FOR1 (BS2000) V2.2A

Fortran-Compiler

Ausgabe Januar 1992

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

Inhalt

1 Einleitung

Der Compiler FOR1 übersetzt FORTRAN-Quellprogramme in Bindemoduln.

Der Sprachumfang des FOR1 umfaßt den Standard ANS FORTRAN 77 (ANSI X3.9-1978) sowie FORTRAN IV mit seinen Erweiterungen. FOR1 bietet gegenüber dem Standard-FORTRAN beträchtlich erweiterte Anwendungsmöglichkeiten. Leistungsfähige Optimierung und ein hohes Maß an Bedienungs- und Testkomfort zeichnen den Compiler aus.

Durch die große Zahl der Einstellungsmöglichkeiten läßt sich der FOR1 auf die unterschiedlichsten Anwendungsfälle abstimmen. Man kann den Ort des Quellprogramms, den Ablauf der Übersetzung, den Aufbau der erzeugten Bindemoduln und die Ausgabe der Protokollisten steuern. Dies bedeutet jedoch keineswegs, daß der Anwender für jeden Übersetzungslauf eine Vielzahl von Angaben machen muß: Der Compiler arbeitet - wo spezielle Angaben fehlen - mit Standardwerten.

Der FOR1-Compiler kann auf zwei unterschiedliche Arten gesteuert werden:
- durch Compileroptionen, die nach Aufruf des Compilers eingegeben werden
- durch Operanden eines SDF-Kommandos

SDF (System Dialog Facility) ist die neue Dialogschnittstelle des BS2000. Mit SDF ist es beispielsweise möglich, die Kommandos über Menüs einzugeben. Informationen zu Form und Bedeutung der zulässigen Angaben erscheinen - falls gewünscht - direkt auf dem Bildschirm.
Zur Verfügung stehen ein SDF-Kommando zum Übersetzen von FOR1-Quellprogrammen und ein SDF-Kommando zum Dynamischen Binden, Laden und Starten übersetzter Programme.

Der FOR1-Compiler unterzieht die Quellprogramme zahlreichen syntaktischen und semantischen Prüfungen. Bei Fehlern während der Übersetzung gibt der Compiler Meldungen aus, die über Fehlerort, Fehlerursache und Korrekturmaßnahmen des Compilers informieren. Bei Fehlern, die während der Laufzeit auftreten, werden vom Laufzeitsystem aussagekräftige Meldungen ausgegeben.

Zum Testen der Programme stehen dem Anwender verschiedene Testhilfen zur Verfügung. Durch Testanweisungen oder Testhilfe-Unterprogramme können Testhilfen bereits in den Quellprogrammtext eingebaut werden. Durch Angabe von Testoptionen können zusätzliche Tests zur Laufzeit veranlaßt werden. Neben diesen FOR1-integrierten Testhilfen kann zur Laufzeit mit der Dialogtesthilfe des BS2000 AID (siehe "AID - Testen von FORTRAN-Programmen" [3]) gearbeitet werden.

Um die Effizienz der ablauffähigen Programme zu erhöhen, führt der Compiler - falls gewünscht - Optimierungsmaßnahmen durch. Der Anwender hat dabei die Wahl zwischen Optimierungsstufen unterschiedlicher Intensität.

Die verschiedenen Einstellungsmöglichkeiten des FOR1 sind untereinander nahezu uneingeschränkt kombinierbar. Das bedeutet z.B., daß auch während der Testphase nicht auf die Optimierung verzichtet werden muß. Bindemoduln, die mit unterschiedlichen Compilereinstellungen erzeugt wurden, sind ablaufverträglich; man kann sie in der Regel problemlos zusammenbinden und ausführen.

1.1 Zielgruppe des Handbuchs

Dieses Handbuch wendet sich an Anwender von FORTRAN im BS2000. Der Leser benötigt Kenntnisse in der Programmiersprache FORTRAN sowie in der einfachen Anwendung des Betriebssystems BS2000.

Der Sprachumfang von FOR1 wird im Handbuch "FOR1-Beschreibung" [21] dargestellt.

Betriebssystemkomponenten werden an den entsprechenden Stellen kurz erläutert.

1.2 Konzept des Handbuchs

In diesem Benutzerhandbuch werden folgende Themen beschrieben:

- Bereitstellen von FORTRAN-Quellprogrammen im BS2000
- Übersetzen mit dem FOR1-Compiler
- Verwalten von Bindemoduln
- Binden zu ablauffähigen Programmen und Laden
- Ablauf von FOR1-Programmen
- Erstellen von FORTRAN-Quellprogrammen im Dialogmodus
- Verwendung von Dateien
- Verkürzen von Programmlaufzeiten (Optimierung)
- Testhilfen
- effizientes Programmieren

Das Handbuch wurde vorwiegend als Nachschlagewerk konzipiert. Bei der Fülle der Übersetzungsmöglichkeiten des FOR1 sollen ein detailliertes Inhaltsverzeichnis und ein ausführliches Stichwortverzeichnis den Zugang zu den Informationen erleichtern.

Zusätzlich informieren einige Übersichtstabellen über den Umfang der zur Verfügung stehenden Übersetzungs- und Ablaufvarianten:

SDF-Syntaxbeschreibung	Abschnitt 1.3.2
SDF-Fragebogen zur Steuerung der Übersetzung mit Verweisen auf entsprechende Compileroptionen	Tab. 2-2 bis 2-14
SDF-Fragebogen zur Steuerung des Ablaufs mit Verweisen auf entsprechende Laufzeitoptionen	Abschnitt 6.2
Compileroptionen mit Verweisen auf entsprechende SDF-Operanden	Abschnitt 2.3.3
Kurzformen für Compileroptionen und Optionswerte	Anhang A.1
Übersetzungsoperanden für die Ein-/Ausgabe von PLAM-Bibliothekselementen	Tab. 1-1
Übersicht über die Compiler-Steueranweisungen	Abschnitt 2.4
Zusammenstellung der Dialogkommandos	Tab. 3-1
Zusammenstellung der Testanweisungen	Abschnitt 7.4.1
Zusammenstellung der Testhilfe-Unterprogramme	Abschnitt 7.5.1
Abbildung der FORTRAN-Sätze in das DVS	Abschnitt 8.4.2

Anwendern, die mit dem FOR1-Compiler nicht vertraut sind, wird empfohlen, zunächst das Kapitel 2 zu lesen und danach die Bedeutungen der interessierenden Compileroptionen bzw. SDF-Operanden nachzuschlagen.

Anwendern, die bereits mit der Bedeutung von FOR1-Optionen vertraut sind, und die nun die Steuerung des Compilers durch SDF-Operanden kennenlernen möchten, wird zur Einführung Abschnitt 2.2.1. und Abschnitt 2.2.2 empfohlen. Welche SDF-Operanden den einzelnen Compileroptionen entsprechen, kann der Tabelle 2-15 in Abschnitt 2.3.3 entnommen werden. Auch die Übersichtstabellen Tab. 2-2 bis 2-14 sowie Tab. 6-1 weisen auf Entsprechungen zwischen SDF-Operanden und Compiler- bzw. Laufzeitoptionen hin.

Außerdem wird jeweils vor der Beschreibung der einzelnen Compiler- bzw. Laufzeitoptionen das Format der entsprechenden SDF-Operanden in einem eigenen Abschnitt dargestellt:

Operanden des Kommandos START-FOR1-COMPILER:

SOURCE	Abschnitt 3.2.2
INCLUDE-LIBRARY	Abschnitt 3.5.2
DIALOG	Abschnitt 3.6.1
SOURCE-PROPERTIES	Abschnitt 4.1.1
COMPILER-ACTION	Abschnitt 4.2.1
MODULE-LIBRARY	Abschnitt 4.3.1
LISTING	Abschnitt 4.6.1
COMPILER-TERMINATION	Abschnitt 4.8.1
MONJV	Abschnitt 4.9
LANGUAGE	Abschnitt 4.10.1
COMPILER	Abschnitt 4.11
TEST-SUPPORT	Abschnitt 7.1
OPTIMIZATION	Abschnitt 9.2.1
FPOOL-LIBRARY	Abschnitt 12.1.1

Operanden des SDF-Kommandos START-FOR1-PROGRAM:

FROM-FILE	Abschnitt 5.1
CPU-LIMIT, TESTOPT, MONJV, OBJECT-CONTINUATION, RUNTIME-OPTIONS	Abschnitt 6.1

1.3 Metasyntax

Bei der Darstellung von Beispiel-Dialogen und Ablaufprotokollen werden Anwendereingaben jeweils durch Fettdruck hervorgehoben.

Für die Darstellung von Anweisungen und Optionen werden in diesem Handbuch metasprachliche Konventionen verwendet, die in den folgenden beiden Abschnitten erläutert werden.

1.3.1 Metasyntax zur Darstellung von Compiler- und Laufzeitoptionen

`OBJECT`

Großbuchstaben bezeichnen Schlüsselwörter, die in dieser Form eingegeben werden müssen.

`name`

Kleinbuchstaben bezeichnen Variablen, die bei der Eingabe durch aktuelle Werte ersetzt werden müssen.

`YES`
`NO`

Die Unterstreichung eines Wertes bedeutet, daß es sich um einen Standardwert handelt, der vom FOR1-Compiler bzw. vom Betriebssystem eingesetzt wird, wenn der Anwender keine Angaben macht.

```
⎧YES⎫
⎩NO ⎭
```

Geschweifte Klammern schließen mehrere mögliche Angaben ein. Aus den angegebenen Größen muß eine Angabe ausgewählt werden. Die Möglichkeiten stehen untereinander. Befindet sich unter den angegebenen Größen ein Standardwert, dann ist keine Angabe erforderlich, wenn der Standardwert gewünscht wird.

`{YES|NO}`

Ein senkrechter Strich zwischen nebeneinander stehenden Angaben bedeutet ebenfalls, daß es sich um verschiedene Möglichkeiten handelt, von denen eine ausgewählt werden muß.

`[ ]`

Eckige Klammern schließen Wahlangaben ein, die weggelassen werden dürfen.

`( )`

Runde Klammern gehören zum Operanden und müssen mit eingegeben werden.

`␣`

Das Zeichen für Leerzeichen wird benutzt, wenn mindestens ein Leerzeichen syntaktisch notwendig ist.

`[,...]`
: Drei Punkte bedeuten, daß die vorausgehende metasprachliche Einheit mehrmals hintereinander wiederholt werden kann.

`:=`
: Eine links vom Zuordnungszeichen stehende Syntaxvariable wird durch die rechts vom Zuordnungszeichen stehende Angabe definiert.

Sonderzeichen
: sind ohne Veränderung zu übernehmen.

1.3.2 SDF-Syntaxbeschreibung

Diese Syntaxbeschreibung basiert auf der SDF-Version 1.4A. Die Syntax der SDF-Kommandosprache wird im folgenden in drei Tabellen erklärt.

Tabelle 1: Metazeichen

In den Kommandoformaten werden bestimmte Zeichen und Darstellungsformen verwendet, deren Bedeutung in Tabelle 1 erläutert wird.

Tabelle 2: Datentypen

Variable Operandenwerte werden in SDF durch Datentypen dargestellt. Jeder Datentyp repräsentiert einen bestimmten Wertevorrat. Die Anzahl der Datentypen ist beschränkt auf die in Tabelle 2 beschriebenen Datentypen.

Die Beschreibung der Datentypen gilt für alle SDF-Kommandos und SDF-Anweisungen. Deshalb werden bei den entsprechenden Operandenbeschreibungen nur noch Abweichungen von Tabelle 2 erläutert.

Tabelle 3: Zusätze zu Datentypen

Zusätze zu Datentypen kennzeichnen weitere Eingabevorschriften für Datentypen. Die Zusätze schränken den Wertevorrat ein oder erweitern ihn.

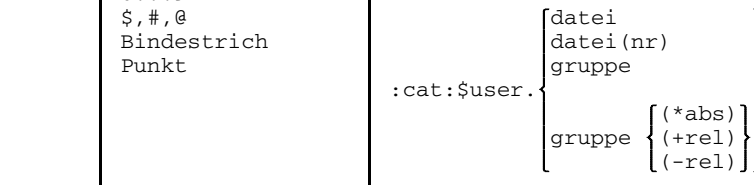
Die Beschreibung der Zusätze zu den Datentypen gilt für alle Kommandos und Anweisungen. Deshalb werden bei den entsprechenden Operandenbeschreibungen nur noch Abweichungen von Tabelle 3 erläutert.

Tabelle 1: Metazeichen

Kennzeichnung	Bedeutung	Beispiele
GROSSBUCHSTABEN	Großbuchstaben bezeichnen Schlüsselwörter. Einige Schlüsselwörter beginnen mit *	TESTOPT = NONE VERSION = *STD
=	Das Gleichheitszeichen verbindet einen Operandennamen mit den dazugehörenden Operandenwerten.	SOURCE-FORMAT = FIXED
< >	Spitze Klammern kennzeichnen Variablen, deren Wertevorrat durch Datentypen und ihre Zusätze beschrieben wird (siehe Tabellen 2 und 3).	LIBRARY = <full-filename 1..54>
Unterstreichung	Der Unterstrich kennzeichnet den Standardwert eines Operanden.	SUMMARY = YES / NO
/	Der Schrägstrich trennt alternative Operandenwerte.	TESTOPT = NONE / AID
(...)	Runde Klammern kennzeichnen Operandenwerte, die eine Struktur einleiten.	LAYOUT = PARAMETER(...)
Einrückung	Die Einrückung kennzeichnet die Abhängigkeit zu dem jeweils übergeordneten Operanden.	SOURCE = NO / YES (...) YES(...) INSERT-ERROR-WEIGHT =
|	Der Strich kennzeichnet zusammengehörende Operanden einer Struktur. Sein Verlauf zeigt Anfang und Ende einer Struktur an. Innerhalb einer Struktur können weitere Strukturen auftreten. Die Anzahl senkrechter Striche vor einem Operanden entspricht der Strukturtiefe.	*LIBRARY-ELEMENT(...) LIBRARY = ,ELEMENT = VERSION =
,	Das Komma steht vor weiteren Operanden der gleichen Strukturstufe.	,SOURCE = NO ,DIAGNOSTICS = NO

list-poss(n)	Aus den list-poss folgenden Operandenwerten kann eine Liste gebildet werden. Ist (n) angegeben, können maximal n Elemente in der Liste vorkommen. Enthält die Liste mehr als ein Element, muß sie in runde Klammern einschlossen werden.	list-poss: <integer 0..99>

Tabelle 2: Datentypen

Datentyp	Zeichenvorrat	Besonderheiten
alphanum-name	A...Z 0...9 $,#,@	muß mit Buchstabe oder Ziffer beginnen
c-string	EBCDIC-Zeichen	ist in Hochkommata einzuschließen; der Buchstabe C kann vorangestellt werden; Hochkommata innerhalb des c-string müssen verdoppelt werden
full-filename	A...Z 0...9 $,#,@ Bindestrich Punkt	Eingabeformat:  :cat: wahlfreie Angabe der Katalogkennung; Zeichenvorrat auf A...Z und 0...9 eingeschränkt; max. 4 Zeichen; ist in Doppelpunkte einzuschließen; Standardwert ist die Katalogkennung, die der Benutzerkennung laut JOIN-Eintrag zugeordnet ist. $user. wahlfreie Angabe der Benutzerkennung; Zeichenvorrat auf A...Z und 0...9 eingeschränkt; max. 8 Zeichen; $ und Punkt müssen angegeben werden; Standardwert ist die eigene Benutzerkennung.

		$. (Sonderfall) System-Standardkennung datei Datei- oder Jobvariablenname; letztes Zeichen darf kein Bindestrich oder Punkt sein; max. 41 Zeichen; muß mindestens A...Z enthalten. #datei (Sonderfall) @datei (Sonderfall) # oder @ als erstes Zeichen kennzeichnet je nach Systemgenerierung temporäre Dateien oder Jobvariablen. datei(nr) Banddateiname nr: Versionsnummer; Zeichenvorrat ist A...Z, 0...9, $, #, @. Klammern müssen angegeben werden. gruppe Name einer Dateigenerationsgruppe (Zeichenvorrat siehe unter "datei") gruppe{(*abs) (+rel) (-rel)} Name einer Dateigeneration (Zeichenvorrat siehe unter "datei") (*abs) absolute Generationsnummer (1-9999) * und Klammern müssen angegeben werden. (+rel) (-rel) relative Generationsnummer (0-99); Vorzeichen und Klammern müssen angegeben werden.
integer	0...9,+,-	+ bzw. - kann nur erstes Zeichen sein.
name	A...Z 0...9 $,#,@	darf nicht nur aus 0...9 bestehen.

Tabelle 3: Zusätze zu Datentypen

Zusatz	Bedeutung
x..y	a) beim Datentyp integer: Intervallangabe x Mindestwert, der für integer erlaubt ist. x ist eine ganze Zahl, die mit einem Vorzeichen versehen werden darf. y Maximalwert, der für integer erlaubt ist. y ist eine ganze Zahl, die mit einem Vorzeichen versehen werden darf. b) bei den übrigen Datentypen: Längenangabe x Mindestlänge für den Operandenwert; x ist eine ganze Zahl. y Maximallänge für den Operandenwert; y ist eine ganze Zahl. x=y Der Operandenwert muß genau die Länge x haben.

1.4 Änderungen gegenüber der Vorgängerversion (FOR1 V2.1A)

Das Handbuch wurde in weiten Teilen neu gegliedert.
Die ISP-Kommandosprache wurde in allen Kapiteln durch die SDF-Kommandosprache ersetzt.

In folgender Tabelle sind die wesentlichen sachlichen Neuerungen und Änderungen mit Verweisen auf die jeweiligen Abschnitte angeführt.
Die über das ganze Handbuch verteilten inhaltlichen und sprachlichen Korrekturen sind nicht eigens genannt.

Stichwort	neu	geän-dert	ent-fallen	Abschnitt
ILCS Programm-Kommunikationsschnittstelle	X			11.1 - 11.7
SDF-Steuerung		X		
- LINKAGE-Operand	X			2.2.3, 4.2.1
- FORTRAN90-CHECK-Operand	X			2.2.3, 4.1.1
- EXTENDED-SYSTEM-Operand			X	
- LISTING-Fragebogen		X		2.2.3
- Unterstützung der symbolischen Versionsbezeichner des LMS	X			2.2.3, 3.2.2 3.6.1, 4.6.1
COMOPT-Steuerung		X		
- LINKAGE-Option	X			4.2.2.6
- FORTRAN90-CHECK-Option	X			4.1.2.8
- EXTENDED-SYSTEM-Option			X	
- Unterstützung der symbolischen Versionsbezeichner des LMS	X			3.2.3, 3.6.2 4.6.2.2
Mathematische Routinen		X		10.3
FPOOL-Funktion GETDATE	X			12.2.10
Unterstützung des Dateiformats NK-SAM	X			8.2.4
Jahreszahlangabe in Compilerlisten		X		4.7, A.6
Dynamisches Bindeladen		X		5.5
PARMOD-Parameter		X		A.9.1
FOR1MODLIB			X	
Beschreibung des Testens mit IDA			X	
SIA (Scientific Instruction Assist)			X	
Auflistung der Bibliotheksmoduln			X	
Auflistung der Fehlermeldungen			X	

1.5 Allgemeine Voraussetzungen für das Übersetzen, Binden und den Programmablauf

Der FOR1-Compiler ist für Anlagen mit einem Hauptspeicher ab 2 MB geeignet. Abhängig von der Größe der Anwenderprogramme ist weiterer Hauptspeicherplatz empfehlenswert.

Zum Erzeugen eines ablauffähigen FOR1-Programms werden folgende Komponenten benötigt:

- der Compiler FOR1
- das Laufzeitsystem FOR1MODLIBS
- für das Vorladen des Compilers durch den Anwender (siehe 1.8):
  ENTER-Prozedur SYSENT.FOR1.022.LOAD1
- Fehlertextdatei für Ein-/Ausgabefehler:
  INCLUDE-Element IFNIOS in der FOR1-Makrobibliothek FOR1MACLIB (siehe A5)
- für die Nutzung des zentralen FPOOLs (siehe 12.2):
  Objektmodulbibliothek FOR1.FPOOLLIB und die zugehörige Datei der Schnittstellenbeschreibungen FOR1.FPOOL

Für das Erzeugen von mehrfachbenutzbaren Moduln steht die Prozedur SYSPRC.FOR1.022.SHARE zur Verfügung (siehe 5.8.1).

Im vorliegenden Handbuch wird davon ausgegangen, daß die oben genannten Komponenten in die Kennung TSOS eingebracht wurden (z.B. Aufruf des Compilers: `/START-PROGRAM FROM-FILE=$FOR1`).

1.6 Systemumgebung des FOR1-Compilers

Bild 1-1 zeigt die Einbettung des FOR1-Compilers im BS2000.

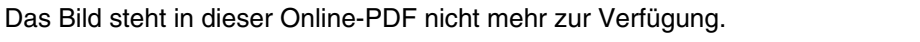


Bild 1-1: Systemumgebung des FOR1-Compilers

FOR1 und seine Lademoduln sind im Betriebssystem BS2000 als Benutzerprogramme einsetzbar (Programmklasse 2 = seitenwechselbar, Programmzustand TU, Speicherklassen 6, 5, 4).

1.7 Verwalten von Programmen in Programmbibliotheken

Der FOR1-Compiler kann auf Programmbibliotheken (PLAM-Bibliotheken) zugreifen. Programmbibliotheken sind PAM-Dateien, die mit der Zugriffsmethode PLAM (Program Library Access Method) bearbeitet werden. Der Zugriff auf PLAM-Bibliothekselemente ermöglicht eine einheitliche und ökonomische Verwaltung unterschiedlicher Elementtypen.

Als Elemente einer PLAM-Bibliothek können (u.a.) folgende Typen angelegt werden:

Typ S Quellprogramme, %INCLUDE-Elemente
Typ M Makros
Typ R Bindemoduln
Typ C Lademoduln
Typ L Bindelademoduln (LLMs)
Typ P Compilerlisten

FOR1 unterstützt PLAM-Bibliothekselemente vom Typ S, R, P und M.

In PLAM-Bibliotheken können alle Elementtypen in *einer* Bibliothek abgelegt werden. Es ist möglich, daß mehrere Elemente den gleichen Namen haben. Diese können durch Typ- oder Versionsbezeichnung unterschieden werden.

Die Datenhaltung in PLAM-Bibliotheken hat folgende Vorteile:

- Einsparung von bis zu 30% Speicherplatz durch das Zusammenlegen verschiedener Elementtypen und zusätzliche Komprimierungstechniken
- kürzere Zugriffszeiten als bei Verwendung herkömmlicher Bibliotheken
- Entlastung des EAM-Speichers (Bindemoduln werden direkt als PLAM-Bibliothekselemente abgelegt)

Übersetzungsoperanden für die Ein-/Ausgabe von PLAM-Bibliothekselementen

Die Eingabe und Ausgabe der verschiedenen PLAM-Bibliothekselemente wird durch Compileroptionen oder SDF-Operanden gesteuert. FOR1 kann folgende Programmzustände bzw. Listen als PLAM-Bibliothekselemente bearbeiten:

Typ	PLAM-Elementtyp	Steuerung durch Compileroption	Beschreibung in Abschnitt	SDF-Operanden
Compileroption	S	OPTIONS	3.3.2	-
Quellprogramme	S	SOURCE	3.2.3	SOURCE
Änderungszeilen	S	UPD	3.4	-
INCLUDE-Quellprogrammteile	S	INCLUDE-LIBRARY	3.5.3	INCLUDE-LIBRARY
Dialogsicherung	S	DIALOG-SAVE	3.6.4	DIALOG, SAVE-FILE
Bindemoduln	R	MODULE-LIBRARY	4.3.2	MODULE-LIBRARY
mehrfachbenutzbare Bindemoduln getrennt von nicht-mehrfachbenutzbaren Bindemoduln	R	SHARE-LIBRARY	4.2.2.2 5.8	COMPILER-ACTION SHAREABLE-CODE OUTPUT-LIBRARY
Listen	P	LIST-OUTPUT	4.6.2.4	LISTING OUTPUT

Tab. 1-1: Compileroptionen und SDF-Operanden für die Ein-/Ausgabe von PLAM-Bibliothekselementen

Auch die Binder DBL, TSOSLNK und BINDER können PLAM-Bibliotheksmoduln verarbeiten, TSOSLNK und BINDER können die erzeugten Moduln in PLAM-Bibliotheken ablegen (siehe Kapitel 5).

1.8 Vorladen des Compilers

FOR1 ist ohne besondere Vorkehrungen mehrfachbenutzbar, d.h. wenn ein Anwender einen Teil (Overlay) des Compilers in den virtuellen Speicher geladen hat, kann dieser Teil von allen anderen Tasks mitbenutzt werden.

Die Teile (Overlays) des Compilers bleiben im virtuellen Speicher, solange sie von mindestens einer Task benutzt werden. Für den jeweils ersten Benutzer eines Overlays wird dieses geladen.

Die Ladezeiten entfallen vollständig, wenn die Overlays von FOR1 durch Vorladen (Stapelaufträge) im virtuellen Speicher gehalten werden. War der Compiler nicht vorgeladen, dann erscheint an der Datensichtstation während der Übersetzung die Meldung:

```
FOR1: COMPILER NOT PRELOADED (BAD LOAD PERFORMANCE)
```

Im SUMMARY-Listing wird im Abschnitt COMPILE TIME der Hinweis "(COMPILER NOT PRELOADED)" gegeben.

Ist der Compiler vorgeladen, dann werden durch den Wegfall der Ladezeiten pro Programmeinheit erheblich CPU-Zeit und Anschaltzeit eingespart.

Den mehrfachbenutzbaren Compiler sollte der Systemverwalter vorladen. Für ein System mit einem Benutzeradreßraum von mehr als zwei Megabyte wird ein Vorlade-Auftrag benötigt. Stehen nur bis zu zwei Megabyte zur Verfügung, wird automatisch ein zweiter Vorlade-Auftrag gestartet.

Die Vorlade-Aufträge werden vorzugsweise beim Systemstart durch die Prozedur

```
/CALL-PROC NAME=SYSPRC.FOR1.022.SYSLOD
```

gestartet. Das Vorladen kann auch vom Anwender durch die Enterdatei

```
/ENTER-JOB FROM-FILE=$SYSENT.FOR1.022.LOAD1,RESOURCES=PAR(CPU-LIMIT=50)
```

gestartet werden.

In dieser Prozedur wird das Programm SYSPRG.FOR1.022.LOAD gestartet, das die Common Memory Pools für die Overlays des Compilers anmeldet. Ist der Benutzeradreßraum zu klein (<2MB), dann wird SYSPRG.FOR1.022.LOAD unterbrochen und die Prozedur SYSENT.FOR1.022.LOAD2 gestartet. Die Prozedur SYSENT.FOR1.022.LOAD2 ruft ihrerseits wieder SYSPRG.FOR1.022.LOAD auf und legt die restlichen Common Memory Pools an. Das unterbrochene Programm SYSPRG.FOR1.022.LOAD wird fortgesetzt. Es hat die Aufgabe, die Overlays in den Common Memory Pools zu halten, wenn der Compiler diese nicht mehr benötigt (daher auch der Ausdruck "Holdertask"). Folgt auf den Start der Vorlade-Aufträge ein Aufruf des Compilers, dann werden die einzelnen Overlays des Compilers in die bereitgestellten Common Memory Pools geladen. Erst wenn alle angesprochenen Overlays des Compilers in die Common Memory Pools geladen sind, wird die Meldung "FOR1: COMPILER NOT PRELOADED(...)" nicht mehr ausge-

geben. Wurde z.B. nur mit COMOPT OPTIMIZE=1 übersetzt, dann wird diese Meldung nach dem Vorladen nicht mehr ausgegeben. Folgt jedoch eine Übersetzung mit COMOPT OPTIMIZE=3, dann wird die Meldung wieder ausgegeben, da ein noch nicht geladenes Overlay des Compilers angesprochen wird.

Vorlade-Aufträge laufen in einer "VPASS-100"-Schleife, d.h. sie befinden sich fast immer in der VPASS-Warteschlange und belasten das System nur minimal. Diese Vorlade-Aufträge werden durch SHUTDOWN oder vorzeitig (vom Systemverwalter) durch SEND-MESSAGE oder durch CANCEL-JOB beendet.

Beispiel: Vorladen des Compilers

```
/ENTER-JOB FROM-FILE=SYSENT.FOR1.022.LOAD1,                                    (1)
 RESOURCES=PAR(CPU-LIMIT=50)

/START-PROG FROM-FILE=$FOR1                                                    (2)

 % BLS0500 PROGRAM 'FOR1', VERSION '2.2A00' OF '91-06-05' LOADED.
 % BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1991 ...
  FOR1: V2.2A00 READY, GIVE COMPILER OPTION
 *COMOPT SOURCE=QUELLE.TEST,END
  FOR1: COMPILER NOT PRELOADED (BAD LOAD PERFORMANCE)
  FOR1: NO ERRORS DURING COMPILATION OF P. U. TEST
  END OF  F O R 1  COMPILATION; CPU TIME USED:   4.649 SEC
/DEL-SYS-FILE FILE-NAME=OMF
/START-PROG FROM-FILE=$FOR1                                                    (3)
 %  BLS0500 PROGRAM 'FOR1', VERSION '2.2A00' OF '91-06-05' LOADED.
 %  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1991 ...
  FOR1:    V2.2A00 READY, GIVE COMPILER OPTION
 *COMOPT SOURCE=QUELLE.TEST, END
   FOR1: NO ERRORS DURING COMPILATION OF P. U. TEST
   END OF  F O R 1  COMPILATION: CPU TIME USED:    4.333 SEC
```

Erläuterung des Beispiels:

(1) Durch Aufruf der ENTER-Prozedur SYSENT.FOR1.022.LOAD1 wird der Vorladeprozeß gestartet. Die ENTER-Prozedur muß in diesem Falle unter der eigenen Kennung stehen.

(2) Bei der ersten Übersetzung des Programms wird der Compiler vorgeladen. Der Hinweis "COMPILER NOT PRELOADED" wird in diesem Übersetzungslauf noch ausgegeben.

(3) Der Compiler ist nun vorgeladen. Der Hinweis "COMPILER NOT PRELOADED" wird nicht mehr gegeben. Der vorgeladene Compiler benötigt weniger CPU-Zeit für die Übersetzung. Die Ersparnis an CPU-Zeit macht sich vor allem bei einem FORTRAN-Programm mit vielen Unterprogrammen bemerkbar. Durch das Vorladen wird dann nur für die erste Programmeinheit Ladezeit benötigt, die Ladezeiten für die weiteren Programmeinheiten entfallen.

1.9 Laufzeitsystem

Wie der FOR1-Compiler ist auch das FOR1-Laufzeitsystem mehrfachbenutzbar.

1.9.1 Struktur

Das FOR1-Laufzeitsystem besteht aus einzelnen Moduln, die zum Großmodul IF@RTS1 zusammengebunden sind. Dieser Großmodul befindet sich zusammen mit einigen Sprachverknüpfungs- und Testhilfemoduln in der Bibliothek FOR1MODLIBS.

Beim Binden werden die benötigten Laufzeitmoduln nicht komplett mit eingebunden, sondern nur Verknüpfungsmoduln eingebunden. Diese Verknüpfungsmoduln ermöglichen, daß beim Programmablauf das eigentliche Laufzeitsystem (der Großmodul IF@RTS1) dynamisch nachgeladen wird. Der Speicherplatz-Bedarf von abgespeicherten FOR1-Programmen wird dadurch erheblich gesenkt.

Das FOR1-Laufzeitsystem hat im wesentlichen folgende Aufgaben:

- Initialisierung der Runtime Communication Area (RTCA) und Beendigung von Programmen
- Durchführen der Ein-/Ausgabe-Operationen
- Bereitstellen von vorgefertigten Funktionen (INTRINSIC-Funktionen) und Unterprogrammen (Testhilfe-Unterprogramme)
- Fehlerbehandlung zur Ablaufzeit

Die vorgefertigten Funktionen umfassen die Standardmoduln für mathematische Funktionen und die Routinen zur Verarbeitung von Zeichenketten. Neben den Funktionen stehen eine Reihe von vorgefertigten Unterprogrammen hauptsächlich zu Testzwecken zur Verfügung.

Der Ein-/Ausgabe-Teil des Laufzeitsystems realisiert die Ein-/Ausgabe-Anweisungen auf drei funktionellen Stufen:

1. FORTRAN-Stufe
   Steuerung der verschiedenen Ein-/Ausgabearten (formatierte, unformatierte, NAMELIST-gesteuerte, listengesteuerte Ein-/Ausgabe).
2. Konversionsstufe
   Konversionen der Daten zwischen ihrer internen und externen Darstellung.
3. Systemanschlußstufe
   Aufruf der Ein-/Ausgabe-Funktionen des Betriebssystems (Zugriffe, Positionierungen).

Die Namen der FOR1-Bibliotheksmoduln werden beim Auftreten von Laufzeitfehlern bzw. bei Ausgabe der Aufrufhierarchie gemeldet.

1.9.2 Laden mit ADD-SHARED-PROGRAM und LOAD-PROGRAM

Das Laufzeitsystem wird vom Systemverwalter mittels der Kommandos

```
/ADD-SHARED-PROG ENTRY-NAME=IF@RTS1, LIB-NAME=$TSOS.FOR1MODLIBS
/LOAD-PROG *MODULE(LIB=$TSOS.FOR1MODLIBS, ELEM=IF@RTS1)
```

mehrfachbenutzbar in den Klasse-4-Speicher geladen.
Der Hauptspeicherbedarf des Lademoduls ist unabhängig von Zahl und Größe der benötigten Laufzeitmoduln. Der Klasse-4-Speicher bleibt bis zu SHUTDOWN belastet.

Ist der Modul IF@RTS1 nicht mehrfachbenutzbar geladen, so wird zum Ablaufzeitpunkt annähernd das gesamte Laufzeitsystem in den Benutzeradreßraum nachgeladen; in diesem Fall ist die Speicherplatzbelegung maximal. Aus diesem Grunde sollte das Laufzeitsystem FOR1MODLIBS stets mehrfachbenutzbar in den Klasse-4-Speicher geladen werden.

Im Klasse-4-Speicher darf nur eine Version des Laufzeitsystems stehen.

1.9.3 Laden über DSSM

Generierung des Subsystemkatalogs

Das mehrfachbenutzbare FOR1-Laufzeitsystem kann vom Systemverwalter über DSSM in den Klasse-4-, den Klasse-5- oder den Klasse-6-Speicher geladen werden. Voraussetzung dafür ist, daß das FOR1-Laufzeitsystem bei der Generierung des Subsystemkatalogs deklariert worden ist.

Das Laden des mehrfachbenutzbaren Laufzeitsystems über DSSM hat folgende Vorteile:

- durch Laden in den Klasse-5- bzw. Klasse-6-Speicher kann der Klasse-4-Speicher entlastet werden;
- innerhalb einer Anwendung kann mehr als eine Version des mehrfachbenutzbaren Laufzeitsystems verwendet werden, indem ein entsprechender neuer Subsystemkatalog geladen wird.

Der Subsystemkatalog wird mit UGEN erzeugt (siehe Handbuch "Systeminstallation" [38]). Im folgenden werden die zur Generierung des Subsystemkatalogs mit UGEN notwendigen Schritte kurz beschrieben:

1. Aufruf des Dienstprogrammes UGEN und Auswahl des UGEN-Zweigs zur Generierung eines Subsystemkatalogs (SSMCAT):

   ```
   /CALL-PROC NAME=$userid.UGEN
    GEN SSC
   ```

2. Definition der Datei für SSMCAT:

   ```
   DSMCAT DSSM-katalog-name,CAT=NEW
   ```

3. Deklarationen der einzelnen Subsysteme:

   – durch direkte Angabe der Deklarationen oder
   – über Subsystem-Eingabedateien (ASSIGN-SYSDTA-Kommando)

   Die FOR1-Laufzeitsystem-Deklarationen sind definiert in den Subsystem-Eingabedateien:

   – $TSOS.SYSSSD.FOR1.022.CL4 für Klasse-4
   – $TSOS.SYSSSD.FOR1.022.CL5 für Klasse-5
   – $TSOS.SYSSSD.FOR1.022.CL6 für Klasse-6.

   definiert.

   Durch das Kommando

   ```
   /ASSIGN-SYSDTA TO-FILE=$TSOS.SYSSSD.FOR1.022.{CL4|CL5|CL6}
   ```

   wird die entsprechende Subsystem-Eingabedatei des FOR1-Laufzeitsystems bei der Generierung des Subsystem-Katalogs verwendet.

4. Beenden des UGEN-Laufs mit der END-Anweisung:

   ```
   END
   ```

Aktivierung und Deaktivierung des FOR1-Laufzeit-Subsystems

Das FOR1LZS-Subsystem muß explizit vom Systemverwalter über das DSSM-Kommando CREATE-SUBSYSTEM aktiviert werden:

```
/CRE-SUBSYS SUBSYS=FOR1LZS, VERSION='0N.NN00'
```

N.NN bezeichnet den Versionsnamen, z.B. 2.2A.

Mit dem DELETE-SUBSYSTEM-Kommando wird ein Subsystem wieder deaktiviert:

```
/DEL-SUBSYS SUBSYS=FOR1LZS
```

Das DSSM-Kommando SHOW-SUBSYSTEM-STATUS gibt Informationen über ein Subsystem:

```
/SHOW-SUBSYS-STA SUBSYS=FOR1LZS
```

Besonderheiten bei der Erweiterung des Subsystemkatalogs

– Wird der Subsystem-Katalog dynamisch um das mehrfachbenutzbare FOR1-Laufzeitsystem erweitert, ist darauf zu achten, daß der Subsystem-Katalog, mit dem das Betriebssystem hochgefahren wurde, höchstens 20 neue Subsysteme aufnehmen kann, mit insgesamt höchstens 100 Eingangspunkten.

– Bei der dynamischen Erweiterung des Subsystem-Katalogs ist darauf zu achten, daß die Reihenfolge der bestehenden Subsysteme nicht verändert und auch keines der Subsysteme gelöscht wird.

  Der Subsystemname des mehrfachbenutzbaren Laufzeitsystems für den Klasse-4-, den Klasse-5- und den Klasse-6-Speicher ist 'FOR1LZS, VERSION 0N.NN00'. N.NN bezeichnet den Versionsnamen, z.B. 2.2A. Durch diese Namens- und Versionsgleichheit können nicht mehrere Subsystem-Deklarationen gleichzeitig zur Erstellung eines Subsystem-Katalogs herangezogen werden. Sollte dies dennoch nötig sein, müssen in den betreffenden Deklarationsdateien die Versionsnummern geändert werden.

2 Steuern des FOR1-Compilers

FOR1 übersetzt FORTRAN-Quellprogramme in Bindemoduln (Objektmoduln). In einem Übersetzungslauf können mehrere Programmeinheiten übersetzt werden.

Mit Hilfe von Übersetzungsoperanden kann der Anwender für jeden Übersetzungslauf die Bedingungen für die Übersetzung festlegen. Übersetzungsoperanden steuern die Eingabe des Quellprogramms, den internen Ablauf der Übersetzung, die Ausgabe der Bindemoduln und die Erstellung und Ausgabe von Protokoll-Listen.

Die Übersetzungsoperanden können auf folgende Arten festgelegt werden:

- mit dem SDF-Kommando START-FOR1-COMPILER (siehe Abschnitt 2.2)
- mit COMOPT-Anweisungen (siehe Abschnitt 2.3)
- mit dem PARAMETER-Kommando in eingeschränktem Umfang (siehe Anhang A.5.2)
- mit Compiler-Steueranweisungen im Quellprogramm einige ergänzende Operanden (siehe Abschnitt 2.4)

Aus der Vielzahl der möglichen Angaben für Übersetzungsoperanden kann jeder Anwender die für seine Anwendung geeignete Variante auswählen. Alle Übersetzungsoperanden haben voreingestellte Standardwerte, die wirksam werden, wenn die betreffende Angabe fehlt. So muß bei einem einfachen Übersetzungslauf (siehe Abschnitt 2.2 und 2.3) zum Beispiel nur die Quellprogrammdatei angegeben bzw. das Quellprogramm eingegeben werden.

Mehrfachangaben von Übersetzungsoperanden

Wird ein Übersetzungsoperand mehrmals spezifiziert, so gilt der zuletzt angegebene Wert. Falls für eine Task ein PARAMETER-Kommando gültig ist, so überschreibt die Angabe einer Compileroption oder eines SDF-Operanden die Angabe in dem entsprechenden PARAMETER-Kommando.

Gültigkeit

Compileroptionen und SDF-Operanden sind nur für den Übersetzungslauf gültig, für den sie angegeben sind.
Das PARAMETER-Kommando gilt entweder bis zum nächsten PARAMETER-Kommando oder in Prozeduren bis zum nächsten STEP-Kommando oder bis zum Taskende.

2.1 Starten des FOR1

Der FOR1-Compiler kann auf zwei Arten gestartet werden:

– Sollen die Übersetzungsoperanden als SDF-Operanden eingegeben werden, so startet der Anwender den Compiler mit dem Kommando START-FOR1-COMPILER unter Angabe der gewünschten Übersetzungsoperanden. Der Compiler beginnt nach Auswertung des Kommandos sofort mit der Übersetzung.

– Sollen die Übersetzungsoperanden als Compileroptionen eingegeben werden, so startet der Anwender den Compiler mit dem Kommando `START-PROGRAM $FOR1`. Nach dem Aufruf gibt FOR1 an der Datensichtstation die Versionsnummer des Compilers aus:

```
% BLS0500 PROGRAM FOR1, VERSION '2.2A00' OF '91-06-05' LOADED.
% BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1991 ...
FOR1: V2.2A00 READY, GIVE COMPILER OPTION
```

  Im Dialog an der Datensichtstation gibt FOR1 nun in jeder Zeile einen Stern aus und erwartet die Eingabe von Compileroptionen. Nach Eingabe der Compileroption END beginnt der Übersetzungsvorgang.

Programmüberwachung durch Jobvariable

Beim Aufruf des Compilers kann eine vorher definierte Jobvariable zur Programmüberwachung angegeben werden. Dadurch kann der Anwender Informationen über die Programmbeendigung des Übersetzungslaufs abfragen (siehe Abschnitt 4.9).

Vorladen des Compilers

Wenn der FOR1 nicht vorgeladen ist, erscheint während der Übersetzung die Meldung

```
FOR1: COMPILER NOT PRELOADED (BAD LOAD PERFORMANCE)
```

Das Vorladen ist sinnvoll, wenn der Compiler oft aufgerufen wird. Bei vorgeladenem Compiler entfallen die Ladezeiten für jeden Anwender, der FOR1 aufruft. Der Compiler kann vom Systemverwalter oder vom Anwender selbst vorgeladen werden (siehe Abschnitt 1.8).

Meldungen des Compilers

Bei fehlerfreier Übersetzung gibt der Compiler für jede übersetzte Programmeinheit folgende Meldung aus:

```
FOR1: NO ERRORS DURING COMPILATION OF P.U. name
```

name ist der Name der übersetzten Programmeinheit.

Am Ende der Übersetzung gibt der Compiler eine Meldung über die verbrauchte CPU-Zeit in Sekunden aus:

```
END OF  F O R 1  COMPILATION; CPU TIME USED: n.nnn SEC
```

Der Anwender kann wählen, ob die Meldungen des Compilers in deutscher oder englischer Sprache ausgegeben werden sollen. Dazu stehen der SDF-Operand LANGUAGE bzw. die Compileroptionen DIALOG und LANGUAGE zur Verfügung. Voreingestellt sind Meldungen in englischer Sprache.

2.2 Steuern mit SDF-Kommandos

2.2.1 Eingeben von SDF-Kommandos

SDF (**S**ystem **D**ialog **F**acility) deckt den Funktionsumfang der bisherigen BS2000-Kommandosprache (ISP-Format) ab. SDF bietet u.a. folgende Möglichkeiten:

- Der Anwender kann SDF-Kommandos über Kommando-Menüs bzw. Operandenfragebögen eingeben, wobei er zwischen drei Stufen mit einem unterschiedlichen Ausmaß an Benutzerführung wählen kann. In diesem "geführten" Dialogmodus kann auch ein Anwender ohne große Vorkenntnisse ein BS2000-Kommando absetzen, da SDF über den Bildschirm die zur Auswahl stehenden Operanden und eine kurze Beschreibung der Bedeutung und der zulässigen Werte ausgibt.
- Der Anwender kann SDF-Kommandos auch ohne Dialogführung eingeben. Er kann die Kommandos abkürzen, sich wahlweise einen Korrekturdialog bei fehlerhaften Eingaben ausgeben lassen oder vom ungeführten Dialog zeitweilig in den geführten Dialog wechseln.

Eine Einführung in SDF mit vielen Beispielen gibt das Handbuch "Einführung in die Dialogschnittstelle (SDF) [16]". Die Benutzerkommandos in der SDF-Kommandosprache werden im Handbuch "Benutzerkommandos (SDF-Format) [12]" beschrieben.

Der Anwender kann sich Informationen über SDF auch am Bildschirm durch Eingabe des Kommandos HELP-SDF? ausgeben lassen. Er erhält dann einen Operandenfragebogen, in dem er auswählen kann, welche speziellen Informationen (z.B. über Arten der Dialogführung, Abkürzungsregeln, Funktionstasten und Anweisungen zur Steuerung des Menüs) er am Bildschirm erhalten möchte.

Die Sprache, in der die Erklärungen in den SDF-Menüs ausgegeben werden, wird vom Systemverwalter eingestellt. Die Sprache der SDF-Meldungen kann vom Anwender mit dem Kommando MODIFY-MSG-ATTRIBUTES festgelegt werden.

Für die Übersetzung und für den Ablauf von FOR1-Programmen stehen je ein SDF-Kommando zur Verfügung:

- START-FOR1-COMPILER ist das SDF-Kommando für das Übersetzen eines FOR1-Quellprogramms. Mit den Operanden dieses Kommandos können bis auf wenige Ausnahmen alle Varianten für die Übersetzung ausgewählt werden, die durch Compileroptionen festlegbar sind.
- START-FOR1-PROGRAM ist das SDF-Kommando für das dynamische Binden, Laden und Starten eines übersetzten FOR1-Programms. Mit den Operanden dieses Kommandos können die wichtigsten Funktionen gewählt werden, die im Aufruf des dynamischen Bindeladers DBL, im Aufruf des statischen Laders ELDE und mit Laufzeitoptionen angegeben werden können.

Geführter Dialog

Im geführten Dialog können SDF-Kommandos über Menüs abgesandt werden. Zu Beginn eines geführten Dialogs wird eine Übersicht sämtlicher Anwendungsbereiche am Bildschirm ausgegeben. Nach Auswahl eines bestimmten Anwendungsbereichs (z.B. PROGRAMMING-SUPPORT) erhält der Anwender eine Übersicht aller Kommandos des ausgewählten Bereichs. Für das ausgewählte Kommando (z.B. START-FOR1-COMPILER) wird wiederum ein Operandenfragebogen ausgegeben, dem weitere Operandenfragebögen und evtl. Unterfragebögen zu einem Operanden folgen können.

Der Anwender kann im geführten Dialog zwischen minimaler, mittlerer oder maximaler Benutzerführung wählen. Diese Stufen unterscheiden sich im Umfang der in den Fragebögen ausgegebenen Informationen. Der geführte Dialog empfiehlt sich vor allem für den Anwender ohne größere Vorkenntnisse über ein Kommando. Dialogführung und direkt am Bildschirm ausgegebene Informationen über Bedeutung und Syntax von Kommandos und Operanden sowie über zulässige und voreingestellte Operandenwerte ersparen häufiges Nachschlagen in Handbüchern. Bei fehlerhafter Eingabe von SDF-Kommandos wird eine Fehlermeldung ausgegeben und ein Korrekturdialog eingeleitet, der ein erneutes Eingeben eines Kommandos erübrigt.
Im geführten Dialog kann in der NEXT-Zeile eines Bildschirms unabhängig vom aktuellen Anwendungsbereich ein beliebiges Kommando eingegeben werden.

Ungeführter Dialog

SDF-Kommandos können nicht nur über Menüs im geführten Dialog eingegeben werden, sondern auch in einer knappen Form analog dem bisherigen BS2000-Kommandoformat. In diesem ungeführten Dialogmodus kann der Anwender zwischen zwei Formen der Kommandoeingabe wählen, je nachdem, ob nur Fehlermeldungen (EXPERT-Form) oder Fehlermeldungen und Korrekturdialog (NO-Form) bei fehlerhafter Eingabe gewünscht werden. Die SDF-Kommandosprache erlaubt Abkürzungen, so daß sich SDF-Kommandos wesentlich reduzieren lassen.

Temporär geführter Dialog

Aus dem ungeführten Dialog kann jederzeit durch die Eingabe eines Fragezeichens in den geführten Dialog gewechselt werden, so daß der Anwender jederzeit fehlende Informationen über Kommandos oder Operanden am Bildschirm einholen kann. Beispielsweise erhält man nach der Eingabe von

```
%CMD:    START-FOR1-COMPILER? TEST
```

den Operandenfragebogen des geführten Dialogs (GUIDANCE = MINIMUM). In der Kopfleiste des Fragebogens ist "SOURCE=TEST" unter "OPERANDEN" bereits registriert. Durch Überschreiben der voreingestellten Operandenwerte mit einem Fragezeichen kann man weitere Informationen über bestimmte Operanden anfordern.

Einstellen des Dialogmodus

Der Dialogmodus wird durch den Operanden GUIDANCE im Kommando MODIFY-SDF-OPTIONS festgelegt. Folgende Dialogmodi können durch diesen Operanden gewählt werden:

GUIDANCE = {UNCHANGED | MAXIMUM | MEDIUM | MINIMUM | EXPERT | NO}

UNCHANGED	Es gilt die bisherige Vereinbarung.
MAXIMUM	Geführter Dialog: maximale Hilfestufe, d.h.: sämtliche Operandenwerte mit Zusätzen, Hilfetexte für Kommandos und Operanden.
MEDIUM	Geführter Dialog: sämtliche Operandenwerte ohne Zusätze, Hilfetexte nur für Kommandos.
MINIMUM	Geführter Dialog: minimale Hilfestufe, d.h.: nur Standardwerte der Operanden, keine Zusätze, keine Hilfetexte.
EXPERT	Ungeführter Dialog: Expertenmodus, d.h.: System fordert mit "/" zur Kommandoeingabe auf; kein Syntaxfehlerdialog; detaillierte Fehlermeldungen; geblockte Kommandoeingabe.
NO	Ungeführter Dialog: System fordert mit "% CMD:" zur Kommandoeingabe auf; Syntaxfehlerdialog (Korrektur fehlerhafter Eingaben ohne Wiederholung des gesamten Kommandos), geblockte Kommandoeingabe (mehrere Kommandos, die durch das logische Zeilenendezeichen getrennt sind, können gleichzeitig abgeschickt werden).

Kommandoeingabe mit Schlüsselwort-Operanden oder Stellungsoperanden

Operandenwerte können als Schlüsselwort-Operanden oder als Stellungsoperanden angegeben werden. Bei Stellungsoperanden muß für jeden weggelassenen Operanden ein Komma eingegeben werden.

Kommando mit Schlüsselwort-Operanden:

```
START-FOR1-COMPILER SOURCE=DAT1,INCLUDE-LIBRARY=*NONE, FPOOL-LIBRARY=*NONE,
DIALOG=YES(LANGUAGE=ENGLISH, DIALOG-INTERRUPT=ERRORS-ONLY,SAVE-FILE=
*STD-NAME (INCLUDE-EXPANSION=YES), LOG-CHANGED-LINES=YES)
```

Kommando mit Stellungsoperanden:

```
/START-FOR1-COMPILER DAT1,,,YES(,ERRORS-ONLY,*STD-NAME(YES),YES)
```

Wird in der Operandenfolge eines Kommandos ein Operand in der Form "Operand=Operandenwert" eingegeben, so müssen alle folgenden Operanden der gleichen Ebene ebenfalls als Schlüsselwort-Operanden angegeben werden.

Da die Reihenfolge der Operanden langfristig nicht garantiert werden kann, ist es ratsam, in Prozeduren nur Schlüsselwort-Operanden zu verwenden.

Abkürzungsregeln

Kommandonamen, Operandennamen und konstante Operandenwerte können folgendermaßen abgekürzt werden:

- von rechts nach links können Zeichen weggelassen werden, z.B. START-FOR1-C statt START-FOR1-COMPILER, SO= statt SOURCE=, *LIBRARY-EL statt *LIBRARY-ELEMENT.
- innerhalb von Teilfolgen können von rechts nach links Zeichen weggelassen werden, z.B. S-F-C statt START-FOR1-COMPILER, S-FO-P statt START-FOR1-PROGRAM, *L-E statt *LIBRARY-ELEMENT.
- die Abkürzungen müssen eindeutig sein; z.B. wird bei Eingabe von START-FOR1 gemeldet, daß dies eine mehrdeutige Abkürzung im Hinblick auf START-FOR1-COMPILER und START-FOR1-PROGRAM ist.

