechnik*. Dort ist auch der Bestellvorgang erklärt. Neu erschienene Titel finden Sie in den *Druckschriften-Neuerscheinungen Datentechnik*.

Beide Veröffentlichungen erhalten Sie regelmäßig, wenn Sie in den entsprechenden Verteiler aufgenommen sind. Wenden Sie sich bitte hierfür an eine Geschäftsstelle unseres Hauses.

Stichwörter

C

D

E

M

Information on this document

On April 1, 2009, Fujitsu became the sole owner of Fujitsu Siemens Computers. This new subsidiary of Fujitsu has been renamed Fujitsu Technology Solutions.

This document from the document archive refers to a product version which was released a considerable time ago or which is no longer marketed.

Please note that all company references and copyrights in this document have been legally transferred to Fujitsu Technology Solutions.

Contact and support addresses will now be offered by Fujitsu Technology Solutions and have the format *...@ts.fujitsu.com*.

The Internet pages of Fujitsu Technology Solutions are available at *http://ts.fujitsu.com/*...
and the user documentation at *http://manuals.ts.fujitsu.com*.

Copyright Fujitsu Technology Solutions, 2009

Hinweise zum vorliegenden Dokument

Zum 1. April 2009 ist Fujitsu Siemens Computers in den alleinigen Besitz von Fujitsu übergegangen. Diese neue Tochtergesellschaft von Fujitsu trägt seitdem den Namen Fujitsu Technology Solutions.

Das vorliegende Dokument aus dem Dokumentenarchiv bezieht sich auf eine bereits vor längerer Zeit freigegebene oder nicht mehr im Vertrieb befindliche Produktversion.

Bitte beachten Sie, dass alle Firmenbezüge und Copyrights im vorliegenden Dokument rechtlich auf Fujitsu Technology Solutions übergegangen sind.

Kontakt- und Supportadressen werden nun von Fujitsu Technology Solutions angeboten und haben die Form *...@ts.fujitsu.com*.

Die Internetseiten von Fujitsu Technology Solutions finden Sie unter *http://de.ts.fujitsu.com/*..., und unter *http://manuals.ts.fujitsu.com* finden Sie die Benutzerdokumentation.

Copyright Fujitsu Technology Solutions, 2009