Informationen über die Abkürzungsregeln erhält man mit dem Kommando HELP-SDF.

2.2.2 Beispiel für einfachen Übersetzungs- und Programmablauf mit SDF-Kommandos

Unter einem einfachen Übersetzungslauf wird hier eine Übersetzung verstanden, bei der mit Ausnahme weniger spezifizierter Übersetzungsoperanden die voreingestellten Operandenwerte wirksam sind. Mit einem einfachen Programmablauf ist das Binden, Laden und Ausführen des übersetzten Programms durch den dynamischen Bindelader DBL gemeint.

```
/START-FOR1-COMPILER SOURCE=QUELLE.TEST                                    (1)
 %  BLS0500 PROGRAM 'FOR1', VERSION '2.2A00' OF '91-06-05' LOADED.
 %  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1991 ...
 FOR1:    V2.2A00 READY,GIVE COMPILER OPTION
 FOR1: COMPILER NOT PRELOADED (BAD LOAD PERFORMANCE)
 FOR1: NO ERRORS DURING COMPILATION OF P.U. TEST
 END OF  F O R 1  COMPILATION; CPU TIME USED: 0.210 SEC.                   (2)
/SET-TASKLIB FOR1MODLIBS                                                   (3)
/START-FOR1-PROGRAM                                                        (4)
 %  BLS0001 DBL VERSION 070 RUNNING
 %  BLS0517 MODUL 'TEST' LOADED
 BS2000  F O R 1  : FORTRAN PROGRAM "TEST "
 STARTED ON 1991-07-16 AT 14:17:39
 BS2000  F O R 1  : FORTRAN PROGRAM "TEST " ENDED PROPERLY AT 14:17:40
 CPU - TIME USED  :       0.0031 SECONDS
 ELAPSED TIME     :       0.8700 SECONDS
```

Erläuterung des Beispiels:

(1) Mit dem SDF-Kommando START-FOR1-COMPILER wird der FOR1 gestartet. Im SOURCE-Operanden wird die katalogisierte Datei QUELLE.TEST als Eingabeort des Quellprogramms festgelegt.

Per Voreinstellung werden die Quellprogrammliste, Diagnoseliste, Datenadreßliste, Optionenliste und Überblicksliste nach SYSLST ausgegeben (LISTING=STD).
Per Voreinstellung wird der Bindemodul im temporären EAM-Bereich (OMF) abgelegt. Aus früheren Übersetzungen stammende Bindemoduln im temporären EAM-Bereich werden automatisch gelöscht (MODULE-LIBRARY=*OMF(DELETE-OLD-CONTENTS=YES).

(2) Der Compiler meldet sich mit Versionsnummer und Datum. Der Compiler meldet, daß FOR1 nicht vorgeladen ist. Während des Übersetzungslaufs wurden keine Fehler gefunden. Die für die Übersetzung gebrauchte CPU-Zeit wird ausgegeben.

(3) Wenn sich die Moduln des Laufzeitsystems nicht in der TASKLIB des Systems befinden, muß die benutzereigene Modulbibliothek als TASKLIB zugewiesen werden, bevor der dynamische Bindelader mit START-FOR1-PROGRAM aufgerufen wird. Die benutzereigene Modulbibliothek heißt hier FOR1MODLIBS.

(4) Mit dem SDF-Kommando START-FOR1-PROGRAM wird der Bindemodul TEST (Name des Programms in der katalogisierten Datei QUELLE.TEST) gebunden, geladen und gestartet.
Da hier kein FROM-FILE-Operand angegeben ist, wird der Bindemodul TEST aus dem temporären EAM-Bereich des laufenen Prozesses genommen (Voreinstellung). Die ausführliche Form dieses Kommandos wäre:

```
START-FOR1-PROGRAM FROM-FILE=*MODULE(LIBRARY=*OMF, ELEMENT=*ALL)
```

Abgekürzte Form:

Durch den Gebrauch der Abkürzungsregeln können die SDF-Kommandos im obigen Beispiel in wesentlich kürzerer Form eingegeben werden.

```
/S-F- QUELLE.TEST
%  CMD0187 ABBREVIATION OF OPERATION NAME 'S-F-' AMBIGUOUS WITH REGARD    (2a)
 TO 'SET-FILE-ATTRIBUTES, SHOW-FILE-LINK, START-FOR1-COMPILER,...
/S-F-C QUELLE.TEST                                                       (2b)
/SE-T FOR1MODLIBS                                                        (3)
/S-FO-P                                                                  (4)
```

(2a) Bei nicht-eindeutiger Abkürzung erfolgt im eingestellten EXPERT-Modus eine Fehlermeldung (in anderen Dialog-Modi zusätzlich ein Korrekturdialog).

(2b) Der SOURCE-Operand wird als Stellungsoperand angegeben.
Kommando- und Operandennamen sowie konstante Operandenwerte können durch Weglassen von Zeichen von rechts nach links abgekürzt werden: START-FOR1-COMPILER wird verkürzt zu S-F-C.

(3) SET-TASKLIB wird verkürzt zu SE-T.

(4) START-FOR1-PROGRAM wird verkürzt zu S-FO-P.

2.2.3 Übersicht: SDF-Kommando START-FOR1-COMPILER und entsprechende Compileroptionen

Die folgenden Tabellen enthalten eine vollständige Auflistung aller Operanden des Kommandos START-FOR1-COMPILER. Den Operanden sind die entsprechenden Compileroptionen gegenübergestellt. Die Darstellung verwendet die in Abschnitt 1.3 festgelegten metasprachlichen Konventionen.

Folgende Compileroptionen haben keine entsprechenden SDF-Operanden:
OPTIONS, UPD, CODE, SUPPLIEDBOUND, UNIT, PAD

Bei einigen SDF-Operanden unterscheiden sich die Voreinstellungen von den Voreinstellungen der entsprechenden Compileroptionen (siehe die Gegenüberstellung in Tabelle 2-2 bis 2-14).

Übersicht: Die Haupt-Operanden des SDF-Kommandos START-FOR1-COMPILER

START-FOR1-COMPILER
SOURCE = ... ,INCLUDE-LIBRARY = ... ,FPOOL-LIBRARY = ... ,DIALOG = ... ,SOURCE-PROPERTIES = ... ,COMPILER-ACTION = ... ,MODULE-LIBRARY = ... ,LISTING = ... ,TEST-SUPPORT = ... ,OPTIMIZATION = ... ,COMPILER-TERMINATION = ... ,MONJV = ... ,LANGUAGE = ...

Tab. 2-1: Haupt-Operanden des SDF-Kommandos START-FOR1-COMPILER

SOURCE-Fragebogen: Eingabeort des Quellprogramms

SDF-Fragebogen	1. Unterfragebogen	2. Unterfragebogen	Entspr. COMOPTs
SOURCE = *SYSDTA			SOURCE = *
= <full-filename 1..54>			= datei
= *LIBRARY-ELEMENT(...)	LIBRARY = <full-filename 1..54> ELEMENT = <full-filename 1..41>(...)	VERSION = *HIGHEST-EXISTING = <alphanum-name 1..24>	*LIBRARY-ELEMENT (LIBRARY = plamlib, ELEMENT = name (VERSION = *HIGHEST EXISTING)) = version))

Tab. 2-2: SDF-Kommando START-FOR1-COMPILER, SOURCE-Fragebogen

INCLUDE-LIBRARY-Fragebogen: Zugriff auf Bibliotheken mit %INCLUDE-Elementen

SDF-Fragebogen	Entsprechende COMOPTs
INCLUDE-LIBRARY = *NONE	INCLUDE[-LIBRARY] = *NO
= list-poss: <full-filename 1..54>	= dateiname bzw. = (dateiname1[,dateiname2][,..])

Tab. 2-3: SDF-Kommando START-FOR1-COMPILER, INCLUDE-LIBRARY-Fragebogen

FPOOL-LIBRARY-Fragebogen: Steuern der FPOOL-Bearbeitung

SDF-Fragebogen	Entsprechende COMOPTs
FPOOL-LIBRARY = *NONE	NOFPOOL
= list-poss: = <full-filename 1..54>	FPOOL = fpoolname bzw. FPOOL = ([fpoolname[,fpoolname][,..]])

Tab. 2-4: SDF-Kommando START-FOR1-COMPILER, FPOOL-LIBRARY-Fragebogen

DIALOG-Fragebogen: Interaktive Analyse

SDF-Fragebogen	1. Unterfragebogen	2. Unterfragebogen	Entsprechende COMOPTs
DIALOG = NO			NODIALOG
			DIALOG = {param (param[,param] (param[,param [,param]]) }
	DIALOG-INTERRUPT = AFTER-ANY-PROG- UNIT		param := E[DIT] = ALL
	= ERRORS-ONLY		= NO
	SAVE-FILE = *NONE		DIALOG-SAVE keine DIALOG-SAVE-Option
	= *STD-NAME(...)	INCLUDE-EXPANSION = NO	= (*STD-FILE ,INCLUDE-EXPANSION = NO)
		= YES	= YES)
	= <full-filename 1..54>(...)	INCLUDE-EXPANSION = NO	= ([FILE=]datei ,INCLUDE- EXPANSIONS = NO)
		= YES	= YES)
	= *LIBRARY-ELEMENT (...)	INCLUDE-EXPANSION = NO	,INCLUDE-EXPANSIONS = NO)
		= YES	= YES)
		LIBRARY = <full-filename 1..54>	= ([FILE=] *LIBRARY-ELEMENT (LIBRARY = plamlib,
		ELEMENT = <full-filename 1..41> (...)	ELEMENT = name
		VERSION = <alphanum-name 1..24>	(VERSION = version))
		= *UPPER-LIMIT	= *UPPER-LIMIT))
	LOG-CHANGED-LINES = NO		CHANGE in LIST- oder LISTFILE- Option nicht angegeben
	= YES		LIST=(CHANGE) oder LISTFILE=listfilename (CHANGE), wenn gleichzeitig COMOPT DIALOG angegeben

Tab. 2-5: SDF-Kommando START-FOR1-COMPILER, DIALOG-Fragebogen

Als Präfix für Dialog-Kommandos ist beim SDF-Fragebogen DIALOG das Zeichen @, bei COMOPT DIALOG das Zeichen % voreingestellt.

SOURCE-PROPERTIES-Fragebogen: Festlegen und Prüfen der Eigenschaften des Quellprogramms beim Übersetzen

SDF-Fragebogen	1. Unterfragebogen	Entsprechende COMOPTs
SOURCE-PROPERTIES = STD		Voreinstellungen des Unterfragebogens
= PARAMETER(...)	COMPILEABLE-COMMENTS = *NONE	CCOM keine CCOM-Option
	= <c-string 1..60>	= 'kommentarmarkierugen'
	LINE-END-COMMENTS = *NONE	LINEEND keine LINEEND-Option
	= <c-string 1..10>	= 'endemarkierungen'
	LANGUAGE-STANDARD = FOR1	STANDARD-CHECK = NO
	= ANS77	= ANS77
	IMPLICIT-DECLARATION = YES	IMPLICIT
	= NO	NOIMPLICIT
	EXPONENT-UNDERFLOW = IGNORED	NOEXPUNDERFLOW
	= ERROR	EXPUNDERFLOW
	SOURCE-FORMAT = FIXED	SOURCE-FORMAT = FIXED
	= FREE	= FREE
	SAVE-CONSTANT = *STD	SAVE-CONSTANT = YES bei OPT=NO,0,1,2 = NO bei OPT=3 oder 4
	= NO	= NO
	= YES	= YES
	FORTRAN90-CHECK = NO	FORTRAN90-CHECK = NO
	= YES	= YES

Tab. 2-6: SDF-Kommando START-FOR1-COMPILER, SOURCE-PROPERTIES-Fragebogen

COMPILER-ACTION-Fragebogen: Festlegen der Eigenschaften des erzeugten Codes

SDF-Fragebogen	1. Unterfragebogen	2. Unterfragebogen	Entspr. COMOPTs
COMPILER-ACTION = SYNTAX-CHECK			NOGEN
= MODULE-GENERATION (...)			Voreinstellungen des Unter-fragebogens
	SHAREABLE-CODE = NO		OBJECT kein SHARE-Operandenwert
	= YES(...)		= SHARE
		OUTPUT-LIBRARY = *MODULE-LIBRARY	SHARE-LIBRARY = *MODULE-LIBRARY
		= <full-filename 1..54>	= plamlib
	MINIMAL-PRECISION = REAL-4(...)		REAL
		EXTERNAL-DATA = NO	= (4)
		= YES	= 4
	= REAL-8(...)		
		EXTERNAL-DATA = NO	= (8)
		= YES	= 8
	= REAL-16(...)		
		EXTERNAL-DATA = NO	= (16)
		= YES	= 16
	CONSTANT-PRECISION = AS-NEEDED		NOTRUNCONST
	= REAL-4		TRUNCONST
	CANCEL-CONDITION		
	= NONE		GEN/NOGEN=FAILURE
	= ERROR		NOGEN = ERROR
	= SEVERE-ERROR		NOGEN = SEVERE
	LINKAGE = STD		LINKAGE = STD
	= FOR1-SPECIFIC		= FOR1-SPECIFIC

Tab. 2-7: SDF-Kommando START-FOR1-COMPILER, COMPILER-ACTION-Fragebogen

MODULE-LIBRARY-Fragebogen: Ausgabeort der erzeugten Bindemoduln

SDF-Fragebogen	1. Unterfragebogen	Entspr. COMOPTs
MODULE-LIBRARY = <full-filename 1..54>		MODULE-LIBRARY =plamlib
= *OMF(...)		=*OMF
	DELETE-OLD-CONTENTS =YES	
	=NO	

Tab. 2-8: SDF-Kommando START-FOR1-COMPILER, MODULE-LIBRARY-Fragebogen

LISTING-Fragebogen: Erzeugen von Compilerlisten

SDF-Fragebogen	1. Unterfragebogen	2. Unterfragebogen	Entspr. COMOPTs
LISTING = NO			NOLIST
= STD			Voreinstellungen der Unterfrage-bögen
= PARAMETER (...)	OPTIONS = YES		LIST = (OPTIONS)
	= NO		
	SOURCE = NO		
	SOURCE = YES(...)	INSERT-ERROR-WEIGHT = NOTE	LIST = (SOURCE) , MSGLEVEL = NOTE
		= WARNING	= WARNING
		= ERROR	= ERROR
	DIAGNOSTICS = NO		
	= YES(...)	MINIMAL-WEIGHT = NOTE	LIST = (DIAG), MSGLEVEL = NOTE
		= WARNING	= WARNING
		= ERROR	= ERROR
	DATA-ALLOCATION-MAP = YES		LIST = (MAP)
	= NO		
	CROSS-REFERENCE = NO		
	= YES		LIST = (XREF)

	EXTERNAL-DICTIONARY = NO		
	= YES		LIST = (ESD)
	ASSEMBLER-CODE = NO		
	= YES		LIST = (OBJECT)
	SUMMARY = YES		LIST = (SUMMARY)
	= NO		
	OPTIMIZED-SOURCE = NO		
	= YES		LIST = (DECOMP)
	SORTING = BY-PROG-UNIT		NOCOLLECT
	= BY-LIST-TYPE		COLLECT = (LIST, LISTFILE)
	LAYOUT = STD		LINECNT=64,EJECT
	= PARAMETER(...)	LINES-PER-PAGE = 64	LINECNT = 64
		= <integer 20..255>	= zahl
		PAGE-EJECT-STMT = ACCEPTED	= EJECT
		= IGNORED	= NOEJECT
		TEXT-SEPARATOR = '|'	TEXT-SEPARATOR = '|'
		= '!'	= '!'
	OUTPUT = *SYSLST		LIST-OUTPUT = *SYSLST
	= <full-filename 1..54>		= listfilename
	= *STD-FILE		= *STD-FILE
	= *LIBRARY- ELEMENT(...)	LIBRARY = <full-filename 1..54> ELEMENT-PREFIX = *NONE = <alphanum-name 1..38> (...)	= *LIBRARY- ELEMENT (LIBRARY = plamlib, ELEMENT- PREFIX = *NONE = prefix
		VERSION = *UPPER-LIMIT = <alphanum-name 1..24>	(VERSION = version)) = UPPER- LIMIT))

Tab. 2-9: SDF-Kommando START-FOR1-COMPILER, LISTING-Fragebogen

TEST-SUPPORT-Fragebogen: Steuern der Testhilfen

SDF-Fragebogen	1. Unterfragebogen	2. Unterfragebogen	Entspr. COMOPTs
TEST-SUPPORT = STD			
= NO			
= PARAMETER (...)	STATEMENT-TABLE = YES		TESTOPT = (STNR)
	= NO		
	TOOL-SUPPORT = NO		SYMTEST = MAP
	= AID		SYMTEST = ALL, OPTIMIZE = NO
	CHECK-CODE = NO		TESTOPT = (STNR)
	= ALL		TESTOPT = ALL
	= YES(...)	PROCEDURE-ARGUMENTS = NO	fehlendes TESTOPT = (ARG)
		= YES	TESTOPT = (ARG)
		ARRAY-BOUNDS = NO	fehlendes TESTOPT = (BOUNDS)
		= YES	TESTOPT = (BOUNDS)
		ARRAY-SUBSCRIPTS = NO	fehlendes TESTOPT = (SUBSCR)
		= YES	TESTOPT = (SUBSCR)
		SUBSTRING-BOUNDS = NO	fehlendes TESTOPT = (STRING)
		= YES	TESTOPT = (STRING)
		BRANCH-STMTS = NO	fehlendes TESTOPT = (CNTRL)
		= YES	TESTOPT = (CNTRL)
		VARIABLE-ASSIGNMENT = NO	fehlendes TESTOPT = (UNDEF)
		= YES	TESTOPT = (UNDEF)
		USER-DEBUG-STMTS = NO	fehlendes TESTOPT = (DEBUG)
		= YES	TESTOPT = (DEBUG)

Tab. 2-10: SDF-Kommando START-FOR1-COMPILER, TEST-SUPPORT-Fragebogen

OPTIMIZATION-Fragebogen: Steuern der Optimierung

SDF-Kommando	1. Unterfragebogen	2. Unterfragebogen	Entspr. COMOPTs
OPTIMIZATION = NO			OPTIMIZE = NO
= LOW			= O
= MEDIUM(...)	CONDITIONAL-LOOPS = IGNORED		= 1
	= RISK-OPTIMIZED		= 2
	OPTIMIZE-PROCEDURES = NO		PROCEDURE- OPTIMIZATION = NO, STD
	= YES		= YES
	= SPECIAL		= SPECIAL- ATTEMPTS
= HIGH(...)	CONDITIONAL-LOOPS = IGNORED		OPTIMIZE = 3
	= RISK-OPTIMIZED		= 4
	OPTIMIZE-PROCEDURES = NO		PROCEDURE- OPTIMIZATION = NO
	= YES		= YES, STD
	= SPECIAL		= SPECIAL- ATTEMPTS
	OPTIMIZATION-HINTS = STD		OPTIMIZE = ({3|4}, FUNCTION-SIDEEFFECT = YES PARAMETER- SIDEEFFECT = NO, REORDER = NO)
	= PARAMETER(...)	REORDER-EXPRESSIONS	REORDER
		= YES	= YES
		= NO	= NO
		FUNCTION-SIDEEFFECTS = YES	FUNCTION- SIDEEFFECT = YES
		= NO	= NO
		ARGUMENT-SIDEEFFECTS = NO	PARAMETER- SIDEEFFECT = NO
		= YES	= YES

Tab. 2-11: SDF-Kommando START-FOR1-COMPILER, OPTIMIZATION-Fragebogen

COMPILER-TERMINATION-Fragebogen: Bedingungen für den Abbruch der Übersetzung

SDF-Fragebogen	1. Unterfragebogen	Entsprechende COMOPTs
COMPILER-TERMINATION = STD		
= PARAMETER(...)	CPU-LIMIT = NONE	
	= <integer 1..32767>	Begrenzung der maximalen Übersetzungszeit im START-PROGRAM-Kommando
	MAX-ERROR-WEIGHT = NONE	ERRKILL = FAILURE
	= ERROR	= ERROR
	= SEVERE-ERROR	= SEVERE
	MAX-ERROR-NUMBER = 100	MAXERR = 100
	= <integer 1..2147483639>	= n

Tab. 2-12: SDF-Kommando START-FOR1-COMPILER, Fragebogen COMPILER-TERMINATION

MONJV-Fragebogen: Überwachen des Übersetzungslaufs mit Jobvariable

SDF-Fragebogen	Entsprechende COMOPTs
MONJV = *NONE	keine Definition einer Jobvariable
= <full-filename 1..54>	Definition einer Jobvariable mit MONJV = jvname

Tab. 2-13: SDF-Kommando START-FOR1-COMPILER, Fragebogen MONJV

LANGUAGE-Fragebogen: Einstellen der Meldungssprache

SDF-Fragebogen	Entsprechende COMOPTs
LANGUAGE = ENGLISH	LANGUAGE=ENGLISH
= DEUTSCH	LANGUAGE=GERMAN

Tab. 2-14: SDF-Kommando START-FOR1-COMPILER, LANGUAGE-Fragebogen

Der letzte Operand des Kommandos START-FOR1-COMPILER heißt COMPILER (siehe Abschnitt 4.11). Dieser Operand ist im geführten Dialog nicht sichtbar und kann nur im NO- oder EXPERT-Modus eingegeben werden.

2.3 Steuern mit Compileroptionen

2.3.1 Eingeben von Compileroptionen

Der Anwender gibt die Compileroptionen in einer oder mehreren COMOPT-Anweisung(en) an. COMOPT-Anweisungen können im Dialogbetrieb direkt am Bildschirm eingegeben werden, sie können in eine Datei geschrieben werden oder in der Quellprogrammdatei stehen. Für die Eingabe von COMOPT-Anweisungen gibt es folgende Möglichkeiten:

Eingeben über SYSDTA:

- SYSDTA liegt auf der Primärzuweisung:
  Im Dialogbetrieb kann der Anwender die COMOPT-Anweisungen nach Starten des Compilers am Bildschirm eingeben. Der Compiler fordert die Compileroptionen explizit an, indem er einen Stern (*) in Spalte 1 vorgibt. Im Batchbetrieb werden die COMOPT-Anweisungen von der Prozedurdatei gelesen.
- SYSDTA wurde einer katalogisierten Datei oder einem Bibliothekselement zugewiesen:
  Die COMOPT-Anweisungen werden aus der Datei oder dem Bibliothekselement gelesen. Ist die zugewiesene Datei die Quellprogrammdatei, so müssen die COMOPT-Anweisungen vor dem Quellprogramm stehen.
- SYSDTA wurde SYSCMD zugewiesen (Übersetzungsprozeduren):
  Die COMOPT-Anweisungen werden aus der Prozedurdatei gelesen.

Die Zuweisung von SYSDTA kann mit dem Kommando ASSIGN-SYSDTA verändert werden (siehe Abschnitt 3.2.1).

Compileroption OPTIONS:

Der Anwender gibt den Eingabeort der COMOPT-Anweisung(en) in der Compileroption OPTIONS an (siehe Abschnitt 3.3.2).

Eine Zusammenstellung aller Compileroptionen befindet sich in Abschnitt 2.3.3.

Format einer COMOPT-Anweisung

```
[*]COMOPT␣option [,option]...
```

- [*]COMOPT ist nur am Anfang einer COMOPT-Anweisung zulässig.
  Im Batchbetrieb muß * in der Prozedurdatei angegeben werden.
- Eine Compileroption *option* besteht aus einem Schlüsselwort, ggf. gefolgt von einem Gleichheitszeichen und einem oder mehreren Optionswerten.

  Beispiel: LIST=(SOURCE, XREF,SUMMARY)
- Die Namen von Optionen und Optionswerten können abgekürzt werden. Von rechts beginnend können beliebig viele Buchstaben weggelassen werden, solange der abgekürzte Name noch eindeutig bleibt. Daneben gibt es zusätzliche Kurzformen, die in Abschnitt 2.3.3 und im Anhang A.1 zusammengestellt sind.
- Treten bei der Abarbeitung einer COMOPT-Zeile Fehler auf, so bleiben die bereits ausgewerteten Optionen dieser Zeile bestehen.
- COMOPT-Anweisungen gelten für genau eine Übersetzung.
- Werden die Compileroptionen über Lochkarten eingegeben, so dürfen nur die Spalten 1-72 verwendet werden.

Beenden der COMOPT-Eingabe

Die COMOPT-Eingabe kann auf folgende Arten beendet werden:

- `[[*]COMOPT] END` als letzte der COMOPT-Anweisungszeilen
- `END` als letztes Schlüsselwort in der Reihe der angegebenen Compileroptionen
- eine beliebige FORTRAN-Quellprogrammzeile

Beispiel:

```
*COMOPT LIST=(SOURCE,XREF,SUMMARY)
*COMOPT LISTFILE=(LIST)
*COMOPT END          bzw. *END
```

oder:

```
[*]COMOPT␣LIST=(SOURCE,XREF,SUMMARY),LISTFILE=(LIST),END
```

NO-Präfix

Die meisten Compileroptionen bestehen aus einem Schlüsselwort und einer Liste von Optionswerten. Dem Schlüsselwort kann das Präfix NO vorangestellt werden. Dadurch wird diese Compileroption nicht für die angegebenen Optionswerte gültig, sondern für die Komplementärmenge unter den möglichen Optionswerten.

Beispiel:

Die LIST-Optionswert-Spezifikation dient zur Steuerung der Ausgabe von Protokollisten.

```
*COMOPT NOLIST=(MAP,XREF,ESD)
```

In diesem Fall werden alle bis auf die drei angegebenen Listen ausgegeben.

Dieses Prinzip gilt auch für leere Optionswert-Listen:

`*COMOPT LIST = ()`	Es werden keine Listen ausgegeben.
`*COMOPT NOLIST=()`	Alle Listen werden ausgegeben.

Bei nicht vorhandener Optionswert-Liste schaltet das NO-Präfix die Option ab. Ohne NO-Präfix gelten die voreingestellten Optionswerte.

`*COMOPT NOLIST`	Es werden keine Listen ausgegeben.
`*COMOPT LIST`	Die Standardlisten werden ausgegeben.

Explizite Anforderung von Optionswerten

Bei einigen Compileroptionen kann der Compiler veranlaßt werden, Optionswerte explizit vom Anwender anzufordern.

Beispiel:

```
*COMOPT SOURCE=/
```

Die Übersetzung wird unterbrochen, so daß der Anwender SYSDTA mit einem ASSIGN-SYSDTA-Kommando auf die Quellprogrammdatei legen kann.

2.3.2 Beispiel für einfachen Übersetzungs- und Programmablauf mit Compileroptionen

Unter einem einfachen Übersetzungslauf wird hier eine Übersetzung verstanden, bei der mit Ausnahme weniger spezifizierter Übersetzungsoperanden die voreingestellten Optionswerte wirksam sind. Mit einem einfachen Programmablauf ist das Binden, Laden und Ausführen des übersetzten Programms durch den dynamischen Bindelader DBL gemeint.

```
/DEL-SYS-FILE OMF                                                          ( 1)
/START-PROGRAM $FOR1                                                       ( 2)
 %  BLS0500 PROGRAM 'FOR1', VERSION '2.2A00' OF '91-06-05'LOADED.          ( 3)
 %  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1991 ...
  FOR1:    V2.2A00 READY, GIVE COMPILER OPTION
*COMOPT SOURCE=QUELLE.TEST                                                 ( 4)
  FOR1: GIVE COMPILER OPTION
*COMOPT LIST
  FOR1: GIVE COMPILER OPTION                                               ( 5)
*COMOPT END
  FOR1: COMPILER NOT PRELOADED (BAD LOAD PERFORMANCE)
  FOR1: NO ERRORS DURING COMPILATION OF P. U. TEST                         ( 6)
  END OF F O R 1  COMPILATION; CPU TIME USED:   0.179 SEC
/SET-TASKLIB FOR1MODLIBS                                                   ( 7)
/START-FOR1-PROGRAM                                                        ( 8)
 %  BLS0001 DBL VERSION 070 RUNNING
 %  BLS0517 MODULE 'TEST' LOADED
 BS2000  F O R 1 : FORTRAN PROGRAM "TEST "
 STARTED ON 1991-07-16 AT 14:20:09
 BS2000  F O R 1 : FORTRAN PROGRAM "TEST " ENDED PROPERLY AT 14:20:11      ( 9)
 CPU - TIME USED :       0.0305 SECONDS
 ELAPSED TIME    :       1.9980 SECONDS
```

Erläuterung des Beispiels

(1) Löschen von eventuell aus früheren Übersetzungen stammenden Bindemoduln im temporären EAM-Bereich. Damit wird sichergestellt, daß der dynamische Bindelader beim nächsten Aufruf den Bindemodul aus der aktuellen Übersetzung verwendet.

(2) FOR1 wird mit dem Kommando START-PROGRAM $FOR1 aufgerufen.

(3) Der Compiler meldet sich mit der Versionsnummer und dem Datum. Er gibt einen Stern aus und erwartet die Eingabe von Compileroptionen.

(4) Die Angabe COMOPT SOURCE=QUELLE.TEST bewirkt, daß das Quellprogramm aus der katalogisierten Datei QUELLE.TEST eingelesen wird.
Wird die Compileroption LIST angegeben, gelten die voreingestellten Optionswerte der LIST-Option. In diesem Fall werden die folgenden Protokollisten nach SYSLST ausgegeben (siehe Abschnitt 4.5):
- Quellprogrammliste (source listing)
- Diagnoseliste (diagnostic listing)
- Datenadreßliste (map listing)
- Überblicksliste (summary listing)
- Liste der Optionen (options listing)

(5) Der Compiler erwartet mit der Ausgabe eines Sterns solange weitere Optionen, bis [COMOPT] END eingegeben wird. [COMOPT] END schließt die Eingabe von Compileroptionen ab. Der Compiler übersetzt das eingelesene Quellprogramm.

(6) Der Compiler meldet, daß der FOR1 nicht vorgeladen ist. Während des Übersetzungslaufs wurden keine Fehler gefunden. Die für die Übersetzung verbrauchte CPU-Zeit wird ausgegeben.

(7) Wenn sich die Moduln des Laufzeitsystems nicht in der TASKLIB des Systems befinden, muß die benutzereigene Modulbibliothek als TASKLIB zugewiesen werden, bevor der Bindelader DBL aufgerufen wird.
Die benutzereigene Modulbibliothek heißt hier FOR1MODLIBS. Sind die FOR1-Laufzeitmoduln in der TASKLIB des Systems vorhanden, so kann diese Anweisung entfallen.

(8) Standardmäßig werden die erzeugten Bindemoduln in den temporären EAM-Bereich (*OMF) ausgegeben. Mit dem Kommando START-FOR1-PROGRAM wird der dynamische Bindelader aufgerufen, der die Bindemoduln bindet, lädt und startet. Der temporäre EAM-Bereich besteht nur für die Dauer der Task, d.h. er wird bei Taskende gelöscht. Sollen die Bindemoduln permanent verfügbar gemacht werden, gibt es zwei Möglichkeiten:
- Die Bindemoduln werden direkt in eine mit der MODULE-LIBRARY-Option angegebene PLAM-Bibliothek ausgegeben (siehe Abschnitt 4.3)
- Die Bindemoduln werden mit dem Bibliotheksprogramm LMS in eine Bindemodulbibliothek eingetragen (siehe Abschnitt 4.5)

(9) Nach dem Ablauf des Programms erfolgt eine Endemeldung mit Ausgabe der CPU-Zeit und der verbrauchten Gesamtzeit.

2.3.3 Übersicht: Compileroptionen und entsprechende SDF-Operanden

In der folgenden Tabelle sind alle Compileroptionen zusammengestellt.
Die Funktion des Präfixes NO ist in Abschnitt 2.3.1 erklärt.

[*]COMOPT...	**Kurzform**	**Bedeutung**	**a) Beschreibung in Abschnitt b) Entspr. SDF-Operanden-Fragebogen c) Entspr. SDF-Operanden-Unterfragebogen**
CCOM='kommentarmarkierungen'	CC	Kennzeichen für die Übersetzung von Kommentarzeilen (max. 60 Zeichen)	a) 4.1.2.1 b) SOURCE-PROPERTIES c) COMPILEABLE-COMMENTS
CODE= { ([SOURCE={EBCDIC / ISO / BCD}] [,UPD={EBCDIC / ISO / BCD}]) / {EBCDIC / ISO / BCD} }	CO	Code des Quellprogramms bzw. der Änderungszeilen	a) 4.1.2.9
{ COLLECT=({LIST / LISTFILE[,LIST]}) / NOCOLLECT }	COL	COLLECT bewirkt ein Zusammenfassen der Listen gleichen Typs für mehrere Programmeinheiten	a) 4.6.2.3 b) LISTING c) SORTING
[NO[COMPATIBLE=]] {BGFOR / BS3FOR}	COMPAT, COM, BGF, BS3	Vermeidung von Unverträglichkeiten zwischen FOR1 und Siemens-BGFOR- bzw. Telefunken-BS3-FORTRAN-Compiler	a) 4.2.2.8
{ DIALOG [={param / (param[,param]) / (param[,param,[param]])}] / NODIALOG } param:={!|?|@|%|#|$ / D|E / E[DIT]={ALL|FIRST|NO}}	D ND	DIALOG schaltet die Interaktive Analyse ein	a) 3.6.2 b) DIALOG LANGUAGE c) DIALOG-INTERRUPT Voreinstellung des Dialogpräfix: @
DIALOG-[SAVE]= ([{*STD [-FILE] / [FILE=]{datei / plamspezifikation}}] [,[INCLUDE-EXPANSIONS=] {NO / YES}]) plamspezifikation:= [LIBRARY-ELEMENT] ([LIBRARY=]plamlib, [ELEMENT=]name[([VERSION=] {*UPPER-LIMIT / version})])	DIALOG-	Ausgeben des Dialogergebnisses in eine Datei	a) 3.6.4 b) DIALOG c) SAVE-FILE

Fortsetzung▷

Fortsetzung

[*]COMOPT...	**Kurzform**	**Bedeutung**	**a) Beschreibung in Abschnitt b) Entspr. SDF-Operanden-Fragebogen c) Entspr. SDF-Operanden-Unterfragebogen**
[NO]EJECT	EJ	Durchführen eines Seitenvorschubs	a) 4.6.2.7 b) LISTING c) LAYOUT, PAGE-EJECT-STMT
END		Abschluß für Compileroptionen	a) 2.3.1
ERRKILL= {E[RROR] | S[EVERE] | F[AILURE]}	EK	Fehlergrad, bei dem der Übersetzungsvorgang abgebrochen werden soll	a) 4.8.2 b) COMPILER-TERMINATION c) MAX-ERROR-WEIGHT
[NO]EXPAND	EX	Steuerung der Ausgabe der mit der %INCLUDE-Anweisung eingefügten Textstellen	a) 4.6.2.8 b) LISTING c) TYPE, SOURCE, INCLUDE-EXPANSION
[NO]EXPUNDERFLOW	EU	Setzen der entsprechenden Programmaske	a) 4.1.2.5 b) SOURCE-PROPERTIES c) EXPONENT-UNDERFLOW
FORTRAN90-CHECK= {YES | NO}	F90	Untersuchen des Quellprogramms auf zu Fortran90 inkompatible Spracherweiterungen	a) 4.1.2.8 b) SOURCE-PROPERTIES c) FORTRAN90-CHECK
{FPOOL [={(fpoolname[,fpoolname]...) | fpoolname}] | NOFPOOL}	F	Steuerung der FPOOL-Bearbeitung	a) 12.1.2 b) FPOOL-LIBRARY
{GEN | NOGEN [={E[RROR] | S[EVERE] | F[AILURE]}]}	G	Kontrolle der Generierung des Objektprogramms	a) 4.2.2.5 b) COMPILER-ACTION c) CANCEL-CONDITION
[NO]IMPLICIT	I	Verbot impliziter Typzuweisungen bei NOIMPLICIT	a) 4.1.2.4 b) SOURCE-PROPERTIES c) IMPLICIT-DECLARATION
INCLUDE [-LIBRARY]= {*NO | (dateiname1[,dateiname2][,...])}	INC	Die INCLUDE-LIBRARY-Option legt den Zugriff auf Bibliotheken hierarchisch fest	a) 3.5.3 b) INCLUDE-LIBRARY
LANGUAGE={ENGLISH|GERMAN}	LA	Sprache, in der ab dem Einlesen der Optionen FOR1-Meldungen ausgegeben werden	a) 4.10.2 b) LANGUAGE
LINKAGE= {STD | FOR1-SPECIFIC}	LNK	Erzeugen von Standardlinkage-Objekten	a) 4.2.2.6 b) COMPILER-ACTIONS c) LINKAGE
LINECNT= {64 | zahl}	LC	Anzahl der Zeilen pro Seite bei den Übersetzerlisten	a) 4.6.2.6 b) LISTING c) LAYOUT, LINES-PER-PAGE
LINEEND='endemarkierungen'	LE	Kennzeichen für Zeilenende	a) 4.1.2.2 b) SOURCE-PROPERTIES c) LINE-END-COMMENTS

Fortsetzung▹

Fortsetzung

[*]COMOPT...	**Kurzform**	**Bedeutung**	**a) Beschreibung in Abschnitt b) Entspr. SDF-Operanden-Fragebogen c) Entspr. SDF-Operanden-Unterfragebogen**
[NO]LIST [=([listenangabe][,...])] listenangabe:={ALL | MIN | NONE | OPTIONS | SOURCE | DIAG | ESD | MAP XREF | ATR | OBJECT | DECOMP | SUMMARY | CHANGE}	L	Auswahl der auf SYSLST auszugebenden Protokollisten	a) 4.6.2.2 b) LISTING c) TYPE CHANGE: b) DIALOG c) LOG-CHANGED-LINES
[NO]LISTFILE [=[listfilename] [({listenangabe | LIST} [,...])]]	LF	Auswahl der auf eine katalogisierte Datei auszugebenden Protokollisten; siehe LIST	a) 4.6.2.5 b) LISTING c) OUTPUT, TYPE
LIST-OUT[PUT]= {listfilename | *STD[-FILE] | *SYSLST | [*LIBRARY-ELEMENT] ([LIBRARY=]plamlib [,[ELEMENT[-PREFIX]=]{prefix | *NONE}] [([VERSION=]{version | *UPPER-LIMIT})]])}	LIST-	Steuerung der Listenausgabe in PLAM-Bibliothek oder katalogisierte Datei	a) 4.6.2.4 b) LISTING c) OUTPUT
MAXERR= {100 | zahl}	ME	Anzahl der Meldungen, bei deren Erreichen der Übersetzungsvorgang abgebrochen wird	a) 4.8.2 b) COMPILER-TERMINATION c) MAX-ERROR-NUMBER
MODULE[-LIBRARY]= {*OMF | plamlib}	MOD	Steuern der Ausgabe von Bindemoduln in EAM-Datei oder PLAM-Bibliothek	a) 4.3.2 b) MODULE-LIBRARY
MSGLEVEL= {N[OTE] | W[ARNING] | E[RROR] (SOURCE={N | W | E} [,DIAG={N | W | E}]) (DIAG={N | W | E} [,SOURCE={N | W | E}])}	MSG	Fehlergrad, von dem die Meldungen ausgegeben werden sollen	a) 4.6.2.1 b) LISTING c) SOURCE, DIAGNOSTICS INSERT-ERROR-WEIGHT MINIMAL-WEIGHT
{OBJECT [= {SHARE | *}] | NOOBJECT}	OBJ, O	Steuerung der Erstellung von Bindemoduln und Festlegung des Ausgabemediums; Erzeugen eines mehrfachbenutzbaren Code-Teils	a) 4.2.2.1 b) COMPILER-ACTION c) SHAREABLE-CODE

Fortsetzung▷

Fortsetzung

[*]COMOPT...	**Kurzform**	**Bedeutung**	**a) Beschreibung in Abschnitt b) Entspr. SDF-Operanden-Fragebogen c) Entspr. SDF-Operanden-Unterfragebogen**
OPT[IMIZE]= { NO / 0 / 1 / 2 / 3 / (3,parameter[,...]) / 4 / (4,parameter[,...]) } parameter:= { FUNC[TION-SIDEEFFECT]= { YES / NO } PARAM[ETER-SIDEEFFECT]= { YES / NO } REORDER= { YES / NO } }	OPT	Steuerung der Optimierung	a) 9.2.2 b) OPTIMIZATION c) CONDITIONAL-LOOPS OPTIMIZATION-HINTS
OPTIO[NS]= { * / datei / bibl(name) / plamspezifikation / / / + } plamspezifikation: siehe SOURCE-Option	OPTIO	Datei, in der weitere Compiler-Optionen stehen	a) 3.3.2
OUTPUT[= { dateiname / (dateiname[,expand]) / ([WS=]dateiname [,OS=dateiname] [,expand]) / ([WS=](dateiname [,expand]) [,OS=(dateiname [,expand])]) }]	OU	Ausgabe auf Datei bei Interaktiver Analyse: Originalprogramm bzw. Arbeitsdatei	a) 3.6.4 b) DIALOG c) SAVE-FILE
[NO]PAD	P	Eingabesätze werden mit Leerzeichen aufgefüllt.	a) 4.2.2.10 b) Voreingestellt ist PAD. Bei LANGUAGE-STANDARD= ANS77 wird NOPAD gesetzt.
PROCEDURE[-OPTIMIZATION]= { STD / NO / YES / SPECIAL[-ATTEMPTS] }	PR	Steuerung der Optimierung beim Aufruf von Prozeduren	a) 9.2.3 b) OPTIMIZATION c) OPTIMIZE-PROCEDURES
REAL= { 4 / (4) / 8 / (8) / 16 / (16) }	R	Festlegung der Mindestlänge für REAL- und COMPLEX-Größen	a) 4.2.2.3 b) COMPILER-ACTION c) MINIMAL-PRECISION

Fortsetzung▷

Fortsetzung

[*]COMOPT...	**Kurzform**	**Bedeutung**	**a) Beschreibung in Abschnitt b) Entspr. SDF-Operanden-Fragebogen c) Entspr. SDF-Operanden-Unterfragebogen**
SAVE-CONSTANT={NO / YES}	SA	Übergabe einer Kopie, falls Konstante als Aktualparameter	a) 4.1.2.7 b) SOURCE-PROPERTIES c) SAVE-CONSTANT
SHARE-LIB[RARY]= {*MODULE-LIBRARY / plamlib}	SH	Ablegen der mehrfach-nutzbaren Bindemoduln in einer gesonderten PLAM-Bibliothek	a) 4.2.2.2 b) COMPILER-ACTION c) SHAREABLE-CODE OUTPUT-LIBRARY
SOURCE[={* / datei / bibl(name) / plamspezifikation / + / (PRIMARY)}]	SRC	Ort des Quellprogramms	a) 3.2.3 b) SOURCE
plamspezifikation:= [*LIBRARY-ELEMENT]([LIBRARY=]plamlib, [ELEMENT=]name[([VERSION=] {*HIGHEST-EXIXTING / version})])	*LIB		c) *LIBRARY-ELEMENT
SOURCE-FORMAT={FIXED|FREE}	SOURCE-SF	Übersetzen eines formatfreien Quell-programms	a) 4.1.2.6 b) SOURCE-PROPERTIES c) SOURCE-FORMAT
ST[AN]D[ARD-CHECK]= {ANS77 / NO}	STD	Prüfen eines FOR1-Quellprogramms auf Abweichungen von ANS FORTRAN 77	a) 4.1.2.3 b) SOURCE-PROPERTIES c) LANGUAGE-STANDARD
[NO]SUPPLIEDBOUND	SUP, SB	Interpretation der Dimensionsangabe 1 als * in Unterprogrammen	a) 4.2.2.9 b) Voreingestellt ist NOSUPPLIEDBOUND.
SYMTEST={NO|MAP|ALL}	SYM	Erzeugen von LSD-Informationen zum symbolischen Testen mit AID	a) 7.6.1 b) TEST-SUPPORT c) TOOL-SUPPORT
[NO]TESTOPT [=([testparameter][,...])] testparameter:={ALL|STNR|ARG| BOUNDS|SUBSCR| STRING|CNTRL|UNDEF| DEBUG}	TO	Auswahl von Test-funktionen	a) 7.3 b) TEST-SUPPORT c) CHECK-CODE
TEXT-SEPARATOR= {'|' / '!'}	TEX	Senkrechte Linien in Listen durch "|" oder durch "!" dargestellt	a) 4.6.2.9 b) LISTING c) LAYOUT
[NO]TRUNCONST	TC	Abschneiden von REAL-Konstanten ohne Exponent	a) 4.2.2.4 b) COMPILER-ACTION c) CONSTANT-PRECISION

Fortsetzung▷

Fortsetzung

[*]COMOPT...	**Kurzform**	**Bedeutung**	**a) Beschreibung in Abschnitt b) Entspr. SDF-Operanden-Fragebogen c) Entspr. SDF-Operanden-Unterfragebogen**
UNIT= ({READ / WRITE / PRINT / PUNCH} =nn [, {READ / WRITE / PRINT / PUNCH} =nn]..)	U	Zuordnung von Dateinummern zu den Ein-Ausgabe-Anweisungen	a) 4.2.2.7
UPD [= {* / _ / datei / bibl(name) / / / +}]	UPD	Ort der Änderungsdatei	a) 3.4

Tab. 2-15: Übersicht: Compileroptionen und entsprechende SDF-Operanden

2.4 Übersicht: Compiler-Steueranweisungen im Quellprogramm

Compiler-Steueranweisungen (compile time statements) steuern die Ausgabe der Quellprogrammliste, die Eingabe des Quellprogrammtextes und die Verarbeitung von FPOOL-Unterprogrammen.

Compiler-Steueranweisungen können zusätzlich zu den Anweisungen der FORTRAN-Sprache in einem FORTRAN-Quellprogramm stehen. Die Compiler-Steueranweisungen werden wie FORTRAN-Anweisungen behandelt.

Die Compiler-Steueranweisungen sind durch ein vorangestelltes Prozentzeichen gekennzeichnet (Ausnahme: die Anweisung *DELETE). Sie werden vom Compiler wie Anweisungen der FORTRAN-Sprache behandelt, erhalten also eine Anweisungsnummer zugeordnet. Auf diese Anweisungsnummer kann sich auch eine eventuell auftretende Fehlermeldung beziehen.

Compiler-Steueranweisungen können an beliebiger Stelle im FORTRAN-Quellprogramm stehen. Eine Compiler-Steueranweisung gilt nur in der Programmeinheit, in der sie angegeben ist.

Compiler-Steueranweisungen können wie die Anweisungen der FORTRAN-Sprache Fortsetzungszeilen haben. Fortsetzungszeilen werden durch einen Eintrag in Spalte 6 markiert.

Die einzelnen Compiler-Steueranweisungen sind entsprechend ihrer inhaltlichen Zugehörigkeit in verschiedenen Abschnitten in diesem Handbuch beschrieben.

Anweisungen zum Steuern der Quellprogrammliste

Format	Beschreibung im Abschnitt	Bedeutung
%EJECT	4.6.3.2	Durchführen eines Seitenvorschubs
%EXPAND {ON | OFF}	4.6.3.1	Steuern der Ausgabe der durch die %INCLUDE-Anweisung eingefügten Texte
%SPACE n	4.6.3.3	Einfügen von Leerzeilen in die Quellprogrammliste
%TITLE ['text']	4.6.3.4	Ausgabe des angegebenen Textes in der Kopfzeile der Quellprogrammliste

Anweisung zum Einfügen von Texten im Quellprogramm

Format	Beschreibung im Abschnitt	Bedeutung
%INCLUDE { name / bibl(name) } [,CODE [={EBCDIC / ISO / BCD}]] [,parameterliste]	3.5.1	Einfügen von Quellprogrammzeilen in das FORTRAN-Quellprogramm

Anweisung zum temporären Ändern von Quellprogrammzeilen

Format	Beschreibung im Abschnitt	Bedeutung
*DELETE k1 [,k2]	3.4	Temporäres Löschen von Quellprogrammzeilen

Anweisungen zum Verarbeiten von FPOOL-Unterprogrammen

Format	Beschreibung im Abschnitt	Bedeutung
%FPOOL fpoolname [(fpoolfkt[,...])]	12.1.3	Angabe der Datei mit den Schnittstellenbeschreibungen aller aufgeführten Funktionen
%NOFPOOL (fpoolfkt [,fpoolfkt,...])	12.1.3	Alle Funktionen werden in der FPOOL-Verarbeitung ausgeschlossen

3 Eingabe des Quellprogramms

In diesem Kapitel werden die folgenden Punkte behandelt:

Abschnitt 3.1,"Erstellen des Quellprogramms", beschreibt die Möglichkeiten, ein Quellprogramm zu erstellen, das der FOR1 lesen soll. Ein Quellprogramm kann in einer Datei stehen oder direkt eingegeben werden. Als Hilfsmittel für die direkte Eingabe steht die Interaktive Analyse zur Verfügung.

Abschnitt 3.2, "Festlegen des Eingabeorts des Quellprogramms", beschreibt die Möglichkeiten, dem Compiler mitzuteilen, von wo das Quellprogramm gelesen werden soll. Das Quellprogramm kann der Systemdatei SYSDTA zugeordnet werden oder in den Übersetzungsoperanden angegeben werden (über die Compileroption SOURCE bzw. das SDF-Kommando START-FOR1-COMPILER).

Abschnitt 3.3, "Festlegen des Eingabeorts der Compileroptionen", beschreibt, wo der Compiler eventuell vorhandene Compileroptionen findet. Compileroptionen können über die Systemdatei SYSDTA oder mit Hilfe der Compileroption OPTIONS zur Verfügung gestellt werden.

Abschnitt 3.4, "Temporäres Ändern eines Quellprogramms: Compileroption UPD", beschreibt, wie der Anwender die von der Quellprogrammdatei gelesene Zeichenfolge durch Angabe von Änderungszeilen für einen speziellen Übersetzungsvorgang verändern kann.

Abschnitt 3.5, "Einfügen von Quellprogrammzeilen", beschreibt, wie Quellprogrammzeilen beim Übersetzungslauf eingefügt werden können. Es stehen die Compiler-Steueranweisung %INCLUDE, der SDF-Operand INCLUDE-LIBRARY und die Compileroption INCLUDE-LIBRARY zur Verfügung.

Abschnitt 3.6, "Eingeben des Quellprogramms mit Interaktiver Analyse", beschreibt, wie der Anwender mit der Compileroption DIALOG oder dem SDF-Operanden DIALOG ein Quellprogramm eingeben, interaktiv analysieren und korrigieren kann.

3.1 Erstellen des Quellprogramms

Ein Quellprogramm kann in einer Quellprogrammdatei stehen oder direkt eingegeben werden. Als Hilfsmittel für die direkte Eingabe steht die Interaktive Analyse zur Verfügung.

3.1.1 Erstellen einer Quellprogrammdatei

Im Normalfall wird ein FORTRAN-Quellprogramm mit einem Editor erstellt und in eine Datei abgespeichert. Der FOR1-Compiler liest das Quellprogramm aus dieser Datei. Das Quellprogramm bleibt in der Datei verfügbar und steht nach der ersten Übersetzung für Änderungen und weitere Übersetzungsläufe bereit. FOR1 kann Quellprogramme aus SAM- und ISAM-Dateien verarbeiten.

Eine Quellprogrammdatei kann in verschiedenen Formen vorliegen:

- als katalogisierte Datei in Form einer SAM- oder ISAM-Datei
- als Element einer Bibliothek
  In Bibliotheken sind Programme in komprimierter Form abgelegt. Die Ein- und Ausgabe von PLAM-Bibliothekselementen vom Typ S kann durch Compileroptionen oder SDF-Operanden gesteuert werden.
- als Gruppendatei (GAM-Datei; GAM = Group Access Method)
  Unter einer Gruppendatei versteht man die Menge der Sätze einer ISAM-Datei, deren Schlüssel mit einer bestimmten Zeichenfolge, dem Namen der Gruppendatei, beginnen (siehe Abschnitt 3.2.3, Beispiel 3).

Bild 3-1 zeigt die verschiedenen Möglichkeiten für das Erstellen einer Quellprogrammdatei:

Im Stapelbetrieb kann mit dem Kommando DATA eine Datei für Quellprogramme erstellt werden, die auf Diskette oder auf Lochkarten gespeichert sind (siehe Handbuch "Benutzerkommandos(ISP-Format)" [11]).

Im Dialogbetrieb kann eine Quellprogrammdatei folgendermaßen erstellt werden:

- mit dem Dateiaufbereiter EDT (siehe Handbuch "EDT-Anweisungen" [20]).
  Der EDT ist dialogorientiert, kann aber auch im Stapelbetrieb angewendet werden. Quellprogrammdateien können mit dem EDT in eine PLAM-Bibliothek abgelegt werden.
- mit der Interaktiven Analyse des FOR1 durch Sichern des direkt eingegebenen Quellprogramms (siehe Abschnitt 3.6).

Das Bild steht in dieser Online-PDF nicht mehr zur Verfügung.

Bild 3-1: Erstellen einer Quellprogrammdatei

Übernehmen von Quellprogrammen von Magnetplatte oder Magnetband

Liegt das Quellprogramm auf Magnetplatte oder Magnetband vor und soll es in eine Datei übernommen werden, dann bietet BS2000 folgende Möglichkeiten:

- Das Kommando COPY-FILE zum Kopieren von Dateien
- Dateiumsetzungsprogramme
  Sie haben vor allem für Datenstrukturen Bedeutung, die vom Standardfall abweichen (z.B. Satzlängen größer als 256 Zeichen).
- Das Softwareprodukt ARCHIVE
  ARCHIVE lagert Dateien auf Magnetbänder oder Magnetplatten aus und ermöglicht die Rekonstruktion der Dateien mit Hilfe dieser Sicherungskopien (siehe Handbuch "ARCHIVE" [4]).
- Das Softwareprodukt PERCON
  PERCON überträgt Daten von einem Datenträger auf einen anderen oder auf mehrere verschiedenen Typs. Gleichzeitig können die Daten aufbereitet und die Dateimerkmale verändert werden (siehe Handbuch "PERCON" [36]).

Ablegen einer Quellprogrammdatei

Quellprogrammdateien können mit Hilfe des Bibliotheksverwaltungsprogramms LMS oder mit Hilfe des Dateiaufbereiters EDT (ab Version 16.1A) in PLAM-Bibliotheken abgelegt werden.

Ändern einer Quellprogrammdatei

Für Änderungen in Quellprogrammdateien gibt es folgende Möglichkeiten:
- EDT (siehe Handbuch "EDT-Anweisungen" [20])
- Interaktive Analyse (siehe Abschnitt 3.6; ein Beispiel für Änderungen im Rahmen der Interaktiven Analyse befindet sich im Abschnitt 3.6.8)

3.1.2 Direktes Eingeben des Quellprogramms

Ein Quellprogramm kann direkt in den Compiler eingegeben werden, d.h. ohne Zwischenspeicherung auf eine katalogisierte Datei.

Dazu wird FOR1 mit dem Kommando START-PROGRAM $FOR1 aufgerufen oder mit dem Kommando START-FOR1-COMPILER ohne Angabe des SOURCE-Operanden. Anschließend kann das Quellprogramm eingegeben werden. Die Eingabe des Quellprogramms muß mit den Zeichen "/*" in den Spalten 1 und 2 abgeschlossen werden.

Beispiel:

Einlesen des Quellprogramms von der Datensichtstation über SYSDTA. Die Quellprogrammeingabe wird durch "/*" abgeschlossen.

```
/START-PROG $FOR1        bzw.  /START-FOR1-COMPILER
% BLS0500 PROGRAM 'FOR1' VERSION '2.2A00' OF '91-06-05' LOADED
% BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1991 ...
FOR1: V2.2A00 READY, GIVE COMPILER OPTION
*       PROGRAM TEST
    .
    .
    .
*       END
*/*
```

Bei direkter Eingabe ist das Quellprogramm nur für eine einzige Übersetzung verfügbar. Zur Fehlerkorrektur muß das gesamte Quellprogramm neu eingegeben werden.

Bei direkter Eingabe ist die Interaktive Analyse empfehlenswert (siehe Abschnitt 3.6), da hier der FOR1-Compiler bei formalen Fehlern, z.B. Tippfehlern, unterbricht. Eingebaute Dateiaufbereiterfunktionen erlauben eine sofortige Korrektur dieser Fehler, und die Übersetzung kann fortgesetzt werden. Ein weiterer Vorteil der Interaktiven Analyse ist, daß das korrigierte Quellprogramm in einer externen Datei gespeichert werden kann.

3.2 Festlegen des Eingabeorts des Quellprogramms

Ein Quellprogramm kann dem Compiler direkt eingegeben werden oder in einer Datei stehen. Der Anwender muß dem Compiler den Eingabeort des Quellprogramms mitteilen, d.h. von wo der Compiler das Quellprogramm lesen soll. Dazu gibt es folgende Möglichkeiten:

- Der Compiler soll das Quellprogramm von der Systemdatei SYSDTA lesen. Dazu wird SYSDTA vor Aufruf des FOR1 mit dem Kommando ASSIGN-SYSDTA der Quellprogrammdatei zugewiesen bzw. bei direkter Eingabe der Datensichtstation. Compileroptionen, die in der Quellprogrammdatei stehen, müssen vor dem Quellprogrammtext stehen.
- Der Anwender gibt den Eingabeort des Quellprogramms im SDF-Operanden SOURCE bzw. in der Compileroption SOURCE an.

3.2.1 Kommando ASSIGN-SYSDTA

Das Betriebssystem BS2000 verwendet standardisierte Systemdateien, um Eingaben oder Ausgaben durchzuführen. SYSDTA ist eine Systemdatei für Eingaben.

BS2000 gibt eine Standardzuweisung vor, die "Primärzuweisung". Die Primärzuweisung ist im Dialogbetrieb die Datensichtstation und im Batchbetrieb die Spoolin-Datei.
Die Primärzuweisung kann verändert werden.

Soll FOR1 Quellprogramme oder Compileroptionen über SYSDTA lesen, so muß der Anwender für die richtige Zuweisung von SYSDTA sorgen, bevor er FOR1 aufruft. Die Zuweisung von SYSDTA wird mit dem Kommando ASSIGN-SYSDTA verändert.

```
ASSIGN-SYSDTA

 TO-FILE = <full-filename 1..54> / *LIBRARY-ELEMENT(...) / *PRIMARY /
           *SYSCMD / *DISKETTE(...)


   *LIBRARY-ELEMENT(...)
     |
     |  LIBRARY = <full-filename 1..51>
     |
     | ,ELEMENT = <full-filename 1..38>(...)
     |
     |    <full-filename 1..38>(...)
     |      |  VERSION = *STD / <text 0..10>
     |      | ,TYPE = STD / D / M
     |
   *DISKETTE(...)
     |
     |  UNIT = *ANY / <alphanum-name 2..2>
     |
     | ,FILE-NAME = <name 1..8>
     |
     | ,VOLUME = list-poss(10): <alphanum-name 1..6>
```

TO-FILE =
Eingabequelle, der SYSDTA zuzuordnen ist.

TO-FILE = <full-filename 1..54>
Name der Datei, der SYSDTA zuzuweisen ist. Die Datei muß folgende Merkmale haben:
- muß bereits katalogisiert sein
- variable Satzlänge
- Zugriffsmethode SAM oder ISAM
- Beginn ISAM-Schlüssel: Byte 5
- Länge ISAM-Schlüssel: 8 Bytes

TO-FILE = *LIBRARY-ELEMENT(...)

LIBRARY = <full-filename 1..51>
Name einer LMS-Bibliothek

ELEMENT = <full-filename 1..38> (...)
Name eines Elements aus der angegebenen Bibliothek. Zusätzlich erlaubtes Zeichen ist der Bindestrich. Er darf jedoch nicht als letztes Zeichen angegeben werden. Die Summe der Längen des Bibliotheksnamens - ohne Katalog- und Benutzerkennung gerechnet- und des Elementnamens darf maximal 39 Zeichen betragen.

VERSION = *STD / <text 0..10>
Ergänzung des Elementnamens durch die Versionsangabe

VERSION = *STD
Höchste Version

TYPE = STD / D / M
Typ des Elements
Standard ist Elementtyp S (Quellprogramme)
D Elementtyp D (Textdaten)
M Elementtyp M (Makros)

TO-FILE = *PRIMARY
Setzt SYSDTA auf die Primärzuweisung zurück

TO-FILE = *SYSCMD
Schaltet SYSDTA mit SYSCMD zusammen, d.h. das System liest über SYSCMD nicht nur Kommandos, sondern auch Daten ein.

TO-FILE = *DISKETTE (...)

UNIT = *ANY / <alphanum-name 2..2>
Mnemotechnischer Gerätename eines Diskettenlaufwerks.
SYSDTA wird dem angegebenen Diskettenlaufwerk zugewiesen.

FILE-NAME = <name 1..8>
Name der Diskettendatei

VOLUME = list-poss(10): <alphanum-name 1..6>
Datenträgerkennzeichen der Diskette. Maximal 10 Datenträgerkennzeichen sind erlaubt.

Eine ausführliche Beschreibung des Kommandos ASSIGN-SYSDTA befindet sich in "Benutzerkommandos (SDF-Format)" [12].

Nach dem Übersetzen ist es sinnvoll, die Zuordnung von SYSDTA zu erneuern. Durch Eingeben von ASSIGN-SYSDTA TO-FILE=*PRIMARY wird die Primärzuweisung wiederhergestellt.

Einschränkung :

Das gleichzeitige Einlesen von Quellprogramm und Änderungszeilen über SYSDTA ist nicht möglich.

Beispiele:

Beispiel 1: Einlesen von einer Datei

Einlesen des Quellprogramms von einer Datei QUELL.MAT:

```
/ASS-SYSDTA TO-FILE=QUELL.MAT
/START-PROG $FOR1
/ASS-SYSDTA TO-FILE=*PRIMARY
```

FOR1 findet in QUELL.MAT keine COMOPT-Zeile mit der Compileroption SOURCE und nimmt daher SYSDTA als Ort des Quellprogramms an (Standard).

Beispiel 2: Einlesen aus einer Bibliothek

Einlesen eines Quellprogramms aus der LMS-Bibliothek LMS.BIBL; Elementname ELE.

```
/ASS-SYSDTA TO-FILE=*LIB-ELEM(LIB=LMS.BIBL,ELEM=ELE)
/START-PROG $FOR1
/ASS-SYSDTA TO-FILE=*PRIMARY
```

Beispiel 3: Zwei Gruppen von Compileroptionen in einer Datei

Zwei Gruppen von Compileroptionen stehen in der Datei OPT. Jede Gruppe enthält den Ort des Quellprogramms und ist durch END abgeschlossen.

Inhalt von OPT:

```
COMOPT SOURCE=SRC.PROG1, ... ,END
COMOPT SOURCE=SRC.PROG2, ... ,END
```

FOR1 soll die Compileroptionen über SYSDTA lesen, der Eingabeort des Quellprogramms ist in der Compileroption SOURCE angegeben.

Übersetzung:

```
/ASS-SYSDTA OPT
/START-PROG $FOR1
/START-PROG $FOR1
/ASS-SYSDTA *PRIMARY
```

Die erste Übersetzung benutzt den ersten Optionensatz, die zweite Übersetzung den zweiten. Die Zuordnung von SYSDTA darf zwischen den Übersetzungen nicht verändert werden. Dadurch bleibt die Positionierung erhalten.

3.2.2 SDF-Operand SOURCE

Mit dem SDF-Operanden SOURCE kann man dem Compiler den Eingabeort einer Quellprogrammdatei mitteilen.

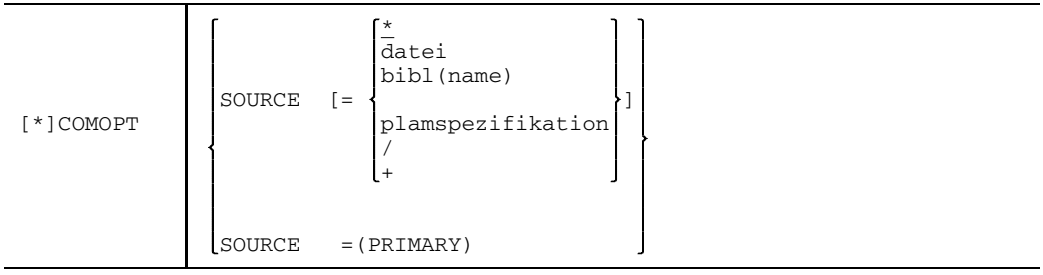
```
START-FOR1-COMPILER

 SOURCE = *SYSDTA / <full-filename 1..54> / *LIBRARY-ELEMENT(...)

   *LIBRARY-ELEMENT(...)
      |
      |  LIBRARY = <full-filename 1..54>
      | ,ELEMENT = <full-filename 1..41>(...)
      |    | VERSION = *HIGHEST-EXISTING / <alphanum-name 1..24>
```

Eine Gegenüberstellung von SDF-Operanden und entsprechenden Compileroptionen enthält Tab. 2-2.

3.2.3 Compileroption SOURCE

Mit der Compileroption SOURCE kann man dem Compiler den Eingabeort des Quellprogramms mitteilen.

Es gilt folgende Voreinstellung:
Fehlt die Compileroption SOURCE oder wird sie ohne Optionswert angegeben, dann wird das Quellprogramm über SYSDTA gelesen.



* Das Quellprogramm wird von SYSDTA gelesen. Standardmäßig gilt SOURCE=*.
Wird das Quellprogramm direkt eingegeben, dann muß die Eingabe mit den Zeichen "/*" in den Spalten 1 und 2 beendet werden (siehe Abschnitt 3.1.2).

datei Name einer katalogisierten Datei, in der das Quellprogramm steht. Maximale Länge einschließlich Katalog- und Benutzerkennung: 54 Zeichen.

bibl(name)

bibl
Name einer PLAM-Bibliothek oder einer GAM-Datei, aus der das Quellprogramm gelesen wird. *bibl* wird zuerst als PLAM-Bibliothek und dann als GAM-Datei interpretiert. *bibl* darf einschließlich Katalog- und Benutzerkennung aus maximal 54 Zeichen bestehen.

name
Name eines PLAM-Bibliothekselements oder Gruppenschlüssel einer GAM-Datei.
- PLAM-Bibliothekselement: *name* darf aus maximal 8 Zeichen bestehen.
- Gruppenschlüssel: *name* darf aus maximal 12 alphanumerischen Zeichen bestehen und muß kürzer als der Gruppenschlüssel sein.

plamspezifikation

Spezifikation eines PLAM-Bibliothekselements

```
[*LIBRARY-ELEMENT] ([LIBRARY=]plamlib,
                        [ELEMENT=]name[([VERSION=]{*HIGHEST-EXISTING})])
                                                  {version           }
```

plamlib
Name einer PLAM-Bibliothek. Maximale Länge einschließlich Katalog- und Benutzerkennung: 54 Zeichen.

name
Name eines PLAM-Bibliothekselements. Der Name darf aus maximal 64 Zeichen bestehen.

version
Versionsbezeichnung des PLAM-Bibliothekselements. Eine Versionsbezeichnung kann maximal 24 Zeichen lang sein.

*HIGHEST-EXISTING
Wenn mehrere Versionen eines Elements vorhanden sind, liest der Compiler standardmäßig die höchste vorhandene Version.

/ Nachdem der Compiler alle Compileroptionen gelesen hat, wird in den Systemmodus unterbrochen. Der Compiler gibt folgende Meldung aus:

```
FOR1: ASSIGN SYSDTA TO READ SOURCE
```

SYSDTA kann nun mit dem Kommando ASSIGN-SYSDTA neu zugewiesen werden. Nachdem der Anwender das Kommando RESUME-PROGRAM eingegeben hat, liest der Compiler das Quellprogramm über SYSDTA und beginnt mit der Übersetzung.

+ Nachdem der Compiler alle Compileroptionen eingelesen hat, gibt er folgende Meldung aus:

```
FOR1: GIVE SOURCE FILE SPECIFICATION - OR ?
```

Nun kann der Anwender über die primäre Zuordnung von SYSDTA - im Dialog die Datensichtstation - den Datenort des Quellprogramms eingeben.

Als Datenort kann angegeben werden:

```
{*                  }
{(SYSCMD)           }
{(PRIMARY)          }
{(CARD)             }
{datei              }
{bibl(name)         }
{plamspezifikation  }
```

*
Das Quellprogramm wird von SYSDTA gelesen. SYSDTA wird nicht neu zugeordnet. Wurde mit dem Kommando ASSIGN-SYSDTA ein Datenort angegeben, dann wird das Quellprogramm von dort gelesen.

(SYSCMD)
Die Systemdatei SYSDTA wird mit SYSCMD zusammengeschaltet. Das Quellprogramm wird von SYSCMD gelesen.

(PRIMARY)
Die Systemdatei SYSDTA wird auf ihre Primärzuweisung zurückgelegt. Das Quellprogramm wird von der Primärzuweisung von SYSDTA gelesen, d.h. von der Datensichtstation.

(CARD)
Der Systemdatei SYSDTA wird ein Kartenleser zugeordnet. Das Quellprogramm wird vom Kartenleser gelesen.

datei
Name der Datei, von der das Quellprogramm gelesen werden soll (Beschreibung siehe oben)

bibl(name)
Spezifikation des PLAM-Bibliothekselements oder Abschnitt der GAM-Datei, von der das Quellprogramm gelesen werden soll (Beschreibung siehe oben)

plamspezifikation
Spezifikation des PLAM-Bibliothekselements, von dem das Quellprogramm gelesen werden soll (Beschreibung siehe oben)

SOURCE=(PRIMARY)
Das Quellprogramm wird von der Datensichtstation im WRITE/READ-Modus eingegeben, unabhängig von der SYSDTA-Zuweisung. Die Angabe ist nur in Verbindung mit COMOPT DIALOG wirksam (siehe Abschnitt 3.6) und ist notwendig, wenn während des Ablaufs einer Prozedur ein Quellprogramm im Dialogmodus geschrieben werden soll.

Hinweis:

Nur bei Gruppendateien (GAM-Dateien, siehe Beispiel 3) dürfen Quellprogramme und Änderungszeilen in derselben Datei stehen.

Beispiele:

Beispiel 1: Einlesen von katalogisierter Datei

Einlesen des Quellprogramms von der katalogisierten Datei QUELL.MAT

```
/START-PROG $FOR1
*COMOPT SOURCE=QUELL.MAT
*END
```

Beispiel 2: Einlesen von Bibliothek

Einlesen des Quellprogramms von der Bibliothek PROGLIB. Name des Elements: PROGA

```
/START-PROG $FOR1
*COMOPT SOURCE=PROGLIB(PROGA)
*END
```

Beispiel 3: Einlesen von Gruppendatei (GAM-Datei)

Die Gruppendatei QUELL.MAT enthält drei Quellprogramme:

```
GGD     (Satzschlüssel 19010000-19900000)
INV     (Satzschlüssel 20010000-20570000)
MAT1    (Satzschlüssel 21010000-21500000)
```

Gruppendateien sind immer vom Typ ISAM. Die Satzschlüssel können im EDT mit der SET-Anweisung (siehe "EDT-Anweisungen" [20]) erzeugt werden.

Das Programm INV besteht aus genau den Zeilen, deren Schlüssel mit der Ziffernfolge 20 beginnt; dieses Programm stellt eine Gruppendatei dar.

Mit den folgenden Anweisungen kann das Programm INV übersetzt werden:

```
/START-PROG $FOR1
*COMOPT SOURCE=QUELL.MAT(20)
*END
```

Der Compiler berücksichtigt aus der Gruppendatei QUELL.MAT nur die Sätze, deren Schlüssel mit 20 beginnen. Führende Nullen sind anzugeben.

Beispiel 4: Einlesen eines PLAM-Bibliothekselements

Mit Hilfe des LMS wird ein FORTRAN-Quellprogramm als PLAM-Bibliothekselement vom Typ S abgelegt. Anschließend wird der FOR1 aufgerufen. In der SOURCE-Option wird das PLAM-Bibliothekselement ELEM1 der Version 001 angegeben.

```
/START-PROG $LMS
$LIB BIBL,BOTH,ANY
$ADDS QUELLE.TEST>ELEM1/001
$END

/START-PROG $FOR1
*COMOPT SOURCE = (BIBL,ELEM1(001)), END
```

Beispiel 5: Compileroptionen und Quellprogramm in verschiedenen Dateien

Die Optionen für die Übersetzung eines Programms stehen auf einer Datei OPT. Der Ort des Quellprogramms soll erst bei Abarbeitung der Optionen angegeben werden. Das Quellprogramm steht auf der Datei QUELL.MAT.

1. Eingabeort des Quellprogramms anfordern mit `COMOPT SOURCE=+`

   Inhalt der Datei OPT:

   ```
   COMOPT SOURCE=+
   COMOPT LIST=(ALL)
   END
   ```

   Übersetzung:

   ```
   /ASS-SYSDTA TO-FILE=OPT
   /START-PROG $FOR1
   FOR1: GIVE SOURCE FILE SPECIFICATION-OR?
   QUELL.MAT
   ```

   In diesem Fall kann der Anwender Informationen über die Eingabemöglichkeiten anfordern, indem er nach der Eingabeaufforderung ein Fragezeichen eingibt.

2. Eingabeort des Quellprogramms anfordern mit `COMOPT SOURCE=/`

   Inhalt der Datei OPT:

   ```
   COMOPT SOURCE=/
   COMOPT LIST=(ALL)
   END
   ```

   Übersetzung:

   ```
   /ASS-SYSDTA TO-FILE=OPT
   /START-PROG $FOR1
   FOR1: ASSIGN SYSDTA TO READ SOURCE
   /ASS-SYSDTA TO-FILE=QUELL.MAT
   /RESUME-PROG
   ```

3.3 Festlegen des Eingabeorts der Compileroptionen

Der Anwender gibt die Compileroptionen in einer oder mehreren COMOPT-Anweisungen an. Es gibt folgende Möglichkeiten, Compileroptionen einzugeben:

- Der Anwender kann die COMOPT-Anweisung(en) direkt eingeben, indem er den Compiler aufruft, ohne vorher mit dem Kommando ASSIGN-SYSDTA die Systemdatei SYSDTA umzuweisen. Der Compiler fordert die Compileroptionen explizit an, indem er einen Stern (*) in Spalte 1 vorgibt.
- Der Anwender kann die COMOPT-Anweisung(en) in eine Datei schreiben und über diese Datei eingeben. Diese Datei kann die Quellprogrammdatei sein oder eine eigene Datei. Mit dem Kommando ASSIGN-SYSDTA wird diese Datei vor Aufruf des Compilers der Systemdatei SYSDTA zugeordnet. Der Compiler liest von der Systemdatei SYSDTA. Compileroptionen, die in einer Quellprogrammdatei stehen, müssen vor dem Quellprogrammtext stehen.
- Der Anwender kann den Eingabeort der COMOPT-Anweisung(en) in der Compileroption OPTIONS angeben (siehe Abschnitt 3.3.2) Für die Compileroption OPTIONS gibt es keinen entsprechenden SDF-Operanden.

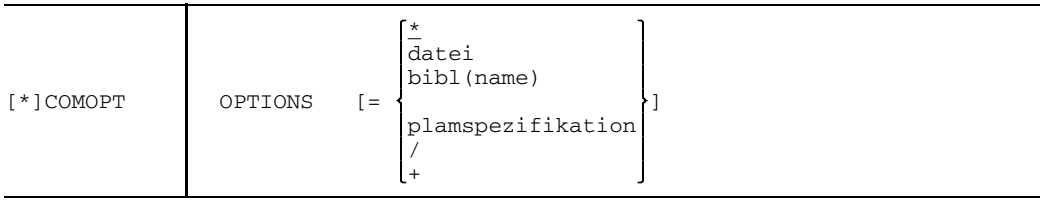
3.3.1 Kommando ASSIGN-SYSDTA

Das Kommando ASSIGN-SYSDTA ist in Abschnitt 3.2.1, "Kommando ASSIGN-SYSDTA" beschrieben.

3.3.2 Compileroption OPTIONS

Mit der Compileroption OPTIONS wird dem Compiler der Eingabeort der Compileroptionen mitgeteilt. Für die Compileroption OPTIONS gibt es keinen entsprechenden SDF-Operanden.

Es gilt folgende Voreinstellung:
Fehlt die Compileroption OPTIONS oder wird sie ohne Optionswert angegeben, dann werden die Compileroptionen von SYSDTA gelesen.



* Die Compileroptionen werden von SYSDTA gelesen.

datei Name einer katalogisierten Datei, in der die Compileroptionen stehen. Maximale Länge einschließlich Katalog- und Benutzerkennung: 54 Zeichen.

bibl(name)

bibl
Name einer PLAM-Bibliothek oder einer GAM-Datei, aus der die Compileroptionen gelesen werden. *bibl* wird zuerst als PLAM-Bibliothek und dann als GAM-Datei interpretiert. *bibl* darf einschließlich Katalog- und Benutzerkennung aus maximal 54 Zeichen bestehen.

name
Name eines PLAM-Bibliothekselements oder Gruppenschlüssel einer GAM-Datei.
- PLAM-Bibliothekselement: *name* darf aus maximal 8 Zeichen bestehen.
- Gruppenschlüssel: *name* darf aus maximal 12 alphanumerischen Zeichen bestehen und muß kürzer als der Gruppenschlüssel sein.

plamspezifikation

Spezifikation eines PLAM-Bibliothekselements

```
[*LIBRARY-ELEMENT] ([LIBRARY=]plamlib,
                    [ELEMENT=]name[([VERSION=]{*HIGHEST-EXISTING})])
                                              {version           }
```

plamlib
Name einer PLAM-Bibliothek. Maximale Länge einschließlich Katalog- und Benutzerkennung: 54 Zeichen.

name
Name eines PLAM-Bibliothekselements. Maximale Länge: 64 Zeichen.

version
Versionsbezeichnung des PLAM-Bibliothekselements. Eine Versionsbezeichnung kann maximal 24 Zeichen lang sein.

*HIGHEST-EXISTING
Wenn mehrere Versionen eines Elements vorhanden sind, liest der Compiler standardmäßig die höchste vorhandene Version.

/ Nachdem der Compiler alle Compileroptionen gelesen hat, wird in den Systemmodus unterbrochen und folgende Meldung ausgegeben:

```
FOR1: ASSIGN SYSDTA TO READ OPTION
```

SYSDTA kann nun mit dem Kommando ASSIGN-SYSDTA neu zugewiesen werden. Nachdem der Anwender das Kommando RESUME-PROGRAM eingegeben hat, liest der Compiler die Compileroptionen von SYSDTA und beginnt mit der Übersetzung.

+ Nachdem der Compiler alle Compileroptionen gelesen hat, gibt er folgende Meldung aus:

```
FOR1: GIVE OPTION FILE SPECIFICATION - OR ?
```

Nun kann der Anwender über die primäre Zuordnung von SYSDTA - im Dialog die Datensichtstation - den Datenort der Compileroptionen eingeben.

Als Datenort kann angegeben werden:

```
⎧*                  ⎫
⎪(SYSCMD)           ⎪
⎪(PRIMARY)          ⎪
⎨(CARD)             ⎬
⎪datei              ⎪
⎪bibl(name)         ⎪
⎩plamspezifikation  ⎭
```

*
Die Compileroptionen werden von SYSDTA gelesen. SYSDTA wird nicht neu zugeordnet. Wurde mit dem ASSIGN-SYSDTA-Kommando ein Datenort angegeben, dann werden die Compileroptionen von dort gelesen.

(SYSCMD)
Die Systemdatei SYSDTA wird mit SYSCMD zusammengeschaltet. Die Compileroptionen werden von SYSCMD gelesen.

(PRIMARY)
Die Systemdatei SYSDTA wird auf ihre Primärzuweisung zurückgelegt. Die Compileroptionen werden von der Primärzuweisung von SYSDTA gelesen, d.h. von der Datensichtstation.

(CARD)
Der Systemdatei SYSDTA wird ein Kartenleser zugeordnet. Die Compileroptionen werden vom Kartenleser gelesen.

datei
Name einer Datei, von der die Compileroptionen gelesen werden sollen (Beschreibung siehe oben)

bibl(name)
Spezifikation eines Bibliothekselements oder Abschnitt einer GAM-Datei, von der die Compileroptionen gelesen werden sollen (Beschreibung siehe oben)

plamspezifikation
Spezifikation eines PLAM-Bibliothekselements, von dem die Compileroptionen gelesen werden sollen (Beschreibung siehe oben)

Beispiele:

Beispiel 1: Compileroptionen von katalogisierter Datei

Einlesen der Compileroptionen von der katalogisierten Datei OPT. Das Quellprogramm steht in der katalogisierten Datei QUELLE.

Inhalt von OPT:

```
COMOPT SOURCE=QUELLE
COMOPT LIST=(ALL)
END
```

Übersetzung:

```
/START-PROG $FOR1
*COMOPT OPTIONS=OPT
*END
```

Beispiel 2: Compileroptionen von Bibliothek

Einlesen der Compileroptionen von der Bibliothek PROGLIB, Name des Elements: OPT. Inhalt von OPT siehe Beispiel 1.

```
/START-PROG $FOR1
*COMOPT OPTIONS=PROGLIB(OPT)
*END
```

Beispiel 3: Anfordern von Compileroptionen bei der Übersetzung

Die Optionen für die Übersetzung eines Programms stehen in der Datei OPT. Sie werden von SYSDTA eingelesen. Weitere Compileroptionen stehen in der Datei OPT1, der Ort dieser Compileroptionen soll erst bei Abarbeitung der Optionen angegeben werden. Das Quellprogramm steht in der Datei QUELLE.

Inhalt der Datei OPT1:

```
COMOPT LIST=(ALL)
END
```

1. Eingabeort der Compileroptionen anfordern mit `COMOPT OPTIONS=+`

   Inhalt der Datei OPT:

   ```
   COMOPT SOURCE=QUELLE
   COMOPT OPTIONS=+
   END
   ```

   Übersetzung:

   ```
   /ASS-SYSDTA TO-FILE=OPT
   /START-PROG $FOR1
   FOR1:GIVE OPTION FILE SPECIFICATION - OR ?
   OPT1
   ```

   In diesem Fall kann der Anwender Informationen über die Eingabemöglichkeiten anfordern, indem er nach der Eingabeaufforderung ein Fragezeichen eingibt.

2. Eingabeort der Compileroptionen anfordern mit `COMOPT OPTIONS=/`

   Inhalt der Datei OPT:

```
COMOPT SOURCE=QUELLE
COMOPT OPTIONS=/
END
```

   Übersetzung:

```
/ASS-SYSDTA TO-FILE=OPT
/START-PROG $FOR1
FOR1: ASSIGN SYSDTA TO READ OPTION
/ASS-SYSDTA TO-FILE=OPT1
/RESUME-PROG
```

3.4 Temporäres Ändern eines Quellprogramms: Compileroption UPD

Der Text eines Quellprogramms kann mit Änderungszeilen für den Ablauf eines Übersetzungsvorgangs geändert werden. Die Änderungen werden in der Quellprogrammliste dokumentiert, haben aber keinen Einfluß auf die Quellprogrammdatei.
Der Eingabeort der Änderungszeilen wird in der Compileroption UPD angegeben. Für die Compileroption UPD gibt es keinen entsprechenden SDF-Operanden.

Voraussetzung

Voraussetzung für das temporäre Ändern eines Quellprogramms ist eine Kennung der Quellprogramm- und Änderungszeilen in den Spalten 73 bis 80. Die Kennung muß aus acht EBCDIC-Zeichen bestehen. Dabei müssen die Kennungen des Quellprogramms und der Änderungszeilen in aufsteigender Reihenfolge vorkommen. Der Stellenwert der einzelnen Zeichen ergibt sich aus der Position des Zeichens im EBCDIC-Zeichenvorrat (A hat niedrigeren Stellenwert als 1). Die Änderungszeilen werden den Kennungen entsprechend in das Quellprogramm eingefügt oder ersetzen die Zeilen mit gleicher Kennung. Sollen mehrere Änderungszeilen hintereinander eingefügt werden, braucht nur die erste Änderungszeile eine entsprechende Kennung aufzuweisen, die nachfolgenden Zeilen ohne Kennung werden unmittelbar dahinter eingesetzt.

*Temporäres Löschen von Quellprogrammzeilen: *DELETE-Anweisung*

Sollen bestimmte Zeilen im Quellprogramm überlesen werden, so kann in den Änderungszeilen folgende Anweisung angegeben werden:

```
*DELETE k1[,k2]
```

k1, k2 achtstellige Kennung, k1 ≤ k2

Alle Zeilen in dem angegebenen Bereich (k1 bis k2) werden überlesen und nicht mehr in der Quellprogrammliste ausgedruckt.

Diese Anweisung muß in Spalte 1 beginnen. In Spalte 8 muß ein Leerzeichen stehen. Die Kennung k1 muß in der Spalte 9 beginnen. Wird die Kennung k2 angegeben, so muß das Komma in Spalte 17 stehen und die Kennung k2 in Spalte 18 beginnen. Im Bereich [k1,k2] dürfen keine Leerzeichen stehen.

Die *DELETE-Anweisung erscheint in der Quellprogrammliste als Kommentarzeile.

*Beispiel: Änderungszeilen mit *DELETE-Anweisung*

Quellprogrammdatei Spalte 73-80

```
      PROGRAM UPDATE                                    AAAA1000
      B=8.                                              AAAA2000
      D=10.                                             AAAA3000
      CONTINUE                                          AAAA4000
      A=B+2.                                            AAAA5000
      C=D/A                                             AAAA6000
      CONTINUE                                          AAAA7000
      A=B/(C*D)                                         AAAA7010
      WRITE(2,*) C                                      AAAA7020
      WRITE(2,*) D                                      AAAA7030
      WRITE(2,*) A                                      AAAA7040
      END                                               AAAA8000
```

Änderungszeilen

```
           WRITE(2,10) A                                AAAA6030
     10    FORMAT(11X,'NENNER = ',F8.2)                 AAAA6060
    *DELETE AAAA7020,AAAA7030                           AAAA6080
           WRITE(2,*) A,B,C,D                           AAAA7040
```

Daraus ergibt sich folgende Quellprogrammliste

```
           PROGRAM UPDATE                               AAAA1000
           B=8.                                         AAAA2000
           D=10.                                        AAAA3000
           CONTINUE                                     AAAA4000
           A=B+2.                                       AAAA5000
           C=D/A                                        AAAA6000
           WRITE(2,10) A                                AAAA6030
     10    FORMAT(11X,'NENNER = ',F8.2)                 AAAA6060
           CONTINUE                                     AAAA7000
           A=B/(C*D)                                    AAAA7010
    *DELETE AAAA7020,AAAA7030                           AAAA6080
           WRITE(2,*) A,B,C,D                           AAAA7040
           END                                          AAAA8000
```

Compileroption UPD

Mit der Compileroption UPD gibt man dem Compiler bekannt, von wo er Änderungszeilen lesen soll. Für die Compileroption UPD gibt es keinen entsprechenden SDF-Operanden.

Es gilt folgende Voreinstellung:

- Fehlt die Compileroption UPD, dann wird angenommen, daß keine Änderungszeilen vorhanden sind.
- Wird die Compileroption UPD ohne Operandenwert angegeben, dann werden die Änderungszeilen von der Datei mit dem LINK-Namen FUPDLINK gelesen. Die Änderungsdatei muß in diesem Fall vor Aufruf des FOR1 mit folgendem Kommando zugewiesen werden:

```
/SET-FILE-LINK LINK-NAME=FUPDLINK,FILE-NAME=dateiname
```

`[*]COMOPT`	`UPD [= {*` / `datei` / `bibl(name)` / `/` / `+}]`

*
: Die Änderungszeilen werden von SYSDTA gelesen.

datei
: Name einer katalogisierten Datei, in der die Änderungszeilen stehen. Maximale Länge einschließlich Katalog- und Benutzerkennung: 54 Zeichen.

bibl(name)

bibl
: Name einer LMS-Bibliothek im OSM-Format, aus der die Änderungszeilen gelesen werden. *bibl* wird zuerst Bibliothek und dann als GAM-Datei interpretiert. *bibl* darf einschließlich Katalog- und Benutzerkennung aus maximal 54 Zeichen bestehen.

name
: Name eines Bibliothekselements oder Gruppenschlüssel einer GAM-Datei.
  - Bibliothekselement: *name* darf aus maximal 8 Zeichen bestehen.
  - Gruppenschlüssel: *name* darf aus maximal 12 alphanumerischen Zeichen bestehen und muß kürzer als der Gruppenschlüssel sein.

/ Nachdem der Compiler alle Compileroptionen gelesen hat, wird in den Systemmodus unterbrochen. Der Compiler gibt folgende Meldung aus:

```
FOR1: ASSIGN SYSDTA TO READ UPDATE
```

SYSDTA kann nun mit dem Kommando ASSIGN-SYSDTA neu zugewiesen werden. Nachdem der Anwender das Kommando RESUME-PROGRAM eingegeben hat, liest der Compiler die Änderungszeilen über SYSDTA und beginnt mit der Übersetzung.

\+ Nachdem der Compiler alle Compileroptionen gelesen hat, gibt er folgende Meldung aus:

```
FOR1: GIVE UPDATE FILE SPECIFICATION-OR?
```

Über die primäre Zuordnung von SYSDTA - im Dialog die Datensichtstation - kann der Datenort der Änderungszeilen eingegeben werden.

Als Datenort kann angegeben werden:

```
*
(SYSCMD)
(PRIMARY)
(CARD)
datei
bibl(name)
```

*

Die Änderungszeilen werden von SYSDTA gelesen. SYSDTA wird nicht neu zugeordnet. Wurde mit dem ASSIGN-SYSDTA-Kommando ein Datenort angegeben, dann werden die Änderungszeilen von dort gelesen.

(SYSCMD)
Die Systemdatei SYSDTA wird mit SYSCMD zusammengeschaltet. Die Änderungszeilen werden von SYSCMD gelesen.

(PRIMARY)
Die Systemdatei SYSDTA wird auf ihre Primärzuweisung zurückgelegt. Die Änderungszeilen werden von der Primärzuweisung von SYSDTA gelesen, d.h. von der Datensichtstation.

(CARD)
Der Systemdatei SYSDTA wird ein Kartenleser zugeordnet. Die Änderungszeilen werden vom Kartenleser gelesen.

datei
Name der Datei, von der die Änderungszeilen gelesen werden sollen (Beschreibung siehe oben).

bibl(name)
Spezifikation eines Bibliothekselements oder Abschnitt einer GAM-Datei, von der die Änderungszeilen gelesen werden sollen (Beschreibung siehe oben).

Einschränkungen:

- Die gleichzeitige Verwendung von COMOPT UPD und %INCLUDE ist unzulässig
- COMOPT UPD und COMOPT DIALOG dürfen nicht gleichzeitig verwendet werden.
- Nur bei Gruppendateien dürfen Quellprogramm und Änderungszeilen in derselben Datei stehen.
- Wird die Compileroption SOURCE-FORMAT=FREE zusammen mit der Compileroption UPD angegeben, dann erfolgt eine Fehlermeldung. Die zuletzt angegebene Compileroption gilt.

Beispiele:

Beispiel 1: Quellprogramm von Datei, Änderungszeilen von Datei

Das Quellprogramm wird von der Datei QUELL.MAT gelesen. Änderungszeilen stehen in der Datei UP mit dem vorgegebenen LINK-Namen FUPDLINK.

```
/SET-FILE-LINK LINK-NAME=FUPDLINK,FILE-NAME=UP
/ASS-SYSDTA *SYSCMD
/START-PROG $FOR1
*COMOPT UPD
*COMOPT SOURCE=/
*END
FOR1: ASSIGN SYSDTA TO READ SOURCE
/ASS-SYSDTA TO-FILE=QUELL.MAT
/RESUME-PROG
```

Beispiel 2: Quellprogramm von Datei, Änderungszeilen von Bibliothek

Das Quellprogramm wird von der Datei QUELL.MAT gelesen, Änderungszeilen stehen in der Datei ELEMUP der Bibliothek LIB. Die Bibliothek LIB ist eine LMS-Bibliothek im OSM-Format.

```
/ASS-SYSDTA *SYSCMD
/START-PROG $FOR1
*COMOPT SOURCE=QUELL.MAT
*COMOPT UPD=+
*END
FOR1: GIVE UPDATE FILE SPECIFICATION-OR?
LIB(ELEMUP)
```

3.5 Einfügen von Quellprogrammzeilen

3.5.1 %INCLUDE-Anweisung

Durch die Compiler-Steueranweisung %INCLUDE werden Quellprogrammzeilen in ein Quellprogramm eingefügt. Die %INCLUDE-Anweisung wird wie eine FORTRAN-Anweisung in den Quellprogrammtext geschrieben. Während des Übersetzungslaufs wird der durch die %INCLUDE-Anweisung spezifizierte Quellprogrammtext an der Stelle eingefügt, an der die %INCLUDE-Anweisung steht. Die %INCLUDE-Anweisung ist nur in der Programmeinheit gültig, in der sie angegeben wird. Der mit %INCLUDE einzufügende Quellprogrammtext kann vor dem Einsetzen in das Quellprogramm verändert werden.

Das Einfügen von Textstellen kann sich über mehrere Ebenen erstrecken, d.h. in einem eingefügten Text können wieder %INCLUDE-Anweisungen stehen. Die Ausgabe des eingefügten Quellprogrammtexts kann durch die Compileroption EXPAND (siehe Abschnitt 4.6.2.8) oder die %EXPAND-Anweisung (siehe Abschnitt 4.6.3.3) gesteuert werden.

```
%INCLUDE  { name       }[,CODE={EBCDIC}][,'az1'='nz1'][,'az2'='nz2'][...]
          { bibl(name) }       { ISO    }
                               { BCD    }
```

name Name eines PLAM-Bibliothekselements vom Typ S, das den einzufügenden Quellprogrammtext enthält. Wird *name* angegeben, dann wird zuerst geprüft, ob ein PLAM-Bibliothekselement dieses Namens vorliegt. Der Zugriff erfolgt nach der Hierarchie von PLAM-Bibliotheken, die im SDF-Operanden INCLUDE-LIBRARY bzw. in der Compileroption INCLUDE festgelegt wurde.

oder

Dateiname einer katalogisierten Datei, die den einzufügenden Quellprogrammtext enthält.

bibl(name)

bibl ist der Name einer PLAM-Bibliothek. Das Bibliothekselement *name* dieser Bibliothek enthält den einzufügenden Quellprogrammtext.

oder

bibl ist der Name einer Gruppendatei (GAM-Datei). Der Gruppenschlüssel *name* kennzeichnet den einzufügenden Quellprogrammtext (siehe Abschnitt 3.2.3, Beispiel 4).

Zunächst wird *bibl* als PLAM-Bibliothek, dann als GAM-Datei aufgefaßt. Eine durch SDF-Operand INCLUDE-LIBRARY bzw. Compileroption INCLUDE festgelegte Suchhierarchie unter den PLAM-Bibliotheken wird bei Angabe von *bibl* nicht berücksichtigt.

CODE Standardmäßig wird angenommen, daß das Quellprogramm im EBCDIC-Code vorliegt. Ein Umsetzen in den ISO- oder den BCD-Code ist möglich.

'az1'='nz1' ['az2'='nz2'] [...]
az : alte Zeichenkette
nz : neue Zeichenkette

Der einzufügende Quellprogrammtext wird vor dem Einsetzen in das Quellprogramm nach den Angaben in dieser Liste geändert. Der Quellprogrammtext wird parallel nach *alte Zeichenkette1* und *alte Zeichenkette2* durchsucht. *alte Zeichenkette1* wird durch *neue Zeichenkette1* ersetzt, *alte Zeichenkette2* durch *neue Zeichenkette2*. Leerzeichen werden beim Suchen und Ersetzen beachtet. Verlängert sich der Quellprogrammtext durch das Ersetzen, so werden bei Bedarf Folgesätze erzeugt.

In der Optionenliste werden nur die explizit in der Compileroption INCLUDE (und nicht die in der %INCLUDE-Anweisung) genannten Namen aufgeführt.

Hinweise:

- Die Anweisungen PROGRAM, SUBROUTINE, FUNCTION, type FUNCTION, BLOCK DATA, END werden ignoriert, wenn sie in einem durch %INCLUDE eingefügten Text vorkommen. Es ist daher nicht möglich, eine oder mehrere Programmeinheiten durch ein %INCLUDE einzufügen.
- Das %INCLUDE-Schlüsselwort darf keine Leerzeichen enthalten.
- Länge von *alte Zeichenkette*: 1 ≤ Länge ≤ 72.
- Länge von *neue Zeichenkette*: 0 ≤ Länge ≤ 32767.
- Reicht *alte Zeichenkette* über eine Satzgrenze, dann wird *alte Zeichenkette* im Text nicht erkannt.
- Schachtelungstiefe der %INCLUDE-Anweisungen ≤ 32767.
- Erstreckt sich eine %INCLUDE-Anweisung über mehrere Zeilen, dann dürfen keine Kommentar- oder Leerzeilen dazwischen stehen.

Beispiel:

Im Programm A werden durch die %INCLUDE-Anweisung Quellprogrammzeilen eingefügt, die in der Datei B stehen. Das INCLUDE-Element B enthält das INCLUDE-Element C, dessen Quellprogrammzeilen vor dem Einsetzen verändert werden. Das INCLUDE-Element C enthält das INCLUDE-Element D.

Programm A:

```
      PROGRAM A
C       DIE INCLUDE-DATEI B
C       WIRD EINGEFUEGT
      %INCLUDE B
      END
```

Datei B:

```
C       TYPDEKLARATION IN
C       DER DATEI B
      REAL M,N
C       INCLUDE-DATEI C
C       WIRD EINGEFUEGT
      %INCLUDE C,'N=3.'='N=5.'
```

Datei C:

```
C       BERECHNUNG IN DER
C       INCLUDE-DATEI C
      M=2.
      N=3.
      X=M*N
C       INCLUDE-DATEI D
C       WIRD EINGEFUEGT
      %INCLUDE D
```

Datei D:

```
C       ERGEBNISAUSGABE IN
C       DER DATEI D
      WRITE*,'ERGEBNIS=',X
```

In der Quellprogrammliste werden die Quellprogrammzeilen des Programms A und alle durch %INCLUDE-Anweisungen eingefügten Quellprogrammzeilen ausgedruckt. Die Quellprogrammzeile 'N=3.' im INCLUDE-Element C wurde zu 'N=5.' geändert. Die Schachtelungstiefe der durch die %INCLUDE-Anweisungen eingefügten Quellprogrammteile wird in der Spalte I der Quellprogrammliste angezeigt (siehe Abschnitt 4.7.2).

```
****  SOURCE  LISTING  ****   SIEMENS-NIXDORF FORTRAN COMPILER    FOR1 V2.2A00DATE = ...
                                                 PROGRAM UNIT: A
DO/IF SEG  STMT  I/H LINE     SOURCE-TEXT

-------------------------------------------------------------------------------

       1/1     1         1  |   PROGRAM A
                           2  |C   DIE INCLUDE-DATEI B
                           3  |C   WIRD EINGEFUEGT
         1     2         4  |   %INCLUDE B
                    1/0    1  |C   TYPDEKLARATION IN
                    1/0    2  |C   DER DATEI B
         1     3  1/0    3  |   REAL M,N
                    1/0    4  |C   INCLUDE-DATEI C
                    1/0    5  |C   WIRD EINGEFUEGT
         1     4  1/0    6  |   %INCLUDE C,'N=3.'='N=5.'
                    2/0    1  |C   BERECHNUNG IN DER
                    2/0    2  |C   INCLUDE-DATEI C
         1     5  2/0    3  |   M=2.
         1     6  2/0    4  |   N=5.
         1     7  2/0    5  |   X=M*N
                    2/0    6  |C   INCLUDE-DATEI D
                    2/0    7  |C   WIRD EINGEFUEGT
         1     8  2/0    8  |   %INCLUDE D
                    3/0    1  |C   ERGEBNISAUSGABE IN
                    3/0    2  |C   DER DATEI D
         1     9  3/0    3  |   WRITE *,'ERGEBNIS=',X
         1    10         5  |   END
```

3.5.2 SDF-Operand INCLUDE-LIBRARY

Der SDF-Operand INCLUDE-LIBRARY legt den Zugriff auf eine Anzahl von Bibliotheken hierarchisch fest. Die Bibliotheken werden in der angegebenen Reihenfolge nach dem Element durchsucht, das in der %INCLUDE-Anweisung angegeben wurde.

```
START-FOR1-COMPILER
```

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

Eine Gegenüberstellung von SDF-Operanden und entsprechenden Compileroptionen enthält Tab. 2-3.

3.5.3 Compileroption INCLUDE-LIBRARY

Die Compileroption INCLUDE-LIBRARY legt den Zugriff auf eine Anzahl von Bibliotheken hierarchisch fest. Die Bibliotheken werden in der angegebenen Reihenfolge nach dem Element durchsucht, das in der %INCLUDE-Anweisung angegeben wurde.

[*] COMOPT	INCLUDE [-LIBRARY] = { *NO / dateiname / (dateiname1[,dateiname2][,...]) }

*NO Es wird angenommen, daß keine Suchhierarchie besteht. In diesem Fall wird der BS2000-Katalog durchsucht.

dateiname1[,dateiname2][,...]
Namen von PLAM-Bibliotheken oder GAM-Dateien, die %INCLUDE-Elemente enthalten. Die Reihenfolge der Bibliotheksnamen legt die Suchhierarchie fest. Bis zu 10 Bibliotheksnamen dürfen angegeben werden. Als letzte Hierarchieebene wird der BS2000-Katalog durchsucht. Wird in der %INCLUDE-Anweisung ein Bibliothekselement mit *bibl(name)* angegeben, dann werden keine Bibliotheken nach der mit der INCLUDE-Option festgelegten Hierarchie durchsucht.

3.6 Eingeben des Quellprogramms mit Interaktiver Analyse

Die Interaktive Analyse ermöglicht die Erstellung formal fehlerfreier Programme im zeilenweisen Dialog über nicht-formatierte Bildschirme. Diese Arbeitsweise wird im folgenden mit "Dialogmodus" bezeichnet. Im Gegensatz dazu ist im folgenden mit "Batchmodus" jene Arbeitsweise gemeint, bei welcher der Anwender für die gesamte Dauer der Übersetzung in die Compiler-Analyse nicht eingreifen kann.

Im Dialogmodus kann ein vom Compiler erkannter Fehler sofort interaktiv behoben werden, mit der korrigierten Anweisung kann der Übersetzungslauf dann fortgesetzt werden. Mit "Update" sind in diesem Abschnitt Änderungen am Quellprogramm während der Übersetzungsunterbrechung gemeint.

Im Vergleich zum Batchmodus hat der Dialogmodus folgende Vorzüge:

- kein Warten auf das Ende der Übersetzung,
- Auswertung von Compilerlisten kann entfallen,
- kein EDITOR nötig, um Fehler im Quellprogramm zu korrigieren,
- keine erneute vollständige Übersetzung des geänderten Quellprogramms (bzw. der Programmeinheit); richtige Programmteile werden nicht mehrfach übersetzt,
- das korrigierte Quellprogramm steht auf Wunsch am Ende der Übersetzung in einer ISAM-Datei zur Verfügung (Option DIALOG-SAVE,OUTPUT).

Die Originaldatei bleibt während der Interaktiven Analyse unverändert. Für notwendige Fehlerkorrekturen verwendet der Compiler ausschließlich eine Kopie der Originaldatei; sie wird im folgenden als "Arbeitsdatei" bezeichnet. Löschen oder Umkatalogisieren der Originaldatei oder der Arbeitsdatei steht nach der Übersetzung im Ermessen des Anwenders.

Der Ablauf der Interaktiven Analyse wird mit Dialog-Kommandos gesteuert, im folgenden auch kurz "Kommandos" genannt.

Hinweis:

Die Interaktive Analyse meldet Fehler, wenn die Anweisungsreihenfolge im Programm nicht der vom Standard geforderten Reihenfolge entspricht (z.B. Verwendung einer Variablen vor ihrer Deklaration).

3.6.1 Steuern der Interaktiven Analyse: SDF-Operand DIALOG

Der SDF-Operand DIALOG dient zum Steuern der Interaktiven Analyse.

```
START-FOR1-COMPILER
```

```
,DIALOG = NO / YES(...)

   YES(...)
      ,DIALOG-INTERRUPT = AFTER-ANY-PROG-UNIT / ERRORS-ONLY

      ,SAVE-FILE = *NONE / *STD-NAME(...) / <full-filename 1..54>(...)/
                   *LIBRARY-ELEMENT(...)

         *STD-NAME(...)
          | INCLUDE-EXPANSION = NO / YES

         <full-filename 1..54>(...)
          | INCLUDE-EXPANSION = NO / YES

         *LIBRARY-ELEMENT(...)
          | INCLUDE-EXPANSION = NO / YES
          | LIBRARY = <full-filename 1..41>
          ,ELEMENT = <full-filename 1..54> (...)
            | VERSION = *UPPER-LIMIT / <alphanum-name 1..24>

      ,LOG-CHANGED-LINES = NO / YES
```

Eine Gegenüberstellung von SDF-Operanden und entsprechenden Compileroptionen enthält Tab. 2-5.

3.6.2 Starten der Interaktiven Analyse: Compileroption DIALOG

Zum Starten der Interaktiven Analyse wird der Compiler wie üblich aufgerufen mit `/START-PROGRAM $FOR1` (oder einem ggf. anderen Namen). Sodann wird der Dialogmodus eingeschaltet mit der Compileroption DIALOG.

```
            ⎧                ⎧param                  ⎫ ⎫
            ⎪ DIALOG      [= ⎨(param[,param])        ⎬]⎪
[*]COMOPT   ⎨                ⎩(param[,param[,param]])⎭ ⎬
            ⎪                                          ⎪
            ⎩ NODIALOG                                 ⎭
```

(NODIALOG ist unterstrichen.)

param:= { kommandopräfix | dialogsprache | editormodus }

kommandopräfix:= { ! | ? | @ | % | # | $ } (Voreinstellung: %)

dialogsprache:= { D | E } (Voreinstellung: E)

Ausgabe der Meldungstexte
- D in deutscher Sprache
- E in englischer Sprache

editormodus:= E[DIT] = { ALL | FIRST | NO } (Voreinstellung: ALL)

Vereinbarung, ob zu Beginn einer neuen Programmeinheit die Steuerung an den Anwender gehen soll, und zwar bei

ALL	vor jeder neuen Programmeinheit
FIRST	nur vor der ersten Programmeinheit
NO	nur im Fehlerfall

Entsprechend dem EDIT-Operanden wird die Interaktive Analyse mit folgender Meldung unterbrochen:

```
'FOR1: NEUE PROGRAMM-EINHEIT - GIB ANWEISUNG ODER @HELP'
```

Der Anwender kann z.B. Kommandos zur Dateibearbeitung oder mit %SYSTEM BS2000-Kommandos eingeben.

Hinweise:

- Voreinstellung bei COMOPT DIALOG: %, E, EDIT=ALL
- Voreinstellung bei SDF-Operand DIALOG: @, E, EDIT=ALL
- COMOPT DIALOG darf nicht gleichzeitig mit der Compileroption COMOPT UPD angegeben werden.

3.6.3 Ablaufskizzierung

Eingabe des Quellprogramms

Das Quellprogramm kann, gesteuert durch die Option SOURCE, auf unterschiedliche Art eingegeben werden:
- von einer Datei
- direkt an der Datensichtstation

Die direkte Eingabe des Quellprogramms an der Datensichtstation zeigt der Anwender an durch

- COMOPT SOURCE=*, fehlende SOURCE-Option oder SOURCE-Option ohne Wert, wenn die Primärzuweisung von SYSDTA die Datensichtstation ist;
- COMOPT SOURCE=(PRIMARY), wenn die Optionen von einer Datei gelesen werden, der SYSDTA zugeordnet wurde.

Der Compiler gibt die Zeilennummer vor. Sodann ist die gesamte Anweisung einzugeben. Zu beachten ist:

- Ein Semikolon als erstes Zeichen einer Eingabezeile wird nach der Eingabe durch 6 Leerzeichen ersetzt. Dadurch wird der auf das Semikolon folgende Text nach der Eingabe auf Spalte 7 positioniert.
- Eingabe nur von Folgezeilen führt notwendigerweise dazu, daß die aktuelle Programmeinheit nochmals übersetzt wird (Erweiterung einer FORTRAN-Anweisung!).
- Eingabe einer Zeilenkennung (Spalten 73-80) ist nicht erlaubt (hierfür: %SET- oder %INSERT-Kommando).
- Die Anweisungslänge ist durch die Bildschirmgröße beschränkt.
- Mehrere FORTRAN-Zeilen bzw. Dialog-Kommandos können auf einmal eingegeben werden, getrennt durch das Zeilenendesymbol für die jeweilige Datenstation (siehe Kommando MODIFY-TERMINAL-OPTIONS; "Benutzerkommandos (SDF-Format)" [12]).
- FORTRAN-Anweisungen, die länger als 72 Zeichen sind, werden vom FOR1-Compiler in das korrekte FORTRAN-Format mit Fortsetzungszeilen umgeformt.
- "/*" in den Spalten 1 und 2 kennzeichnet das Eingabeende.
- Das Quellprogramm steht dem Anwender nach der interaktiven Übersetzung auf Wunsch als Datei (WS) zur Verfügung; der Dateiname ist in der Option OUTPUT und/oder im Kommando %SAVE bzw. %WRITE anzugeben.
- Weitere Informationen: siehe Abschnitt 3.6.6, "Update der Arbeitsdatei".

Eingabe von Dialog-Kommandos

Dialog-Kommandos können eingegeben werden
- jederzeit während der direkten Programmeingabe an der Datenstation
- bei der Programmeingabe von einer Datei in Abhängigkeit vom EDIT-Operand: entweder vor der Übersetzung, nachdem das gesamte Quellprogramm in die Arbeitsdatei eingelesen ist (EDIT=FIRST) oder vor der Übersetzung jeder einzelnen Programmeinheit (EDIT=ALL).
- im "Update"-Zustand.

Ablauf der Interaktiven Analyse

Der Compiler liest das Quellprogramm in eine Arbeitsdatei. Er überprüft die einzelnen Anweisungen der Reihe nach.
Erkennt er einen syntaktischen Fehler, so zeigt er am Bildschirm die betreffende Anweisung, kennzeichnet die Fehlerposition und gibt einen Meldungstext aus. Der Compiler befindet sich jetzt im sogenannten "Update-Zustand", in dem der Anwender FORTRAN-Anweisungen oder Kommandos (zur Interaktiven Analyse) eingeben kann. Er hat damit folgende Möglichkeiten:

- Update der Arbeitsdatei (Fehlerkorrektur)
- Kein Update und Fortsetzung der Interaktiven Analyse bei der nächsten Anweisung bzw. Neubeginn der Analyse in der aktuellen Programmeinheit (Ignorieren bisheriger Korrekturen)
- Abbruch des Dialogmodus und Übergang in den Batchmodus.
- Abbruch der Übersetzung und/oder Ausgabe von Protokollen

Nach einer Korrektur setzt die Interaktive Analyse bei der betreffenden Anweisung auf, d.h. die Syntax dieser Anweisung wird erneut überprüft.

Ausgabe von Protokollen bzw. Dateien

Folgende Protokolle bzw. Dateien können wahlweise ausgegeben werden:
- Übersetzungsprotokoll (wie im Batchmodus)
- Änderungsprotokoll: Liste der geänderten, eingefügten oder gelöschten Zeilen, bezogen auf das Originalprogramm. Sie enthält auch die eingegebenen Kommandos zur Interaktiven Analyse
- Originalprogrammdatei (sinnvoll bei direkter Eingabe am Datensichtgerät): Bei direkter Eingabe vom Datensichtgerät entspricht die Ersteingabe dem Originalprogramm
- Arbeitsdatei (Include-Expansionen wahlweise)

Die Ausgabe steuern die Optionen LIST, LIST-OUTPUT, LISTFILE, OUTPUT, DIALOG-SAVE bzw. der SDF-Operand LISTING.

3.6.4 Steuern der Ausgabe der Interaktiven Analyse: Compileroptionen DIALOG-SAVE und OUTPUT

Compileroption DIALOG-SAVE

```
[*]COMOPT   DIALOG-SAVE = ([ { *STD[-FILE]                          } ]
                           { [FILE=] { datei                        } }
                           {         { plamspezifikation            } }

                          [,[INCLUDE-EXPANSIONS=] { NO  } ])
                                                  { YES }
```

*STD[-FILE]
: Bei Angabe von *STD-FILE wird das Dialogergebnis in eine Datei mit dem Namen FOR1.SAV.WS.prog[.tsn[.time]] geschrieben (siehe OUTPUT-Option).

datei
: Name einer katalogisierten Datei, in die die Arbeitsdatei geschrieben wird. Maximale Länge einschließlich Katalog- und Benutzerkennung: 54 Zeichen.

plamspezifikation
: Spezifikation eines PLAM-Bibliothekselements, in das die Arbeitsdatei geschrieben wird:

```
[*LIBRARY-ELEMENT] ([LIBRARY=]plamlib,
                    [ELEMENT=]name[([VERSION=] { *UPPER-LIMIT } )])
                                               { version      }
```

plamlib
: Name einer PLAM-Bibliothek. Maximale Länge einschließlich Katalog- und Benutzerkennung: 54 Zeichen.

name
: Name eines PLAM-Bibliothekselements. Der Name darf aus maximal 64 Zeichen bestehen.

version
: Versionsbezeichnung des PLAM-Bibliothekselements. Eine Versionsbezeichnung kann maximal 24 Zeichen lang sein.

*UPPER-LIMIT
: Die Arbeitsdatei wird mit der höchstmöglichen Versionsbezeichnung eingetragen.

INCLUDE-EXPANSIONS

=YES Der Quellprogrammtext der INCLUDE-Elemente wird mit ausgegeben, die zugehörigen INCLUDE-Anweisungen erscheinen als Kommentar.

=NO Standardmäßig werden die INCLUDE-Elemente nicht expandiert ausgegeben.

Einschränkung:

Die DIALOG-SAVE-Option darf nicht gleichzeitig mit der OUTPUT-Option angegeben werden.

Compileroption OUTPUT

```
[*]COMOPT | OUTPUT [={ dateiname                                          }]
                     { (dateiname [,expand])                              }
                     { ( WS= dateiname [,OS=dateiname] [,expand])         }
                     { ([WS=] (dateiname [,expand])                       }
                     {       [,OS= (dateiname [,expand])])                }
```

dateiname

bezieht sich bei den ersten beiden Spezifikationen auf die Arbeitsdatei.

Sind WS-Dateiname und OS-Dateiname gleich, so überschreibt die geänderte Arbeitsdatei die Originaldatei

- im Dialogbetrieb auf Wunsch; Abfrage!
- im Stapelbetrieb in jedem Fall.

Ist kein Dateiname spezifiziert, so erzeugt der Compiler Standardnamen in folgender Form:

FOR1.SAV.WS.prog[.tsn[.time]]	für die Arbeitsdatei; Linkname: EDWSLINK

prog	Name der Programmeinheit
tsn	Task Sequence Number, 4stellig
time	Startzeit der Übersetzung in der Form hhmmss

Die Angaben tsn, time werden bei Bedarf schrittweise angehängt, um eindeutige Dateinamen zu generieren.

Eine OS-Datei wird nur erzeugt, wenn der OS-Operand spezifiziert wird. Der Linkname für die OS-Datei ist EDOSLINK. Die Zuordnung einer Datei über diesen Linknamen hat Vorrang gegenüber der *dateiname*-Angabe im OS-Operanden.

expand := [NO]EXPAND (Include-Expansionen)

Voreinstellung: NOEXPAND
Bei EXPAND werden die Include-Elemente expandiert in die Arbeitsdatei übertragen und die zugehörigen %INCLUDE-Anweisungen als Kommentarzeilen ausgewiesen.

Bei der dritten Spezifikation bezieht sich "expand" auf die Arbeitsdatei und auf das Originalprogramm.

WS Arbeitsdatei (Worksource)

OS Orginalprogramm (Originalsource)

Originalprogrammdatei und Arbeitsdatei werden erst nach der Übersetzung aller Programmeinheiten erstellt. Zwischenzeitliche Sicherung ist mit dem Kommando %SAVE bzw. %WRITE möglich.

Existiert bereits eine Ausgabedatei mit gleichem Namen, so wird diese
- im Dialogmodus nach einer Abfrage ggf. überschrieben,
- im Batchmodus überschrieben.

Die vom Compiler generierten Zeilennummern werden nicht als Indizes verwendet, sondern bei der Ausgabe auf ISAM-Dateien werden neue Indizes generiert (standardmäßige Schlüssellänge: 8 byte).

Einschränkung:

Die OUTPUT-Option darf nicht gleichzeitig mit der DIALOG-SAVE-Option angegeben werden.

3.6.5 Fehleranzeige

Bei der Interaktiven Analyse liest und analysiert der Compiler die einzelnen Anweisungen eines Quellprogramms der Reihe nach; deshalb muß die Reihenfolge gemäß Standard FORTRAN 77 eingehalten werden. Alle Deklarationsanweisungen müssen am Programmanfang stehen. Bei einer fehlerhaften Anweisung wird die Übersetzung unterbrochen, und zwar nach der Analyse der gesamten Anweisung.

Der Compiler zeigt am Datensichtgerät folgende Zeilen:

1. Zeile die fehlerhafte Anweisung einschließlich der Zeilennummer.
2. Zeile die Fehlernummer und Markierung der Fehlerposition.
3. Zeile den Meldungstext.

(Genauer: Ausgegeben wird die 1. Zeile der fehlerhaften Anweisung sowie eine weitere Zeile für den Fall, daß dort eine Fehlerposition markiert ist. Hat die Anweisung Folgezeilen, so kann sich der Anwender wahlweise die gesamte Anweisung mit dem Kommando %PRINT ausgeben lassen. Ggf. werden auch mehrere Fehler angezeigt.)

Überschreitet die Ausgabe eine Bildschirmseite, so kann der Anwender wahlweise (Abfrage!) einen weiteren Bildschirm ausgeben lassen, wobei der alte überschrieben wird.

Der Anwender hat zwei Möglichkeiten, auf die Fehleranzeige zu reagieren:

- Eingabe einer FORTRAN-Anweisung,
- Eingabe eines Kommandos (zur Interaktiven Analyse).

Da nicht alle möglichen Fehler während der syntaktischen Analyse erkannt werden, erhält der Anwender die Kontrolle zu folgenden Zeitpunkten:

- nach der formalen Analyse zur Korrektur nicht definierter Anweisungsmarken,
- nach der semantischen Analyse zur Korrektur von Semantikfehlern,
- nach Ausgabe des Bindemoduls in den temporären EAM-Bereich zur Korrektur von Fehlern, die bei der Interpretation von DATA-Anweisungen auftraten (vor Erstellung der Übersetzungslisten).

In jedem dieser drei Fälle muß nach einer Korrektur die gesamte Programmeinheit neu übersetzt werden, nicht jedoch vorhergehende Programmeinheiten.

Im dritten Fall liegt am Ende der Übersetzung der Modul für die betreffende Programmeinheit mehrfach vor, was zu Binde-/Ladefehlern führen kann. Falls es sich um die erste Programmeinheit handelt, kann dieses Problem umgangen werden, indem vor einer neuen Übersetzung der fehlerhafte Modul gelöscht wird (Kommando: %SYS DEL-SYS-FILE OMF). Ansonsten erreicht der Anwender durch Übernehmen der Moduln aus dem temporären EAM-Bereich in eine Bibliothek die Eliminierung der fehlerhaften Moduln, da diese durch die dahinterstehenden, korrigierten Moduln überschrieben werden.

3.6.6 Update der Arbeitsdatei

Mit der Ausgabe eines Syntaxfehlers durch die Interaktive Analyse beginnt der Update-Zustand. Die Übersetzung ist für die Dauer eines Updates unterbrochen. Der Anwender kann nun sein Quellprogramm in der vom Compiler automatisch angelegten Arbeitsdatei korrigieren im weitesten Sinne, d.h. also Fehler beheben, Zeilen verschieben, FORTRAN-Anweisungen hinzufügen oder löschen.
Die Eingabe des Anwenders bestimmt, ob der Compiler im Update-Zustand bleibt oder ob der Compiler die Übersetzung fortsetzt.

Beendet wird der Update-Zustand durch:
- %CONTINUE
- %BATCH
- %STOP
- %RESTART
- die alleinige Korrektur von Zeilen der fehlerhaften aktuellen FORTRAN-Anweisungen
- Drücken der DÜ-(bzw. ENTER-)Taste (entspricht %CONTINUE)
- die Eingabe von "/*" in den Spalten 1 und 2 (=Ende der Dateneingabe vom Datensichtgerät).

Erhalten wird der Update-Zustand bei:
- jedem beliebigen anderen Kommando
- Eingabe einer FORTRAN-Anweisung.

Quellprogramm verändern (auf der Arbeitsdatei):
- Zeilen ersetzen (%SET)
- Zeilen löschen (%DELETE)
- Zeilen einfügen (%INSERT)
- Zeilen kopieren (%COPY)
- Zeilen verschieben (%MOVE)

Weitere Funktionen sind:
- Einsehen der Arbeitsdatei und Blättern (%PRINT).
- Einsehen der Fehlerdatei (%PRINT ERROR) und Blättern; in dieser Datei sind alle anstehenden Fehler aufgelistet.
- Arbeitsdatei sichern (%SAVE bzw. % WRITE).
- Einsehen der HELP-Datei (%HELP); sie beschreibt die Interaktive Analyse.
- Arbeitsdatei neu durchnumerieren (%RENUMBER).
- Wechseln des Verarbeitungszustandes (%BATCH; %STOP).
- Ausführen von BS2000-Kommandos (%SYSTEM).
- Ignorieren aller durchgeführten Korrekturen in der aktuellen Programmeinheit (%RESTART).

Fertig übersetzte Programmeinheiten sind vor Veränderungen im Rahmen der Interaktiven Analyse geschützt und können daher nicht mehr mit den beschriebenen Kommandos angesprochen werden (Ausnahme: %SAVE bzw. %WRITE).

3.6.6.1 Zeilennumerierung

Die Zeilen werden in der Reihenfolge des Einlesens durchnumeriert. Die Zeilennummer besteht aus einer maximal 8stelligen Dezimalzahl, wobei zur besseren Lesbarkeit vor den letzten vier Stellen ein Dezimalpunkt eingefügt wird. Beim Einlesen beginnt die Numerierung mit 1.0000, und die Zeilennummer erhöht sich jeweils um die aktuelle Schrittweite (Standardwert 1.0000) bei jeder weiteren Zeile.

Die Zeile mit der niedrigsten Zeilennummer kann mit %, die Zeile mit der höchsten Zeilennummer kann mit $ angesprochen werden.
Handelt es sich bei den Zeilen um Zeilen aus einem Includeelement, so erhalten diese Zeilen ein Präfix. Dieses Präfix kennzeichnet die Zeilen als Includeelementzeilen. Die Includeelemente werden dabei fortlaufend durchnumeriert, wobei mit 1 begonnen wird. Die Zeilennumerierung innerhalb eines Include-Elementes beginnt ebenfalls bei 1.0000 mit der Schrittweite 1.0000.

Beispiel: I3.0012.0000
Interpretation: Zeile 12 des 3. Includeelements

Reicht die Standardnumerierung nicht aus, so muß der Anwender auf Anforderung des Compilers ein geeignetes %RENUMBER-Kommando eingeben.

Eingabe von Zeilennummern:

Bei der Eingabe von Zeilennummern können im Rahmen der Eindeutigkeit folgende Abkürzungen getroffen werden:
- führende Nullen vor dem Dezimalpunkt können entfallen
- restliche Nullen hinter dem Dezimalpunkt können entfallen. Dabei wird die Zeilennummer bei fehlendem Dezimalpunkt mit ".0000" ergänzt.

Beispiel:
Zulässige Angaben für Zeilennummern sind: %2; %02.; %2.01; %2.0

Ausgabe von Zeilennummern:

Bei der Ausgabe von Zeilennummern werden führende Nullen vor dem Dezimalpunkt unterdrückt, mit Ausnahme der ersten Stelle vor dem Dezimalpunkt. Nach dem Dezimalpunkt werden immer vier Stellen ausgegeben.

Aktuelle Zeilennummer:

Die aktuelle Zeilennummer ist die letzte angezeigte Zeilennummer bei einer Fehlermeldung.

Zeilenbereich:

Der Zeilenbereich [ln1,ln2] umfaßt alle besetzten Zeilen mit Nummer k, wobei für k gilt: ln1 ≤ k ≤ ln2.
Anstelle eines Zeilenbereichs ist in einem Kommando auch eine Liste von Zeilenbereichen erlaubt, d.h. mehrere Zeilenbereiche, getrennt durch Kommata.
ln1 bzw. ln2 kann auch eine Konstruktion der Form %+ln1 oder $-ln1 oder $+ln2 sein.
Ist der Bereich leer, so erfolgt eine Fehlermeldung.

Schrittweite:

Die neue Zeilennummer wird aus der vorhergehenden Zeilennummer durch Erhöhen um die aktuelle Schrittweite gebildet. Ist die neue Zeile bereits belegt, so erfolgt eine Abfrage, ob die betreffende Zeile überschrieben werden kann (%INSERT, %COPY, %MOVE).

Der Compiler setzt die aktuelle Schrittweite standardmäßig auf 1.0000.

Ist bei einem Kommando die Angabe einer Zeilennummer möglich, so wird entweder die spezifizierte Schrittweite als aktuelle genommen, oder die Schrittweite hängt von der Anzahl der in der Zeilennummer angegebenen Dezimalziffern ab.

Beispiel:
Für k=2 wird die Schrittweite 1 angenommen, für k=2.4 die Schrittweite 0.1, ebenso für k=2.0, für k=2.40 die Schrittweite 0.01 usw.

3.6.6.2 Eingeben von Anweisungen und Kommandos

Korrekturen können auch an Zeilen vorgenommen werden, deren Nummer nicht die aktuelle Zeilennummer ist. Dies geht entweder per Kommando (welches Zeilenbereichsangaben zuläßt) oder durch Überschreiben ausgegebener Zeilen (wie im EDT). In diesem Fall verwendet man entweder die in der Fehleranzeige ausgegebene FORTRAN-Anweisung oder aber die mit Hilfe des %PRINT-Kommandos gezeigten Anweisungen.

Vorgriff-/Rückgriff-Update

Bezieht sich der Update auf eine Zeile mit einer kleineren oder gleichen Zeilennummer als die aktuelle Zeilennummmer, so handelt es sich um einen "Rückgriff-Update", ansonsten liegt ein "Vorgriff-Update" vor. Ein Rückgriff-Update ist nur innerhalb der aktuellen Programmeinheit erlaubt. Er bewirkt, daß die gesamte Programmeinheit bis zur aktuellen Zeilennummer neu übersetzt wird. Bereits übersetzte Programmeinheiten können nicht mehr geändert werden.

Kettung von Anweisungen

Mehrere Anweisungen können in eine Zeile geschrieben werden, wenn sie durch das Zeichen für Zeilenende getrennt sind (wie im EDT). Pro Zeile können 72 Zeichen überschritten werden. Auch können so gekettete Anweisungen über mehr als eine Zeile gehen, höchstens aber über einen ganzen Bildschirm. Der Compiler löst solche Verkettungen in Einzelanweisungen auf, für die er eigene Zeilennummern vergibt, die jedoch nicht in der Arbeitsdatei erkenntlich sind. Am Ende einer solchen Kette von Anweisungen kann auch ein den Update beendendes Kommando stehen. Für Zeilen, die durch Zeilenende getrennt sind, werden Zeilennummern vergeben. Lediglich durch Zeilenumbruch erzeugte Zeilen sind in der Arbeitsdatei nicht mit eigener Nummer versehen. Dort wird die Eingabezeile in ihrer vollen Länge gehalten.

Positionieren auf Spalte 7

Wird ein Semikolon als erstes Zeichen einer Eingabezeile eingegeben, so wird es nach der Eingabe durch 6 Leerzeichen ersetzt. Dadurch wird der auf das Semikolon folgende Text nach der Eingabe auf Spalte 7 positioniert.

Zeilenlänge

Bei einem Update ist die Eingabe der gesamten Zeile (der gesamten Anweisung) notwendig. Eine Zeilenlänge größer 72 Zeichen ist dabei zulässig. Ein Umbruch erfolgt dann in Spalte 72, wobei die notwendigen Folgezeilen vom Compiler generiert werden. Die Zeilenlänge ist durch die Bildschirmgröße begrenzt. Hat die zu korrigierende Anweisung in Spalte 73-80 eine Kennung, so wird diese auch in die Folgezeilen eingetragen, es sei denn, im Kommando ist eine andere Kennung verlangt. Am Beginn der Zeile müssen allerdings die FORTRAN-Eingabeformat-Konventionen eingehalten werden. Kommentare mittels LINEEND-Option sind nur erlaubt, falls die Eingabezeile nicht länger als 72 Zeichen ist.

3.6.6.3 Zeilenumbruch

Besteht die Eingabe von der Datenstation aus mehr als 72 Zeichen, so erfolgt nach Spalte 72 der Umbruch, und zwar automatisch durch den Compiler. Liegt eine Anweisung vor, so beginnt jede notwendige Folgezeile mit einem &-Zeichen in Spalte 6. Handelt es sich um eine Kommentarzeile, so beginnt jede notwendige Folgezeile mit einem C in Spalte 1, vier Leerzeichen und einem &-Zeichen in Spalte 6. CCOM-Zeilen werden berücksichtigt.
Dieser Umbruch wird jedoch nur im Quellprogramm-Listing und in der Quellprogramm-Ausgabedatei sichtbar. Am Datensichtgerät wird die Zeile ohne Umbruch ausgegeben.

Beispiel:

```
                        Spalte 6  ...          Spalte 72
                           |                      |
                           v                      v
Eingabe:                    A=B+  ...           C**2+4

Speicherung in              A=B+  ...           C**
der Arbeitsdatei:          &2+4

Eingabe:              C   LANGE   ...        KOMMENTARZEILE

Speicherung in        C   LANGE   ...        KOMMEN
der Arbeitsdatei:     C    &TARZEILE

Eingabe (COMOPT
LINEEND vereinbart):        A=B   ...        ";" LINEEND

Speicherung in              A=B   ...        ";" L
der Arbeitsdatei:          &INEEND
```

Dies führt zu einem Fehler, da die Folgezeile nicht mehr als Kommentar behandelt wird.

```
Eingabe:              CC    A=B+  ...           C**2+4

Speicherung in        CC    A=B+  ...           C**
der Arbeitsdatei:     CC   &2+4
```

3.6.6.4 Analyse eines Update

Der Compiler fährt mit der Interaktiven Analyse bei der gewünschten Zeile fort (genauer: bei der ersten Anweisung dieser Zeile), die Zeilennummer ist maßgebend; sie muß nicht die aktuelle Zeilennummer sein. Der Update wird in der Reihenfolge der Korrekturen von der Interaktiven Analyse abgearbeitet und ggf. bei einem Fehler unterbrochen.

Zeilenkennung in den Spalten 73-80

Ist in entsprechenden Kommandos eine Zeilenkennung spezifiziert, so wird sie ab Spalte 73 in die zugehörige Zeile eingefügt, auch in die vom Compiler generierten Folgezeilen. Apostrophe in der Zeilenkennung müssen verdoppelt werden. Weniger als acht Zeichen werden ab Spalte 73 linksbündig eingetragen und die Zeile bis Spalte 80 mit Leerzeichen aufgefüllt. Enthält die Zeilenkennung kein Zeichen (Angabe von ’’) oder mehr als acht Zeichen, so erfolgt eine Fehlermeldung.

3.6.6.5 Fehler in einem Include-Element

Sie müssen zusätzlich außerhalb der Interaktiven Analyse behoben werden.
Im Dialogmodus kann hingegen nur die Arbeitsdatei korrigiert werden, wobei Korrekturen an einer Include-Expansion, die an mehreren Stellen vorkommt, getrennt voneinander durchgeführt werden müssen. Modifikationen in einer Include-Anweisung wirken nicht auf das Include-Element selbst.

3.6.7 Kommandos zum Steuern der Interaktiven Analyse

Die mit "Kommandos" bezeichneten Eingaben steuern die Interaktive Analyse. Sie werden im folgenden auch "Dialog-Kommandos" genannt, um sie von BS2000-Kommandos zu unterscheiden.

3.6.7.1 Regeln für die Eingabe von Kommandos

Im sogenannten Kommandomodus werden alle Dialog-Kommandos mit Ausnahme der "Blätterkommandos" eingegeben. Blätterkommandos werden im sogenannten Blättermodus eingegeben; Kennzeichen am Datensichtgerät: "*+-0". Den Wechsel vom Kommandomodus in den Blättermodus bewirkt das Kommando %PRINT.

Kommandos für die Interaktive Übersetzung sollten nur an der Datenstation eingegeben werden. Im Originalprogramm sollten nur Kommentarzeilen, FORTRAN-Anweisungen oder Compiler-Steueranweisungen stehen.

Dialog-Kommandos, die im Originalprogramm stehen, werden zwar an der jeweiligen Stelle ausgeführt, führen aber im Batchmodus zu Fehlern, da sie dort nicht bekannt sind (OPTION NODIALOG). Außerdem werden sie nicht in die Ausgabedatei übernommen.

Fehlerhafte Kommandos werden abgewiesen; eine Fehlermeldung weist auf die Fehlerursache hin.

Kommandos überschreiben nur dann vorhandene Zeilen, wenn der Anwender eine entsprechende Abfrage bestätigt (%INSERT, %COPY, %MOVE).

Die alleinige Betätigung der DÜ-(bzw. ENTER-)Taste wirkt wie
- %CONTINUE im Kommandomodus,
- * im Blättermodus (siehe Kommando %PRINT).

Kommandopräfix

Die Kommandos beginnen mit einem Präfix. Dies kann der Anwender in der Option DIALOG definieren. Eines der Zeichen !, ?, #, $, %, @ ist zulässig. Die Voreinstellung des Präfixes ist "%", wenn die Interaktive Analyse über die Option DIALOG gesteuert wird, und "@", wenn die Interaktive Analyse über den SDF-Operand DIALOG gesteuert wird.
Während eines Übersetzungslaufs kann das Kommandopräfix durch folgende Eingabe geändert werden:

```
aktuelles Kommandopräfix: neues Kommandopräfix
```

Beispiel:
Das Standardzeichen "%" soll gegen das Kommandopräfix "#" ausgetauscht werden; Eingabe: %:#.

Bei Eingabe eines Quellprogramms von einer Datei wird ein Kommandopräfix in Spalte 6 als Fortsetzungszeichen einer FORTRAN-Zeile interpretiert.
Bei Eingabe von der Datenstation führt eine Verdoppelung des Präfixes dazu, daß die Zeile als Datenzeile interpretiert wird.

Abkürzungen

Kommandonamen können durch Weglassen restlicher Buchstaben so weit von rechts abgekürzt werden, wie sie noch eindeutig identifiziert werden können. Überdies gibt es einige spezielle Abkürzungen (siehe Abschnitt 3.6.7.2, Tabelle 3-1).

Leerzeichen

Sie können bei Kommandoeingabe weggelassen werden. Ausnahme: Folgt auf ein Kommando ein mit einem Buchstaben beginnender Operandenwert, so ist dieser durch ein Leerzeichen abzusetzen; z.B. %PRINT␣ERRORS. Es dürfen jedoch auch beliebig viele Leerzeichen eingestreut werden.

3.6.7.2 Zusammenstellung der Dialog-Kommandos

Die folgende Übersicht stellt alle Dialog-Kommandos zusammen. Die kürzestmöglichen Kommandonamen sind angezeigt.

Kommando	Kurz-form	Bedeutung
%BATCH [{L|LF|WS|CP|OS|PU|ALL} [,...]]	B	Abbruch der Interaktiven Analyse, Verzweigen in den Batchmodus
%CONTINUE	CON	Fortsetzen der Übersetzung nach einer Unterbrechung
%COPY ln1[-ln2] [,ln3[-ln4]][,...] {, |TO} ln5[-ln6][,ln7[-ln8]][,...] [(s)] [,]['id'[(incr)]]	COP	Kopieren eines Zeilen-bereichs
%DELETE ln1[-ln2] [,ln3[-ln4]]...	D	Löschen eines Zeilen-bereichs
%HELP	H	Kurzbeschreibung aller Dialog-Kommandos
%INSERT ln1[(s)] [,]['id'[(incr)]] [:string]	IN	Einfügen eines Zeilen-bereichs
%LOWER { ON / OFF }	L L ON L OF	Kleinbuchstaben zulassen

Fortsetzung▷

Fortsetzung

Kommando	Kurz-form	Bedeutung
%MOVE ln1[-ln2] [,ln3[-ln4]][,...] {, |TO} ln5[-ln6][,ln7[-ln8]][,...] [(s)] [,]['id'[(incr)]]	M	Verschieben eines Zeilen-bereichs
%PRINT [{ln1[-ln2] [,ln3[-ln4]]... [,[NO]EXPAND] E[RRORS]}]	P	Ausgabe eines Zeilenbereichs bzw. der Fehlerdatei; danach Blättern möglich
+ +n - -n -- ++ * 0 !		Blätterkommandos, nach einem %PRINT-Kommando Verketten der Blätter-kommandos
%:{@|#|?|!|%|$}		Kommandopräfix ändern
%RENUMBER [ln1 [(s)]]	R	Neunumerierung
%RESTART	RES	Neustart bei der aktuellen Programmeinheit des Origi-nalprogramms
%SAVE ['dateiname' [, [NO]EXPAND]]	SA	Retten der Arbeitsdatei (ISAM)
%[SET] ln1 [(s)] [,] ['id'[(incr)]] [:string]	SE	Ändern einer Zeile
%STOP [{L|LF|WS|CP|OS|PU|ALL} [,...]]	ST	Beenden der Übersetzung
%SYSTEM ['BS2000-Kommando']	SY	Ausführen von BS2000-Kommandos
%WRITE ['dateiname'[,[NO]EXPAND]]	W	Retten der Arbeitsdatei (SAM)

Tab. 3-1: Zusammenstellung der Dialogkommandos

3.6.7.3 BATCH (Fortsetzung im Batchmodus)

```
%BATCH [{L | LF | WS | CP | OS | PU | ALL} [, ...]]
```

L	Listing
LF	Listfile
WS	Arbeitsdatei (Worksource)
CP	Änderungsprotokoll (Changeprotocol)
OS	Originaldatei (Originalsource)
PU	Batchmodus für die aktuelle Programmeinheit (Program Unit)
ALL	L,LF,WS,CP,OS

Die Interaktive Analyse wird abgebrochen, und nach dem Beenden der Übersetzung im Batchmodus werden die spezifizierten Protokolle ausgegeben. Bei Angabe von "PU" wird der Batchmodus nur für die aktuelle Programmeinheit eingeschaltet.

Listing und Listfile werden im Standardumfang ausgegeben, soweit sie nicht schon in der Compileroption angefordert wurden.

Bei der Spezifikation des Änderungsprotokolls spielt die Angabe von L bzw. bzw. LF eine Rolle:

- Sind L und/oder LF spezifiziert, so wird das Änderungsprotokoll in diese Listen eingefügt
- Ist weder L noch LF spezifiziert, so wird das Änderungsprotokoll nach SYSLST ausgegeben

3.6.7.4 CONTINUE (Fortsetzen der Übersetzung)

```
%CONTINUE
```

Dieses Kommando setzt nach der durch den Fehler bedingten Unterbrechung die Übersetzung fort.

Dabei ist zu beachten, daß bei einem vorherigen Rückgriff-Update der Compiler am Anfang der aktuellen Programmeinheit aufsetzt. Dies wird durch einen entsprechenden Hinweis angezeigt. Ansonsten setzt der Compiler die Übersetzung mit der nächsten Anweisung fort. Eine korrigierte Anweisung wird dabei nochmals übersetzt, da die Interaktive Analyse mehr als eine reine Syntaxprüfung umfaßt (z.B. Struktur von DO-Schleifen).

3.6.7.5 COPY (Kopieren eines Zeilenbereichs)

```
%COPY ln1 [-ln2] [,ln3[-ln4]][,...]{, | TO} ln5[-ln6][,ln7[-ln8]][,...]
      [(s)] [,] ['id'[(incr)]]
```

ln1,...	Zeilennummer
s	Schrittweite
id	Zeilenkennung
incr	Inkrement für Zeilenkennung

Der durch die Zeilennummern ln1 und ln2 bestimmte Kopierbereich wird an die durch die Zeilennummer ln5 spezifizierte Stelle kopiert. Die neuen Zeilennummern werden dabei aus ln5 und s bestimmt. Ist s nicht spezifiziert, so wird die aktuelle Schrittweite zur Bestimmung der neuen Zeilennummer genommen. Wird nur ln1 spezifiziert, oder stimmen ln1 und ln2 überein, so wird die Zeile ln1 kopiert.

Eine angegebene Zeilenkennung wird ab Spalte 73 linksbündig eingetragen, und zwar auch in vom Compiler generierte Folgezeilen.
Apostrophe innerhalb der Zeilenkennung müssen verdoppelt werden. Hat sie mehr als 8 Zeichen oder ist sie leer (d.h. entweder '␣' oder "), so erfolgt eine Fehlermeldung. Ein ausgegebenes Inkrement wird auf den numerischen Wert der Zeilenkennung bei jeder durch Kopie erzeugten Zeile aufaddiert (von rechts).

Das Kopierkommando wird nicht ausgeführt (Ausgabe einer Fehlermeldung), wenn die maximale Zeilennummer erreicht ist, oder der Kopierbereich leer ist.

Bevor vorhandene Zeilen vom Kommando überschrieben werden, wird der Anwender nach einer Bestätigung gefragt. Entsprechend der Antwort wird das Kommando ausgeführt oder abgelehnt.

3.6.7.6 DELETE (Löschen eines Zeilenbereichs)

```
%DELETE [ln1 [-ln2][,ln3[-ln4]]...]
```

ln1,...	Zeilennummer

Gelöscht wird der Zeilenbereich [ln1-ln2]...

Das Kommando %DELETE ohne Angabe von Zeilennummern bewirkt das Löschen des gesamten Zeilenbereiches. Um ein irrtümliches Löschen des Gesamtbereiches auszuschließen, wird vor der Ausführung dieses Kommandos zurückgefragt.

3.6.7.7 HELP (Kurzbeschreibung aller Dialog-Kommandos)

```
%HELP
```

Eine Kurzbeschreibung aller Kommandos wird am Datensichtgerät ausgegeben.

3.6.7.8 INSERT (Einfügen eines Zeilenbereichs)

```
%INSERT ln1 [(s)] [,] ['id'[(incr)]] [:string]
```

ln1	Zeilennummer
s	Schrittweite
id	Zeilenkennung (Spalten 73-80)
incr	Inkrement für Zeilenkennung
string	Zeichenfolge (FORTRAN-Zeile)

Der Compiler bestimmt aus ln1 und s die zu generierenden Zeilennummern und erwartet eine sequentielle Eingabe. Dabei gibt der Compiler die Zeilennummer vor.

Ist die Schrittweite nicht spezifiziert, so wird die aktuelle Schrittweite genommen.

Eine angegebene Zeilenkennung wird ab Spalte 73 linksbündig eingetragen, und zwar auch in vom Compiler generierte Folgezeilen. Apostrophe innerhalb der Zeilenkennung müssen verdoppelt werden. Hat sie mehr als 8 Zeichen oder ist sie leer (d.h. entweder '␣' oder "), so erfolgt eine Fehlermeldung. Ein angegebenes Inkrement wird auf den numerischen Wert der Zeilenkennung bei jeder durch Kopie erzeugten Zeile aufaddiert (von rechts).
Im Gegensatz zum Kommando %SET werden niemals Zeilen überschrieben.

3.6.7.9 LOWER (Einstellen der Groß-/Kleinschreibung)

```
%LOWER {OFF|ON}
```

%LOWER OFF	Der Compiler setzt am Datensichtgerät eingegebene Kleinbuchstaben in Großbuchstaben um.
%LOWER ON	Der Compiler unterscheidet am Datensichtgerät eingegebene Klein- und Großbuchstaben.

3.6.7.10 MOVE (Verschieben eines Zeilenbereichs)

```
%MOVE ln1 [-ln2] [,ln3[-ln4]][,...]{, | TO} ln5[-ln6][,ln7[-ln8]][,...]
         [(s)] [,] ['id'[(incr)]]
```

ln1,...	Zeilennummer
s	Schrittweite
id	Zeilenkennung
incr	Inkrement für Zeilenkennung

Der Verschiebebereich [ln1,ln2] wird an die durch die Zeilennummer ln5 spezifizierte Stelle verschoben. Die neuen Zeilennummern werden dabei aus ln5 und s bestimmt. Ist s nicht spezifiziert, so wird die aktuelle Schrittweite zur Bestimmung der neuen Zeilennummer genommen. Wird nur die Zeile ln1 spezifiziert, oder stimmen ln1 und ln2 überein, so wird die Zeile ln1 verschoben.

Eine angegebene Zeilenkennung wird ab Spalte 73 linksbündig eingetragen, und zwar auch in vom Compiler generierte Folgezeilen. Apostrophe innerhalb der Zeilenkennung müssen verdoppelt werden. Hat sie mehr als 8 Zeichen oder ist sie leer (d.h. entweder '␣' oder "), so erfolgt eine Fehlermeldung. Ein angegebenes Inkrement wird auf den numerischen Wert der Zeilenkennung bei jeder durch Kopie erzeugten Zeile aufaddiert (von rechts).

Das Verschiebekommando wird in folgenden Fällen nicht ausgeführt:
- die maximale Zeilennummer ist erreicht
- der zu verschiebende Bereich ist leer

Bevor vorhandene Zeilen vom Kommando verändert werden, wird der Anwender nach einer Bestätigung gefragt. Entsprechend der Antwort wird das Kommando ausgeführt oder abgelehnt.

3.6.7.11 PRINT (Ausgabe eines Zeilenbereichs/der Fehlerdatei; Blättern)

```
%PRINT  [{ ln1 [-ln2] [,ln3[-ln4]]... [,expand] }]
         { E[RRORS]                             }
```

ln1,... Zeilennummer, speziell: % für Anfang, $ für Ende

expand:= [NO] EXPAND

EXPAND Include-Elemente werden expandiert (Voreinstellung).
NOEXPAND Include-Elemente werden nicht expandiert.

ERRORS Ausgabe der Fehlerliste

Ausgegeben wird der Zeilenbereich [ln1,ln2] der Arbeitsdatei oder bei ERRORS der Beginn der Fehlerdatei. Damit wird vom Kommandomodus in den Blättermodus verzweigt.

Blättern in dem mit %PRINT ausgegebenen Zeilenbereich

Folgende Kommandos sind im Blättermodus möglich:

```
+    eine Bildschirmseite vorwärts
+n   n Zeilen vorwärts (n ist ganze Zahl)
-    eine Bildschirmseite rückwärts
-n   n Zeilen rückwärts (n ist ganze Zahl)
--   Positionieren zum Anfang des gegebenen Zeilenbereichs (%)
++   Positionieren ans Ende des gegebenen Zeilenbereichs ($)
*    eine Bildschirmseite vorwärts; am Ende des gegebenen
     Zeilenbereichs Wechsel in den Kommandomodus; im Gegensatz dazu
     erhalten die zuvor genannten Kommandos den Blättermodus
0    Beenden des Blättermodus, Wechsel in den Kommandomodus
```

Blätterkommandos können durch ein Ausrufezeichen miteinander verkettet werden.

Beispiel:

%PRINT % zeigt die Zeile mit der niedrigsten Nummer.
%P %-$ zeigt die gesamte Arbeitsdatei.
++!- zeigt das Ende des vorgegebenen Zeilenbereichs.

3.6.7.12 RENUMBER (Neunumerierung)

```
%RENUMBER [ ln1 [(s)] ]
```

ln1	Zeilennummer
s	Schrittweite

Ein spezifiziertes s wird als neue aktuelle Schrittweite genommen.
Die Zeilen der Arbeitsdatei werden - mit ln1 beginnend - mit der aktuellen Schrittweite neu numeriert.
Fehlen ln1 und s, so wird bei 1.0000 begonnen und die aktuelle Schrittweite auf 1.0000 gesetzt.

3.6.7.13 RESTART (Neustart)

```
%RESTART
```

Der Compiler beginnt erneut die Übersetzung am Anfang der aktuellen Programmeinheit, wobei er alle in dieser Programmeinheit vorgenommenen Änderungen zurücknimmt (Rückstellen auf Originalprogramm). Die Optionen bleiben unverändert.

Wurde das Originalprogramm direkt am Datensichtgerät eingegeben, so ist keine erneute Eingabe nötig. Der Compiler übersetzt/analysiert alle bisherigen FORTRAN-Zeilen noch einmal, sofern sie nicht per Kommando (%INSERT, %MOVE,...) erzeugt wurden.

3.6.7.14 SAVE bzw. WRITE (Retten der Arbeitsdatei)

```
{%SAVE|%WRITE}['dateiname'[, [NO]EXPAND ]]
```

SAVE	Einrichten einer ISAM-Datei.
WRITE	Einrichten einer SAM-Datei.
dateiname	Dateiname gemäß BS2000-Konvention. Ist er nicht angegeben, so wird der Name aus der Option OUTPUT (WS) bzw. der Standardname genommen.
EXPAND	Include-Elemente werden expandiert. Dabei werden die %INCLUDE-Anweisungen als Kommentarzeilen ausgegeben.
NOEXPAND	Include-Elemente werden nicht expandiert (Voreinstellung).

Die aktuelle Arbeitsdatei wird unter dem angegebenen Namen eingerichtet und damit gesichert.
Existiert schon eine BS2000-Datei mit dem angegebenen Namen, so wird die Erlaubnis des Anwenders eingeholt, um die Datei zu überschreiben.

Das Originalprogramm kann nicht durch das %SAVE bzw. %WRITE-Kommando überschrieben werden, wenn es von SYSDTA gelesen wurde, Element einer Include-Bibliothek oder einer Gruppendatei (GAM-Datei) ist.

3.6.7.15 SET (Ändern einer Zeile)

```
%[SET] ln1 [(s)] [,] ['id'[(incr)]] [:string]
```

ln1	Zeilennummer
s	Schrittweite
id	Zeilenkennung
incr	Inkrement für Zeilenkennung
string	Zeichenfolge (FORTRAN-Zeile)

Mit diesem Kommando läßt sich der Inhalt einer vorhandenen Zeile ln1 ändern. Gleichzeitig kann die aktuelle Schrittweite verändert werden. Dabei wird die Zeile mit der spezifizierten Zeichenfolge beschrieben (ab Spalte 1).

Eine bereits existierende Zeilenkennung wird übernommen und ggf. in erforderliche Folgezeilen eingetragen, sofern keine Zeilenkennung im Kommando spezifiziert wurde.

Eine angegebene Zeilenkennung wird ab Spalte 73 linksbündig eingetragen, und zwar auch in vom Compiler generierten Folgezeilen. Apostrophe innerhalb der Zeilenkennung müssen verdoppelt werden. Hat sie mehr als 8 Zeichen oder ist sie leer (d.h. entweder '␣' oder "), so erfolgt eine Fehlermeldung. Ein angegebenes Inkrement wird auf den numerischen Wert der Zeilenkennung bei jeder durch Kopie erzeugten Zeile aufaddiert (von rechts).

Hinweise:

- Existiert keine Zeile mit der Nummer ln1, so wird eine neue Zeile erzeugt.
- Notwendige Folgezeilen erzeugt der Compiler.
- Ein spezifiziertes s wird als aktuelle Schrittweite genommen.
- "SET" darf bei Kommandoeingabe fehlen.

3.6.7.16 STOP (Beenden der Übersetzung)

```
%STOP  [{L | LF | WS | CP | OS | PU | ALL} [, ...]]
```

L	Listing
LF	Listfile
WS	Arbeitsdatei (Worksource)
CP	Liste der Änderungen (Changeprotokoll)
OS	Originaldatei (Originalsource)
PU	– Abbruch der Übersetzung der aktuellen Programmeinheit – keine Listenausgabe – Fortsetzung bei der nächsten Programmeinheit.
ALL	L,LF,WS,CP,OS

Die Übersetzung wird abgebrochen, und die angegebenen Protokolle werden ausgegeben. Ausnahme: "PU".

Ist bei Programmeingabe von der Datenstation die aktuelle Programmeinheit noch nicht vollständig, so beendet sich der Compiler abnormal. Es wird folgende Meldung ausgegeben: `STOP REQUESTED DURING COMPILATION OF P.U. ...` Zustandsanzeige einer evtl. gesetzten Jobvariablen: $A2001

Listing und Listfile werden im Standardumfang ausgegeben, soweit sie nicht schon in der LIST- bzw. LISTFILE-Option angefordert wurden.

Bei der Spezifikation der Liste der Änderungen spielt die Angabe von L bzw. LF eine Rolle. Sind L und/oder LF spezifiziert, so wird das Änderungsprotokoll in diese Listen eingefügt. Ist weder L noch LF spezifiziert, so wird die Liste der Änderungen allein nach SYSLST ausgegeben.

3.6.7.17 SYSTEM (Ausführen von BS2000-Kommandos)

```
%SYSTEM   ['[/]BS2000-Kommando']]
```

Das in Apostrophen gegebene BS2000-Kommando wird ausgeführt.

Ist kein BS2000-Kommando angegeben, so wird in den System-Modus verzweigt (BREAK); Rückkehr mit dem Kommando RESUME-PROGRAM.

3.6.8 Beispiel zur Interaktiven Analyse

Im folgenden ist skizziert, wie man mit der Interaktiven Analyse arbeitet. Das Beispiel umfaßt drei Abschnitte:

1. Neuerstellen eines Programms
2. Testlauf dieses Programms
3. Erweiterung dieses Programms

Die Abschnitte A.6.2 bis A.6.10 im Anhang zeigen Listen zu diesem Programm.

```
/DEL-SYS-FILE OMF
/START-PROG $FOR1
 %  BLS0500 PROGRAM 'FOR1', VERSION '2.2A00' OF '91-06-05' LOADED.
 %  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1991 ...
  FOR1:    V2.2A00 READY, GIVE COMPILER OPTION
*COMOPT D=(D,@),OUT=WS.X,LF=LF.X(SRC,D,OP,XR),END                          (1)
 @   1.    :      PROGRAM DIALOG                                           (2)
 @   2.    :      INTEGER I1,I2
 @   3.    :      REEL A(10),B
 @   3.0000:         REEL A(10),B
           ........ 1                                                      (3)
 1 FEHLER    ANWEISUNGS-SCHLUESSELWORT VERSCHRIEBEN: REAL VERWENDET
 @   3.    :      REAL A(10),B                                             (4)
 @   4.    :****
 @   5.    :      DO 10 I=1.5<      B(I)=A(I)+I                            (5)
 @   7.    :10    A(I)=I
 @   8.    :      WRITE (2,11) A
 @   9.    :11    FORMAT (' **',5F5.2)
 @  10.    :      CALL SUBA(A(1))
 @  11.    :      END
 @   7.0000:10       A(I)=I                                                (6)
   WARNUNG   NICHT ANGESPROCHENE MARKE #10
 @   7.    :@P                                                            (7)
 @   1.0000:         PROGRAM DIALOG
 @   2.0000:         INTEGER I1,I2
 @   3.0000:         REAL A(10),B
 @   4.0000:****
 @   5.0000:         DO 10 I=1.5
 @   6.0000:            B(I)=A(I)+I
 @   7.0000:10       A(I)=I
 @   8.0000:         WRITE (2,11) A
 @   9.0000:11       FORMAT (' **',5F5.2)
 @  10.0000:         CALL SUB(A(1))
 @  11.0000:         END
 @   7.    :                                                              (8)
 @   5.0000:         DO 10 I=1,5                                           (9)
 @   5.0001:
@CON                                                                      (10)
 FOR1: RECOMPILIERUNG DER AKTUELLEN P.U. INITIIERT
 @  11.0000:          END
   WARNUNG   STATEMENT FUNKTION B UNBENUTZT                               (11)
 @  11.    :@I3.5                                                         (12)
@I3.5
   !
 FOR1:  FEHLER, FALSCHE INCLUDE NUMMER
 @  11.    :@IN3.5                                                        (13)
 @   3.5   :      DIMENSION B(5)
 @   3.6   :@R                                                            (14)
 @  13.    :@P1-2
 @   1.0000:         PROGRAM DIALOG
 @   2.0000:         INTEGER I1,I2
 @  13.    :@D2                                                           (15)
 FOR1: 1 ZEILE(N) GELOESCHT
 @  13.    :@CON

 FOR1: RECOMPILIERUNG DER AKTUELLEN P.U. INITIIERT
 FOR1: LIST FILE ERZEUGT = LF.X
```

```
 FOR1: KEINE FEHLER WAEHREND UEBERSETZUNG DER P.U. DIALOG
@  13.    :      SUBROUTINE SUBA(X)                                    (16)
@  14.    :      Y=X*X<      END
 FOR1: KEINE FEHLER WAEHREND UEBERSETZUNG DER P.U. SUBA
@  16.    :/*                                                          (17)
 FOR1: WS FILE ERZEUGT = WS.X
 ENDE DER  F O R 1  UEBERSETZUNG;  BENOETIGTE CPU ZEIT: 1.142 SEC.

/START-PROG FROM-FILE=*MODULE(*OMF)                                   (18)
 %  BLS0001 ### DBL VERSION 070 RUNNING
 %  BLS0517 MODULE 'DIALOG' LOADED
BS2000  F O R 1 : FORTRAN PROGRAM "DIALOG"
STARTED ON 1991-09-03 AT 14:23:53
** 1.00 2.00 3.00 4.00 5.00
** 0.00 0.00 0.00 0.00 0.00
BS2000  F O R 1 : FORTRAN PROGRAM "DIALOG  " ENDED PROPERLY AT 14:23:55
CPU - TIME USED :       0.0205 SECONDS
ELAPSED TIME    :       1.7630 SECONDS

/DEL-SYS-FILE OMF
/START-PROG $FOR1
 %  BLS0500 PROGRAM 'FOR1', VERSION '2.2A00' OF '91-06-05' LOADED
 %  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1991 ...
 FOR1:    V2.2A00 READY, GIVE COMPILER OPTION
*COMOPT SRC=WS.X,D=(D,@,E=FIRST),OUT=WS.X1,END                        (19)
 FOR1: NEUE PROGRAMM-EINHEIT - GIB ANWEISUNG ODER @HELP @P
@   1.0000:         PROGRAM DIALOG
@   2.0000:         REAL A(10),B
@   3.0000:         DIMENSION B(5)
@   4.0000:****
@   5.0000:         DO 10 I=1,5
@   6.0000:         B(I)=A(I)+I
@   7.0000:10       A(I)=I
@   8.0000:         WRITE (2,11) A
@   9.0000:11       FORMAT (' **',5F5.2)
@  10.0000:         CALL SUB(A(1))
@  11.0000:         END
@  12.0000:         SUBROUTINE SUB(X)
@  13.0000:         Y=X*X
@  14.0000:         END
@  15.    :@D7                                                         (20)
 FOR1: 1 ZEILE(N) GELOESCHT
@  15.    :@CON
@  11.0000:          END
  SCHWER    OFFENER DO-BLOCK, WIRD DURCH EIN MARKIERTES CONTINUE ABGESCHLOSSEN
@  11.    :@RES                                                        (21)
 FOR1: RECOMPILIERUNG DER AKTUELLEN P.U. INITIIERT MIT ORIGINAL-QUELLE
 FOR1: NEUE PROGRAMM-EINHEIT -GIB ANWEISUNG ODER @HELP @M8-9TO13.9     (22)
 FOR1: ZIEL-BEREICH DER VERSCHIEBE-LISTE UEBERLAPPT EXISTIERENDE ZEILEN - UEBERSCHREIBEN? (J/N)
N
 FOR1: 0 ZEILE(N) VERSCHOBEN
@  15.    :@M8-9T13.5
 FOR1: 2 ZEILE(N) VERSCHOBEN
@  13.7   :@P13-14
@  13.0000:          Y=X*X
@  13.5000:          WRITE (2,11) A
@  13.6000:11        FORMAT (' **',5F5.2)
@  14.0000:          END
@  13.7   :@CON
 FOR1: KEINE FEHLER WAEHREND UEBERSETZUNG DER P.U. DIALOG
 FOR1: WS FILE ERZEUGT = WS.X1
 FOR1: KEINE FEHLER WAEHREND UEBERSETZUNG DER P.U. SUB
 ENDE DER  F O R 1  UEBERSETZUNG;  BENOETIGTE CPU ZEIT: 0.613 SEC.    (23)
```

Erläuterung des Beispiels:

(1) Start der Interaktiven Analyse; Optionen: DIALOG, OUTPUT, LISTFILE; erwünscht sind deutsche Fehlermeldungstexte; Kommandopräfix @.

(2) Eingabeaufforderung. Das folgende Programm wird direkt am Datensichtgerät eingegeben.

(3) In Zeile 3 wird ein Fehler markiert. Der Meldungstext wird ausgegeben.

(4) Fehlerkorrektur: nochmalige Eingabe.

(5) Kettung von zwei FORTRAN-Anweisungen. Der Compiler löst sie in zwei Einzelanweisungen auf; vgl. aktuelle Zeilennummern 5 und 7.

(6) Dies ist eine Folge der fehlerhaften Anweisungszeile 5.

(7) PRINT-Kommando.

(8) Die aktuelle Zeilennummer bleibt 7.

(9) Direktes Überschreiben der Zeile 5 im Ausgabebereich; implizite Schrittweite 0.0001, wie die nächste Zeilennummer zeigt.

(10) CONTINUE-Kommando.

(11) B wird als FUNCTION-Name interpretiert, weil die Dimensionsvereinbarung fehlt.

(12) Das INSERT-Kommando kann nicht zu "I" abgekürzt werden.

(13) Einfügen der Dimensionsvereinbarung.

(14) RENUMBER-Kommando.

(15) DELETE-Kommando.

(16) Die Programmeingabe kann fortgesetzt werden.

(17) Ende der Eingabe. Das Programm wird in der Datei WS.X abgelegt.

(18) Programmablauf.

(19) Programmerweiterung; Eingabedatei WS.X; Ausgabe der Arbeitsdatei in WS.X1. Die Steuerung wird nur vor der ersten Programmeinheit verlangt (EDIT-Operand).

(20) Löschen der Zeile 7 führt zu einem schweren Fehler.

(21) RESTART-Kommando; Aufsetzen auf dem Stand der Datei WS.X.

(22) MOVE-Kommando mit impliziter Schrittweite 0.1. Da vorhandene Zeilen überschrieben werden, wird das Kommando abgeändert nochmal eingegeben.

(23) Das CONTINUE-Kommando führt zum Ende der Interaktiven Analyse, weil EDIT=FIRST gesetzt war - im Gegensatz zum ersten Analyselauf vgl. bei (1).

4 Übersetzen des Quellprogramms

4.1 Festlegen und Prüfen der Eigenschaften des Quellprogramms

4.1.1 SDF-Operand SOURCE-PROPERTIES

```
START-FOR1-COMPILER

,SOURCE-PROPERTIES = STD / PARAMETER(...)

   PARAMETER(...)

      COMPILEABLE-COMMENTS = *NONE / <c-string 1..60>
     ,LINE-END-COMMENTS = *NONE / <c-string 1..10>
     ,LANGUAGE-STANDARD = FOR1 / ANS77
     ,IMPLICIT-DECLARATION = YES / NO
     ,EXPONENT-UNDERFLOW = IGNORED / ERROR
     ,SOURCE-FORMAT = FIXED / FREE
     ,SAVE-CONSTANT = *STD / YES / NO
     ,FORTRAN90-CHECK = YES / NO
```

Eine Gegenüberstellung von SDF-Operanden und entsprechenden Compileroptionen enthält Tab. 2-6.

4.1.2 Festlegen und Prüfen der Eigenschaften des Quellprogramms durch Compileroptionen

4.1.2.1 CCOM-Option

[*]COMOPT	CCOM = 'kommentarmarkierungen'

Alle Kommentarzeilen, die in Spalte 2 eines der angeführten Zeichen aufweisen, werden nach Ersetzen der Spalten 1 und 2 durch zwei Leerzeichen als normale FORTRAN-Anweisungen behandelt und übersetzt. Als Kommentarmarkierungen sind bis zu 60 abdruckbare Zeichen des EBCDIC-Zeichenvorrats zulässig; ausgenommen ist das Leerzeichen.
Mit der Angabe COMOPT CCOM = ' ' können die vereinbarten Kommentarmarkierungen wieder gelöscht werden.

Wird die Compileroption SOURCE-FORMAT=FREE zusammen mit der Option CCOM angegeben, dann erfolgt eine Fehlermeldung. Die zuletzt angegebene Option gilt.

4.1.2.2 LINEEND-Option

[*]COMOPT	LINEEND = 'endemarkierungen'

Jedes einzelne der in Endemarkierungen angegebenen Zeichen übernimmt die Funktion einer Endemarke, sofern es in einer der Spalten 7-72 außerhalb von Character- oder Hollerith-Konstanten auftritt. Alles, was hinter einem solchen Zeichen in der Programmzeile steht, wird vom Compiler als Kommentar behandelt.
Bis zu 10 Zeichen können als Endemarkierungen in der LINEEND-Option vereinbart werden. Die in *endemarkierungen* angegebenen Zeichen müssen im EBCDIC-Zeichenvorrat liegen, dürfen aber nicht dem FORTRAN-Zeichensatz entstammen. Auch das Prozentzeichen (%) darf nicht verwendet werden, da es für FOR1-Compiler-Steueranweisungen und -Testanweisungen benötigt wird.

Leerzeichen in der LINEEND-Option werden ignoriert, können also nicht als Endemarkierung vereinbart werden. Zulässig sind z.B. das Ausrufungszeichen, das Fragezeichen oder der senkrechte Strich.

Mit der Angabe COMOPT LINEEND = ' ' können die vereinbarten Endemarkierungen wieder gelöscht werden.

Wird die Compileroption SOURCE-FORMAT=FREE zusammen mit der Option LINEEND angegeben, dann erfolgt eine Fehlermeldung. Die zuletzt angegebene Option gilt.

4.1.2.3 STANDARD-CHECK-Option

Durch Angabe der Option STANDARD-CHECK=ANS77 wird ein FORTRAN-Quellprogramm auf Abweichungen vom Standard ANS FORTRAN 77 überprüft.

[*]COMOPT	ST[AN]D[ARD-CHECK] = { ANS77 | NO }

Abweichungen des FOR1-Sprachumfangs vom ANS-FORTRAN-77-Sprachumfang werden als WARNINGs ausgegeben. Standardmäßig werden keine Abweichungen vom Standard ANS FORTRAN 77 gemeldet.

Zur Unterscheidung von den sonstigen Fehlermeldungen haben die Abweichungsmeldungen nach dem Fehlertyp und der Fehlernummer den Zusatz: ANS FORTRAN 77 DEVIATION; vgl. z.B.:

```
FA206  ANS FORTRAN 77 DEVIATION: MORE THAN 19 CONTINUATION LINES NOT ALLOWED
```

Ist die Compileroption MSGLEVEL=ERROR eingeschaltet, dann werden keine Meldungen über Abweichungen ausgegeben.

Einschränkungen:

Erscheint die Variable I aus der Anweisung ASSIGN n TO I in Anweisungen außer einer FORMAT- oder GOTO-Anweisung, dann wird dies vom Standardchecker nicht als Abweichung vom Standard ANS FORTRAN 77 gemeldet.

Wird die Compileroption SOURCE-FORMAT=FREE zusammen mit der Option STANDARD-CHECK=ANS77 angegeben, dann erfolgt eine Fehlermeldung. Die zuletzt angegebene Option gilt.

Beispiel: STANDARD-CHECK-Option

Das Programm STDTEST wird mit COMOPT STD=ANS77 übersetzt. Mit COMOPT DIALOG=D oder COMOPT LANGUAGE=GERMAN erhält man deutsche Meldungen der Abweichungen von Standard ANS77. Im SOURCE-Listing und im DIAGNOSTIC-Listing werden die Abweichungen angezeigt.

```
  ****  SOURCE  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER     FOR1 V2.2A00  DATE = ...  TIME = 11:49:53    PAGE    1
                                                PROGRAM UNIT: STDTEST
 DO/IF SEG   STMT  I   LINE        SOURCE-TEXT                                                    COL73-80 RECORD-ID.

*        1      1         1        PROGRAM STDTEST                                                        00010000           *
***** WARNUNG (FA203) *******              -1-                                                    *ANS FORTRAN 77 ABWEICHUNG*****
                        2  C                                                                             00010100
         1      2         3        INTEGER I,J                                                            00020000
*        1      3         4        REAL  A(3,3) , B(3) ,E(3) /3*0./                                       00030000           *
***** WARNUNG (FA249) *******                                  -1-                                    *ANS FORTRAN 77 ABWEICHUNG*****
*        1      4         5        NAMELIST /NAM/ A,B                                                     00040000           *
***** WARNUNG (FA230) *******                                                                           *ANS FORTRAN 77 ABWEICHUNG*****
***** WARNUNG (SA088) *******                                                                           *ANS FORTRAN77 ABWEICHUNG******
*        1      5         6        WRITE *, 'BITTE WERT FUER MATRIX A(3,3) UND VEKTOR B(3)',              00050000           *
*        1                7       1         'IM NAMELIST-FORMAT EINGEBEN /NAM/'                          00060000           *
***** WARNUNG (FA259) *******                                                                           *ANS FORTRAN 77 ABWEICHUNG*****
*        1      6         8        READ (1,NAM)                                                           00070000           *
                        9  C                                                                             00070100
***** WARNUNG (SA089) *******                                                                           *ANS FORTRAN77 ABWEICHUNG******
         2      7        10        DO 100, I=1,3                                                          00080000
   1     2      8        11        DO 100, J=1,3                                                          00090000
   2     3      9        12           E(I) = A(I,J) * B(J) + E(I)                                         00100000
   2     5     10        13   100  CONTINUE                                                              00110000
                         14  C                                                                           00120000
         6     11        15        WRITE (2,'(1X,3E20.4E2)') E                                            00130000
         6     12        16        STOP                                                                   00140000
         6     13        17        END                                                                    00150000


  *** DIAGNOSTIC  LISTING ***      SIEMENS-NIXDORF FORTRAN COMPILER     FOR1 V2.2A00  DATE = ...  TIME = 11:49:53    PAGE    2
                                                PROGRAM UNIT: STDTEST
 DO/IF SEG   STMT  I   LINE        SOURCE-TEXT                                                    COL73-80 RECORD-ID.

*        1      1         1        PROGRAM STDTEST                                                        00010000           *
                              ..... 1 ..........................................................
    1 WARNUNG (FA203)  ANS FORTRAN 77 ABWEICHUNG: NAMENSLAENGE DARF NICHT GROESSER 6 SEIN
*        1      3         4        REAL  A(3,3) , B(3) ,E(3) /3*0./                                       00030000           *
                              ............................... 1 ....................................
    1 WARNUNG (FA249)  ANS FORTRAN 77 ABWEICHUNG: INITIALISIERUNG IN EXPLIZITER TYPANWEISUNG NICHT ERLAUBT
*        1      4         5        NAMELIST /NAM/ A,B                                                     00040000           *
      WARNUNG (FA230)  ANS FORTRAN 77 ABWEICHUNG: NAMELIST-ANWEISUNG IM STANDARD UNBEKANNT
      WARNUNG (SA088)  ANS FORTRAN77 ABWEICHUNG: NAM IST NAMELIST-BEZEICHNER
*        1      5         6        WRITE *, 'BITTE WERT FUER MATRIX A(3,3) UND VEKTOR B(3)',              00050000           *
*        1                7       1         'IM NAMELIST-FORMAT EINGEBEN /NAM/'                          00060000           *
      WARNUNG (FA259)  ANS FORTRAN 77 ABWEICHUNG: WRITE-ANWEISUNG OHNE STEUERPARAMETERLISTE IST NICHT ERLAUBT
*        1      6         8        READ (1,NAM)                                                           00070000           *

      WARNUNG (SA089)  ANS FORTRAN77 ABWEICHUNG: NAM IST NAMELIST-BEZEICHNER
```

4.1.2.4 IMPLICIT-Option

[*]COMOPT	[NO]IMPLICIT

Die Option NOIMPLICIT verbietet die implizite Typzuordnung von FORTRAN-Variablen. IMPLICIT-Anweisungen und undefinierte Variablen werden mit einer Fehlermeldung "Error" versehen.

Die Type-Voreinstellung von INTRINSIC-Funktionen wird hiervon nicht berührt.

4.1.2.5 EXPUNDERFLOW-Option

[*]COMOPT	[NO]EXPUNDERFLOW

Diese Option setzt die entsprechende Programmaske für Underflow.
Diese Programmaske wird nur einmal gesetzt. Die Option hat deshalb nur bei einem FORTRAN-Hauptprogramm eine Wirkung und gilt für alle Unterprogramme.

Bei NOEXPUNDERFLOW (Voreinstellung) wird das Programm bei Unterlauf nicht unterbrochen. Das entsprechende Datenelement wird auf 0 gesetzt.

Bei EXPUNDERFLOW wird das Programm bei Unterlauf mit einer Fehlermeldung unterbrochen.

4.1.2.6 SOURCE-FORMAT-Option

Mit der Option SOURCE-FORMAT wird festgelegt, ob FOR1 ein FORTRAN-Quellprogramm mit spaltenorientiertem Format der Programmzeilen (FIXED) oder ein formatfreies Quellprogramm (FREE) übersetzen soll.

[*]COMOPT	SOURCE-FORMAT = { FIXED / FREE }

FIXED Das Format des Quellprogramms folgt den Spaltenvorschriften und Regeln der FOR1-Beschreibung [21].

FREE Das Quellprogramm liegt formatfrei vor.

Werden die Compileroptionen STANDARD-CHECK=ANS77, LINEEND, UPD, CCOM und DIALOG einerseits und SOURCE-FORMAT=FREE andererseits angegeben, dann erfolgt eine Fehlermeldung. Die zuletzt angegebene Option gilt.

4.1.2.7 SAVE-CONSTANT-Option

[*]COMOPT	SAVE-CONSTANT=$\left\{\begin{matrix}\text{YES}\\ \\ \text{NO}\end{matrix}\right\}$

Mit dieser Option kann die Art der Übergabe von Konstanten an Unterprogramme gesteuert werden.

Voreinstellung: YES bei Angabe von OPTIMIZE = NO, 0, 1, 2
NO bei Angabe von OPTIMIZE = 3, 4

YES Ist ein Aktualparameter eine Konstante, dann wird an das Unterprogramm die Adresse einer Kopie dieser Konstanten übergeben. Damit wird ein Überschreiben der Konstanten im Unterprogramm verhindert.

NO Es wird keine Kopie angelegt, sondern die Adresse der Konstanten wird übergeben. Dadurch kann die Konstante durch das Unterprogramm verändert werden.

Beispiel:

```
    PROGRAM CON
    PARAMETER (CONST=7.5)
    DO 10 I=1,5
    CALL SUB1 (CONST)
 10 WRITE *, ' HAUPTPROGRAMM: CONST = ',CONST
    END
    SUBROUTINE SUB1 (X)
    REAL*4 X
    X=X+2.5
    WRITE *, ' UNTERPROGRAMM: X     = ',X
    RETURN
    END
```

Mit SAVE-CONSTANT=YES wird die Kopie der Konstanten CONST an das Unterprogramm übergeben. Die Konstante ist vor Überschreiben im Unterprogramm geschützt, da nur die Kopie bei der Rückgabe des Wertes an das Hauptprogramm überschrieben wird. Bei jedem Schleifendurchlauf wird ausgegeben:

```
    UNTERPROGRAMM: X     = 0.10000000E+02
    HAUPTPROGRAMM: CONST = 0.75000000E+01
```

Mit SAVE-CONSTANT=NO wird dagegen der im Unterprogramm veränderte Wert an die Konstante CONST im Hauptprogramm zurückgegeben. Man erhält folgende Ausgabe:

```
    UNTERPROGRAMM: X     = 0.10000000E+02
    HAUPTPROGRAMM: CONST = 0.10000000E+02
    UNTERPROGRAMM: X     = 0.12500000E+02
    HAUPTPROGRAMM: CONST = 0.12500000E+02
```

```
UNTERPROGRAMM: X     = 0.15000000E+02
HAUPTPROGRAMM: CONST = 0.15000000E+02
UNTERPROGRAMM: X     = 0.17500000E+02
HAUPTPROGRAMM: CONST = 0.17500000E+02
UNTERPROGRAMM: X     = 0.20000000E+02
HAUPTPROGRAMM: CONST = 0.20000000E+02
```

4.1.2.8 FORTRAN90-CHECK-Option

[*]COMOPT	FORTRAN90-CHECK = {YES|NO}

Durch die Angabe der Option FORTRAN90-CHECK wird ein FORTRAN-Quellprogramm daraufhin untersucht, ob es FOR1-Erweiterungen enthält, die der Fortran90-Compiler nicht mehr unterstützt. Falls solche Erweiterungen im Quellprogramm auftreten, werden WARNINGs ausgegeben. Zur Unterscheidung von den sonstigen Fehlermeldungen haben diese Abweichungsmeldungen nach dem Fehlertyp und der Fehlernummer den Zusatz: FORTRAN90 DEVIATION, vgl. z.B.:

```
FA301    FORTRAN90 DEVIATION: NESTED BOOLEAN IF-STATEMENTS
```

Da die Abweichungsmeldungen vom Fehlergrad WARNING sind, wird die Ausgabe unterdrückt falls die Option MSGLEVEL=ERROR eingeschaltet ist.

Eine Beschreibung der FOR1-Erweiterungen, die vom Fortran90-Compiler nicht mehr unterstützt werden, enthält Abschnitt 10.2. Dort sind auch die Texte der entsprechenden Abweichungsmeldungen verzeichnet.

4.1.2.9 CODE-Option

[*]COMOPT	CODE = { ([SOURCE={EBCDIC|ISO|BCD}][,UPD={EBCDIC|ISO|BCD}]) / {EBCDIC|ISO|BCD} }

Die Option gibt den Code an, in dem das Quellprogramm und die Änderungszeilen der UPDATE-Datei vorliegen. Wird kein Schlüsselwort SOURCE oder UPD angegeben, so gilt der Code sowohl für das Quellprogramm als auch für die Änderungszeilen

Für die Compileroption CODE gibt es keinen entsprechenden SDF-Operanden.

4.2 Festlegen der Eigenschaften des erzeugten Codes

4.2.1 SDF-Operand COMPILER-ACTION

```
START-FOR1-COMPILER

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

   MODULE-GENERATION(...)
      |  SHAREABLE-CODE = NO / YES(...)
      |
      |    YES(...)
      |     |  OUTPUT-LIBRARY = *MODULE-LIBRARY / <full-filename 1..54>
      |
      |  ,MINIMAL-PRECISION = REAL-4(...) / REAL-8(...) / REAL-16(...)
      |
      |    REAL-4(...)
      |     |  EXTERNAL-DATA = NO / YES
      |
      |    REAL-8(...)
      |     |  EXTERNAL-DATA = NO / YES
      |
      |    REAL-16(...)
      |     |  EXTERNAL-DATA = NO / YES
      |
      |  ,CONSTANT-PRECISION = AS-NEEDED / REAL-4
      |
      |  ,CANCEL-CONDITION = NONE / ERROR / SEVERE-ERROR
      |
      |  ,LINKAGE = STD / FOR1-SPECIFIC
```

Eine Gegenüberstellung von SDF-Operanden und entsprechenden Compileroptionen enthält Tab. 2-7.

4.2.2 Festlegen der Eigenschaften des erzeugten Codes durch Compileroptionen

4.2.2.1 OBJECT-Option

Mit der OBJECT-Option kann man steuern, ob
- überhaupt Bindemoduln erzeugt werden;
- falls Bindemoduln erzeugt werden, diese in den EAM-Bereich abgelegt werden;
- mehrfachbenutzbare Bindemoduln erzeugt werden.

[*]COMOPT	{ [NOOBJECT] / [OBJECT [= ({ [SHARE] / [*] })]] }

NOOBJECT
: Es werden keine Bindemoduln erzeugt.

OBJECT=(SHARE)
: Mehrfachbenutzbare Bindemoduln werden erzeugt. (siehe Abschnitt 5.8).

OBJECT=(*)
: Standardmäßig werden Bindemoduln erzeugt und in die temporäre EAM-Datei der laufenden Task eingetragen. Die Bindemoduln werden zu den bereits in der EAM-Datei befindlichen Moduln hinzugefügt. Falls OBJECT=(*) *explizit* angegeben ist *und* MODULE-LIBRARY spezifiziert ist, so ist die Reihenfolge der Eingabe entscheidend: Die später getroffene Angabe des Ablageorts überschreibt die vorher getroffene. In solchen Fällen gibt der Compiler fogende WARNING aus:

```
MA 45  OBJECT OUTPUT CONFLICT
```

Hinweis:

Bei Angabe von COMOPT NOOBJECT wird kein Bindemodul erzeugt. Deshalb werden bei dieser Einstellung folgende Listen nicht erzeugt und können somit nicht ausgegeben werden:
- Liste der externen Namen (ESD LISTING)
- Datenadreßliste (MAP LISTING)
- Querverweisliste (XREF LISTING)
- Attributliste (ATTRIBUTE LISTING)
- Objektliste (OBJECT LISTING)

4.2.2.2 SHARE-LIBRARY-Option

Wurden durch die Option OBJECT=(SHARE) mehrfachbenutzbare Bindemoduln erzeugt, so können diese durch die SHARE-LIBRARY-Option in einer gesonderten PLAM-Bibliothek abgelegt werden.

`[*]COMOPT`	`SHARE-LIB[RARY] = {*MODULE-LIBRARY / plamlib}`

*MODULE-LIBRARY
: Mehrfachbenutzbare und nicht mehrfachbenutzbare Bindemoduln werden in gleicher Weise abgelegt. Sie werden entweder in die mit der MODULE-LIBRARY-Option angegebene PLAM-Bibliothek oder standardmäßig in die temporäre EAM-Datei ausgegeben.

plamlib
: Name einer PLAM-Bibliothek, in der die mehrfachbenutzbaren Bindemoduln abgelegt werden. Die nicht mehrfachbenutzbaren Bindemoduln werden nach den Angaben in der MODULE-LIBRARY-Option abgelegt. Die Namen der mehrfachbenutzbaren Bindemoduln werden nach den gleichen Regeln bestimmt wie bei der MODULE-LIBRARY-Option. Maximale Länge einschließlich Katalog- und Benutzerkennung: 54 Zeichen.

4.2.2.3 REAL-Option

Durch Übersetzung einer Programmeinheit mit der REAL-Option kann ohne Änderung des Quellprogramms die Genauigkeit der darin enthaltenen Gleitpunktgrößen vom Typ REAL oder COMPLEX erhöht werden. Dies kann erforderlich sein z.B. bei Über- oder Unterlauf, zur Konvergenzerzwingung von iterativen Verfahren unter numerisch extremen Bedingungen, bei Portierung von FORTRAN-Programmen usw.

Ungeklammerte Option

`[*]COMOPT`	`REAL = {4 / 8 / 16}`

Bei der Option REAL=8 werden alle REAL-Variablen, -Konstanten, -Funktionen und -Felder der Länge 4 in Größen der Länge 8, alle COMPLEX-Daten der Länge 8 in Größen der Länge 16 umgewandelt. Größen vom Typ REAL*16 und COMPLEX*32 bleiben unverändert.
Bei der Option REAL=16 haben alle REAL-Größen die Länge 16, alle COMPLEX-Größen die Länge 32.

Einschränkungen

- Einige INTRINSIC-Funktionen (z.B. AMAXO) existieren nicht in allen REAL-Längen. In diesem Fall gibt der Compiler eine Meldung aus.
- Bei COMMON und EQUIVALENCE verändert sich die Anordnung der Elemente. Der Compiler kann die Richtigkeit nicht überprüfen.
- Wenn Programmeinheiten zusammengebunden werden, die mit unterschiedlichen REAL-Optionen übersetzt wurden, können Fehler durch unterschiedliche Funktions- und Parameterlängen entstehen. Unterschiedliche Parameter können im gerufenen Unterprogramm mit der Option TESTOPT=(ARG,...) aufgedeckt werden.

Geklammerte Option

`[*]COMOPT`	`REAL = { (4) / (8) / (16) }`

Bei den Optionen REAL=(8) und REAL=(16) werden wie bei den Optionen REAL=8 und REAL=16 die Längen der Konstanten, Variablen und Felder vergrößert.

Im Unterschied zur ungeklammerten Option bleiben unverändert:
- Variablen und Felder, wenn sie in COMMON stehen.
- Variablen und Felder, wenn sie aktuelle oder formale Parameter von Unterprogrammen (SUBROUTINEs) oder externen Funktionen sind.
- Aufrufe externer Funktionen, Definitionen und Eingänge von Funktionen.
- INTRINSIC-Funktionen, wenn sie als aktuelle Parameter in einem Unterprogrammaufruf mitgegeben wird.

Besonderheiten

- Wenn INTRINSIC-Funktionen verlängert werden, können möglicherweise die Parameter kurz bleiben. In diesem Falle wird zur Laufzeit Typkonversion durchgeführt (ohne Warnung zur Compile- und Laufzeit).
- Variablen oder Felder, die erst durch Ersatz der formalen Parameter in Formelfunktionen aktuelle Parameter von Benutzerfunktionen werden, bleiben nicht kurz, sondern werden verlängert (Beispiel 1).
- REAL- bzw. COMPLEX-Konstanten als Aktualparameter werden stets verlängert. REAL- bzw. COMPLEX-Variablen werden, falls sie als Aktualparameter auftreten, nicht verlängert. Zusammengesetzte variable REAL- bzw. COMPLEX-Ausdrücke als Aktualparameter bleiben jedoch nur dann kurz, wenn sie keine Konstanten der Datentypen REAL oder COMPLEX enthalten, ansonsten werden sie verlängert (siehe Beispiel 3 und 4). Wird dies bei der Programmverknüpfung nicht beachtet, so ist ohne TESTOPT=(ARG,...) mit fehlerhaftem Ablauf, mit TESTOPT=(ARG,...) mit einer Fehlermeldung zu rechnen.

Beispiel 1:

```
REAL*4 F,A,B,X
F(X)=1.+A(X)                          A(X) sei der Aufruf einer externen
B =1.                                 Funktion.
B =F(B) → (eingesetzt) B=1.+A(B)      B erhält mit der Option REAL=(8) den
                                      Typ REAL*8 und bleibt nicht REAL*4,
                                      obwohl B durch Einsetzen in die Formel-
                                      funktion F(X) Aktualparameter der
                                      externen Funktion A wird. Der Compiler
                                      erkennt dies nämlich zu spät.
                                      A behält jedoch die Länge 4.
```

Beispiel 2:

```
REAL*4 F,A,B,X
F(X)=X+A(B)
B =1.
B =F(B) → (eingesetzt) B=B+A(B)       B behält die Länge REAL*4 mit der
                                      Option REAL=(8).
```

Beispiel 3:

```
COMPLEX*8 C,D                         Option REAL=(8)
CALL F(C,D+1)                         Beide Parameter haben den Typ COMPLEX*8.
CALL F(C,D+(1,0))                     Nur der erste Parameter bleibt kurz, der
                                      zweite Parameter wird verlängert, da eine
                                      Konstante des Typs COMPLEX enthalten ist.
```

Beispiel 4:

```
COMPLEX*8 C,D                         Option REAL = (8)
CALL F(C,D,D+C)                       Alle drei Parameter haben den Typ COMPLEX*8.
```

4.2.2.4 TRUNCONST-Option

[*]COMOPT	[NO]TRUNCONST

Mit der Option NOTRUNCONST werden REAL-Konstanten ohne REAL-Exponent mit mehr als 7 gültigen Dezimalziffern intern als REAL*8-Konstanten, solche mit mehr als 16 gültigen Dezimalziffern als REAL*16-Konstanten dargestellt.

Dabei werden bis zu 32 gültige Dezimalziffern verwendet. Analoges gilt für COMPLEX-Konstanten.

4.2.2.5 GEN-Option

[*]COMOPT	{ GEN / NOGEN [=E[RROR] / S[EVERE] / F[AILURE]] }

GEN Der Compiler generiert Bindemoduln.

NOGEN **ohne** Operandenwert-Angabe: Der Compiler führt nur die syntaktische und semantische Prüfung durch. Es werden keine Bindemoduln generiert. Folgende Listen können nicht erzeugt und ausgegeben werden: Liste der externen Namen, Datenadreßliste, Querverweisliste, Attributliste und Objektliste.

mit Operandenwertangabe: Bei Auftreten eines Fehlers vom Fehlergrad der GEN-Option oder schwerwiegender werden keine Bindemoduln generiert.

Hinweis:

Das Präfix NO bedeutet hier **nicht**, daß die Operanden-Komplementärmenge gemeint ist.

4.2.2.6 LINKAGE-Option

Mit der LINKAGE-Option kann festgegt werden, nach welchen Konventionen die Schnittstellen der erzeugten Moduln gestaltet werden sollen.

[*]COMOPT	LINKAGE = {STD|FOR1-SPECIFIC}

LINKAGE = STD
Es werden standardisierte ILCS-Schnittstellen erzeugt (Inter Language Communication Services = Standard Linkage). Nähere Informationen zum Standard-Linkage-Konzept enthält Kapitel 11.

LINKAGE = FOR1-SPECIFIC
Es werden herkömmliche FOR1-spezifische Schnittstellen erzeugt (wie bei FOR1-Versionen ≤ 2.1).

Eine Übersetzung mit dem (voreingestellten) Wert LINKAGE = STD ist nur möglich, wenn zusätzlich gilt:
- PROCEDURE-OPTIMIZATION = NO
- NOCOMPATIBLE [={BGFOR | BS3FOR}]

Werden unverträgliche Optionswerte angegeben, bestimmt die Eingabereihenfolge die tatsächlich verwendeten Werte:

Falls eine Option eingegeben wird, die zu einer vorher eingegebenen Option unverträglich ist, dann wird der Wert der vorher eingegebenen Option entsprechend angepaßt. So wird z.B. durch die Eingabe von PROCEDURE-OPTIMIZATION=YES oder COMPATIBLE=BGFOR eine zuvor gesetzte Option LINKAGE=STD auf LINKAGE=FOR1-SPECIFIC umgeschaltet.

In solchen Fällen gibt der COMPILER Warnungen aus:

- `MA 43 LINKAGE=FOR1-SPECIFIC EXPECTED`
  wird ausgegeben, falls vor Eingabe der zu LINKAGE=STD unverträglichen Option die LINKAGE-Option *nicht explizit* spezifiziert wurde, z.B. bei:
  ```
  *COMOPT SOURCE=TEST,COMPATIBLE=BGFOR,END
  ```

- `MA 20 ILLEGAL OPTION COMBINATION`
  wird ausgegeben falls neben der zu LINKAGE=STD unverträglichen Option LINKAGE=STD *explizit* angegeben wird, z.B. bei:
  ```
  *COMOPT SOURCE=TEST,LINKAGE=STD,COMPATIBLE=BGFOR,END
  ```

Es kann auch vorkommen, daß *beide* Meldungen ausgegeben werden, nämlich dann, wenn LINKAGE=STD *nach* der hierzu unverträglichen Option explizit angegeben wird, z.B. bei:

```
*COMOPT SOURCE=TEST,COMPATIBLE=BGFOR,LINKAGE=STD,END
```

Da der Compiler die angegebenen Compileroptionen linear auswertet, wird bei der Auswertung dieser COMOPT-Anweisung zunächst durch COMPATIBLE=BGFOR die zu diesem Zeitpunkt noch geltende Voreinstellung LINKAGE=STD in LINKAGE=FOR1-SPECIFIC geändert und MA 43 ausgegeben. Weil jedoch die explizite Angabe von LINKAGE=STD folgt, wird LINKAGE=FOR1-SPECIFIC wieder in LINKAGE=STD zurückgewandelt, NOCOMPATIBLE gesetzt und MA 20 ausgegeben.

Hinweis:

Eine Sonderbehandlung erfährt die Optionsbelegung PROCEDURE-OPTIMIZATION=STD. Falls die Option LINKAGE=STD explizit gesetzt wird, interpretiert der Compiler PROCEDURE-OPTIMIZATION=STD als PROCEDURE-OPTIMIZATION=NO. Auf diese Tatsache wird mit folgender Meldung hingewiesen:

```
MA 42 PROC-OPT=NO BECAUSE LINK=STD
```

4.2.2.7 UNIT-Option

`[*]COMOPT`	`UNIT=( {READ / WRITE / PRINT / PUNCH} = nn [, {READ / WRITE / PRINT / PUNCH} = nn]...)`

nn ganzzahliger Wert, 0 ≤ nn ≤ 99

Diese Option vereinbart Zuordnungen zwischen Ein-/Ausgabe-Anweisungen und Dateinummern. Für Ein-/Ausgabe-Anweisungen, in denen im FORTRAN-Quellprogamm keine Dateinummer spezifiziert ist, werden die in der UNIT-Option angegebenen Dateinummern gültig.

Zu dieser Compileroption gibt es keinen entsprechenden SDF-Operanden.

4.2.2.8 COMPATIBLE-Option

`[*]COMOPT`	`[NO[COMPATIBLE=]] {BGFOR / BS3FOR}`

Die COMPATIBLE-Option dient zur Vermeidung von Inkompatibilitäten (Unverträglichkeiten) zwischen FOR1-Compiler und Siemens-BGFOR- bzw. TR440-BS3-FORTRAN-Compiler.

COMPATIBLE = BGFOR:

1. OPTION NOTRUNCONST wird eingeschaltet.
2. OPTION PAD wird eingeschaltet.
3. Jede DO-Schleife wird mindestens einmal durchlaufen (nichtabweisende Schleife).
4. Für alle Dateien wird BLANK=’ZERO’ voreingestellt.

COMPATIBLE = BS3FOR:

Jede DO-Schleife wird mindestens einmal durchlaufen (nichtabweisende Schleife).

Zu dieser Compileroption gibt es keinen entsprechenden SDF-Operanden.

Hinweise:

Das Präfix NO bedeutet hier **nicht**, daß die Operanden-Komplementärmenge gemeint ist.
COMOPT COMPATIBLE und COMOPT LINKAGE=STD sind unverträglich (siehe 4.2.2.6).

4.2.2.9 SUPPLIEDBOUND-Option

[*]COMOPT	[NO]SUPPLIEDBOUND

Die Option SUPPLIEDBOUND bewirkt, daß in Unterprogrammen die Dimensionsangabe "1" eines eindimensionalen Feldes als "*" interpretiert wird. Dies gilt nicht für Felder, die als COMMON- Bereiche deklariert wurden.

Zu dieser Compileroption gibt es keinen entsprechenden SDF-Operanden.

4.2.2.10 PAD-Option

[*]COMOPT	[NO]PAD

Bei Angabe von COMOPT PAD werden Eingabesätze, die kürzer sind als im Programm festgelegt wurde, mit Leerzeichen aufgefüllt. Die Option COMPATIBLE=BGFOR schaltet implizit die Option PAD ein, sofern nicht explizit NOPAD angegeben wurde.

Zur dieser Compileroption gibt es keinen entsprechenden SDF-Operanden. Bei Verwendung des SDF-Kommandos START-FOR1-COMPILER ist PAD voreingestellt. Bei LANGUAGE-STANDARD=ANS77 wird NOPAD gesetzt.

Beispiel:

```
.
.
.
CHARACTER SATZ*100
READ (10,FMT='(A100)') SATZ
.
.
.
```

Hat der Eingabesatz eine Länge kleiner 100 byte, dann erfolgt eine I/O-Fehlermeldung, sofern nicht COMOPT PAD angegeben wurde:

```
IO3A: RECORD POINTER OUTSIDE I/O BUFFER
```

4.3 Bestimmen des Ausgabeorts des erzeugten Bindemoduls

4.3.1 SDF-Operand MODULE-LIBRARY

```
START-FOR1-COMPILER

,MODULE-LIBRARY = *OMF(...) / <full-filename 1..54>

   *OMF(...)
      DELETE-OLD-CONTENTS = YES / NO
```

Eine Gegenüberstellung von SDF-Operanden und entsprechenden Compileroptionen enthält Tab. 2-8.

4.3.2 Compileroption MODULE-LIBRARY

Die bei einem Übersetzungslauf erzeugten Bindemoduln können sowohl in die temporäre EAM-Datei der laufenden Task als auch in eine PLAM-Bibliothek ausgegeben werden. Die Ausgabe der Bindemoduln kann durch die MODULE-LIBRARY-Option gesteuert werden.

`[*]COMOPT`	`MODULE[-LIBRARY] = {*OMF / plamlib}`

*OMF Standardmäßig werden Bindemoduln in die temporäre EAM-Datei ausgegeben.

plamlib Name einer PLAM-Bibliothek, in die die erzeugten Bindemoduln ausgegeben werden sollen. Wenn die PLAM-Bibliothek noch nicht existiert, wird sie angelegt. *plamlib* darf aus maximal 54 Zeichen (Länge des vollqualifizierten Dateinamens) bestehen.

Jeder Bindemodul erhält als Bibliothekselement vom Typ R einen Namen, der aus dem Namen der Programmeinheit gebildet wird:

- Der Elementname ist der Name der Programmeinheit, wenn dieser nicht mehr als 7 Zeichen enthält.
- Der Elementname besteht aus den ersten 4 und den letzten 3 Zeichen des Namens der Programmeinheit, wenn dieser mehr als 7 Zeichen enthält.
- Der Elementname lautet $PU#nnn, beginnend mit nnn=000, wenn ein Hauptprogramm keine PROGRAM-Anweisung enthält.
- Für unbenannte BLOCK DATA-Unterprogramme werden Elemente mit den Namen der COMMON-Blöcke erzeugt.

Besteht bereits ein PLAM-Bibliothekselement desselben Namens, so wird es überschrieben, ohne daß eine Meldung erfolgt.

4.4 Struktur und Namensgebung von Bindemoduln

Bindemoduln werden entweder direkt in eine PLAM-Bibliothek abgelegt oder in den EAM-Bereich geschrieben.

Bindemoduln setzen sich aus 80 byte langen Sätzen zusammen. Diese Sätze lassen sich in fünf verschiedene Typen einteilen:

- ESD-Sätze (external symbol dictionary):
  Die ESD-Sätze enthalten Informationen über die externen Namen, die in dem Modul definiert werden oder auf die in dem Modul Bezug genommen wird, und über die COMMON-Bereiche, die in dem Modul auftreten. Die ESD-Sätze müssen zu Beginn eines Moduls stehen.
- TXT-Sätze (text):
  Die TXT-Sätze enthalten den zu ladenden Objektprogrammtext, d.h. die Speicherabbilder der Befehlsfolgen, Konstanten und Initialwerte von Daten.
- RLD-Sätze (relocation dictionary):
  Die RLD-Sätze enthalten Informationen über die Adreßkonstanten in dem Modul. Diese Adreßkonstanten müssen dann vom Binder bzw. vom dynamischen Bindelader entsprechend der Lage des Moduls modifiziert werden.
- LSD-Sätze (list for symbolic debugging): Die LSD-Sätze enthalten Informationen für das symbolische Testen in Objektmoduln. Diese Informationen sind für das Arbeiten mit der Dialogtesthilfe AID des BS2000 notwendig. LSD-Sätze werden nur erzeugt, wenn die Option SYMTEST=ALL oder der SDF-Operand TOOL-SUPPORT=AID angegeben wird.
- END-Satz:
  Der END-Satz wird immer am Ende des Bindemoduls generiert.

Das Format der Sätze wird in "Systemkonventionen" [39] ausführlich beschrieben.

Für jede Programmeinheit erzeugt FOR1 beim Übersetzungsvorgang einen oder mehrere Abschnitte (Control SECTions). Diese Abschnitte sind die kleinsten beim Binden verlagerbaren Einheiten.

Die von FOR1 erzeugten Abschnitte sind:

- Code- und Konstanten-Abschnitt
  Für jede Programmeinheit außer BLOCK DATA gibt es genau einen Code- und einen Konstanten-Abschnitt. Sie enthalten die erzeugten Befehlsfolgen und alle vom Anwender definierten und vom Compiler erzeugten Konstanten.
  - Bei der Übersetzung *ohne* COMOPT=(SHARE) wird *ein* Bindemodul mit dem Namen der Programmeinheit (prog) erzeugt, in dem Code- und Konstantenabschnitt mit enthalten sind (neben Datenabschnitt und ggf. COMMON-Abschnitten).
  - Bei der Übersetzung *mit* COMOPT=(SHARE) wird für Code- und Konstantenabschnitt ein gesonderter Bindemodul erzeugt. Der Name dieses Moduls ist der Name der Programmeinheit, mit dem Zeichen "@" auf Länge 8 aufgefüllt (prog[@]...).

  Für BLOCK DATA-Programmeinheiten gibt es keinen Code- und Konstanten-Abschnitt.
- Daten-Abschnitt
  Für jede Programmeinheit gibt es genau einen Daten-Abschnitt. Er enthält alle vom Anwender oder Compiler angelegten veränderlichen Daten und ihre Initialwerte.
  - Bei der Übersetzung *ohne* COMOPT=(SHARE) wird *ein* Bindemodul mit dem Namen der Programmeinheit erzeugt, in dem der Datenabschnitt mit enthalten ist (neben Codeabschnitt, Konstantenabschnitt und ggf. COMMON-Abschnitten).
  - Bei der Übersetzung *mit* COMOPT OBJECT=(SHARE) wird ein eigener Bindemodul für Datenabschnitt (und ggf. COMMON-Abschnitte) erzeugt, der den Namen der Programmeinheit trägt.
- COMMON-Abschnitt
  Für jeden COMMON-Block in einer Programmeinheit oder einer BLOCK DATA-Programmeinheit erzeugt FOR1 einen COMMON-Abschnitt, der den Namen des Commonblocks enthält. Dieser Abschnitt enthält alle Daten des COMMON-Blocks und ihre Initialwerte.
  - Bei der Übersetzung *ohne* COMOPT=(SHARE) wird *ein* Bindemodul erzeugt, in dem die COMMON-Abschnitte mit enthalten sind.
  - Bei der Übersetzung *mit* COMOPT=(SHARE) sind die COMMON-Abschnitte im selben Modul wie der Datenabschnitt.
  - Bei unbenannten BLOCK-DATA-Programmeinheiten wird für jeden COMMON-Abschnitt ein eigener Modul erzeugt.

4.5 Verwalten von Bindemoduln

Der FOR1-Compiler kann Bindemoduln direkt in PLAM-Bibliotheken ausgeben. Der Name der PLAM-Bibliothek wird in der MODULE-LIBRARY-Option festgelgt (siehe 4.3).

Ist die MODULE-LIBRARY-Option nicht spezifiziert, so gibt der Compiler die erzeugten Bindemoduln in die temporäre Bindemoduldatei (Object Modul File-OMF) im EAM-Bereich aus. Diese Datei besteht nur für die Dauer der Task. Man kann die Bindemoduln jedoch noch nachträglich aus dem EAM-Bereich in PLAM-Bibliotheken übertragen (siehe Beispiel).

LMS

Das Bibliotheksverwaltungsprogramm LMS ermöglicht das Erstellen und Bearbeiten von Programmbibliotheken (PLAM-Bibliotheken). In Programmbibliotheken lassen sich u.a. Quellprogramme, INCLUDE-Dateien, UPD-Dateien, Bindemoduln, Lademoduln, LLMs und Listen verwalten.

Nähere Informationen über den Funktionsumfang finden sich in Anhang A.10.4. Eine ausführliche Beschreibung enthält das Handbuch "LMS" [25].

Beispiel:

```
/ASSIGN-SYSDTA TO-FILE=QUELLE.TEST
/START-PROG $FOR1
/ASSIGN-SYSDTA TO-FILE=*PRIMARY                (1)
/START-PROG $LMS                               (2)
$LIB BIBL,USAGE=OUT,STATE=NEW                  (3)
$ADDR *OMF>ELEM1                               (4)
$END                                           (5)
```

Erläuterung des Beispiels:

(1) Vor Aufruf des LMS wird SYSDTA auf die Primärzuweisung zurückgelegt.
(2) Aufruf des LMS.
(3) Mit der LIB-Anweisung wird der Name der neuen Ausgabebibliothek zugewiesen.
(4) Mit der ADD-Anweisung wird der Modul aus dem EAM-Bereich als Element ELEM1 vom Typ R in die zugewiesene Ausgabebibliothek aufgenommen.
(5) Die END-Anweisung beendet den LMS-Lauf.

Das Bibliothekselement ELEM1 der Bibliothek BIBL kann mit dem statischen Binder TSOSLNK, dem Binder BINDER oder dem dynamischen Bindelader DBL weiterbearbeitet werden (siehe Kap. 5).

4.6 Erzeugen von Compilerlisten

Der Compiler erzeugt (falls gewünscht) beim Übersetzungsvorgang eine Reihe von Listen mit Informationen über die Struktur des Quell- und des Objektprogramms, Informationen über die Übersetzungsfehler und Informationen über den Ablauf der Übersetzung. Der Anwender kann durch Angabe von Compileroptionen oder SDF-Operanden die Ausgabe dieser Listen steuern. Er kann entscheiden, welche Listen ausgegeben werden und wohin diese Listen geschrieben werden. Inhalt und Druckbild der einzelnen Listen werden in Abschnitt 4.7 beschrieben, Beispiele für Compilerlisten befinden sich im Anhang A.6.

4.6.1 Steuern von Meldungen und Listen: SDF-Operand LISTING

```
START-FOR1-COMPILER

,LISTING = STD / NO / PARAMETER(...)

   PARAMETER(...)
      OPTIONS = YES / NO
      ,SOURCE = NO / YES(...)
         YES(...)
           INSERT-ERROR-WEIGHT = NOTE / WARNING / ERROR
      ,DIAGNOSTICS = NO / YES(...)
         YES(...)
           MINIMAL-WEIGHT = NOTE / WARNING / ERROR
      ,DATA-ALLOCATION-MAP = YES / NO
      ,CROSS-REFERENCE = NO / YES
      ,EXTERNAL-DICTIONARY = NO / YES
      ,ASSEMBLER-CODE = NO / YES
      ,SUMMARY = YES / NO
      ,OPTIMIZED-SOURCE = NO / YES
      ,SORTING = BY-PROG-UNIT / BY-LIST-TYPE
```

Fortsetzung>

Fortsetzung

```
,LAYOUT = STD / PARAMETER(...)

   PARAMETER(...)
     | LINES-PER-PAGE = 64 / <integer 20..255>
     |,PAGE-EJECT-STMT = ACCEPTED / IGNORED
     |,TEXT-SEPARATOR = '|' / '!'

,OUTPUT = *SYSLST / <full-filename 1..54> / *STD-FILE /
          *LIBRARY-ELEMENT(...)

   *LIBRARY-ELEMENT(...)
     | LIBRARY = <full-filename 1..54>
     |,ELEMENT-PREFIX = *NONE / <alphanum-name 1..38> (...)
     |    | VERSION = *UPPER-LIMIT / <alphanum-name 1..24>
```

Eine Gegenüberstellung von SDF-Operanden und entsprechenden Compileroptionen enthält Tab. 2-9.

4.6.2 Steuern von Meldungen und Listen durch Compileroptionen

4.6.2.1 MSGLEVEL-Option (Diagnosemeldungen in Abhängigkeit vom Fehlergrad)

`[*]COMOPT`	`MSGLEVEL = { N[OTE] | W[ARNING] | E[RROR] / (SOURCE={N|W|E}[,DIAG={N|W|E}]) / (DIAG={N|W|E}[,SOURCE={N|W|E}]) }`

Die Option steuert die Ausgabe der Diagnosemeldungen in der Quellprogrammliste oder in der Diagnoseliste in Abhängigkeit vom Fehlergrad. Es können fünf verschiedene Fehlergrade auftreten:

- N [OTE] Bemerkung
  Ursache von Bemerkungen sind nicht Fehler des FORTRAN-Quellprogramms im Hinblick auf die FOR1-Sprache. Bemerkungen melden z.B. Hinweise über Optimierungsmöglichkeiten. Da Bemerkungen nicht auf Fehlern des Quellprogramms beruhen, braucht der Anwender wegen Bemerkungen nichts an seinem Programm zu verändern, um einen einwandfreien Ablauf des übersetzten Programms zu erreichen. Bemerkungen werden in der Quellprogramm- oder Diagnoseliste nur ausgegeben, wenn in der MSGLEVEL-Option der Fehlergrad NOTE angegeben wurde.
- W [ARNING] Warnung
  Warnungen beziehen sich auf Stellen im Quellprogramm, die zwar sprachlich richtig sind, aber i.a. auf logische Fehler hindeuten. Warnungen werden z.B. bei spezifizierten, aber nicht verwendeten Datenelementen, bei möglichen Endlos-Schleifen, bei Anweisungen, die nie ausgeführt werden können, etc. ausgegeben. Beim Ablauf von Programmen, für die beim Übersetzen Warnungen ausgegeben werden, können Fehler auftreten, oder es kann unbeabsichtigtes Verhalten auftreten. Der Anwender sollte daher überprüfen, ob die angesprochenen Stellen im Quellprogramm seinen Absichten entsprechen oder auf Grund von Fehlern entstanden sind.
- E [RROR] Fehler
  Hier handelt es sich um Fehler des Quellprogramms im Hinblick auf die FOR1-Sprache. Die angezeigten Fehler können jedoch vom Compiler korrigiert werden, so daß Anweisungen, in denen ein Fehler auftritt, trotzdem übersetzt werden können. Die vom Compiler durchgeführten korrigierenden Maßnahmen werden im Diagnosetext (in der vollständigen Form) ausgegeben. Beispiele für Fehler sind fehlende schließende Klammern in arithmetischen Ausdrücken, zu lange Namen von Datenelementen, etc.
  Die vom Compiler durchgeführten korrigierenden Maßnahmen können den Sinn einer Anweisung eventuell verändern. Die Fehlerursachen sollen daher vom Anwender auf jeden Fall behoben werden.

- S [EVERE] schwerer Fehler
  Schwere Fehler werden gemeldet, wenn der Compiler Fehler erkennt, diese jedoch nicht sinnvoll korrigieren kann. Der Compiler trifft selbständig Maßnahmen, wie z.B. Zuweisung voreingestellter Werte, damit er weiter übersetzen kann. Im ungünstigsten Fall wird eine fehlerhafte FORTRAN-Anweisung ganz durch eine CONTINUE-Anweisung ersetzt.
- F [AILURE] Abbruchfehler
  Abbruchfehler führen zum sofortigen Abbruch des Übersetzungsvorgangs. Abbruchfehler können auf Grund von Compiler- und Systemfehlern auftreten (z.B. Adressierungsfehler).

Es werden alle Diagnosemeldungen vom in der MSGLEVEL-Option angegebenen Fehlergrad oder schwerwiegender ausgegeben.

MSGLEVEL	Möglicher Fehlergrad der ausgegebenen Meldungen
N	N, W, E, S, F
W	W, E, S, F,
E	E, S, F

4.6.2.2 LIST-Option (Auswahl der Listen)

In der LIST-Option wird angegeben, welche Listen in die Systemdatei SYSLST ausgegeben werden sollen. Mit der LIST-Option werden auch die Listen ausgewählt, die in die mit der LIST-OUTPUT-Option festgelegte Datei ausgegeben werden.

[*]COMOPT	[NO]LIST [=([listenangabe] [,...])]

listenangabe:= {ALL|MIN|OPTIONS|SOURCE|DIAG|ESD|MAP|XREF|
ATR|OBJECT|DECOMP|SUMMARY|CHANGE|NONE}

Ausgabe:

ALL	Sämtliche Listen mit Ausnahme der Decompilerliste
MIN	Optionenliste, Diagnoseliste und Überblicksliste
SOURCE	Quellprogrammliste
DIAG	Diagnoseliste
ESD	Liste der externen Namen
MAP	Datenadreßliste
XREF	Querverweisliste
ATR	Attributliste
OBJECT	Objektliste
DECOMP	Decompilerliste
SUMMARY	Überblicksliste
OPTIONS	Optionenliste
CHANGE	Liste der Änderungen (siehe "Interaktive Analyse"; Abschnitt 3.6)
NONE	Keine Listen

Bei mehreren LIST-Optionen gilt die zuletzt angegebene. Die Reihenfolge der Operanden hat keinen Einfluß auf die Reihenfolge der ausgegebenen Listen. Die Wechselwirkung von PARAMETER-Kommando und LIST-Option bzw. SDF-Operand LISTING muß gegebenenfalls beachtet werden. Die Bedeutung des NO-Präfixes wird in Abschnitt 2.3.1 erklärt. Die Decompilerliste wird nur bei OPTIMIZATION = 3 | 4 erzeugt und nur bei expliziter Angabe des Operandenwerts DECOMP ausgegeben. Sämtliche Listen werden somit nur durch LIST=(ALL,DECOMP) ausgegeben.

Folgende Fälle können unterschieden werden:

- COMOPT LIST mit angegebenen Operandenwerten:
  Die gewünschten Listen werden ausgegeben.
- COMOPT LIST ohne Operandenangabe:
  Es gelten die Standardparameter. Die Listen SOURCE, DIAG, MAP, SUMMARY und OPTIONS werden ausgegeben ("Standardlisten").

- COMOPT LIST=(NONE):
  keine Listenausgabe.
- COMOPT NOLIST mit angegebenen Operanden:
  Die den Operanden entsprechenden Listen werden unterdrückt; alle übrigen werden ausgegeben.
- COMOPT NOLIST ohne Operandenangabe:
  Die LIST-Option wird ausgeschaltet; keine Listenausgabe.

Bei den Compileroptionen NOGEN sowie NOOBJECT werden folgende Listen nicht erzeugt, da sie von der Generierung des Objektcodes abhängig sind und daher nicht ausgegeben werden können:
- Liste der externen Namen (ESD LISTING)
- Datenadreßliste (MAP LISTING)
- Querverweisliste (XREF LISTING)
- Attributliste (ATTRIBUTE LISTING)
- Objektliste (OBJECT LISTING)

Beispiele:

keine LIST-Option
: Keine Listenausgabe auf SYSLST.

COMOPT LIST
: Die den Standardparametern entsprechenden Listen werden auf SYSLST ausgegeben.

COMOPT LIST = (SOURCE,DIAG)
: Quellprogramm- und Diagnoseliste werden auf SYSLST ausgegeben.

COMOPT NOLIST=(ATR,OBJECT)
: Alle Listen - außer der Attribut- und der Objektliste - werden auf SYSLST ausgegeben.

COMOPT NOLIST oder COMOPT LIST = ( )
: Es werden überhaupt keine Listen auf SYSLST ausgegeben.

COMOPT NOLIST = ( )
: Alle Listen werden auf SYSLST ausgegeben.

4.6.2.3 COLLECT-Option (Ordnen der Listen)

```
              ⎧            ⎧L[IST]                      ⎫ ⎫
              ⎪COLLECT=(  ⎨                            ⎬)⎪
[*]COMOPT     ⎨            ⎩L[IST]F[ILE]  [,L[IST]]    ⎭ ⎬
              ⎪                                          ⎪
              ⎩NOCOLLECT                                 ⎭
```

LIST
Die mit der LIST-Option ausgewählten Listen werden für alle von einem Übersetzungsvorgang erfaßten Programmeinheiten gesammelt und nach Typ geordnet ausgegeben.

LISTFILE
Die mit der LISTFILE-Option ausgewählten Listen werden für alle von einem Übersetzungsvorgang erfaßten Programmeinheiten gesammelt und nach Typ geordnet ausgegeben.

NOCOLLECT
Die Listen werden nach Programmeinheiten geordnet ausgegeben.

Die COLLECT-Option ist nur dann wirksam, wenn die Listen unmittelbar über SYSLST ausgegeben, oder in eine ISAM-Datei abgelegt werden.

Wird in eine ISAM-Datei ausgegeben, dann werden die ausgegebenen Listen mit Hilfe des ISAM-Schlüssels sortiert. Wenn die COLLECT-Option gesetzt ist, wird in der ersten Stelle des Schlüssels folgende Kennzeichnung für den Typ der Liste eingetragen:

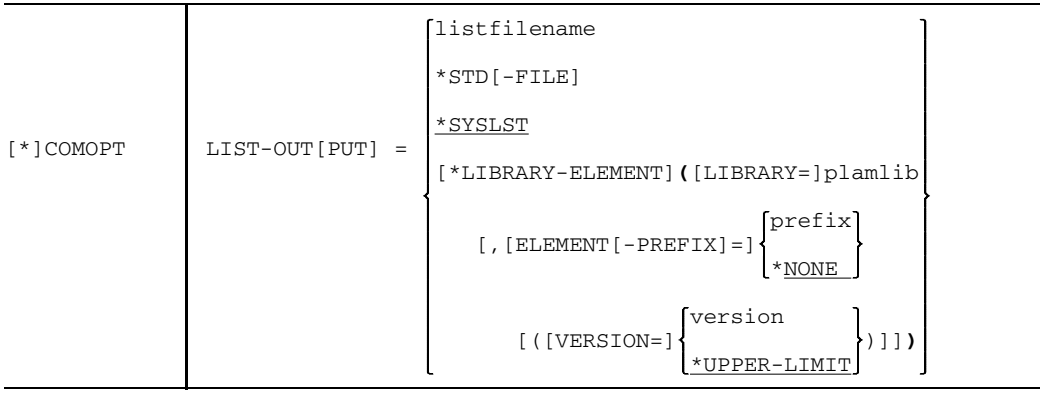
SOURCE	0	XREF	5
DIAG	1	ATR	5
CHANGE	2	OBJECT	6
ESD	3	DECOMP	7
MAP	4	SUMMARY	8
		OPTIONS	9

Attributliste und Querverweisliste (XREF) haben die gleiche Kennzeichnung.
Die Informationen der Attributliste sind vollständig in der Querverweisliste enthalten. Werden beide Listen verlangt, wird daher nur die Querverweisliste ausgegeben.

4.6.2.4 LIST-OUTPUT-Option (Ausgabeort der Listen)

Mit der LIST-OUTPUT-Option kann der Ort der Listenausgabe festgelegt werden. Die Art und Zusammensetzung der Listen wird mit der LIST-Option ausgewählt. Die LIST-OUTPUT-Option darf nicht gleichzeitig mit der LISTFILE-Option verwendet werden.

Wurde vor der Übersetzung einer Datei der LINK-Namen SAVLINK zugeordnet, dann werden die Listen in diese Datei ausgegeben und nicht in die mit der LIST-OUTPUT-Option angegebene Datei.



listfilename

Name einer katalogisierten ISAM-Datei, in die die Listen ausgegeben werden sollen. Die ISAM-Datei muß Sätze variabler Länge mit einer Schlüssellänge von 8 byte haben. Der Datei *listfilename* wird der LINK-Name SAVLINK zugeordnet, der nach Beendigung des Compilerlaufs wieder freigegeben wird.

*STD[-FILE]

Bei Angabe von *STD-FILE werden die Listen in eine Datei mit dem Namen `SAVLST.FOR1.prog[.tsn[.time]]` ausgegeben.
Der Dateiname wird wie folgt gebildet:

prog	Name der ersten Programmeinheit
tsn	vierstellige Task Sequence Number
time	Startzeit des Übersetzungsvorgangs in der Form: hhmmss

Die Qualifizierung des Dateinamens mit *tsn* und *time* wird nur durchgeführt, wenn es für die Eindeutigkeit des Katalogeintrags notwendig ist. Der Ausgabedatei wird der LINK-Name SAVLINK zugeordnet, der nach Beendigung des Compilerlaufs wieder freigegeben wird.

*SYSLST

Standardmäßig werden die Listen auf die Systemdatei SYSLST ausgegeben.

*LIBRARY-ELEMENT
Die Listen werden in ein PLAM-Bibliothekselement (vom Typ P) ausgegeben.

plamlib Name der PLAM-Bibliothek. Maximale Länge einschließlich Katalog- und Benutzerkennung: 54 Zeichen.

[ELEMENT-PREFIX=]
Teilqualifizierung des Elementnamens zur Hervorhebung einer Auswahl von Bibliothekselementen.

prefix Zeichenfolge, durch die eine Teilqualifizierung des Elementnamens definiert wird. *prefix* darf maximal 38 Zeichen lang sein.

*NONE Standardmäßig wird keine Teilqualifzierung vorgenommen.

[VERSION=]
Versionsbezeichnung des Bibliothekselements

version Zeichenfolge, die die Version bezeichnet; maximal 24 Zeichen lang.

*UPPER-LIMIT
Die erzeugten Listen werden mit der höchstmöglichen Versionsbezeichnung eingetragen.

Die Namensgebung der PLAM-Bibliothekselemente hängt von der mit der COLLECT-Option gewählten Sortierung der Listen ab:

– Falls die Listen nach Programmeinheiten geordnet werden (Voreinstellung) werden alle Listen eines Übersetzungslaufs in **einem** Bibliothekselement mit dem Namen *prog* abgelegt:

prog bei COMOPT OBJECT=(NOSHARE):
Name der ersten Programmeinheit bzw. $PU#000 bei Fehlen des Namens;
bei COMOPT OBJECT=(SHARE):
Name der ersten Programmeinheit, mit dem Zeichen "@" auf die Länge 8 aufgefüllt, bzw. $PU#000@ bei Fehlen des Namens.

– Falls die Listen nach Typ geordnet werden (Angabe von COLLECT=(LIST)) wird pro gewünschten Listentyp ein Bibliothekselement mit dem Namen *prog.type* angelegt:

prog.type
prog wird wie oben beschrieben gebildet, *type* gibt den Typ der im Bibliothekselement enthaltenen Listen an.

4.6.2.5 LISTFILE-Option (Ausgabe in katalogisierte Datei)

Mit der LISTFILE-Option kann die Ausgabe der Protokollisten in eine ISAM-Datei gesteuert werden. Die auszugebenden Listen können ausgewählt werden.
Die LISTFILE-Option darf nicht gleichzeitig mit der LIST-OUTPUT-Option verwendet werden.

[*]COMOPT	[NO]LISTFILE [=[listfilename] [({listenangabe / LIST} [,...])]]

listfilename

Name einer katalogisierten ISAM-Datei, in die die Listen ausgegeben werden sollen. Die ISAM-Datei muß Sätze variabler Länge mit einer Schlüssellänge von 8 byte haben. Wird *listfilename* nicht angegeben, oder entspricht die angegebene Datei nicht den Anforderungen, so wird vom Compiler eine eigene Datei mit folgendem Standardnamen erzeugt:

```
SAVLST.FOR1.prog[.tsn[.time]]
```

prog Name der ersten Programmeinheit
tsn vierstellige Task Sequence Number
time Startzeit des Übersetzungsvorgangs in der Form hhmmss

In der Optionenliste erscheint in diesem Fall statt des Standardnamens "*STD-FILE". Die Qualifizierung des Dateinamens mit *tsn* und *time* wird nur durchgeführt, wenn es für die Eindeutigkeit des Katalogeintrags notwendig ist.

listenangabe:= {ALL|MIN|OPTIONS|SOURCE|DIAG|ESD|MAP|XREF|
ATR|OBJECT|DECOMP|SUMMARY|CHANGE|NONE}

Die Operanden haben dieselbe Bedeutung wie in der LIST-Option. Die Angabe COMOPT LISTFILE=*listfilename* ohne Operanden bewirkt die Ausgabe der Listen OPTIONS, SOURCE, MAP, SUMMARY und gegebenenfalls DIAG in die Datei *listfilename*. Die Angabe COMOPT LISTFILE ohne Operanden bewirkt die Ausgabe dieser Listen in eine vom Compiler erzeugte Datei mit einem Standarddateinamen.

Falls der Anwender COMOPT COLLECT=(LF) angegeben hat, werden die Listen nach ihrem Typ zusammengefaßt ausgegeben.

LIST Jene Listen, die auf SYSLST ausgegeben werden, werden auch in die angegebene Datei *listfilename* geschrieben.

Die erzeugte Datei kann z.B. mit folgendem Kommando ausgedruckt werden:

```
/PRINT-FILE listfilename, FILE-PART=PAR(FROM=9), LAYOUT-CONTR=
PAR(CONTR-CHAR=EBCDIC)
```

4.6.2.6 LINECNT-Option (Zeilenzahl pro Seite)

[*]COMOPT	LINECNT = { 64 / zahl }

zahl ganzzahliger Wert

Die Option steuert die Anzahl der Zeilen, die pro Seite ausgegeben werden.
Die angegebene Zahl muß größer als 19 und kleiner als 256 sein (19 < zahl < 256).

4.6.2.7 EJECT-Option (Seitenvorschub)

[*]COMOPT	[NO]EJECT

Die Option beeinflußt die Durchführung eines Seitenvorschubs.

EJECT Jeder Seitenvorschub wird wirklich durchgeführt.

NOEJECT
Anstelle von Seitenvorschüben zwischen verschiedenen Listen werden jeweils drei Trennzeilen erzeugt. Seitenvorschübe auf Grund von %EJECT werden ignoriert.

4.6.2.8 EXPAND-Option (Auflistung von Einschüben)

[*]COMOPT	[NO]EXPAND

Die Option steuert die Ausgabe der mittels der %INCLUDE-Anweisung eingefügten Textstellen im Quellprogramm (EXPAND-Modus).

EXPAND
Die im Quellprogramm angegebenen %EXPAND-Anweisungen werden ausgeführt; gleichzeitig wird dadurch der EXPAND-Modus eingeschaltet.

NOEXPAND
Die im Quellprogramm angegebenen %EXPAND-Anweisungen verlieren ihre Wirkung; es werden keinerlei %INCLUDE-Einfügungen aufgelistet.

4.6.2.9 TEXT-SEPARATOR-Option (Darstellungsweise von senkrechten Linien)

[*]COMOPT	TEXT-SEPARATOR = { '|' / '!' }

Die Option legt fest, wie senkrechte Linien in Compilerlisten dargestellt werden.

'|' Senkrechte Linien werden in Compilerlisten durch das Zeichen "|" dargestellt. Im deutschen Zeichensatz entspricht dem Zeichen "|" das Zeichen "ö".

'!' Senkrechte Linien werden in Compilerlisten durch das Zeichen "!" dargestellt.

4.6.3 Steuern der Quellprogrammliste mit Compiler-Steueranweisungen

Das Druckbild der Quellprogrammliste wird mit den folgenden Compiler-Steueranweisungen (compile time statements) gesteuert, die in das Quellprogramm geschrieben werden. Die Wirksamkeit der Anweisungen %EJECT und %EXPAND hängt von den Optionen COMOPT EJECT bzw. COMOPT EXPAND ab.

Compiler-Steueranweisungen werden wie alle anderen FORTRAN-Anweisungen behandelt. Sie sind nur in der Programmeinheit gültig, in der sie angegeben sind.

Für Compiler-Steueranweisungen gilt im Unterschied zu FORTRAN-Anweisungen:
- Beendigung durch Kommentar- oder Leerzeile
- Schlüsselwort darf keine Leerzeichen enthalten

4.6.3.1 %EXPAND-Anweisung

```
%EXPAND  {ON }
         {   }
         {OFF}
```

%EXPAND ON
: Der EXPAND-Modus wird eingeschaltet, d.h. durch die %INCLUDE-Anweisung eingefügte Quellprogrammteile werden in der vom Compiler erzeugten Quellprogrammliste (Source-Text) mit aufgeführt.

%EXPAND OFF
: schaltet den EXPAND-Modus aus.

Eine %EXPAND-Anweisung wirkt auf %INCLUDE-Anweisungen der gleichen Schachtelungstiefe und vererbt sich auf %INCLUDE-Anweisungen tieferer Schachtelungsebenen. Es ist aber innerhalb einer tieferen Ebene möglich, den EXPAND-Modus **aus**zuschalten. Diese Einstellung gilt dann für die betreffende, sowie noch tiefere Schachtelungsebenen. Nicht möglich ist es jedoch, den EXPAND-Modus innerhalb einer tieferen Schachtelungsebene **ein**zuschalten.

Beispiel:

```
PROGRAM A

%EXPAND ON       Datei B         Datei C         Datei D
%INCLUDE B
%INCLUDE D
END              B1 = B2         C1 = C2         D1 = D2
                 %INCLUDE C      %EXPAND OFF
                                 %INCLUDE D
```

```
Resultat:       übersetztes     Quellprogrammliste      INCLUDE-Ebene
                Programm

                PROGRAM A         PROGRAM A
                %EXPAND ON        %EXPAND ON
                %INCLUDE B        %INCLUDE B
                B1 = B2           B1 = B2                  1
                %INCLUDE C        %INCLUDE C               1
                C1 = C2           C1 = C2                  2
                %EXPAND OFF       %EXPAND OFF              2
                %INCLUDE D        %INCLUDE D               2
                D1 = D2                                   (3)
                %INCLUDE D        %INCLUDE D
                D1 = D2           D1 = D2                  1
                END               END
```

Innerhalb der INCLUDE-Ebene 2 wird der EXPAND-Modus ausgeschaltet. Deshalb erscheint die INCLUDE-Ebene 3 nicht in der Quellprogrammliste.

4.6.3.2 %EJECT-Anweisung

```
%EJECT
```

Die Anweisung bewirkt unter der Option EJECT einen Seitenvorschub in der Quellprogrammliste.

4.6.3.3 %SPACE-Anweisung

```
%SPACE n
```

n ganzzahliger Wert: $0 \leq n \leq 255$

Die Anweisung bewirkt das Einfügen von n Leerzeilen in der Quellprogrammliste. Es werden aber höchstens so viele Leerzeilen eingefügt, wie noch auf dieselbe Seite passen. Die Leerzeilen werden in der Quellprogrammliste nur ausgedruckt, wenn im PRINT-FILE-Kommando CONTROL-CHARACTER=EBCDIC angegeben wird.

4.6.3.4 %TITLE-Anweisung

```
%TITLE ['text']
```

Die Wirkung der Anweisung ist wie bei %EJECT. Zusätzlich wird der angegebene Text auf jeder Seite der Quellprogrammliste derselben Programmeinheit in der Kopfzeile ausgegeben. Die Länge des Textes ist maximal 55 Zeichen. Soll innerhalb des Textes ein Apostroph stehen, so muß er zweimal geschrieben werden.

Steht eine %TITLE-Anweisung innerhalb der ersten n Zeilen eines Quellprogramms (n: 64 bzw. in der LINECNT-Option festgelegte Zeilenzahl), dann wird *text* auf der ersten Seite der Quellprogrammliste ausgegeben und es erfolgt kein Seitenvorschub.

4.7 Beschreibung der Compilerlisten

In diesem Abschnitt wird der Inhalt und das Druckbild der durch LIST und LISTFILE erzeugten Listen beschrieben. Die angegebenen Listen beziehen sich auf denselben Übersetzungsvorgang.

Bei allen Listen werden am Beginn jeder Seite zwei Überschriftszeilen geschrieben:

- In der ersten Zeile erscheinen eine Bezeichnung der Art der Liste, ein Titel, das Datum und die Uhrzeit des Beginns der Übersetzung sowie eine laufende Seitennumerierung. Bei der Quellprogrammliste kann der standardmäßig vordefinierte Titel "SIEMENS-NIXDORF FORTRAN COMPILER FOR1 Vn.nn" durch einen vom Anwender in der %TITLE-Anweisung im FORTRAN-Quellprogramm spezifizierten Titel ersetzt werden.
- In der zweiten Zeile wird der Name der Programmeinheit ausgegeben, auf die sich die Liste bezieht. Das Datum wird im ISO-Format yyyy-mm-tt ausgegeben.

4.7.1 Optionenliste (OPTIONS LISTING)

Die Ausgabe dieser Liste wird durch Angabe des Operandenwertes OPTIONS in der LIST- oder der LISTFILE-Option erreicht.

Standardmäßig wird die Optionenliste auf SYSLST ausgegeben. Die Optionenliste besteht aus drei Teilen:

- Umgebung der Task (environment)
- Liste der angegebenen Optionen (options-file)
- Liste der in Kraft befindlichen Optionen (options in effect)

Anhang A.6.10 zeigt eine Optionenliste.

In der Optionenliste erscheinen nur die durch COMOPTs festgelegten Operandenwerte, auch wenn durch %-Anweisungen (%INCLUDE, %FPOOL) andere Operandenwerte in Kraft sind.

Bei der Umgebung einer Task wird neben anderen Informationen die Task Sequence Number (TSN) der Task ausgegeben.
Diese vierstellige Zahl wird beim LOGON-Kommando einer Task zugeteilt und ist bis zum LOGOFF-Kommando gültig. Die TSN ist ein Bestandteil des vom Compiler für eine Datei generierten Namens, wenn vom Anwender kein Dateiname spezifiziert wurde.

In der Liste der angegebenen Optionen stehen auch die Meldungen über fehlerhafte Optionen.

In der Liste der in Kraft befindlichen Optionen werden jene Optionen, für die der Anwender keine oder keine gültige Angaben gemacht hat und für die daher die vordefinierten Standardwerte gelten, durch ein D (Default) gekennzeichnet. Ein P kennzeichnet Optionen, die durch das PARAM-Kommando (siehe A.4) aktiviert wurden.

Die Optionenliste erscheint am Ende der Gesamtliste.

4.7.2 Quellprogrammliste (SOURCE LISTING)

Die Ausgabe der Quellprogrammliste wird durch die Angabe des Operandenwertes SOURCE in der LIST- oder LISTFILE-Option erreicht.

Standardmäßig wird diese Liste auf SYSLST ausgegeben.
Die Abschnitte A.6.1 und A.6.2 zeigen Quellprogrammlisten.

Einträge in den Spalten der Quellprogrammliste haben die im folgenden dargestellte Bedeutung. Wird in einer Zeile für eine Spalte kein Eintrag gemacht, so gilt der zuletzt oberhalb gemachte Eintrag in dieser Spalte.

Spalte	**Bedeutung**
DO/IF	Schachtelungstiefe der DO-Schleifen (ausgenommen implizites DO) und des BLOCK IF
SEG	Nummer des höchsten Segments, das in der FORTRAN-Anweisung erreicht wird. Unter einem Segment versteht man die größtmögliche Folge von Anweisungen, die genau einen Eingangspunkt aufweist. Beim Ablauf des Programms werden immer alle Anweisungen eines Segments sequentiell durchlaufen. Die Durchnumerierung der Segmente eines übersetzten Programms beginnt mit 1. Eine Ein-/Ausgabe-Anweisung mit implizitem DO kann mehrere Segmente umfassen. Auf Grund der möglichen Verlagerung von Anweisungsteilen bei der Optimierung von Schleifen kann eine Anweisung aus dem ursprünglichen Segment in ein anderes geraten oder auf mehrere Segmente verteilt werden (siehe 9.3.5). In diesem Fall erscheint in der Quellprogrammliste keine der verlagerten Segmentnummern. Die angegebenen Segmentnummern dienen zur Interpretation der Ergebnisse der Testanweisung %COUNT (siehe 7.4.7). Falls die Compileroptionen OPT={3|4} oder SOURCE-FORMAT=FREE angegeben wurden, werden keine Segmentnummern ausgegeben.
STMT	Fortlaufende Nummer der FORTRAN-Anweisung, gezählt je Programmeinheit, beginnend bei 1 (keine Quellprogrammzeile).

I/H Schachtelungstiefe/Hierarchiestufe

I Schachtelungstiefe der durch die %INCLUDE-Anweisung eingefügten Textteile.

H Hierarchiestufe des PLAM-INCLUDE-Elements. H bezieht sich auf die in der INCLUDE-Option festgelegten Reihenfolge der PLAM-Bibliotheken.

So bezeichnet beispielsweise 1/3 die in der INCLUDE-Option an dritter Stelle genannte PLAM-Bibliothek. Das INCLUDE-Element hat die Schachtelungstiefe 1.

LINE Fortlaufende Zeilennummer, gezählt für die gesamte Übersetzung, beginnend bei 1. Bei Textteilen, die durch die %INCLUDE-Anweisung eingefügt wurden, wird die Zeilennumerierung separat wieder mit 1 begonnen.
Die Zeilennummern stimmen also in der Regel mit den laufenden Nummern in den betreffenden Bibliothekselementen überein und erleichtern damit Änderungen und Korrekturen.

Ausnahmen sind temporär geänderte Quellprogramme (hinter der ersten geänderten Zeile) und durch %INCLUDE eingefügte Elemente, wenn in ihnen auf Grund der Textsubstitution Folgezeilen erzeugt werden (ab der ersten erzeugten Folgezeile).

RECORD-ID. Schlüssel der ISAM-Sätze, falls das Quellprogramm von einer ISAM-Datei gelesen wurde.

Anzeige im Fehlerfall

Beim Übersetzungsvorgang werden vom Compiler möglicherweise Diagnosemeldungen erzeugt, die in der Quellprogrammliste erscheinen. (Meldungen über fehlerhafte Optionen stehen in der Optionenliste.) Die Diagnosemeldungen stehen unmittelbar hinter den Zeilen, auf die sie sich beziehen, sind aber so gestaltet, daß sie das Listenbild des eigentlichen Quellprogrammtextes nicht stören. Die betreffenden Anweisungen selber werden durch einen Stern an den beiden Rändern der Seite markiert. Falls sich die Diagnosemeldung auf eine bestimmte Spaltenposition in der Anweisung bezieht, wird in der Diagnosemeldung diese Spalte markiert. Die Markierung ist gleichzeitig eine Numerierung der Diagnosemeldungen für eine Anweisung. Diagnosemeldungen, die sich auf keine bestimmte Spaltenposition in der Anweisung beziehen, werden am Ende der Meldungen für diese Anweisung geschrieben.

Am linken Rand wird der Fehlergrad angezeigt und eine Fehlernummer ausgegeben.

Am rechten Rand wird ein kurzer Diagnosetext ausgegeben, der in der Regel zur Korrektur ausreicht.

Die auf Grund der Diagnose vom Compiler ergriffene Maßnahme findet sich im vollen Meldungstext in der Diagnoseliste.

Mit der MSGLEVEL-Option kann bestimmt werden, ab welchem Fehlergrad die Meldungen in der Quellprogrammliste erscheinen sollen.

4.7.3 Diagnoseliste (DIAGNOSTIC LISTING)

Die Ausgabe der Diagnoseliste wird durch Angabe des Parameterwertes DIAG in der LIST- oder LISTFILE-Option erreicht.

Standardmäßig wird die Diagnoseliste auf SYSLST ausgegeben. In der Diagnoseliste werden die Meldungen über fehlerhafte Optionen und alle beim Übersetzen eines Quellprogramms erzeugten Meldungen ausgegeben.

Anhang A.6.1 zeigt eine Diagnoseliste.

Die Anweisungen, auf die sich die Meldungen beziehen, werden in der Diagnoseliste in der gleichen Form wie in der Quellprogrammliste ausgegeben. Zusätzlich zur Diagnose werden in der Diagnoseliste die Korrekturmaßnahmen des Compilers aufgelistet.

Bezieht sich eine Meldung auf eine bestimmte Spalte einer Quellprogrammzeile, so wird die Spaltenposition durch eine Nummer in der Zeile darunter markiert. Beziehen sich mehrere Meldungen auf dieselbe Spalte einer Quellprogrammzeile, so wird die Spaltenposition nur einmal markiert, und zwar mit der Nummer, die der ersten dieser Meldungen entspricht.

4.7.4 Liste der externen Namen (ESD LISTING)

Die Ausgabe dieser Liste wird durch Angabe des Operandenwertes ESD in der LIST- oder LISTFILE-Option erreicht.

Standardmäßig wird diese Liste auf SYSLST ausgegeben. Ist COMOPT NOGEN oder COMOPT NOOBJECT gesetzt, kann keie ESD-Liste erzeugt werden, da sie von der Generierung des Objektcodes abhängig ist.

In dieser Liste stehen Informationen über die externen (bindefähigen) Namen der Programmeinheit. Sie dienen zur Verbindung der Programmeinheit mit anderen Programmeinheiten, Laufzeitroutinen und fremden Prozeduren. Die ausgegebenen ESD-Informationen entsprechen den ESD-Sätzen, die beim Übersetzungsvorgang erzeugt werden.

Anhang A.6.4 zeigt eine Liste der externen Namen.

Die Spalten der Liste der externen Namen haben folgende Bedeutung:

Spalte	**Bedeutung**
IDENTIFIER	Externer Name. Dieser kann entweder vom Anwender angegeben sein, z.B. in einer EXTERNAL-Anweisung, oder er kann vom Compiler erzeugt werden, wie z.B. der Name des Datenabschnitts für die Programmeinheit, oder er kann vorgegeben sein, wie z.B. der Eingangspunkt in einer vordefinierten Funktion. Falls ein vom Anwender angegebener Name länger als 7 Zeichen ist, wird er für den entsprechenden ESD-Satz auf 7 Zeichen verkürzt, indem nur die ersten 4 und die letzten 3 Zeichen genommen werden. In der Liste wird der verkürzte Name angegeben. Es liegt in der Verantwortlichkeit des Anwenders, daß bei dieser Verkürzung keine gleichen Namen entstehen. Innerhalb der Programmeinheit gilt jedoch der vollständige Name.
ESID	Nummer des externen Namens. Die externen Namen werden für jeden Modul mit 1 beginnend durchnumeriert.
TYPE	Art des externen Namens. Es gibt vier Arten von externen Namen: SD Abschnittsdefinitionen (section definition). Sie beziehen sich auf die Namen der Programmabschnitte (CSECTS) der Programmeinheiten. Die Namensgebung für die erzeugten Programmabschnitte ist in Abschnitt 4.4 beschrieben. CM Namen von COMMON-Bereichen außerhalb von BLOCK DATA-Programmeinheiten. (COMMON-Block-Anforderungen). Bei einem unbenannten COMMON-Bereich werden 8 Leerzeichen als Name genommen. VC V-Konstanten, externe Namen, die auf Code-Abschnitte und Code-Eingangspunkte anderer Programmeinheiten verweisen. Sie werden zum Aufruf von Unterprogrammen benutzt. LD Eingangspunkte (label definition), d.h. externe Namen, die in der Programmeinheit definiert werden (ENTRY-Anweisung). Beim Binden wird versucht, jeder Anforderung einer externen Definition (V-Konstante, Anforderung eines COMMON-Blocks) eine vorhandene Definition eines externen Namens (Abschnittsdefinition, Eingangspunkt) zuzuordnen.

DISPL — Distanz zum Modulbeginn bei Definitionen von externen Namen. Diese Distanz wird sedezimal in byte angegeben.
Bei Anforderungen von externen Definitionen wird die Distanz jener Stelle zum Modulbeginn angegeben, in die die Adresse des angeforderten COMMON-Blocks bzw. Unterprogramms abgelegt wird (Adreßkonstante).

LENGTH — Länge eines Abschnitts oder eines COMMON-Blocks, sedezimal in byte. Bei V-Konstanten und Eingangspunkten wird keine Längenangabe gemacht.

4.7.5 Datenadreßliste (MAP LISTING)

Die Ausgabe der Datenadreßliste wird durch Angabe des Operandenwerts MAP in der LIST- oder LISTFILE-Option erreicht. Standardmäßig wird diese Liste auf SYSLST ausgegeben. Ist COMOPT NOGEN oder COMOPT NOOBJECT gesetzt kann keine Datenadreßliste erzeugt werden, da sie von der Generierung des Objektcodes abhängt.

In der Datenadreßliste werden Informationen über die Anordnung der in der Programmeinheit verwendeten Daten unter den Spaltenüberschriften SYMBOL,TYPE und ADR ausgegeben. Die Datenadreßliste ist nach den Abschnitten, die für eine Programmeinheit erzeugt werden, unterteilt (CODE + CONSTANTS SECTION, LOCAL DATA SECTION, COMMON DATA SECTION). Anhang A.6.5 zeigt eine Datenadreßliste.

Für den Code- und Konstanten-Abschnitt werden Informationen über vorkommende Anweisungsmarken ausgegeben. Bei Anweisungsmarken, die in gesetzten oder berechneten Sprüngen vorkommen sowie bei aufgerufenen Funktionen und Unterprogrammen werden auch die Adressen, in denen die Adressen der oben angegebenen Elemente abgelegt sind, beschrieben. Für den Datenabschnitt werden Informationen über die Daten in dieser Programmeinheit, sowie bei Feldern und CHARACTER-Variablen auch über deren Deskriptoren ausgegeben.

Für den COMMON-Abschnitt werden nur Informationen über die Daten und über Deskriptoren von dynamischen Feldern im COMMON-Bereich ausgegeben. Die Informationen über die restlichen zugehörigen Deskriptoren befinden sich im Datenabschnitt.

Die Datenadreßliste wird in zwei Formen ausgegeben:
- alphabetisch sortiert
- nach steigender Distanz der Elemente zum Modulbeginn sortiert.

Die Spalten der Datenadreßliste haben folgende Bedeutung:

Spalte	**Bedeutung**
SYMBOL	Name des beschriebenen Elements Deskriptoren werden durch ein angehängtes .D, Adressen durch ein angehängtes .A gekennzeichnet.
TYPE	Eigenschaften der beschriebenen Elemente.
ADR	Distanz der beschriebenen Elemente zum Modulbeginn, sedezimal in byte.

4.7.6 Querverweisliste (XREF LISTING)

Die Ausgabe dieser Liste wird durch Angabe des Operandenwertes XREF in der LIST- oder LISTFILE-Option erreicht. Standardmäßig wird diese Liste nicht ausgegeben. Ist COMOPT NOGEN oder COMOPT NOOBJECT gesetzt, kann keine Querverweisliste erzeugt werden, da sie von der Generierung des Objectcodes abhängig ist.

Anhang A.6.6 zeigt eine Querverweisliste.

Die Querverweisliste liefert Informationen über Eigenschaften und Referenzen zu jedem symbolischen Namen und zu jeder Anweisungsmarke, die in der Programmeinheit vorkommen. Unter der Überschrift IDENTIFIER sind Namen nacheinander eingetragen:
- symbolische Namen, alphabetisch sortiert
- Anweisungsmarken, aufsteigend sortiert.

Zu jedem Namen gibt es unter den folgenden Spalten weitere Einträge:

Spalte	**Bedeutung**
DISPL	Distanz der beschriebenen Elemente zum Modulbeginn, sedezimal in byte.
DESCR	Distanz der Deskriptoren und Adressen zum Modulbeginn, sedezimal in byte.
SPEC	Anweisungsnummer, bei der der Name deklariert wurde. "*" bedeutet, daß der Name nicht explizit deklariert wurde.
ATTRIBUTES AND REFERENCES	1. Eigenschaften des Namens 2. Liste von Anweisungsnummern, bei denen der Name vorkommt. Vor manchen Nummern sind Markierungen eingetragen: – "/" besagt, daß der Name dort deklariert wird; es gibt drei Deklarationsmöglichkeiten (Typanweisung, DIMENSION-Anweisung, COMMON-Anweisung). – "=" besagt, daß sich der Wert der Variablen beim Programmablauf möglicherweise ändert (Zuweisung, aktuelles Argument, E/A-Liste usw.). Mehrfache Markierungen zur gleichen Anweisungsnummer zeigen entsprechend mehrfache Wertzuweisungen innerhalb einer einzigen Anweisung an. Bekommt ein- und dieselbe Anweisungsmarke sowohl markierte als auch nicht-markierte Einträge, so werden zuerst die unmarkierten ausgewiesen.

4.7.7 Attributliste (ATTRIBUTE LISTING)

Die Ausgabe dieser Liste wird durch Angabe des Parameterwertes ATR in der LIST- oder LISTFILE-Option erreicht. Standardmäßig wird diese Liste nicht ausgegeben. Ist COMOPT NOGEN oder COMOPT NOOBJECT gesetzt, kann keine Attributliste erzeugt werden, da sie von der Generierung des Objektcodes abhängt.

Die Informationen der Attributliste sind vollständig in der Querverweisliste enthalten. Werden beide Listen verlangt, wird daher nur die Querverweisliste ausgegeben.

4.7.8 Objektliste (OBJECT LISTING)

Standardmäßig wird diese Liste nicht ausgegeben.

Die Ausgabe dieser Liste wird durch Angabe des Operandenwertes OBJECT in der LIST- oder LISTFILE-Option erreicht.

Falls die Option NOGEN oder die Option NOOBJECT in Kraft ist, wird kein Objektprogrammtext erzeugt und es kann daher keine Objektliste ausgegeben werden.

In der Objektliste wird der vom Compiler erzeugte Objektprogrammtext für den Code- und Konstanten-Abschnitt in Form einer Assemblerliste ausgegeben. Da eine BLOCKDATA-Programmeinheit keinen Code- und Konstanten-Abschnitt enthält, wird für eine BLOCKDATA-Programmeinheit keine Objektliste ausgegeben.

Anhang A.6.7 zeigt eine Objektliste.

Der Aufbau der Objektliste entspricht den Listen, die vom Assembler erzeugt werden.

Die Spalten der Objektliste haben folgende Bedeutung:

Spalte	**Bedeutung**
FLG	Markierung der Befehlsfolgen, die für eine fehlerhafte FORTRAN-Anweisung erzeugt wurden. In der Kommentarzeile, die den Beginn der Befehlsfolge für eine FORTRAN-Anweisung anzeigt, steht in diesem Fall ein Stern.
DISPL	Distanz zum Modulbeginn, sedezimal in byte.
OPERATION	Objektprogrammtext in sedezimaler Form.
ADDR1,ADDR2	Adressen, die im Befehl verwendet werden. Diese Adressen sind die Summen der Inhalte der Basisregister und der Distanz. Sie geben die Distanz zum Modulbeginn an, entsprechen also den Einträgen in der Spalte DISPL.

STMNT Numerierung der Befehle.

ASSEMBLY CODE Erzeugter Assemblercode.
Die vorkommenden Zahlen werden dezimal ausgegeben.

SYMB.ADDR1, SYMB.ADDR2 Angabe der im FORTRAN-Quellprogramm verwendeten Namen.
Bei Konstanten wird der aufbereitete Wert ausgegeben. Die Form dieser Ausgabe hängt vom Typ der Konstanten ab.
Da die Ausgabe für einen Befehl die Zeilengrenze nicht übersteigt, können längere Konstanten abgeschnitten werden.
Vom Compiler erzeugte Größen haben folgende Form:

```
%Tnnnnnnnn      temporäre Hilfsgröße
                nnnnnnnn     Hexadezimalziffern

%Vnn            Compilervariable
                nn           Dezimalziffern

%Lnnnnn         interne Anweisungsmarke (Label)
                nnnnn        Dezimalziffern

%Inn            Iterationszähler bei DO-Schleifen
                nn           Dezimalziffern
```

Die Anweisungsmarken des FORTRAN-Quellprogramms erscheinen ebenfalls im Assemblercode, sie sind durch das vorangestellte Zeichen # gekennzeichnet.
Bei Verzweigung in einen anderen Modul wird beim Ladebefehl der entsprechenden Adresse der Modulname ausgegeben.

Zusätzlich werden in den Assemblercode Kommentarzeilen eingefügt. Diese Kommentarzeilen stehen am Beginn von Segmenten und FORTRAN-Anweisungen, sowie am Beginn von sogenannten Code-"Scheiben" (SLICE-Nr.), d.h. Code-Bereichen mit fester Stellung des Code-Basisregisters. Die Kommentarzeilen am Beginn einer Anweisung geben die Art der Anweisung an und das entsprechende FORTRAN-Statement aus dem FORTRAN-Quellprogrammm. Bei Anweisungen, die bei der Optimierung verlagert wurden, wird als Art der Anweisung MOVED STMT ausgegeben.

4.7.9 Decompilerliste (DECOMPILER LISTING)

Die Ausgabe dieser Liste wird nur durch explizite Angabe des Operandenwerts DECOMP in der LIST- oder LISTFILE-Option erreicht. Bei Angabe von (ALL) in der LIST- oder LISTFILE-Option wird die Decompilerliste nicht ausgegeben. Eine Voraussetzung für die Ausgabe der Decompilerliste ist, daß OPT=3 oder 4 angegeben wurde.

Die Optimierung hat Codeveränderungen (siehe Kap. 9) zur Folge, so daß beim Testen mit Dialogtesthilfen nicht mehr ohne weiteres auf das Quellprogramm Bezug genommen werden kann. Durch die Optimierung kann z.B. die Anweisungsreihenfolge verändert werden, eine Anweisung kann in mehrere Anweisungen aufgeteilt werden oder kann ganz wegfallen. Eine Ablaufverfolgung oder das Setzen von Testpunkten auf der Grundlage des Quellprogramms ist deshalb nicht mehr eindeutig möglich.

In der Decompilerliste können durch die Optimierung (bei OPTIMIZE= 3 oder 4) entstandene Änderungen dargestellt werden, da die Decompilerliste aus den Informationen erstellt wird, die die Codegenerierung von der Optimierung erhält. Mit der Decompilerliste liegt eine "Highlevel"-Beschreibung des Objektcodes vor, die das Testen eines optimierten Programms (das eigentlich nicht mehr testbar ist) erleichtert. Wann Variablenwerte in Register geladen oder wann und ob Registerinhalte gespeichert werden, läßt sich mit Sicherheit jedoch nur anhand der Objektliste nachvollziehen.

Die Decompilerliste weist gegenüber dem ursprünglichen Quellprogramm einige Besonderheiten auf:

1. Die Decompilerliste enthält nicht nur Anweisungen aus dem ursprünglichen Quellprogramm, sondern auch Variablen und Hilfsgrößen, die der Compiler intern anlegt. Zur Unterscheidung von den Variablen des FORTRAN-Quellprogramms beginnen die Namen der internen Variablen mit dem Zeichen "%":

```
%Tnnnnnnnn     temporäre Hilfsgröße
               nnnnnnnn    Hexadezimalziffern

%Vnn           Compilervariable
               nn          Dezimalziffern

%Inn           Iterationszähler bei DO-Schleifen
               nn          Dezimalziffern
```

   Diese internen Variablen werden unter denselben Namen in der Objekt-Liste aufgeführt.

   Neben den internen Variablen treten in der Decompiler-Liste interne Anweisungsmarken auf:

```
Lnnnnn         interne Anweisungsmarke (Label)
               nnnnn       maximal fünfstellige Dezimalzahl
```

   In der Objektliste wird eine interne Anweisungsmarke in der Form %Lnnnnn dargestellt.
   Für die internen Variablen und Hilfsgrößen wird keine LSD-Information erzeugt.

2. Alle Felder - mit Ausnahme von dynamischen Feldern - werden in der Decompilerliste als eindimensionale Felder deklariert. Die Ober- und Untergrenzen des Feldes werden relativ zu der gedachten Adresse von A(0,0,...,0) bestimmt (siehe Abschnitt 9.3.4) und in der Decompilerliste ausgegeben.

3. Die Adressierung eines Feldelements erfolgt bei allen Feldern so, als ob das Feld ein eindimensionales Feld wäre. In der Decompilerliste wird die Adressierung nicht über den Index eines Feldelements dargestellt, sondern über die Adressierung in byte, dividiert durch die Länge des Datentyps.

   Bei konstanten Indizes und konstanter Feldelementlänge wird in der Decompilerliste das Ergebnis dieser Division ausgegeben. (Im Objektcode wird diese Division nicht durchgeführt, weil auf Objektebene nur in byte adressiert wird.)

4. Bei der Optimierung entstehen durch die Indexexpansion (siehe 9.3.4) gemeinsame Teilausdrücke (Feldmultiplikatoren). Bei Feldern mit variablen Grenzen als Formalparameter und bei dynamischen Feldern sind die betreffenden Feldmultiplikatoren zur Übersetzungszeit noch nicht bekannt. Diese erst zur Laufzeit berechenbaren Feldmultiplikatoren werden in der Decompilerliste in folgender Form dargestellt:

```
%array02, %array03, ... , %arrayn
           Feldmultiplikatoren des Feldes mit dem Namen array

           array       Name eines dynamischen Feldes oder eines
                       formalen Feldes mit variabler Grenze

           n           Dimension des Feldes array
```

5. EQUIVALENCE- und DATA-Anweisungen sowie BLOCKDATA-Unterprogramme werden nicht zurückübersetzt, d.h. die Decompilerliste enthält diese Anweisungen nicht.

6. Ein-/Ausgabe-Anweisungen werden nicht in READ- oder WRITE-Anweisungen zurückübersetzt; stattdessen werden die Aufrufe der entsprechenden Laufzeitroutinen angezeigt.

7. Maschinenabhängige Optimierungen, d.h. Registerallokationen (das Halten von häufig verwendeten Variablen in Registern), werden nicht in der Decompilerliste ausgegeben.

Beispiele

Beispiel 1: Deklaration von Feldern

```
  ****  SOURCE  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER ...
                                                      PROGRAM UNIT: TEST
 DO/IF SEG   STMT  I/H LINE        SOURCE-TEXT
-------------------------------------------------------------------------
*      1/1      1          1       PROGRAM TEST
         1      2          2       INTEGER * 4 DYNARRAY(:,:)
         1                 3      *           ,ARRAY(-1:1,-2:2)
         1      3          4       END

  *** DECOMPILER  LISTING ***      SIEMENS-NIXDORF FORTRAN COMPILER ...
                                                      PROGRAM UNIT: TEST
    STMT  | DECOMPILED TEXT
-------------------------------------------------------------------------
       1  |       PROGRAM TEST
       1  |       ABNORMAL
       1  |       INTEGER  *  4 DYNARRAY (:,:)
          |*      *** DECLARATION OF ARRAY MULTIPLIERS ***
       1  |       INTEGER  * 4 %DYNARRAY02
          |*      *** END OF ARRAY MULTIPLIER DECLARATION ***
       1  |       INTEGER  *  4 ARRAY (-7:7)
          |***** STATEMENT 3 (END) ****************
       3  |       END
```

Für das dynamische Feld DYNARRAY(:,:) wird der Feldmultiplikator %DYNARRAY02 angelegt. Das Feld ARRAY(-1:1,-2:2) wird in der Decompilerliste als eindimensionales Feld dargestellt.

Beispiel 2: Felder in Unterprogrammen

```
 ****  SOURCE  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER ...
                                                    PROGRAM UNIT: SUBARR
DO/IF SEG   STMT  I/H LINE        SOURCE-TEXT
-------------------------------------------------------------------------
     1/1      1         1        SUBROUTINE SUBARR ( ARR1, ARR2, ARR3 )
       1      2         2        INTEGER * 4 ARR1(-5:5,M,N)
       1                3       *           ,ARR3(0:10,0:N)
       1      3         4        INTEGER * 2 ARR2(0:10,0:*)
       1      4         5        COMMON      M,N
       1      5         6        ARR1(1,1,1) = 5
       1      6         7        ARR2(I,J) = 5
       1      7         8        END

 *** DECOMPILER  LISTING ***      SIEMENS-NIXDORF FORTRAN COMPILER ...
                                                    PROGRAM UNIT: SUBARR
  STMT  DECOMPILED TEXT
-------------------------------------------------------------------------
     1         SUBROUTINE SUBARR ( /ARR1 /,/ARR2 /,/ARR3 /)
     1         ABNORMAL
     1         INTEGER  *  4 M
     1         INTEGER  *  4 N
     1         INTEGER  *  4 I
     1         INTEGER  *  4 J
     1         COMMON /   / M, N
       *       *** DECLARATION OF AUXILIARY VARIABLES ***
     1         INTEGER * 4 %T00010000
     1         INTEGER * 4 %T00010044
     1         INTEGER * 4 %T000100CC
     1         INTEGER * 4 %T00010110
     1         INTEGER * 4 %T00010154
       *****   STATEMENT 2 (DIMENSION) **********
     2         INTEGER * 4 ARR1(-5+
               .                  (11)+
               .                  (11*(M )) :
               .                  5+(11)*(M)+
               .                  (11*(M ))*(N))
       *       *** DECLARATION OF ARRAY MULTIPLIERS ***
     2         INTEGER * 4 %ARR103
     2         %ARR103 = 11*(M )
       *       *** END OF ARRAY MULTIPLIER DECLARATION ***
       *****   STATEMENT 2 (DIMENSION) **********
     2         INTEGER * 4 ARR3(0+0 :
               .                  10+(11)*(N))
       *****   STATEMENT 3 (DIMENSION) **********
     3         INTEGER * 2 ARR2(*)
       *****   STATEMENT 1 (ENTRY) **************
       *****   STATEMENT 5 (ASSIGNMENT) *********
     5         %T00010000=11+%ARR103
     5         %T00010044=%T00010000*4
     5         ARR1((4+%T00010044)/4)=5
       *****   STATEMENT 6 (ASSIGNMENT) *********
     6         %T000100CC=J*11
     6         %T00010110=I+%T000100CC
     6         %T00010154=%T00010110*2
     6         ARR2(%T00010154/2)=5
       *****   STATEMENT 7 (END) ****************
     7         END
```

ARR1(-5:5,M,N) ist ein formales Feld mit variablen Grenzen. ARR1 wird als eindimensionales Feld deklariert, wobei die Unter- und Obergrenze nach Abschnitt 9.3.4 folgendermaßen dargestellt werden:

$A_s=A_0 + l(s_1*m_0 + ... + s_n*m_{n-1})$

Die Grenzen werden relativ zu A_0 dargestellt:

```
ARR1( u1*m0                                   ARR1( -5*1
     +u2*m1                                         +1*11
     +u3*m2 ) :         entspricht                 +1* (11*M)) :
    ( o1*m0                                       (  5*1
     +o2*m1                                         +M*11
     +o3*m2 )                                       +N*(11*M))
```

Der Ausdruck 11*M erscheint hier mehrfach und wird als Feldmultiplikator %ARR103 genutzt.

Für ARR3 errechnen sich entsprechend die Grenzen

$(u_1*m_0 + u_2*m_1 : o_1*m_0 + o_2*m_1) = (0 + 0 : (10*1) + (N*11))$

Die Adressierung des Feldelements ARR1 erfolgt mit Hilfe von temporären Hilfsgrößen und mit Hilfe des Feldmultiplikators %ARR103 = 11*M. Die Adresse des Feldelements ARR1(1,1,1) relativ zur gedachten Adresse A0 ist

$l(s_1*m_0 + s_2*m_1 + s_3*m_2) = 4*(1*1 + 1*11 + 1*11*M)$

Mit %T00010000 = 11 + (11*M) ergibt sich ARR1(4 + %T00010044) = 5.
Der Index-Ausdruck "4+%T00010044" bezieht sich auf die Adressierung in byte, so daß dieser Wert durch die Feldelementlänge 4 dividiert werden muß, um den Indexlistenwert zu erhalten: ARR1((4+%T00010044)/4) = 5. Für ARR2(I,J) wird nach demselben Vorgehen ARR2(I*1+J*11) bestimmt. Die Hilfsgröße wird entsprechend der Länge der INTEGER-2-Größe mit 2 multipliziert und anschließend wieder dividiert, um den Indexlistenwert zu erhalten.

Beispiel 3: CHARACTER-Variable

```
 ****  SOURCE  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER ...
                                                  PROGRAM UNIT: CHAR
DO/IF SEG   STMT  I/H LINE        SOURCE-TEXT
-------------------------------------------------------------------
     1/1      1         1        SUBROUTINE CHAR ( CHAR1, CHAR2, CHAR3 )
       1      2         2        CHARACTER * (*)   CHAR1
       1      3         3        CHARACTER * (*,V) CHAR2
       1      4         4        CHARACTER * (N)   CHAR3
       1      5         5        COMMON      N, M
       1                6
       1      6         7        CHAR1(:)    = CHAR2(:M)
       1      7         8        CHAR3(M:) = CHAR1(M-1:M+3)
       1      8         9        END

 *** DECOMPILER  LISTING ***      SIEMENS-NIXDORF FORTRAN COMPILER ...
                                                  PROGRAM UNIT: CHAR
   STMT  DECOMPILED TEXT
-------------------------------------------------------------------
      1        SUBROUTINE CHAR ( /CHAR1 /,/CHAR2 /,/CHAR3 /)
      1        ABNORMAL
      1        CHARACTER* (*) CHAR1
      1        CHARACTER*( * ,V) CHAR2
      1        INTEGER  *  4 N
      1        INTEGER  *  4 M
      1        COMMON /   / N, M
      *      *** DECLARATION OF AUXILIARY VARIABLES ***
      1        INTEGER * 4 %T00010000
      1        INTEGER * 4 %T00010044
      1        INTEGER * 4 %T00010088
      1        INTEGER * 4 %T000100CC
      1        INTEGER * 4 %T00010110
      1        INTEGER * 4 %T00010154
      1        INTEGER * 4 %T00010198
      1        INTEGER * 4 %T000101DC
           ***** STATEMENT 4 (CHAR) ***************
      4        CHARACTER*(N) CHAR3
           ***** STATEMENT 1 (ENTRY) **************
           ***** STATEMENT 6 (ASSIGNMENT) *********
      6        CHAR1=CHAR2(:M)
           ***** STATEMENT 7 (ASSIGNMENT) *********
      7        %T00010110=M-1
      7        %T00010154=M+3
      7        CHAR3(M:)=CHAR1(%T00010110:%T00010154)
           ***** STATEMENT 8 (END) ****************
      8        END
```

Die aufgeführten temporären Hilfsgrößen werden für die Teilkettenverarbeitung benötigt. Die Anfangs- und Endposition der Teilkette werden durch temporäre Hilfsgrößen ersetzt.

Beispiel 4: Ausgabe-Anweisungen

```
 ****  SOURCE  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER    FOR1 V2.2A00
                                                    PROGRAM UNIT: IO
DO/IF SEG   STMT  I/H LINE        SOURCE-TEXT
-------------------------------------------------------------------------------
      1/1      1         1       SUBROUTINE IO ( IAR )
        1      2         2       INTEGER * 4 IAR(10),
        1                3      *           SCHRITTWEITE
        1      3         4       N = 2
        1      4         5       WRITE(10,FMT=99) N
        3      5         6       WRITE(2,FMT='(I4)') (IAR(J),J=1,10)
        3      6         7       RETURN
        3      7         8  99   FORMAT(I4)
        3      8         9       END

 *** DECOMPILER  LISTING ***      SIEMENS-NIXDORF FORTRAN COMPILER ...
                                                    PROGRAM UNIT: IO
   STMT   DECOMPILED TEXT
-------------------------------------------------------------------------------
      1         SUBROUTINE IO ( /IAR /)
      1         ABNORMAL
      1         INTEGER  *  4 IAR (1:10)
      1         INTEGER  *  4 SCHRITTWEITE
      1         INTEGER  *  4 N
      1         INTEGER  *  4 J
      1         INTEGER  *  4 %I1
            *     EXTERNAL PROC. REFERENCE IF@XICA
            *     EXTERNAL PROC. REFERENCE IF@XFCO
            *     EXTERNAL PROC. REFERENCE IF@XTCA
            *     *** DECLARATION OF AUXILIARY VARIABLES ***
      1         INTEGER * 4 %T00010154
      1         INTEGER * 4 %T00010220
            ***** STATEMENT 1 (ENTRY) **************
            ***** STATEMENT 3 (ASSIGNMENT) *********
      3         N=2
            *     ******** BEGIN OF I/O - STATEMENT *******
            ***** STATEMENT 4 (WRITE) **************
      4         CALL IF@XICA('FORMATTED,SEQUENTIAL␣FILE ,WRITE/ENCODE',
                .              UNIT      = 10,
                .              FMT       = '(I4)')
      4         CALL IF@XFCO(N)
      4         CALL IF@XTCA
            *     ******** END OF I/O - STATEMENT *******
            *     ******** BEGIN OF I/O - STATEMENT *******
            ***** STATEMENT 5 (WRITE) **************
      5         CALL IF@XICA('FORMATTED,SEQUENTIAL␣FILE ,WRITE/ENCODE',
                .              UNIT      = 2,
                .              FMT       = '(I4)')
            *     ******** END OF I/O - STATEMENT *******
            ***** STATEMENT 5 (MOVED STMT) *********
      5         %T00010220=4
      5         %I1=10
      5   L12   CALL IF@XFCO(IAR((%T00010220)/4))
            ***** STATEMENT 5 (INCR STMT) **********
      5         %T00010220=%T00010220+4
      5         %I1 = %I1- 1
      5         IF (%I1 .NE. 0) GOTO L12
      5   L13   CALL IF@XTCA
            ***** STATEMENT 6 (RETURN) *************
      6         RETURN
      6         END
```

Ein-/Ausgabeanweisungen werden durch die Aufrufe der entsprechenden Laufzeitroutinen dargestellt. Zusätzlich werden die jeweiligen Parameter der Laufzeitroutinen (Typ, Unit, Format, usw.) aufgeführt.

Beispiel 5: DO-Schleife

```
  ****  SOURCE  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER ...
                                                      PROGRAM UNIT: LOOP
 DO/IF SEG   STMT  I/H LINE        SOURCE-TEXT
-------------------------------------------------------------------------
*      1/1      1         1       SUBROUTINE LOOP (N)
         1      2         2       INTEGER * 4 IAR(10),
         1                3      .           SCHRITTWEITE
         1      3         4       DO 10,I=1,N,SCHRITTWEITE
   1     3      4         5  10      IAR(I) = 5
         3      5         6       END

  *** DECOMPILER  LISTING ***      SIEMENS-NIXDORF FORTRAN COMPILER ...
                                                      PROGRAM UNIT: LOOP
    STMT  | DECOMPILED TEXT
-------------------------------------------------------------------------
       1  |       SUBROUTINE LOOP ( N )
       1  |       ABNORMAL
       1  |       INTEGER  *  4 N
       1  |       INTEGER  *  4 IAR (1:10)
       1  |       INTEGER  *  4 SCHRITTWEITE
       1  |       INTEGER  *  4 I
       1  |       INTEGER  *  4 %I1
       1  |       INTEGER  *  4 %V1
       1  |       INTEGER  *  4 %V2
          |    *     *** DECLARATION OF AUXILIARY VARIABLES ***
       1  |       INTEGER * 4 %T00010000
       1  |       INTEGER * 4 %T00010044
       1  |       INTEGER * 4 %T000100CC
       1  |       INTEGER * 4 %T00010110
       1  |       INTEGER * 4 %T00010154
          |    ***** STATEMENT 1 (ENTRY) **************
          |    ***** STATEMENT 3 (DO) *****************
       3  |       %T00010000=N-1
       3  |       %T00010044=%T00010000+SCHRITTWEITE
       3  |       %I1=%T00010044/SCHRITTWEITE                    (1)
       3  |       IF (%I1 .LE. 0) GO TO L13
          |    ***** STATEMENT 4 (MOVED STMT) *********
       4  |    L14   %T00010110=4                                (2)
       4  |       %T00010154=SCHRITTWEITE*4
          |    ***** STATEMENT 4 (ASSIGNMENT) *********
       4  |    L3    CONTINUE                                    (7)
       4  |    10    IAR(%T00010110/4)=5                         (3)
          |    ***** STATEMENT 4 (INCR STMT) **********
       4  |       %T00010110=%T00010110+%T00010154               (4)
          |    ***** STATEMENT 4 (DOEND) **************
       4  |       %I1 = %I1- 1                                   (5)
       4  |       IF (%I1 .NE. 0) GOTO L3                        (6)
       4  |    L15   CONTINUE
          |    ***** STATEMENT 5 (END) ****************
       5  |    L13   END
```

In der Decompilerliste wird die Schleifeninitialisierung, Schleifensteuerung und Schleifenfortschaltung einer optimierten DO-Schleife sichtbar (vgl. Kap. 9):

(1) Die Anzahl der Schleifendurchläufe wird als
(Endwert - Anfangswert + Schrittweite) / Schrittweite
bestimmt. Dieser Wert wird mit Hilfe von temporären Hilfsgrößen berechnet und dem Iterationszähler %I1 zugewiesen.
(2) Der Anfangswert und die Schrittweite der Schleife werden vor dem Schleifenbereich zugewiesen und in byte ausgedrückt.
(3) Im Schleifenbereich wird die Laufvariable %T00010110 wieder durch die Länge dividiert, um den Indexlistenwert zu erhalten.
(4) Die Laufvariable wird um die Distanz zwischen den Feldelementen erhöht.
(5) Der Iterationszähler wird um 1 vermindert
(6) und auf Null geprüft.
(7) Ist %I1 ungleich Null, wird der Schleifenbereich erneut durchlaufen.

Beispiel 6: Optimierungen im Schleifenbereich

```
 ****  SOURCE  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER ...
                                                    PROGRAM UNIT: OPT
DO/IF SEG   STMT  I/H LINE        SOURCE-TEXT
-------------------------------------------------------------------------
     1/1      1         1        PROGRAM OPT
       1      2         2        DO 1 I=1,5
  1    1      3         3        L=7
  1    2      4         4        M=M+N*L
  1    2      5         5        K=I*3+L*4
  1    4      6         6      1 N=N*7+K
       4      7         7        END
*** DECOMPILER  LISTING ***      SIEMENS-NIXDORF FORTRAN COMPILER ...
                                                    PROGRAM UNIT: OPT
  STMT  DECOMPILED TEXT
-------------------------------------------------------------------------
     1        PROGRAM OPT
     1        ABNORMAL
     1        INTEGER  *  4 I
     1        INTEGER  *  4 L
     1        INTEGER  *  4 M
     1        INTEGER  *  4 N
     1        INTEGER  *  4 K
     1        INTEGER  *  4 %I1
         *    *** DECLARATION OF AUXILIARY VARIABLES ***
     1        INTEGER * 4 %T00010000
     1        INTEGER * 4 %T00010044
     1        INTEGER * 4 %T00010198
         ***** STATEMENT 2 (DO) *****************
         ***** STATEMENT 3 (MOVED STMT) *********
     3        L=7                                                  (1)
         ***** STATEMENT 5 (MOVED STMT) *********
         ***** STATEMENT 5 (MOVED STMT) *********
     5        %T00010198=31                                        (4)
     5        %I1=5
         ***** STATEMENT 3 (ASSIGNMENT) *********
         ***** STATEMENT 4 (ASSIGNMENT) *********
     4   L3   %T00010000=N*7                                       (2)
     4        M=M+%T00010000                                       (3)
         ***** STATEMENT 5 (ASSIGNMENT) *********
     5        K=%T00010198                                         (6)
         ***** STATEMENT 6 (ASSIGNMENT) *********
     6   1    N=%T00010000+K                                       (3)
         ***** STATEMENT 5 (INCR STMT) **********
     5        %T00010198=%T00010198+3                              (5)
         ***** STATEMENT 6 (DOEND) **************
     6        %I1 = %I1- 1
     6        IF (%I1 .NE. 0) GOTO L3
         ***** STATEMENT 7 (END) ****************
     7   L5   END
```

In diesem Beispiel (vgl. 9.4.1) werden mit OPT=3 weitergehende Optimierungen im Schleifenbereich durchgeführt, die die Decompilerliste wiedergibt:

(1) Die Zuweisung L=7 wird vor den Schleifenbereich gestellt.
(2) L wird durch die Konstante 7 ersetzt.
(3) Damit kann (N*7) als gemeinsamer Teilausdruck der Anweisungen 4 und 6 erkannt und durch %T00010110 ersetzt werden.
(4) Der Anfangswert von K für I=1 ist %T00010198=1*3+28=31.
(5) Die Multiplikation mit der Laufvariablen I in Anweisung 5 wird durch eine Addition ersetzt, in der als Inkrement jeweils 3 addiert wird. Dadurch wird die Fortschaltung der Laufvariablen überflüssig und
(6) die Berechnung des Wertes von K wird zu der Zuweisung der temporären Hilfsgröße %T00010198 vereinfacht.

4.7.10 Überblicksliste (SUMMARY LISTING)

Die Ausgabe dieser Liste wird durch Angabe von SUMMARY in der LIST- oder LISTFILE-Option erreicht.

Standardmäßig wird diese Liste auf SYSLST ausgegeben.

Anhang A.6.8 zeigt eine Überblicksliste.

Wenn mehrere Programm-Einheiten in einem Lauf übersetzt werden, wird zusätzlich zur Überblicksliste für jede Programm-Einheit nach Beendigung der Übersetzung der letzten Programm-Einheit eine Gesamt-Überblicksliste erstellt, die die statistischen Daten für die Gesamt-Übersetzung enthält, wie

- Liste aller erzeugten Moduln
- Summen der jeweiligen Fehler einer Fehlerklasse
- gesamte CPU- und ELAPSED-Zeit etc.
- den Hinweis "(COMPILER NOT PRELOADED)", wenn der Compiler nicht vorgeladen wurde.

Die Gesamt-Überblicksliste ist genauso aufgebaut wie die Überblicksliste.

Anhang A.6.9 zeigt eine Gesamt-Überblicksliste.

4.7.11 Liste der Änderungen (CHANGE LISTING)

Die Liste der Änderungen enthält alle Kommando-Eingaben sowie ersetzte und eingefügte Quellprogrammzeilen.

Die Ausgabe dieser Liste wird durch Angabe von CHANGE in der LIST- oder LISTFILE-Option erreicht.

CHANGE ist nur wirksam in Verbindung mit COMOPT DIALOG.

Anhang A.6.3 zeigt eine Liste der Änderungen. Sie wurde im Beispiel im Abschnitt 3.6.8 erzeugt.

4.8 Abbruch des Übersetzungslaufs

4.8.1 SDF-Operand COMPILER-TERMINATION

```
START-FOR1-COMPILER

,COMPILER-TERMINATION = STD / PARAMETER(...)
   PARAMETER(...)
      CPU-LIMIT = NONE / <integer 1..32767>
      ,MAX-ERROR-WEIGHT = NONE / ERROR / SEVERE-ERROR
      ,MAX-ERROR-NUMBER = 100 / <integer 1..2147483639>
```

Eine Gegenüberstellung von SDF-Operanden und entsprechenden Compileroptionen enthält Tab. 2-12.

4.8.2 Abbruch des Übersetzungslaufs: Compileroptionen ERRKILL und MAXERR

ERRKILL-Option

[*]COMOPT	ERRKILL = { E [RROR] / S [EVERE] / F [AILURE] }

Die ERRKILL-Option legt fest, daß die Übersetzung eines Programms bei Auftreten eines Fehlers vom Fehlergrad der ERRKILL-Option oder schwerwiegender abgebrochen werden soll.

ERRKILL	Abbruch bei Fehler vom Grad
E	E,S,F
S	S,F
F	F

MAXERR-Option

[*]COMOPT	MAXERR = { 100 / n }

n ganzzahliger Wert ≤ 2^{31}-1

Nach Auftreten der in der Option angegebenen Anzahl von Fehlern (ERRORs) wird der Übersetzungsvorgang abgebrochen.

4.9 Überwachen des Übersetzungslaufs durch Jobvariable: SDF-Operand MONJV

START-FOR1-COMPILER
,MONJV = *NONE / <full-filename 1..54>

Eine Gegenüberstellung von SDF-Operanden und entsprechenden Compileroptionen enthält Tab. 2-13.

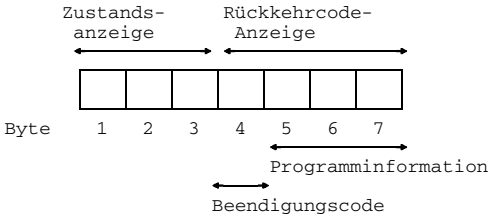
Bedeutung der Anzeigen von Jobvariablen

Mit Hilfe des Softwareprodukts JV (Jobvariablen) können Aufträge und Programme im BS2000 überwacht und gesteuert werden (siehe "Jobvariablen" [24]). Der Anwender definiert eine überwachende Jobvariable, die er bei einem LOGON-, ENTER-JOB oder START-PROGRAM-Kommando als Operand angibt. Das Betriebssystem trägt in diese Jobvariable Informationen über den aktuellen Zustand eines Programms ("Zustandsanzeige") und weitere auf Programmebene definierte Informationen ("Rückkehrcode-Anzeige") ein. Nach dem Programmende können diese Informationen vom Anwender abgefragt werden; weitere Aufträge und Programme können in Abhängigkeit von diesen Informationen gesteuert werden.

Eine Jobvariable kann sowohl eine FOR1-Übersetzung als auch den Ablauf eines FORTRAN-Programms (siehe Abschnitt 6.5.3) überwachen. Zustandsanzeige und Rückkehrcode-Anzeige der überwachenden Jobvariablen werden von FOR1 bzw. vom FORTRAN-Objektprogramm aus mit Parametern des TERM-Makros versorgt.

Die Zustandsanzeige der Jobvariablen wird in Abhängigkeit vom Parameter des MODE-Operanden, die Rückkehrcode-Anzeige in Abhängigkeit vom Parameter des URETCD-Operanden gesetzt (siehe "Makroaufrufe an den Ablaufteil" [26]).

Die Jobvariablen für die Programmüberwachung haben folgenden Aufbau:



Die Zustandsanzeige wird linksbündig in den ersten 3 Bytes der Jobvariablen gesetzt.

$T␣ Das Programm wurde normal beendet.

$A␣ Das Programm wurde nicht normal beendet. Diese Anzeige wird ebenfalls bei Programmabbruch durch das System gesetzt.

$R␣ Nach dem Starten eines Programms wird die Zustandsanzeige auf "$R" gesetzt.

Die Rückkehrcode-Anzeige wird in Byte 4 bis 7 der Jobvariablen eingetragen. Das erste Byte der Rückkehrcode-Anzeige enthält den Beendigungscode. Bei fehlendem URETCD-Parameter des TERM-Makros (siehe "Makroaufrufe an den Ablaufteil" [26]) oder bei Programmabbruch durch das System werden 4 Blanks in die Rückkehrcode-Anzeige eingetragen.

Nach einem FOR1-Übersetzungslauf kann der Beendigungscode folgende Informationen enthalten:

0 Der Compiler hat den Übersetzungslauf normal beendet. Es wurden keine Warnungen oder Fehler festgestellt.

1 Der Compiler hat den Übersetzungslauf normal beendet. Es wurden jedoch Warnungen oder Fehler (ERRORS, SEVERE ERRORS) bei der Übersetzung des Quellprogramms gemeldet. Die Ergebnisse der Übersetzung sind nur bedingt verwendbar.

2 Der Compiler ist fehlerfrei gelaufen. Die Übersetzung wurde jedoch vorzeitig aufgrund der Angaben in der MAXERR- oder ERRKILL-Option abgebrochen. Die Ergebnisse der Übersetzung sind nur bedingt verwendbar.

3 Der Compiler hat den Übersetzungslauf wegen eines Compilerfehlers definiert beendet. Ergebnisse des Übersetzungslaufs sind nicht vorhanden oder nicht verwertbar.

Bytes 5 bis 7 des Rückkehrcodes enthalten die Programminformation. Nach einem FOR1-Übersetzungslauf kann die Programminformation folgende Angaben enthalten:

000 Der Compiler hat den Übersetzungslauf normal beendet. Bei der Übersetzung des Quellprogramms wurden keine Fehlermeldungen (NOTE, WARNING, ERROR, SEVERE oder FAILURE) ausgegeben.

001 Der Compiler hat den Übersetzungslauf normal beendet. Es sind Bemerkungen (NOTES) gemeldet worden.

002 Der Compiler hat den Übersetzungslauf normal beendet. Es sind Warnungen (WARNINGS) gemeldet worden.

003 Der Compiler hat den Übersetzungslauf normal beendet. Es sind Fehler (ERRORS) gemeldet worden, für die eine Korrektur unternommen wurde.

004 Der Compiler hat den Übersetzungslauf normal beendet. Es sind schwere, nicht korrigierbare Fehler (SEVERE ERRORS) aufgetreten.

006 Der Compiler hat den Übersetzungslauf wegen schwerer Fehler kontrolliert beendet. Objektprogramme sind nicht vorhanden oder nicht verwertbar.

Die folgende Tabelle zeigt die Zuordnung zwischen Zustandsanzeige, Beendigungscode und Programminformation:

Zustands-anzeige	Beendigungs-code	Programm-information	Bemerkung
$T␣	0	000	Keine Fehler, WARNINGS, NOTES bei der Übersetzung.
$T␣	0	001	Es wurden NOTES bei der Übersetzung gemeldet.
$T␣	1	002	Es wurden WARNINGS bei der Übersetzung gemeldet.
$T␣	1	003	Es wurden ERRORS bei der Übersetzung gemeldet.
$T␣	1	004	Es wurden SEVERE ERRORS bei der Übersetzung gemeldet.
$A␣	2	003	Die Übersetzung wurde wegen ERRKILL=E oder Erreichen des MAXERR-Limits vorzeitig abgebrochen.
$A␣	2	004	Die Übersetzung wurde wegen ERRKILL=S oder Erreichen des MAXERR-Limits vorzeitig abgebrochen.
$A␣	3	006	Die Übersetzung wurde wegen eines Compilerfehlers vorzeitig definiert abgebrochen (FATAL ERRORS).
$A␣	- (undefiniert)	—— (undefiniert)	Die Übersetzung wurde wegen eines schwerwiegenden Compilerfehlers undefiniert abgebrochen (DUMP).

Tab. 4.1: Zustandsanzeige, Beendigungscode und Programminformation von Jobvariablen

Beispiel:

Mit Hilfe der Jobvariablen JOBVAR1 wird ein FOR1-Lauf überwacht. Der Binder soll nach der Übersetzung nur aufgerufen werden, wenn der Compiler überhaupt keine Meldungen oder nur NOTES ausgegeben hat.

```
/BEGIN-PROC LOG=C, PAR=YES(PROC-PAR=(&PROGRAM), ESC-CHAR=C'&')
/REMARK UEBERSETZEN UND BINDEN EINES FORTRAN-PROGRAMMS
/DEL-SYS-FILE OMF
/ASSIGN-SYSDTA TO-FILE=*SYSCMD
/CRE-JV JOBVAR1                                                    (1)
/START-PROG $FOR1, MONJV=JOBVAR1                                   (2)
*COMOPT SOURCE=QUELLE.&PROGRAM
*COMOPT END
/SET-JOB-STEP
/SHOW-JV JV-NAME(JOBVAR1)                                          (3)
/SKIP-COMM TO-LABEL=ENDE, IF=JV(CONDITION=(JOBVAR1,5,3)>'001')     (4)
/START-PROG $TSOSLNK
PROG LADE.&PROGRAM
INCLUDE *
RESOLVE ,$FOR1MODLIBS
END
/.ENDE DEL-JV JV-NAME(JOBVAR1)                                     (5)
/END-PROC
```

Erläuterung des Beispiels:

(1) Die Jobvariable JOBVAR1 wird in den Katalog eingetragen.

(2) Mit dem START-PROGRAM-Kommando wird die Jobvariable JOBVAR1 dem aufzurufenden Programm als programmüberwachende Jobvariable zugeordnet.

(3) Durch das SHOW-JV-Kommando wird der Wert der Jobvariablen auf SYSOUT ausgegeben.

(4) Mit dem SKIP-COMMANDS-Kommando wird geprüft, ob die Programminformation (5. bis 7. Byte) der Jobvariablen einen Wert größer "001" enthält: in diesem Fall hat der Compiler Warnungen oder Fehler gemeldet. Trifft diese Bedingung zu, dann wird zur Anweisung mit der Marke ".ENDE" verzweigt.

Das Ablaufprotokoll hat beim Auftreten von ERRORS folgendes Aussehen:

```
.
.
.
(IN)     /SHOW-JV JV-NAME(JOBVAR1)
(OUT)    %$T 1003
(  )
(IN)     /SKIP-COMM TO-LABEL=ENDE, IF=JV(CONDITION=(JOBVAR1,5,3)>'001')
(OUT)    %  CJC0010 SKIP COMMANDS: CONDITION = TRUE
(IN)     /.ENDE DEL-JV JV-NAME(JOBVAR1)
(IN)     /END-PROC
.
.
```

Sind keine Fehlermeldungen oder nur NOTES (Programminformation ’000’ oder ’001’) ausgegeben worden, dann wird der Binder aufgerufen. Das Ablaufprotokoll hat in diesem Fall folgendes Aussehen:

```
.
.
.
(IN)      /SHOW-JV JV-NAME(JOBVAR1)
(IN)      /GETJV (JOBVAR1,1),CHAR
(OUT)     %$T 0000
(  )
(IN)      /SKIP-COMM TO-LABEL=ENDE, IF=JV(CONDITION=(JOBVAR1,5,3)>’001’)
(OUT)     %  CJC0011 SKIP COMMANDS: CONDITION = FALSE
(IN)      /START-PROG $TSOSLNK
.
.
```

(5) Mit dem DELETE-JV-Kommando wird der Jobvariablen-Eintrag im Katalog gelöscht.

4.10 Einstellen der Meldungssprache

4.10.1 SDF-Operand LANGUAGE

START-FOR1-COMPILER
,LANGUAGE = ENGLISH / DEUTSCH

Eine Gegenüberstellung von SDF-Operanden und entsprechenden Compileroptionen enthält Tab. 2-14.

4.10.2 Compileroption LANGUAGE

[*]COMOPT	LANGUAGE={ENGLISH|GERMAN}

ENGLISH

Ab dem Einlesen der Compileroptionen werden Meldungen des FOR1 in englischer Sprache ausgegeben.

GERMAN

Ab dem Einlesen der Compileroptionen werden Meldungen des FOR1 in deutscher Sprache ausgegeben.

4.11 SDF-Operand COMPILER

Als letzter Operand des SDF-Kommandos START-FOR1-COMPILER steht der Operand

```
COMPILER = $FOR1 / <full-filename 1..54>
```

zur Verfügung. Er bietet die Möglichkeit, den Dateinamen des Compilers anzugeben, wenn dieser von der voreingestellten Bezeichnung abweicht.

Der Operand COMPILER ist im geführten Dialog nicht sichtbar und kann nur im NO- oder EXPERT-Modus angegeben werden.

5 Binden, Laden und Starten

Durch den FOR1-Compiler wird ein FORTRAN-Quellprogramm in einen Bindemodul (= zu bindender Modul) übersetzt. Bindemoduln stehen in einer PLAM-Bibliothek, einer Bindemodulbibliothek oder in der temporären EAM-Datei (*OMF) zur Verfügung.

Bindemoduln bestehen bereits aus Maschinencode, können aber in dieser Form noch nicht ablaufen, da der Maschinencode noch nicht vollständig ist. Jeder Bindemodul enthält Verweise auf externe Adressen (Externverweise), d.h. auf weitere Moduln, die ihn zur Ausführung ergänzen müssen.

Die zusätzlich benötigten Moduln sind Moduln des Laufzeitsystems (siehe Abschnitt 1.9) und eventuell weitere Bindemoduln, wie z.B. getrennt übersetzte Quellprogramme oder Unterprogramme in anderen Sprachen.

Die wichtigste Funktion des Binders besteht darin, die für den Lademodul erforderlichen Bindemoduln aus den verschiedenen Quellen (Dateien, Bibliotheken) abzurufen und miteinander zu verknüpfen. Die Verknüpfung besteht darin, daß der Binder jeden Bindemodul um diejenigen Adressen ergänzt, die sich auf Bereiche außerhalb des Bindemoduls beziehen.

Diesen Vorgang nennt man **Binden**. Als Ergebnis des Bindens entsteht ein Lademodul (= zu ladender Modul) oder ein Bindelademodul (LLM). Diese müssen noch in den Hauptspeicher geladen und gestartet werden. Bindelademoduln werden in diesem Handbuch nicht beschrieben, eine ausführliche Beschreibung befindet sich im Handbuch "Binder-Lader-Starter" [13].

Für das Binden, Laden und Starten gibt es folgende Möglichkeiten:

Statischer Binder TSOSLNK

TSOSLNK bindet einen oder mehrere Bindemoduln zu einem Lademodul und speichert diesen Lademodul in einer katalogisierten Datei oder in einer PLAM-Bibliothek (als Element vom Typ C) ab.
Um vom TSOSLNK erzeugte Lademoduln ausführen zu können, müssen sie mit dem Lader ELDE in den Hauptspeicher geladen werden.

Binder BINDER

Der BINDER bindet Moduln (Bindemoduln, Bindelademoduln) zu einer logisch und physikalisch strukturierten ladbaren Einheit zusammen. Diese Einheit bezeichnet man als Bindelademodul (Link and Load Module, LLM). Der BINDER speichert Bindelademoduln in PLAM-Bibliotheken (Elementtyp L) ab.
Der BINDER steht ab Betriebssystemversion V10 zur Verfügung, eine genaue Beschreibung befindet sich im Handbuch "Binder-Lader-Starter" [13].

Dynamischer Bindelader DBL

DBL bindet in einem Arbeitsgang Moduln (Bindemoduln, Bindelademoduln) zu einer temporär ladbaren Einheit, lädt diese in den Hauptspeicher und startet sie. Das gebundene Programm steht nach der Ausführung nicht mehr zur Verfügung. Dynamisches Binden und Laden ist vor allem in der Testphase zweckmäßig.

SDF-Kommando START-FOR1-PROGRAM

Die Operanden dieses Kommandos steuern die wichtigsten Funktionen des dynamischen Bindeladers DBL und des statischen Laders ELDE. Es können Bindemoduln und Lademoduln verarbeitet werden.

5.1 Binden, Laden, Starten: SDF-Kommando START-FOR1-PROGRAM, Operand FROM-FILE

Der Operand FROM-FILE des SDF-Kommandos START-FOR1-PROGRAM bietet folgende Möglichkeiten:

– Ein mit FOR1 erzeugter Bindemodul kann mit DBL gebunden, geladen und gestartet werden (*MODULE(...)/<full-filename 1..54>).

– Ein mit TSOSLNK erzeugter Lademodul kann mit ELDE geladen und gestartet werden (PHASE(...)).

```
START-FOR1-PROGRAM

FROM-FILE = <full-filename 1..54> / *MODULE(...) / *PHASE(...)

  *MODULE(...)
    |
    |  LIBRARY = *OMF / *STD / <full-filename 1..54>
    |
    | ,ELEMENT = *ALL / <full-filename 1..32>
    |
    | ,PROGRAM-MODE = 24 / ANY

  *PHASE(...)
    |
    |  LIBRARY = <full-filename 1..54>
    |
    | ,ELEMENT = <full-filename 1..41>(...)
    |      |  VERSION = *HIGHEST-EXISTING / <alphanum-name 1..24>
```

Tab. 5-1: SDF-Operand FROM-FILE (Binden und Laden)

Eine Gegenüberstellung von SDF-Operanden und entsprechenden Kommandos befindet sich im folgenden Abschnitt.

5.2 Übersicht: SDF-Operand FROM-FILE und entsprechende DBL- bzw. ELDE-Steuerung

Die folgende Tabelle stellt den SDF-Operanden des Kommandos START-FOR1-PROGRAM die entsprechenden Operanden im Aufruf des ELDE und DBL gegenüber. Die Darstellung der SDF-Operanden folgt der Metasyntax in Abschnitt 1.3.2

SDF-Fragebogen	1. Unterfragebogen	2. Unterfragebogen	Entspricht im ELDE	Entspricht im DBL
FROM-FILE = <full-filename 1..54>			katalogisierte Datei, in der der Lademodul steht	
= *MODULE(...)	LIBRARY = *OMF			temporäre EAM-Bindemodul-datei
	= *STD			Bindemodulbib-liothek Suchhierarchie des DBL bei fehlender An-gabe der Bin-demodulbiblio-thek
	= <full-filename 1..54>			Name der Bindemodul-bibliothek
	ELEMENT = *ALL			Bindemodul Alle Binde-moduln der angegebenen Bibliothek
	= <full-filename 1..32>			Name des Bindemoduls
	PROGRAM-MODE = 24			PROG-MOD = 24
	= ANY			= ANY
= *PHASE(...)	LIBRARY = <full-filename 1..54>		Name der Lade-modulbiblio-thek	
	ELEMENT = <full-filename 1..41>(...)	VERSION = <alphanum..name 1..24>	Name des Lade-moduls Versionsbe-zeichnung des Bibliotheks-elements	
		= *HIGHEST-EXISTING	Höchste Version	

Tab. 5-2: SDF-Fragebogen FROM-FILE (Angaben über den Binde-/Lademodul)

5.3 Statisches Binden (Binder TSOSLNK)

Der statische Binder TSOSLNK bindet:

- einen oder mehrere Bindemoduln zu einem Lademodul und speichert diesen Lademodul entweder in eine katalogisierte Datei oder in eine PLAM-Bibliothek (als Element vom Typ C). Hierzu dient die PROGRAM-Anweisung des TSOSLNK.
  Der Lademodul wird vom statischen Lader ELDE geladen und gestartet.
- mehrere Bindemoduln zu einem einzigen vorgebundenen Modul (Großmodul) und speichert diesen im temporären EAM-Bereich (*OMF) oder in einer PLAM-Bibliothek (als Element vom Typ R). Hierzu dient die MODULE-Anweisung (siehe Handbuch "TSOSLNK" [41]).
  Der erzeugte vorgebundene Modul dient als Eingabe entweder für den TSOSLNK oder den dynamischen Bindelader DBL.

Steueranweisungen für den Binder TSOSLNK

Hier wird nur eine Auswahl der Steueranweisungen für den TSOSLNK aufgeführt. Eine vollständige Beschreibung enthält das Handbuch "TSOSLNK" [41].

```
/START-PROG $TSOSLNK                                                    (1)
                     ⎧FILENAM=datei                                  ⎫
 PROGRAM programm [,⎨                                                 ⎬]  (2)
                     ⎩LIB[RARY]=bibl[,ELEM[ENT]=element[(version)]]  ⎭

                   [,SYMTEST={ALL|N|MAP}]                               (3)

                   [,MAP={Y|N}]                                         (4)

                   [,LOADPT={adresse|*XS}]                              (5)

          ⎧modul[(version)][,bibl]       ⎫
          ⎪(modul[(version)],...)[,bibl] ⎪
          ⎪,bibl                         ⎪
 INCLUDE  ⎨                              ⎬                             (6)
          ⎪modul,*                       ⎪
          ⎪(modul,...),*                 ⎪
          ⎪*                             ⎪
          ⎩                              ⎭

          ⎧modul       ⎫
 RESOLVE [⎨            ⎬],bibl                                        (7)
          ⎩(modul,...)  ⎭

 END                                                                    (8)
```

Erläuterung:

(1) Aufruf des Binders TSOSLNK.

(2) Die PROGRAM-Anweisung legt den Namen des Lademoduls fest und wo der Lademodul abgelegt werden soll.

programm Hier ist der Name anzugeben, den der Lademodul erhalten soll. Ist kein weiterer Operand (*FILENAM* oder *LIB*) angegeben, so erhält die katalogisierte Datei diesen Namen.

FILENAM=datei
Mit *datei* wird ein Name gewählt, den die katalogisierte Datei erhalten soll, in die der Lademodul abgelegt wird. Die maximale Länge einschließlich cat-id und user-id beträgt 54 Zeichen.

LIB=bibl [,ELEM=element]
Der Lademodul wird in der PLAM-Bibliothek namens *bibl* als Element vom Typ C unter dem Namen *element* abgelegt. Ist nur der Operand *LIB* angegeben, wird *programm* als Elementname angenommen. Existiert die Bibliothek *bibl* noch nicht, dann wird sie eingerichtet.

version Versionsbezeichnung des PLAM-Bibliothekselements *element*.

(3) SYMTEST=ALL
Durch SYMTEST=ALL können symbolische Adressen beim Testen mit der interaktiven Testhilfe AID angesprochen werden. FOR1-Programme können mit einer AID-Version ≥ 1.0C symbolisch getestet werden. Voraussetzung für das symbolische Testen ist, daß bei der Übersetzung die Compileroption SYMTEST=ALL angegeben wurde.

SYMTEST=N
Das Programm kann nicht symbolisch getestet werden.

SYMTEST=MAP
Der Binder erzeugt eine Objekt-Strukturliste, die mit in den Lademodul geschrieben wird. Mit dieser Information ist ein minimales symbolisches Testen möglich. Symbolisches Testen in vollem Umfang ist nur möglich, wenn AID die symbolischen Informationen noch nachlädt.

(4) Mit MAP=Y wird eine Programmübersicht nach SYSLST ausgegeben, die Informationen über Größe, Länge und Adressen der eingegebenen Bindemoduln enthält.

(5) LOADPT=adresse legt eine virtuelle Adresse (sedezimal: X'...') fest, an die der Lader ein Programm laden soll. Fehlt dieser Operand, dann wird die virtuelle Adresse X'000000' angenommen.
LOADPT=*XS legt die Ladeadresse des Programms im Adreßraum oberhalb 16 Megabyte fest. Dabei müssen alle Programmabschnitte (CSECTs) das Attribut RMODE=ANY haben. Der statische Lader ELDE wird dann das Programm im 31-Bit-Adressierungsmodus starten.

(6) Die INCLUDE-Anweisung legt die Bindemoduln fest, die der TSOSLNK binden soll.

Bindemoduln können stehen:
- im temporären EAM-Bereich (*OMF), wenn sie während der laufenden Task von FOR1 erzeugt wurden.
- in einer Bindemodulbibliothek, die vom Bibliotheksverwaltungsprogramm LMS erzeugt wurde.
- in einer PLAM-Bibliothek, die mit dem Bibliotheksverwaltungsprogramm LMS oder durch die Compileroption MODULE-LIBRARY erstellt wurde.

modul Name des Bindemoduls, der aus dem temporären EAM-Bereich (*OMF) bzw. der Bindemodul- oder PLAM-Bibliothek *bibl* eingelesen werden soll. Gibt man mehrere Bindemoduln an (maximal 20), muß man sie in runde Klammern einschließen.

version Versionsbezeichnung des Bindemoduls *modul*. Die Versionsbezeichnung ist nur für PLAM-Bibliotheken gültig. Läßt man die Angabe *version* weg, so wird der Bindemodul mit der höchsten Versionsbezeichnung eingebunden.

bibl Name der Bindemodul- bzw. PLAM-Bibliothek, aus der die Bindemoduln eingelesen werden sollen.
Läßt man die Angabe *modul* weg, so liest der Binder alle Bindemoduln der Bibliothek ein.
Läßt man die Angabe *bibl* weg, so sucht der Binder den Bindemodul *modul* in der Bibliothek TASKLIB.

* Temporärer EAM-Bereich (*OMF) der laufenden Task. Läßt man die Angabe *modul* weg, so liest der Binder alle Bindemoduln ein, die sich im temporären EAM-Bereich befinden.

(7) Mit der RESOLVE-Anweisung werden dem Binder die Bibliotheken mitgeteilt, die mit dem Autolink-Verfahren nach bisher unbefriedigten Externverweisen durchsucht werden sollen.

Autolink-Verfahren des TSOSLNK:

Findet der TSOSLNK in einem Bindemodul Externverweise, die nicht durch die Moduln befriedigt werden können, die in INCLUDE-Anweisungen angegeben wurden, so geht er nach folgendem Autolink-Verfahren vor:

- Zuerst sucht der TSOSLNK, ob in einer RESOLVE-Anweisung eine Bibliothek in Verbindung mit dem Externverweis angegeben wurde.
- Kann der TSOSLNK im ersten Schritt den Externverweis nicht befriedigen, so durchsucht er sämtliche Bibliotheken, die in RESOLVE-Anweisungen angegeben wurden. Dabei wird die letzte RESOLVE-Anweisung zuerst berücksichtigt, die vorletzte als zweite usw.
  Bibliotheken, die nicht durchsucht werden sollen, können durch EXCLUDE-Anweisungen von der Suche ausgeschlossen werden.
- Ist es dem TSOSLNK auch im zweiten Schritt nicht gelungen, den Externverweis zu befriedigen, durchsucht er die Bibliothek TASKLIB, sofern dies nicht durch die Anweisung NCAL oder eine entsprechende EXCLUDE-Anweisung verhindert wurde. Falls es unter der Benutzerkennung der laufenden Task keine Bibliothek namens TASKLIB gibt, verwendet der TSOSLNK die Bibliothek des Systems, $TSOS.TASKLIB.

Sind auch nach dem Autolink-Verfahren noch unbefriedigte Externverweise vorhanden, gibt der TSOSLNK ihre Namen in in einer Liste nach SYSOUT und SYSLST aus.

Ist das FOR1-Laufzeitsystem nicht Bestandteil der Bibliothek TASKLIB des Anwenders oder der System-TASKLIB, so muß es in einer RESOLVE-Anweisung angegeben werden:

```
RESOLVE ,$userid.FOR1MODLIBS
```

Wird der Funktionspool FPOOL verwendet, dann muß die Bindemodulbibliothek $userid.FOR1.FPOOLLIB in einer RESOLVE-Anweisung angegeben werden:

```
RESOLVE ,$userid.FOR1.FPOOLLIB
```

(8) Die Eingaben für den TSOSLNK müssen mit der END-Anweisung abgeschlossen werden.

Beispiel:

Übersetzen:

```
/START-PROG $FOR1
*COMOPT SOURCE=QUELL.MAT
*COMOPT MODULE-LIBRARY=PLAM.LIB                    (1)
*COMOPT FPOOL=fpool
*END
```

Binden mit TSOSLNK:

```
/START-PROG $TSOSLNK
*PROGRAM PROGAB,FILENAM=OBJ.MAT                    (2)
*INCLUDE (MODA,MODB),PLAM.LIB                      (3)
*RESOLVE (FUNKT1,FUNKT2),FUNKLIB                   (4)
*RESOLVE ,FOR1MODLIBS                              (5)
*RESOLVE ,$TSOS.FOR1.FPOOLLIB                      (6)
*END
/
.
.
.
```

Erläuterung des Beispiels:

(1) Bei der Übersetzung werden die Bindemoduln MODA und MODB erzeugt und in die PLAM-Bibliothek PLAM.LIB geschrieben.

(2) Der Lademodul soll den Namen PROGAB erhalten und soll in der katalogisierten Datei OBJ.MAT stehen.

(3) Anweisung, die Bindemoduln MODA und MODB aus der Bibliothek PLAM.LIB zusammenzubinden.

(4) Die Externverweise FUNKT1 und FUNKT2 werden mit Bindemoduln aus der Bibliothek FUNKLIB aufgelöst Falls aus FUNKLIB ein BLOCKDATA-Modul (zur Initialisierung eines benannten COMMON-Bereichs) eingebunden werden soll, muß dies mit einer INCLUDE-Anweisung geschehen.

(5) Die Externverweise auf Moduln des FOR1-Laufzeitsystems werden aufgelöst. Es werden nur Adaptermoduln eingebunden. Das eigentliche Laufzeitsystem wird zur Laufzeit nachgeladen. Falls sich das FOR1-Laufzeitsystem in der System-TASKLIB befindet, oder in der TASKLIB der Benutzerkennung, kann Anweisung (5) entfallen.

(6) Die Externverweise auf Moduln aus dem FPOOL werden aufgelöst.

Binden von BLOCK DATA-Unterprogrammen

BLOCK DATA-Unterprogramme müssen beim Binden mit dem TSOSLNK mit eingebunden werden.
Stehen die BLOCK DATA-Unterprogramme in der temporären EAM-Datei, dann werden sie durch die Anweisung "INCLUDE *" mit eingebunden.
Stehen die BLOCK DATA-Unterprogramme nicht in der temporären EAM-Datei, dann müssen sie in der INCLUDE-Anweisung explizit aufgeführt werden.

Als Name des Bindemoduls in der INCLUDE-Anweisung ist bei einer benannten BLOCK DATA-Programmeinheit der Name dieser Programmeinheit anzugeben. Bei einer unbenannten BLOCK DATA-Programmeinheit müssen die Namen aller COMMON-Blöcke dieser Programmeinheit in INCLUDE-Anweisungen aufgeführt werden.

Beispiel: Binden eines unbenannten BLOCK DATA-Unterprogramms

Quellprogramm in der katalogisierten Datei QUELL.TEST:

```
PROGRAM TEST
COMMON /A/I,/B/R
WRITE *,I,R
END
BLOCK DATA
COMMON /A/I,/B/R
DATA I,R/1,2.2/
END
```

Übersetzen:

```
/START-PROG $FOR1
*COMOPT SOURCE=QUELL.TEST
*COMOPT MODULE-LIBRARY=PLAM.LIB
*END
```

Binden:

```
/START-PROG $TSOSLNK
*PROGRAM TEST,FILENAM=L.TEST
*INCLUDE TEST,PLAM.LIB
*INCLUDE A,PLAM.LIB
*INCLUDE B,PLAM.LIB
*RESOLVE ,$TSOS.FOR1MODLIBS
*END
```

Initialisieren von benannten COMMON-Blöcken

Ein benannter COMMON-Block kann nicht nur in einem BLOCK DATA-Unterprogramm, sondern in jeder Programmeinheit mit Werten vorbesetzt werden. Wird gleichzeitig in mehreren Programmeinheiten derselbe COMMON-Block initialisiert, so meldet dies der Binder (TSOSLNK; DBL) mit der Meldung "DUPLICATE CSECT". Wird diese Bindermeldung ignoriert, so kann dies bei Overlay-Systemen zu ungewollten Reinitialisierungen führen.
Falls ein COMMON-Block in sämtlichen Programmeinheiten initialisiert wird, kann der Binder diesen nicht mehr als COMMON-Block erkennen. Die entsprechende Information geht verloren.

5.4 Statisches Laden (Lader ELDE)

Damit ein vom TSOSLNK erzeugter Lademodul ablaufen kann, muß er in den Hauptspeicher geladen werden. Für diese Aufgabe steht im BS2000 der Lader ELDE zur Verfügung.
ELDE wird aufgerufen, wenn ein START-PROGRAM-Kommando oder ein LOAD-PROGRAM-Kommando eingegeben wird, das sich auf eine katalogisierte Datei oder ein Element einer PLAM-Bibliothek (Elementtyp C) bezieht:

- Das START-PROGRAM-Kommando weist den ELDE an, den Lademodul in den Speicher zu laden und zu starten.
- Das LOAD-PROGRAM-Kommando weist den ELDE an, den Lademodul in den Speicher zu laden, ohne ihn zu starten. Dadurch lassen sich vor dem Programmablauf weitere Kommandos eingeben, etwa zur Testhilfe. Das Programm kann daraufhin mit dem RESUME-PROGRAM-Kommando gestartet werden.

Aufruf des ELDE

Hier wird nur eine Auswahl der Angaben der Kommandos START-PROGRAM und LOAD-PROGRAM beschrieben. Eine vollständige Beschreibung der beiden Kommandos befindet sich im Handbuch "TSOSLNK" [41].

```
START-PROGRAM bzw. LOAD-PROGRAM

 FROM-FILE = <full-filename 1..54> / *PHASE(...)

   *PHASE(...)
      |  LIBRARY = <full-filename 1..54>
      | ,ELEMENT = <full-filename 1..41>
      | ,VERSION = *STD / <text 1..24>

,TEST-OPTIONS = NONE / AID
,MONJV = *NONE / <full-filename 1..54>
```

FROM-FILE =
Gibt die Eingabequelle an

FROM-FILE = <full-filename 1..54>
Eingabequelle ist die katalogisierte Datei, die den vom TSOSLNK erzeugten Lademodul enthält.

FROM-FILE = *PHASE(...)
Eingabequelle ist eine PLAM-Bibliothek, die den vom TSOSLNK erzeugten Lademodul als Element vom Typ C enthält.

LIBRARY = <full-filename 1..54>
Name der PLAM-Bibliothek, in der der Lademodul gespeichert ist.

ELEMENT = <full-filename 1..41>
Name des Bibliothekselements, in dem der Lademodul gespeichert ist. Das Bibliothekselement muß den Elementtyp C haben.

VERSION =
Gibt die Versionsbezeichnung des Elements an.

VERSION = *STD
Das Element mit der höchsten Versionsbezeichnung wird genommen.

VERSION = <text 1..24>
Explizite Angabe der Elementversion.

TEST-OPTIONS =
Gibt an, ob symbolische Adressen im Quellprogramm beim Testen mit AID verwendet werden dürfen. Mit symbolischen Adressen sind nur Programme testbar, für die beim Übersetzen LSD-Informationen erzeugt wurden (Compileroption SYMTEST=ALL).

TEST-OPTIONS = NONE
Die LSD-Informationen werden nicht in den Lademodul übernommen. Symbolisches Testen mit AID ist noch möglich, wenn AID mit einer %SYMLIB-Anweisung eine Bibliothek angegeben wird, die die LSD-Informationen enthält.

TEST-OPTIONS = AID
Erlaubt die Verwendung von symbolischen Adressen des Quellprogramms beim Testen mit AID.

MONJV = *NONE / <full-filename 1..54>
Name einer Jobvariablen, die das Programm überwachen soll. Bei Angabe von *NONE wird das Programm nicht mit einer Jobvariablen überwacht. Beim Ablauf hinterlegt das Programm in der Rückkehrcode-Anzeige dieser Jobvariablen einen Code, der über mögliche Ablauffehler Aufschluß gibt. Die einzelnen Codes und ihre Bedeutung sind in einer Tabelle in Abschnitt 4.9 zusammengefaßt. Dieser Operand steht nur Anwendern mit dem Softwareprodukt "Jobvariablen" zur Verfügung (siehe Abschnitt A.10.5).

Beispiel:

Der im Beispiel in Abschnitt 5.3 erzeugte Lademodul PROGAB kann durch folgendes Kommando geladen und ausgeführt werden:

```
/START-PROGRAM FROM-FILE=OBJ.MAT
```

OBJ.MAT: katalogisierte Datei, in die der Lademodul PROGAB geschrieben wurde.

5.5 Dynamisches Bindeladen (Bindelader DBL)

Mit dem dynamischen Bindelader DBL werden in einem Arbeitsgang Moduln (Bindemoduln, Bindelademoduln) temporär zu einer ladbaren Einheit gebunden, dann in den Speicher geladen und ausgeführt. Die erzeugte Ladeeinheit wird nach dem Programmablauf automatisch gelöscht.
Die Arbeitsweise des DBL ist im Handbuch "Binder-Lader-Starter" [13] ausführlich beschrieben.

Der DBL arbeitet in zwei Betriebsmodi. Der Betriebsmodus wird mit dem Operanden RUN-MODE der Kommandos START-PROGRAM und LOAD-PROGRAM ausgewählt.

RUN-MODE=STD
In diesem Betriebsmodus arbeitet der DBL voll kompatibel zum DLL. DLL wird nur bis einschließlich BS2000 V9.5 ausgeliefert. Der Betriebsmodus RUN-MODE=STD ist voreingestellt.

RUN-MODE=ADVANCED
In diesem Betriebsmodus kann der DBL auch Bindelademoduln (LLMs) verarbeiten und unterstützt die neuen Funktionen ab BS2000 V10. Dieser Betriebsmodus wird hier nicht beschrieben. Eine ausführliche Beschreibung befindet sich im Handbuch "Binder-Lader-Starter" [13].

Aufruf des DBL

Der DBL wird aufgerufen, wenn ein START-PROGRAM-Kommando oder LOAD-PROGRAM-Kommando eingegeben wird, das sich auf den temporären EAM-Bereich, ein Element einer Bindemodulbibliothek oder ein Element einer PLAM-Bibliothek (Elementtyp R) bezieht:

- Das START-PROGRAM-Kommando weist den DBL an, den Bindemodul zu binden, zu laden und sofort zu starten.
- Das LOAD-PROGRAM-Kommando weist den DBL an, den Bindemodul zu binden und zu laden. Nach dem Binde-Lade-Vorgang können weitere Kommandos (z.B. Testhilfe-Kommandos) eingegeben werden. Anschließend startet der Anwender das Programm mit dem Kommando RESUME-PROGRAM.

Im folgenden ist nur eine Auswahl der Angaben der Kommandos START-PROGRAM und LOAD-PROGRAM beschrieben. Eine vollständige Beschreibung der beiden Kommandos befindet sich im Handbuch "Binder-Lader-Starter" [13].

```
START-PROGRAM  bzw. LOAD-PROGRAM
```

```
FROM-FILE = *MODULE(...)

  *MODULE(...)
      LIBRARY = *STD / *OMF / <full-filename 1..54>
     ,ELEMENT = *ALL / <full-filename 1..8>
     ,PROGRAM-MODE = 24 / ANY
     ,TEST-OPTIONS = NONE / AID
     ,MONJV = *NONE / <full-filename 1..54>
```

FROM-FILE = *MODULE (...)
Der dynamische Bindelader DBL wird aufgerufen.

LIBRARY =
Gibt die Eingabequelle an, aus der Bindemoduln geholt werden. Eingabequelle für Bindemoduln kann der temporäre EAM-Bereich (*OMF), eine Bindemodulbibliothek oder eine PLAM-Bibliothek (Elemente vom Typ R) sein.

LIBRARY = *STD
Eingabequelle ist die Bibliothek, die mit dem SET-TASKLIB-Kommando zugewiesen wurde. Wird der Bindemodul dort nicht gefunden, so wird die Bibliothek namens TASKLIB der aktuellen Task und anschließend die System-TASKLIB ($TSOS.TASKLIB) durchsucht.

LIBRARY = *OMF
Eingabequelle ist der temporäre EAM-Bereich.

LIBRARY = <full-filename 1..8>
Name einer Bindemodulbibliothek oder einer PLAM-Bibliothek, die als Eingabequelle verwendet wird.

ELEMENT =
Legt die Moduln fest, die aus der angegebenen Bibliothek geholt werden sollen.

ELEMENT = *ALL
Nur zulässig für Bindemoduln aus dem temporären EAM-Bereich. Alle Bindemoduln werden aus dem EAM-Bereich geholt.

ELEMENT = <full-filename 1..32>
Name des Bindemoduls
full-filename darf sein:
- Name eines Bindemoduls
- Name eines Programmabschnitts (CSECT-Name)
- Name einer Einsprungadresse (ENTRY-Name)
- Name eines COMMON-Bereichs

Elemente von PLAM-Bibliotheken müssen den Elementtyp R haben.

PROGRAM-MODE =
Bestimmt, in welchem Teil des Adreßraums (oberhalb oder unterhalb 16Mbyte) das Programm geladen werden soll.

PROGAM-MODE = 24
Der Modul wird unterhalb 16 Megabyte geladen. Die Programmausführung erfolgt im 24-Bit-Adressierungsmodus. Externverweise werden nur mit CSECTs oder ENTRYs befriedigt, die unterhalb 16 Megabyte liegen. Das Laden eines Programms mit 31-Bit-Adressierungsmodus (AMODE=31) wird mit einer Fehlermeldung abgebrochen.

PROG-MODE = ANY
Der Modul kann oberhalb oder unterhalb 16 Megabyte geladen werden. Die Ladeadresse wird unter Auswertung der Eigenschaften RMODE und AMODE festgelegt: Ist RMODE=24, dann liegt die Ladeadresse unterhalb 16 Megabyte. Ist RMODE=ANY und AMODE=ANY, dann liegt die Ladeadresse oberhalb 16 Megabyte.

TEST-OPTIONS =
Gibt an, ob symbolische Adressen im Quellprogramm beim Testen mit AID verwendet werden dürfen.
Mit symbolischen Adressen können nur Programme getestet werden, für die bei der Übersetzung LSD-Informationen erzeugt wurden (mit Compileroption SYMTEST=ALL).

TEST-OPTIONS = NONE
Die LSD-Informationen werden nicht in den Lademodul übernommen. Symbolisches Testen mit AID ist noch möglich, wenn AID mit einer %SYMLIB-Anweisung eine Bibliothek angegeben wird, die die LSD-Informationen enthält.

TEST-OPTIONS = AID
Symbolische Adressen können beim Testen mit AID angesprochen werden.

MONJV = *NONE / <full-filename 1..54>
Name einer Jobvariablen, die das Programm überwachen soll. Bei Angabe von *NONE wird das Programm nicht mit einer Jobvariablen überwacht.
Beim Ablauf hinterlegt das Programm in der Rückkehrcode-Anzeige dieser Jobvariablen einen Code, der über mögliche Ablauffehler Aufschluß gibt. Die einzelnen Codes und ihre Bedeutung sind in einer Tabelle in Abschnitt 4.9 zusammengefaßt.
Dieser Operand steht nur Anwendern mit dem Softwareprodukt "Jobvariablen" zur Verfügung (siehe Abschnitt A.10.5).

Beispiel:

Übersetzen:

```
/DEL-SYS-FILE OMF
/START-PROG $FOR1
*COMOPT SOURCE=QUELL.MAT
*COMOPT OBJECT=(*)                                    (1)
*END
```

Binden, Laden, Starten:

```
/SET-TASKLIB FOR1MODLIBS                              (2)
/START-PROG FROM-FILE=*MODULE(LIB=*OMF)               (3)
```

Erläuterung des Beispiels:

(1) Die erzeugten Bindemoduln werden im temporären EAM-Bereich (*OMF) abgelegt.

(2) Zuweisen des FOR1-Laufzeitsystems. Dies ist nur nötig, wenn sich das FOR1-Laufzeitsystem nicht in der TASKLIB der aktuellen Task bzw. in der TASKLIB des Systems befindet.

(3) Alle Bindemoduln aus dem temporären EAM-Bereich sollen mit dem DBL gebunden, geladen und gestartet werden (ELEMENT=*ALL ist voreingestellt).

5.6 Speicherbelegung von gestarteten Programmen

Der Speicherbereich eines gestarteten Programms umfaßt den erzeugten Lademodul und den zu Beginn des Ablaufs angelegten Speicherbereich. Das Programm wird unter Kontrolle des FOR1-Laufzeitsystems ausgeführt.
Bild 5-1 zeigt die Speicherbelegung für Objektprogramme, die ohne explizite Angabe einer Ladeadresse geladen wurden.

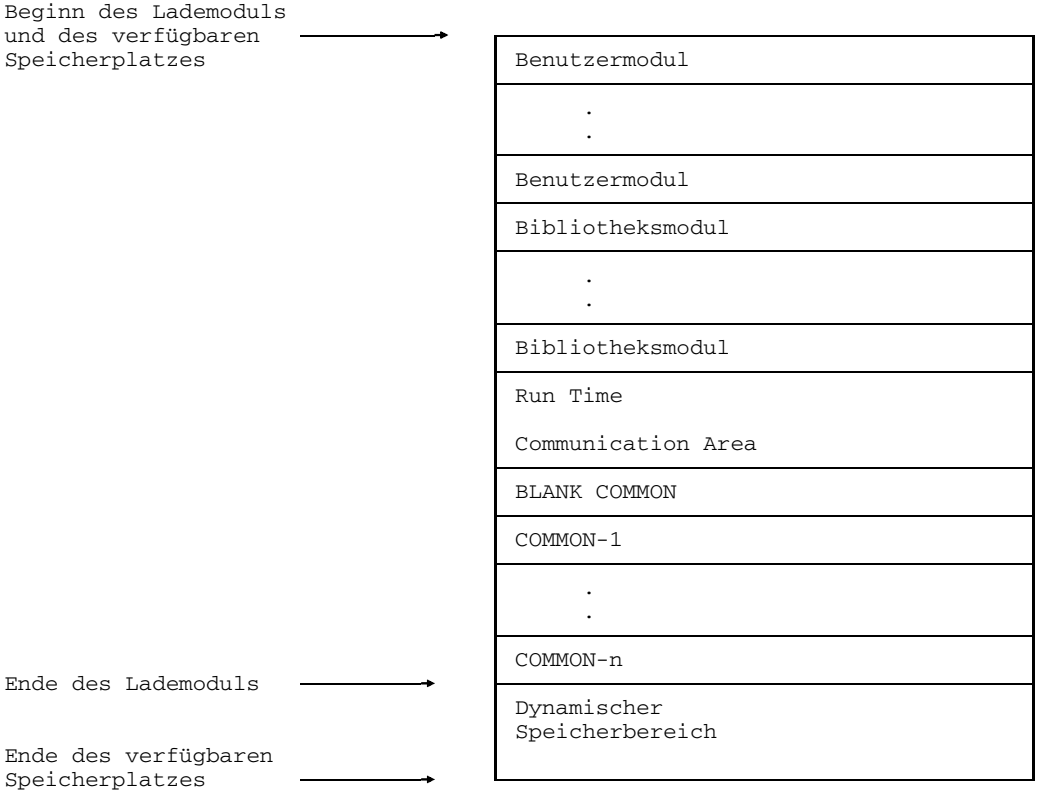


Bild 5-1: Speicherbelegung eines gestarteten Progaramms

Benutzermoduln

Die aus den Bindemoduln durch Auflösen der Externverweise entstandenen Moduln.

Bibliotheksmoduln

Die vom Binder beim Auflösen der Externverweise hinzugefügten Moduln (aus der FOR1-Laufzeitbibliothek oder anderen Modulbibliotheken).

Run Time Communication Area (RTCA)

In der RTCA sind Informationen abgelegt, die dazu dienen, den Ablauf des gestarteten Programms zu steuern. Die RTCA wird von der Programminitialisierungsroutine nach den Benutzer- und Bibliotheksmoduln angelegt und ist ungefähr 4 KB groß. Folgende Informationen befinden sich unter anderem in der RTCA:

- Tabelle der Ein-/Ausgabe-Einheiten.
  Diese Tabelle enthält für jede Dateinummer einen Verweis auf die zugehörige Ein-/Ausgabe-Routine und zeigt den Typ der zugehörigen Datei an.
- Hash-Tabelle für den Zugriff zu den Dateideskriptoren.
  Dateideskriptoren werden vom FOR1-Laufzeitsystem für jede Datei im dynamischen Speicherbereich angelegt.
- Adresse des "aktuellen" Dateideskriptors, d.h. Datei, auf die gerade zugegriffen wird.
- Adresse der "aktuellen" Ein-/Ausgabe-Routine
- Adresse für den Fehlerausgang bei Ein-/Ausgabe-Operationen
- Informationen über die laufende Task, wie z.B. Datum, Uhrzeit, verbrauchte CPU-Zeit

COMMON-n, BLANK COMMON

Gemeinsamer Speicherbereich, der vom Binder für die COMMON-Bereiche angelegt wurde.

Dynamischer Speicherbereich

Der dynamische Speicherbereich wird für folgende Zwecke verwendet:

Ein-/Ausgabe-Pufferbereiche

Die Übertragung von und zu einer Datei erfolgt nicht für jeden Satz separat, sondern es wird immer der Inhalt eines Pufferbereichs auf einmal übertragen.

Konversionspuffer

Bei den meisten Ein-/Ausgabe-Operationen sind Konversionen zwischen der internen und externen Darstellung von Daten notwendig. Die Konversionsroutinen benutzen bei längeren Daten dynamisch angelegte Konversionspuffer.

Dateideskriptoren

In einem Dateideskriptor werden die Eigenschaften einer Datei im FORTRAN-Programm beschrieben. Dies dient dazu, um die Zulässigkeit einer Ein-/Ausgabe-Operation prüfen zu können und die für die INQUIRE-Anweisung benötigten Informationen bereitzuhalten. Außerdem ist die zuletzt durchgeführte Ein-/Ausgabe-Operation auf diese Datei eingetragen, um die Gültigkeit der Aufeinanderfolge von Ein-/Ausgabe-Operationen prüfen zu können. Bei jedem Zugriff auf eine Datei wird der zugehörige Dateideskriptor modifiziert.

Dateisteuerblock (FCB)

Der Dateisteuerblock ist der dominierende Verständigungsbereich aller Ein-/Ausgabe-Operationen. Er beschreibt die Eigenschaften einer Datei aus der Sicht des Datenverwaltungssystems. Für jede Datei, EAM-Dateien und Systemdateien ausgenommen, existiert ein Dateisteuerblock.

FORMAT-Deskriptoren bei variablem FORMAT

Die Ergebnisse der Interpretation von variablen Formaten werden in Form von FORMAT-Deskriptoren gespeichert, um die einmal durchgeführte Interpretation evtl. öfter verwenden zu können.

5.7 Binder BINDER

Der BINDER bindet Moduln (Bindemoduln, Bindelademoduln) zu einer logisch und physikalisch strukturierten ladbaren Einheit zusammen. Diese Einheit bezeichnet man als Bindelademodul (Link and Load Module, LLM). Der BINDER speichert Bindelademoduln in PLAM-Bibliotheken (Elementtyp L) ab.
Der BINDER steht ab BS2000 V10 zur Verfügung. Eine genaue Beschreibung des BINDER befindet sich im Handbuch "Binder-Lader-Starter"[13].

5.8 Mehrfachbenutzbare Programme

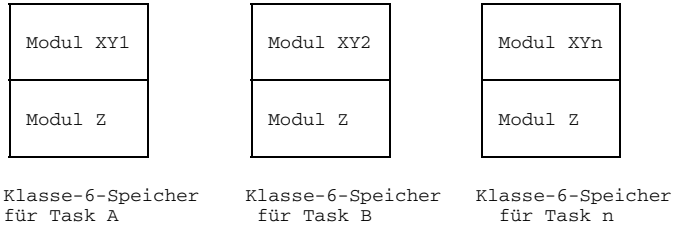
Bei großen Programmen kann es von Vorteil sein, einzelne Programmabschnitte, auf die mehrere Anwender (Tasks) gemeinsam zugreifen, mehrfachbenutzbar (shareable) zu machen.

Mehrfachbenutzbare Programmteile haben folgende Vorteile:
- Ersparnis von Speicherplatz (der mehrfachbenutzbare Modul steht nur einmal im Arbeitsspeicher)
- Zeitersparnis durch reduziertes Paging.

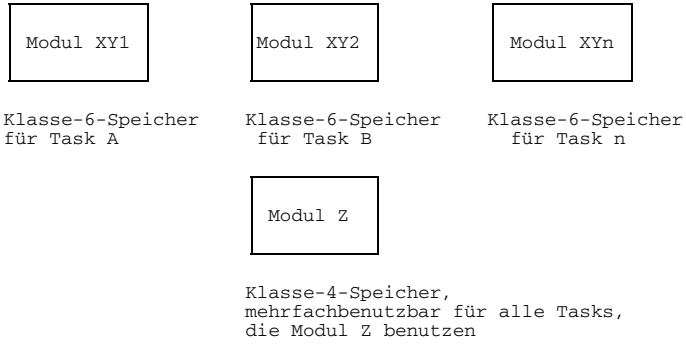
Programmabläufe mit mehrfachbenutzbaren und nicht-mehrfachbenutzbaren Programmen

Die folgenden Bilder veranschaulichen Programmabläufe mit mehrfachbenutzbaren und nicht-mehrfachbenutzbaren Programmen:

Speicherbelegung bei nicht-mehrfachbenutzbaren Programmen (Modul Z wird dreimal in den Klasse-6-Speicher geladen):



Speicherbelegung bei mehrfachbenutzbaren Programmen (Modul Z wird einmal in den Klasse-4-Speicher geladen):



Erzeugen von mehrfachbenutzbaren Programmen

Mehrfachbenutzbare Programme werden folgendermaßen erzeugt:

- Das Quellprogramm wird mit der Compileroption OBJECT=(SHARE) übersetzt. Diese Compileroption trennt den mehrfachbenutzbaren Teil des Quellprogramms (Codeteil) vom nicht-mehrfachbenutzbaren Teil (Datenteil).

  FOR1 erzeugt einen mehrfachbenutzbaren Bindemodul und einen nicht-mehrfachbenutzbaren Bindemodul.
  Der mehrfachbenutzbare Codeteil darf keine Adreßverweise auf den nicht-mehrfachbenutzbaren Datenteil enthalten. Die Verbindung zwischen Codeteil und Datenteil wird deshalb so hergestellt, daß ein kleiner Codeteil in den Datenteil verlagert wird. In diesem Codeteil werden die Register für die Adressen von Code- und Datenteil versorgt, danach wird in den Codeteil verzweigt (siehe Bild 5-2).

- Mehrfachbenutzbarer Bindemodul und nicht-mehrfachbenutzbarer Bindemodul werden in PLAM-Bibliotheken abgelegt.

- Mehrfachbenutzbarer und nicht-mehrfachbenutzbarer Bindemodul können entweder mit der SHARE-Prozedur SYSPRC.FOR1.022.SHARE oder ohne die SHARE-Prozedur weiterverarbeitet werden. Diese beiden Wege sind in Abschnitt 5.8.1 und 5.8.2 beschrieben.

  Der Systemverwalter erklärt den mehrfachbenutzbaren Teil mit dem Kommando ADD-SHARED-PROGRAM als mehrfachbenutzbar. Der mehrfachbenutzbare Teil wird in den Klasse-4-Speicher geladen, sobald der erste Anwender ihn anfordert. Dort bleibt er solange verfügbar, bis alle Tasks durch SHUTDOWN beendet werden.

  Der nicht-mehrfachbenutzbare Teil wird pro Task in den Klasse-6-Speicher geladen.

Einschränkungen:

- Die Compileroptionen OBJECT=(SHARE) und SYMTEST=ALL schließen sich gegenseitig aus. Werden beide angegeben, so gilt die letzte Angabe und es erscheint eine Fehlermeldung.

- Ein Eingreifen mit der symbolischen Testhilfe AID ist nicht möglich, da mehrfachbenutzbare Programmteile im Klasse-4-Speicher stehen.

- Bei Sprachverknüpfungen dürfen mehrfachbenutzbare und nicht-mehrfachbenutzbare Programme nicht gemischt werden.

```
**************************************
**        D A T A    A R E A         **
**************************************
**    NON-SHARE ENTRY CODE PART       **
**************************************
*
MUPS     CSECT
         USING *,15
         STM   0,12,20(13)
         LR    9,13                  (1) Laden der Adresse des Datenteils
         LA    13,24(0,15)           (2) Laden der Adresse des Codeteils
         L     11,20(0,15)
         MVI   0(9),236
         BCR   15,11                 (3) Verzweigen in den Codeteil
         DROP  15
         DC    V(MUPS@@@@)
**************************************
**         NON-SHARE DATA PART       **
**************************************
*
         DS    0D
         USING *,13
         DS    312C
         ORG   MUPS+96
         DC    A(I@@@RTCA)             RUNTIME COMMUNICATION AREA
         ORG   MUPS+104
         DC    X'03'
         DC    'MUPS   '
               .
               .
               .
```

```
**************************************
**        C O D E    A R E A         **
**************************************
*
MUPS@@@@ EQU   *
*
***** STATEMENT 1 (ENTRY) *************
*       INTEGER FUNCTION MUPS(J)
*                                                  **** SEGMENT 1 ****
* CODE SLICE BEGIN *                    (SLICE 1)
         USING *,15
         USING *,11
         L     12,16(0,11)
         BC    15,20(0,11)
         DC    AL1(03)
         DC    CL7'MUPS   '
         DC    A(MUPS####)
         USING MUPS####,12
         USING MUPS,13
         LR    15,11
         STM   14,15,12(9)
         LR    14,11
         ST    13,8(0,9)
         CLI   0(13),0
         BC    8,76(0,11)
         STM   12,0,104(9)
               .
               .
               .
```

Bild 5-2: Generierter Code mit COMOPT OBJECT=(SHARE); Auszüge aus dem OBJECT-Listing

5.8.1 Mehrfachbenutzbare Programme mit Hilfe der Prozedur SYSPRC.FOR1.022.SHARE

5.8.1.1 Vorgehensweise

1. Übersetzen mit der Compileroption OBJECT=(SHARE)

   Sollen mehrfachbenutzbare Programme erzeugt werden, so muß dies bereits bei der Übersetzung eingeleitet werden, indem folgende Compileroption angegeben wird:

   ```
   COMOPT OBJECT=(SHARE)
   ```

   Diese Compileroption trennt den mehrfachbenutzbaren Teil des Quellprogramms (Codeteil) vom nicht-mehrfachbenutzbaren Teil (Datenteil).

   FOR1 erzeugt einen mehrfachbenutzbaren Bindemodul und einen nicht-mehrfachbenutzbaren Bindemodul.

   Der Name des mehrfachbenutzbaren Bindemoduls entspricht dem Namen der Programmeinheit und ist rechtsbündig mit @ auf acht Zeichen ergänzt.

   Der nicht-mehrfachbenutzbare Bindemodul erhält den Namen der Programmeinheit.

2. Ablegen von mehrfachbenutzbaren und nicht-mehrfachbenutzbaren Bindemoduln in PLAM-Bibliotheken. Der Anwender hat folgende Möglichkeiten:

   – Der Anwender benutzt bei der Übersetzung die Compileroptionen SHARE-LIBRARY und MODULE-LIBRARY.

     Mehrfachbenutzbare Bindemoduln werden entsprechend der Angaben in der Compileroption SHARE-LIBRARY abgelegt (siehe Abschnitt 4.2.2.2). Fehlt die Compileroption SHARE-LIBRARY, so werden die mehrfachbenutzbaren Bindemoduln entsprechend der Angaben in der Compileroption MODULE-LIBRARY abgelegt.

     Nicht-mehrfachbenutzbare Bindemoduln werden entsprechend der Angaben in der Compileroption MODULE-LIBRARY abgelegt (siehe Abschnitt 4.3.2).

   – Nach der Übersetzung werden mit dem Bibliotheksverwaltungsprogramm LMS die mehrfachbenutzbaren und nicht-mehrfachbenutzbaren Bindemoduln in PLAM-Bibliotheken gebracht.

3. SHARE-Prozedur einsetzen

   Bevor die SHARE-Prozedur SYSPRC.FOR1.022.SHARE aufgerufen wird, muß der Anwender die Namen der mehrfachbenutzbaren Bindemoduln, die zu einem mehrfachbenutzbaren Modul zusammengefügt werden sollen, in eine Datei schreiben (siehe Beispiel). Die Namen werden ohne @ angegeben.

   Nun ruft der Anwender die SHARE-Prozedur auf.

   Die SHARE-Prozedur erstellt zwei Adaptermoduln, einen Adaptermodul für den mehrfachbenutzbaren Modul und einen Adaptermodul für den nicht-mehrfachbenutzbaren Modul. Diese Adaptermoduln stellen die Verbindung zwischen mehrfachbenutzbarem Modul und nicht-mehrfachbenutzbarem Modul her.

   Die SHARE-Prozedur bindet den mehrfachbenutzbaren Adaptermodul und den mehrfachbenutzbaren Bindemodul zu einem mehrfachbenutzbaren Modul zusammen und legt den nicht-mehrfachbenutzbaren Adaptermodul in der angegebenen Bibliothek ab.

   Die SHARE-Prozedur kann sowohl im Stapelprozeß als auch im Dialogprozeß aufgerufen werden. Im Stapelprozeß müssen bei Aufruf der SHARE-Prozedur die Parameter angegeben werden. Im Dialogprozeß können die Parameter entweder beim Aufruf angegeben werden, oder aber sie werden bei Prozedurablauf über Prompting am Bildschirm angefordert.

   Aufruf und Parameter der SHARE-Prozedur sind in Abschnitt 5.8.1.2 beschrieben.

4. Modul als mehrfachbenutzbar erklären und in den Klasse-4-Speicher laden

   Der von der SHARE-Prozedur erzeugte mehrfachbenutzbare Modul muß noch vom Systemverwalter als mehrfachbenutzbar erklärt werden (Kommando ADD-SHARED-PROGRAM) und in den Klasse-4-Speicher geladen werden. Dies ist im Handbuch "BS2000 Systemverwaltung [40] beschrieben.

5. Binden der nicht-mehrfachbenutzbaren Moduln mit TSOSLNK

   Der Anwender muß noch den nicht-mehrfachbenutzbaren Bindemodul, den nicht-mehrfachbenutzbaren Adaptermodul und ggf. weitere Bindemoduln mit dem Binder TSOSLNK zu einem Lademodul binden.

6. Starten des Programms

   Der Anwender startet das Programm mit dem Lader ELDE durch Aufrufen des nicht-mehrfachbenutzbaren Lademoduls:

```
/START-PROGRAM FROM-FILE=    ...      bzw.
/LOAD-PROGRAM FROM-FILE=     ...
```

5.8.1.2 Parameter der Prozedur SYSPRC.FOR1.022.SHARE

```
/CALL-PROCEDURE SYSPRC.FOR1.022.SHARE, [SHRNAMES=name[,X[,breg]]
                                ,LIBN=libn,LIBS=libs, LIBT=libt
                                ,ADAPTS=adapts,ADAPTN=adaptn,SHRMOD=shrmod
                               [,HELP={YES|NO}][,ER={YES|NO}][,XS = {YES|NO}]
                               [,MACLIB = {$TSOS.MACROLIB|libname}]]
```

name Name der Datei, die die Namen der mehrfachbenutzbaren Moduln enthält. *name* wird auf 8 Zeichen rechtsbündig mit "@" ergänzt. Ohne Angabe von *X* und *breg*: Basisregister ist 11.

X Basisregister ist 15.

breg Basisregister wird wie Sprungregister zu den mehrfachbenutzbaren Moduln benutzt.

libn Name der nicht-mehrfachbenutzbaren Modulbibliothek.
Diese Modulbibliothek enthält die nicht-mehrfachbenutzbaren Moduln und die von der SHARE-Prozedur erzeugten nicht-mehrfachbenutzbaren Adaptermoduln.

libs Name der mehrfachbenutzbaren Modulbibliothek.
Diese Modulbibliothek enthält die mehrfachbenutzbaren Moduln und die von der SHARE-Prozedur erzeugten mehrfachbenutzbaren Adaptermoduln.

libt Typ der Modulbibliotheken (LMS oder LMR), in denen die Moduln abgespeichert sind.

adapts Name, den der mehrfachbenutzbare Adaptermodul erhalten soll.
Dieser Modul wird für die Verbindung der nicht-mehrfachbenutzbaren und mehrfachbenutzbaren Programmteile benötigt. Er wird von der SHARE-Prozedur vor den mehrfachbenutzbaren Modul gebunden.

adaptn Name, den der nicht-mehrfachbenutzbare Adaptermodul erhalten soll.
Dieser Modul wird für die Verbindung der nicht-mehrfachbenutzbaren und mehrfachbenutzbaren Programmteile benötigt.

shrmod Name, den der mehrfachbenutzbare Modul erhalten soll.
shrmod enthält den mehrfachbenutzbaren Bindemodul und den mehrfachbenutzbaren Adaptermodul.

HELP=<u>YES</u> / NO

Standard ist YES. Nach dem Start der SHARE-Prozedur wird eine kurze Erklärung der Bedeutung der einzelnen Parameter ausgegeben.

ER=<u>YES</u> / NO

Standard ist YES. Die erstellten Hilfsdateien T.T.T.T.SHR, T.T.T.T.LNK und T.T.T.T.LST.SHR.&SHRMOD werden gelöscht.

XS=<u>NO</u> / YES

=<u>NO</u>: das Programm wird unterhalb 16 MB geladen.
=YES: das Programm wird oberhalb 16 MB geladen.

MACLIB=<u>$TSOS.MACROLIB</u> / libname

Mit dem MACLIB-Operanden wird die System-Makrobibliothek für die Assemblierung der Adaptermoduln zugewiesen. Zulässige Operandenwerte sind $TSOS.MACROLIB (Voreinstellung) oder ein anderer Bibliotheksname *libname*. Falls XS=YES gesetzt wird, muß eine Makrobibliothek mit Moduln aus einer BS2000-Version ≥ 9.0 angegeben werden.

Portabilität

Der Name des mehrfachbenutzbaren Moduls und die dazugehörige Modulbibliothek können aus Portabilitätsgründen geändert werden, obgleich sie beim Erstellen mit Hilfe der SHARE-Prozedur fixiert wurden.

Dies ist möglich durch eine Änderung der betroffenen Namen im nicht-mehrfachbenutzbaren Adapter. Dort sind die Namen auf der Relativ-Adresse 0 abgelegt in der Form:

```
DC CL8'MODS'
DC CL54'MODBIBS'
```

Die Änderung ist dann mit Hilfe von LMS bei der Übertragung von Bindemoduln, bzw. mit Hilfe von DPAGE bei Übertragung von Lademoduln möglich.

Die Adressen der einzelnen Bindemoduln können dem Binderprotokoll entnommen werden.

5.8.1.3 Beispiel

Das Hauptprogramm PROG steht in der Datei S.PROG und soll nicht-mehrfachbenutzbar übersetzt werden.
Die Unterprogramme SUB1 und SUB2 stehen in der Datei S.SUB und sollen mehrfachbenutzbar übersetzt werden.

```
 PROGRAM  PROG
 INTEGER A,B
 CALL SUB1(A,B)
 IF (A.EQ.100 .AND. B.EQ.20) WRITE(2,*) A,B,'AUFRUF UND
-RUECKSPRUNG OK'
 END

 SUBROUTINE SUB1(X,Y)
 INTEGER X,Y
 X=10
 Y=10.50
 CALL SUB2(X,Y)
 IF (X.EQ.100 .AND. Y.EQ.20) WRITE (2,*) X,Y,'AUFRUF UND
-RUECKSPRUNG AUS SUB2 OK'
 RETURN
 END

 SUBROUTINE SUB2(X,Y)
 INTEGER X,Y
 X=X*X
 Y=Y+Y
 WRITE (2,*) 'HIER SUBROUTINE SUB2'
 RETURN
 END
```

1. Übersetzen:

   Übersetzen des Hauptprogramms PROG:

   ```
   /START-PROG $FOR1
   *COMOPT SOURCE=S.PROG,LIST=(SRC,D,OP,XR)
   *END
   ```

   Übersetzen der Unterprogramme SUB1 und SUB2:

   Durch COMOPT OBJECT=(SHARE) werden mehrfachbenutzbarer Codeteil und nicht-mehrfachbenutzbarer Datenteil getrennt.

   ```
   /START-PROG $FOR1
   *COMOPT SOURCE=S.SUB,LIST=(SRC,D,OP,XR)
   *COMOPT OBJECT=(SHARE)
   *END
   ```

2. Ablegen der mehrfachbenutzbaren und nicht-mehrfachbenutzbaren Bindemoduln:

   Die nicht-mehrfachbenutzbaren Bindemoduln werden in die Bibliothek LIB.NOSHARE abgelegt. Die mehrfachbenutzbaren Bindemoduln werden in der Bibliothek LIB.SHARE abgelegt.

   Hauptprogramm:

```
/START-PROG $FOR1
*COMOPT SOURCE=S.PROG,MODULE-LIBRARY=LIB.NOSHARE,END
```

   Unterprogramme:

```
/START-PROG $FOR1
*COMOPT SOURCE=S.SUB,OBJECT=(SHARE),SHARE-LIBRARY=LIB.SHARE
*COMOPT MODULE-LIBRARY=LIB.NOSHARE,END
```

3. SHARE-Prozedur:

   Die Namen der mehrfachbenutzbaren Moduln werden mit dem EDT vor Aufruf der SHARE-Prozedur in eine Datei mit dem Namen SHRNAM geschrieben.

```
/START-PROG $EDT
 SUB1
 SUB2
 @W'SHRNAM'
 @H
```

   Aufruf der SHARE-Prozedur SYSPRC.FOR1.022.SHARE:

```
/CALL-PROC SYSPRC.FOR1.022.SHARE, SHRNAMES=SHRNAM,LIBN=LIB.NOSHARE,
 LIBS=LIB.SHARE,LIBT=LMS,ADAPTS=ADAPTS,ADAPTN=ADAPTN,
 SHRMOD=SHRMOD,HELP=NO,ER=NO
```

   Zur Verbindung der mehrfachbenutzbaren und nicht-mehrfachbenutzbaren Programmteile werden zwei Adaptermoduln erstellt. Der Adaptermodul namens ADAPTS wird vor den mehrfachbenutzbaren Bindemodul gebunden. Der entstandene mehrfachbenutzbare Modul erhält den Namen SHRMOD. Der Adaptermodul namens ADAPTN für den nicht-mehrfachbenutzbaren Bindemodul wird in der Bibliothek LIB.NOSHARE abgelegt.

4. Mehrfachbenutzbar erklären und Laden des mehrfachbenutzbaren Moduls namens SHRMOD ist Aufgabe des Systemverwalters (siehe "BS2000 Systemverwaltung" [40]).

5. Binden der nicht-mehrfachbenutzbaren Moduln mit TSOSLNK:

```
/START-PROG $TSOSLNK
 PROG PROG,FILENAM=L.PROG              (1)
 INCLUDE PROG,LIB.NOSHARE              (2)
 INCLUDE ADAPTN,LIB.NOSHARE
 RESOLVE ,LIB.NOSHARE                  (3)
 RESOLVE ,FOR1MODLIBS
 END
```

   (1) Der Lademodul soll den Namen PROG erhalten und in der katalogisierten Datei L.PROG abgelegt werden.

   (2) Der nicht-mehrfachbenutzbare Bindemodul PROG und der nicht-mehrfachbenutzbare Adaptermodul namens ADAPTN sollen eingebunden werden. Beide befinden sich in der Bibliothek LIB.NOSHARE.

   (3) Externverweise sollen mit den Bibliotheken LIB.NOSHARE und FOR1MODLIBS aufgelöst werden.

6. Starten des Programms durch Starten des nicht-mehrfachbenutzbaren Lademoduls:

```
/SET-TASKLIB FOR1MODLIBS
/START-PROGRAM FROM-FILE=L.PROG
```

5.8.2 Mehrfachbenutzbare Programme ohne die Prozedur SYSPRC.FOR1.022.SHARE

5.8.2.1 Vorgehensweise

1. Übersetzen mit der Compileroption OBJECT=(SHARE) (siehe Abschnitt 5.8.1.1).
2. Ablegen der mehrfachbenutzbaren und nicht-mehrfachbenutzbaren Bindemoduln in PLAM-Bibliotheken (siehe Abschnitt 5.8.1.1).
3. Mehrfachbenutzbar erklären und Laden des mehrfachbenutzbaren Bindemoduls in den Klasse-4-Speicher.
   Der Systemverwalter erklärt den mehrfachbenutzbaren Bindemodul mit dem Kommando ADD-SHARED-PROGRAM als mehrfachbenutzbar und lädt ihn in den Klasse-4-Speicher (siehe Handbuch "BS2000 Systemverwaltung" [40]).
4. Starten des Programms
   Der Anwender startet bzw. lädt das Programm mit dem dynamischen Bindelader DBL durch Aufrufen des nicht-mehrfachbenutzbaren Bindemoduls:
   ```
   /START-PROGRAM FROM-FILE=*MODULE ...
   ```
   bzw.
   ```
   /LOAD-PROGRAM FROM-FILE=*MODULE ...
   ```

5.8.2.2 Beispiel

Das Hauptprogramm PRNOSHR steht in der Datei SOURCE.PRNOSHR und soll nicht-mehrfachbenutzbar übersetzt werden.
Die Unterprogramme SUBSHR1, SUBSHR2 und SUBSHR3 stehen in der Datei SOURCE.SUB123 und sollen mehrfachbenutzbar übersetzt werden.

```
PROGRAM PRNOSHR
WRITE (2,*) 'START PRNOSHR'
CALL SUBSHR1 ()
WRITE (2,*) 'END PRNOSHR'
END

SUBROUTINE SUBSHR1 ()
WRITE (2,*) '   START SUBSHR1'
CALL SUBSHR2 ()
WRITE (2,*) '   END   SUBSHR1'
END

SUBROUTINE SUBSHR2 ()
WRITE (2,*) '      START SUBSHR2'
CALL SUBSHR3 ()
WRITE (2,*) '      END   SUBSHR2'
END

SUBROUTINE SUBSHR3 ()
WRITE (2,*) '         START SUBSHR3'
WRITE (2,*) '         END   SUBSHR3'
END
```

1. Übersetzen und Ablegen der Bindemoduln in PLAM-Bibliotheken

   Unterprogramme SUBSHR1, SUBSHR2 und SUBSHR3:

   Durch COMOPT OBJECT=(SHARE) werden mehrfachbenutzbarer Codeteil und nicht-mehrfachbenutzbarer Datenteil getrennt.
   Der mehrfachbenutzbare Bindemodul wird in der Bibliothek SHR.LIB abgelegt.
   Der nicht-mehrfachbenutzbare Bindemodul wird in der Bibliothek MOD.LIB abgelegt.

```
/START-PROG $FOR1
*COMOPT SOURCE=SOURCE.SUB123
*COMOPT OBJECT=(SHARE),MODULE-LIBRARY=MOD.LIB,SHARE-LIBRARY=SHR.LIB
*COMOPT END
```

   Hauptprogramm PRNOSHR:

   Das Hauptprogramm PRNOSHR wird übersetzt und der erzeugte Bindemodul wird in die Bibliothek MOD.LIB abgelegt:

```
/START-PROG $FOR1
*COMOPT SOURCE=SOURCE.PRNOSHR,MODULE-LIBRARY=MOD.LIB
*COMOPT END
```

2. Mehrfachbenutzbar erklären und Laden des mehrfachbenutzbaren Moduls in den Klasse-4-Speicher ist Aufgabe des Systemverwalters (siehe Handbuch "BS2000 Systemverwaltung" [40])

3. Zuweisen der TASKLIB und Starten des Programms mit DBL:

```
/SET-TASKLIB LIBRARY=FOR1MODLIBS
/START-PROG FROM-FILE=*MODULE(LIBRARY=MOD.LIB,ELEMENT=PRNOSHR)
```

   Beim Ablauf des Programms erhält man folgende Ausgaben nach SYSOUT:

```
START PRNOSHR
   START SUBSHR1
      START SUBSHR2
         START SUBSHR3
         END   SUBSHR3
      END   SUBSHR2
   END  SUBSHR1
END PRNOSHR
```

6 Programmablauf

6.1 Steuern des Programmablaufs: Operanden des SDF-Kommandos START-FOR1-PROGRAM

```
START-FOR1-PROGRAM

 FROM-FILE = (der FROM-FILE-Operand ist in Abschnitt 5.1 beschrieben)
,CPU-LIMIT = JOB-REST / <integer 1..32767>
,TESTOPT = NONE / AID
,MONJV = *NONE / <full-filename 1..54>
,OBJECT-CONTINUATION = NO / YES  1)
,RUNTIME-OPTIONS = NO / YES(...)
   YES(...)
      LINE-OVERPRINT = LASER / YES / NO
      ,SYSDTA-UNIT = (1,5,97) / list-poss: <integer 0..99>
      ,SYSOPT-UNIT = (7,98) / list-poss: <integer 0..99>
      ,SYSLST-UNIT = (6,99) / list-poss: <integer 0..99>
      ,SYSIPT-UNIT = 8 / list-poss: <integer 0..99>
      ,SYSOUT-UNIT = 2 / list-poss: <integer 0..99>
      ,FOR1-COUNT-UNIT = 6 / list-poss: <integer 0..99>
      ,START = NOT-XS / XS
      ,EXPONENT-UNDERFLOW = UNCHANGED / YES / NO
```

1) OBJECT-CONTINUATION=YES kann nur im Batchbetrieb angegeben werden.

6.2 Übersicht: SDF-Operand RUNTIME-OPTIONS und entsprechende Laufzeitoptionen

SDF-Fragebogen	1. Unterfragebogen	Entspr. Laufzeitoption
RUNTIME-OPTIONS = NO		/PARAMETER CARD = NO
= YES(...)		= YES
	LINE-OVERPRINT = LASER	RUNOPT OVERPRINT = LASER
	= YES	= YES
	= NO	= NO
	SYSDTA-UNIT = (1,5,97)	Standard-Dateinummern für SYSDTA: 1,5,97
	= list-poss: <integer 0..99>	SUBSTITUTE, DTA: n,n,...,END
	SYSOPT-UNIT = (7,98)	Standard-Dateinummern für SYSOPT: 7,98
	= list-poss: <integer 0..99>	SUBSTITUTE, OPT: n,n,...,END
	SYSLST-UNIT = (6,99)	Standard-Dateinummern für SYSLST: 6,99
	= list-poss: <integer 0..99>	SUBSTITUTE, LST: n,n,...,END
	SYSIPT-UNIT = 8	Standard-Dateinummer für SYSIPT: 8
	= list-poss: <integer 0..99>	SUBSTITUTE, IPT: n,n,...,END
	SYSOUT-UNIT = 2	Standard-Dateinummer für SYSOUT: 2
	= list-poss: <integer 0..99>	SUBSTITUTE, OUT: n,n,...,END

	FOR1-COUNT-UNIT = 6	Ausgabe des %COUNT-Protokolls über SYSLST
	= list-poss: <integer 0..99>	SUBSTITUTE, CNT: n,n,...,END
	START = XS	RUNOPT START = XS
	= NOT-XS	keine RUNOPT START
	EXPONENT-UNDERFLOW = UNCHANGED	RUNOPT EXPONENT-UNDERFLOW keine RUNOPT EXPONENT-UNDERFLOW
	= YES	= YES
	= NO	= NO

Tab. 6-1: SDF-Fragebogen RUNTIME-OPTIONS und entsprechende Laufzeitoptionen

6.3 Steuern des Programmablaufs durch Laufzeitoptionen

Durch die Angabe von Laufzeitoptionen [RUNtime-OPTionen] kann der Anwender nach dem Aufruf des Programms den Programmablauf beeinflussen.

Durch Laufzeitoptionen kann der Anwender:

- die vordefinierte Zuordnung von Ein-/Ausgabe-Einheiten zu Systemdateien (Standard-FORTRAN-Dateien) verändern,
- die Interpretation von Vorschubsteuerzeichen für die Drucker regeln,
- die FOR1-STXIT-Behandlung bei der FOR1-Initialisierung unterdrücken,
- den Maschinenadreßmodus beim Ablauf eines FOR1-Programms auf 31 ändern,
- die Einstellung der Exponenten-Unterlauf-Behandlung aktualisieren.

6.3.1 Eingeben von Laufzeitoptionen

Die Eingabe von Laufzeitoptionen wird durch den Operanden CARD des PARAMETER-Kommandos möglich:

/PARAM[ETER]	CARD = { YES / <u>NO</u> }

CARD = YES
: Mit der Angabe CARD=YES kann der Anwender Laufzeitoptionen nach dem Aufruf des Programms eingeben.

  Die Laufzeitoptionen werden entweder

  - aus einer Datei mit dem LINK-Namen FOR1RUN gelesen (/SET-FILE-LINK LINK-NAME=FOR1RUN, FILE-NAME=dateiname) oder
  - von SYSDTA, wenn der LINK-Name nicht vergeben wird.

CARD = <u>NO</u>
: entspricht der Voreinstellung. Der Anwender gibt keine der unten aufgelisteten Laufzeitoptionen an. Für den Programmablauf kommen die Standardwerte zur Anwendung.

Das Kommando bleibt gültig bis LOGOFF, SET-JOB-STEP oder bis zum nächsten Kommando PARAM CARD =...

Die Eingabe der Laufzeitoptionen wird mit END abgeschlossen. Insgesamt darf die Eingabe bis zu 320 Zeichen umfassen.

Beispiel:

```
/PARAM CARD = YES
/START-PROGRAM programm
 GIVE RUNOPT OR END CARD OR ?        (1)
 RUNOPT OVERPRINT = YES
 GIVE RUNOPT OR END CARD OR ?
 A,DTA:11,END                        (2)
```

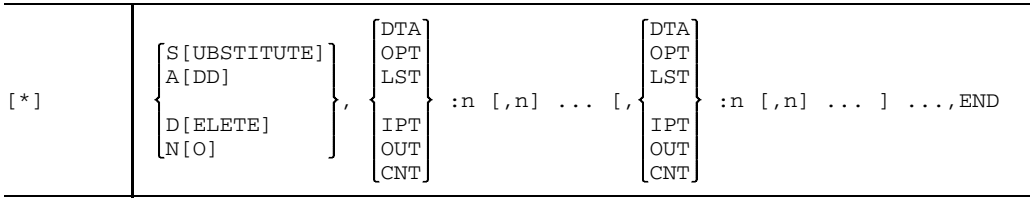
(1) Bei einem Stapelauftrag wird die Ausgabe der Aufforderung "GIVE RUNOPT OR..." unterdrückt, die Eingabemöglichkeit ist jedoch vorhanden.

(2) Wenn die Standardzuordnung von BS2000-System-Dateien geändert wird, muß die entsprechende Laufzeitoption die letzte RUNOPT-Eingabe sein, da Laufzeitoptionen, die die Zuordnung ändern, mit END abgeschlossen werden müssen.

Bei Eingabe eines "?" wird folgende Hilfsinformation ausgegeben:

```
STANDARD PRECONNECTION IS:
SYSDTA  :  1 , 5 , 97
SYSOPT  :  7 , 98
SYSLST  :  6 , 99
SYSIPT  :  8
SYSOUT  :  2
%COUNT  :  6
PLEASE CHANGE THIS PRECONNECTION IN THE FORM:
S<UBSTITUTE>/A<DD>/D<ELETE>/N<O>,(DTA:MM,NN,...)(,OPT:OO,PP,...)
(,LST:...)(,IPT:...)(,OUT:...)(,CNT:...),END
( ) MEANS OPTIONAL
```

6.3.2 Ändern von Dateinummern: Laufzeitoptionen SUBSTITUTE, ADD, DELETE und NO



S[UBSTITUTE]
Die vordefinierten Dateinummern (siehe Abschnitt 8.3.3.1) werden für die spezifizierten Standard-Dateien durch die angegebenen Nummern ersetzt.

Einschränkung:
Keine Dateinummer kann gleichzeitig mehreren BS2000-Systemdateien zugeordnet sein.

A[DD] Die vordefinierten Dateinummern werden für die spezifizierten Standard-Dateien um die angegebenen Nummern erweitert.

D[ELETE]

Die angegebenen Dateinummern werden aus der Menge der vordefinierten Dateinummern entfernt. Ist der UNIT-Parameter einer Ein-/Ausgabe-Anweisung ein Stern ("*"), und wird die für "*" voreingestellte Dateinummer entfernt, so wird diese durch eine andere vordefinierte Dateinummer ersetzt. Löscht man alle voreingestellten Dateinummern zu einer Standard-Datei, so tritt bei einer Ein-/Ausgabe-Anweisung mit "*" als UNIT-Parameter ein Fehler (IC02) auf.

N[O] Es werden keine Änderungen der Dateinummern gewünscht.

n ganzzahliger Wert, $0 \leq n \leq 99$

Hinweis:

Da diese Eingabe durch ein END abgeschlossen wird, müssen andere RUNOPT-Eingaben vorher erfolgen.

Beispiele:

Eingabe	*Wirkung*
`SUBSTITUTE,DTA:10,37,END`	Die Systemdatei SYSDTA erhält die Dateinummern 10 und 37 zugeordnet. Die vordefinierten Nummern 1, 5 und 97 gelten nicht mehr für SYSDTA.
`ADD,OPT:11,12,OUT:13,END`	Für die Systemdatei SYSOPT sind neben den vordefinierten Nummern 7 und 98 auch die Nummern 11 und 12 gültig. Analog gelten für SYSOUT die Nummern 2 und 13.
`S,CNT:15,END`	Die Ausgabe des dynamischen Programmprotokolls (%COUNT) erfolgt nicht über Dateinummer 6 (SYSLST), sondern über Dateinummer 15.
`DELETE,DTA:1,END`	Die voreingestellte Dateinummer 1 wird gelöscht. Die FORTRAN-Anweisung "READ *,A" bewirkt z.B. nun eine Eingabe über die Dateinummer 5.

6.3.3 Steuerung der Vorschubzeichen-Erzeugung für Ausgabe nach SYSLST: RUNOPT OVERPRINT

Für die Ausgabe nach SYSLST regelt RUNOPT OVERPRINT die Erzeugung von Drucker-Vorschubzeichen. Dabei werden FORTRAN-Steuerzeichen in sedezimale Vorschubzeichen umgesetzt (siehe Tab. 6-2). Diese sedezimalen Vorschubzeichen werden dann beim Drucken interpretiert.

`[*]RUNOPT`	`O[VERPRINT] [=[{Y[ES] / N[O] / L[ASER]}]]`

O=YES Ein FORTRAN-Steuerzeichen ungleich "+" erzeugt eine zusätzliche Datenzeile, die das sedezimale Vorschubzeichen enthält (Zweizeilenlösung).
"+" bewirkt nur, daß keine zusätzliche Datenzeile erzeugt wird; der Vorgängersatz wird jedoch nicht überdruckt.

O=NO Das Überdrucken von Datensätzen wird ausgeschlossen. Für Datensätze mit einem Vorschubzeichen ungleich "+" ergibt sich ein deutlicher CPU- und Elapsed-Zeit-Gewinn, da nur eine Zeile pro FORTRAN-Vorschubzeichen ausgegeben wird (Einzeilenlösung).
Das Vorschubzeichen "+" wird als "␣" interpretiert.

O=LASER
Voreinstellung: Das physikalische Überdrucken eines Datensatzes auf Laserdrucker ist genau einmal möglich. Sätze werden nicht sofort ausgegeben, sondern gepuffert (Einzeilenlösung mit Pufferung). Ein mit dem Vorschubzeichen "+" ausgegebener Datensatz überdruckt physikalisch seinen Vorgängersatz. Der Versuch, zwei unmittelbar aufeinanderfolgende Datensätze mit dem Vorschubzeichen "+" auszugeben, wird abgewehrt und führt zum Abbruch des Druckvorgangs.

Bei mehrfacher Angabe von RUNOPT OVERPRINT gilt die letzte. Ist der Operandenteil des OVERPRINT-Operanden leer, z.B. RUNOPT O oder RUNOPT O = , so gilt er als nicht angegeben, und es bleibt bei der Voreinstellung.

Die folgende Tabelle zeigt die Umsetzung der FORTRAN-Steuerzeichen:

FORTRAN-Vorschub-zeichen	Vorschub	Umsetzung in Steuerzeichen		
		O V E R P R I N T		
		YES	NO	LASER
+	Kein Vorschub	00aaaa	40aaaa	00aaaa
'␣'	1 Zeile	41 00bbbb	40bbbb	01bbbb
0	2 Zeilen	42 00cccc	41cccc	02cccc
1	1 Seite (1. Zeile der nächsten Seite)	C1 00dddd	C1dddd	81dddd
{aaaa|bbbb|cccc|dddd}		Daten		

Tab. 6-2: Umsetzung der FORTRAN-Steuerzeichen

Vorschubsteuerung mit RUNOPT OVERPRINT=LASER

Mit [*]RUNOPT OVERPRINT=LASER wird die physikalische Ausgabe eines Datensatzes verzögert, bis der nachfolgende Satz vorliegt. Dieser löst die Ausgabe des zunächst gepufferten Satzes aus, und zwar mit dem Steuerzeichen X'0n' für Vorschub nach dem Drucken. Der Wert für n wird aus dem Steuerzeichen des Nachfolger-Satzes errechnet, dessen Ausgabe wiederum verzögert wird.
Bei der Pufferung des ersten Satzes wird ein Pseudo-Satz mit X'C1' (eine Seite Vorschub) bzw. X'4n' ausgegeben, wobei n über das Steuerzeichen des ersten Satzes errechnet wird.

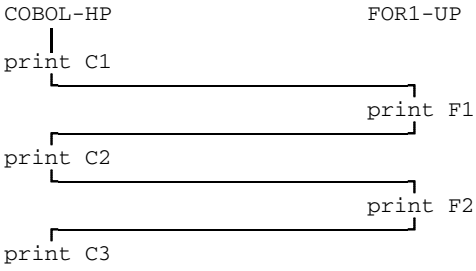
Pufferleerung

Der Anwender kann einen wegen OVERPRINT=LASER gepufferten Satz ausgeben über den Aufruf der FOR1-RTS-Routine I$PRINT.

FORTRAN-Anweisung:CALL I$PRINT

Durch den geeigneten Einsatz dieser Anweisung können Verschiebungen des Druckbilds verhindert werden, die sich eventuell aus der Verwendung anderssprachiger Programmeinheiten ergeben.

Beispiel 1:



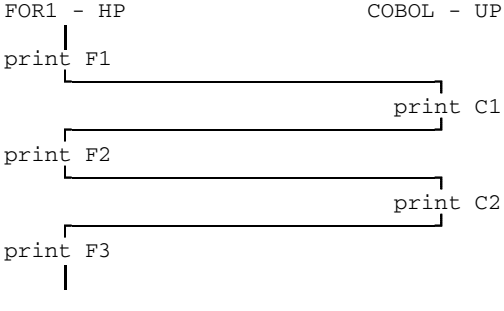
Es ergibt sich folgende Verschiebung (angegeben sind nur die Vorschübe der FORTRAN-Datensätze):

SOLL-Druckbild:	Erzeugtes Druckbild (0=LASER):
C1	C1
(Vorschub von F1)	(Vorschub von F1)
F1	C2
C2	F1
(Vorschub von F2)	(Vorschub von F2)
F2	C3
C3	

Das Vorschubzeichen für F1 wird als Pseudo-Satz ausgegeben. Der Satz F1 dagegen wird bis zum Vorliegen von Satz F2 gepuffert und dann mit dem Vorschubsteuerzeichen von F2 ausgegeben. Satz F2 bleibt im Puffer, da kein FORTRAN-Datensatz folgt.

Abhilfe: Die korrekte Ausgabe der FORTRAN-Datensätze muß über die Anweisung zur Pufferleerung (s.o.) gesteuert werden.

Beispiel 2:



Für das Druckbild ergibt sich:

Soll-Druckbild:	Erzeugtes Druckbild (0=LASER):
(Vorschub von F1)	(Vorschub von F2)
F1	C1
C1	F1
(Vorschub von F2)	(Vorschub von F2)
F2	C2
C2	F2
(Vorschub von F3)	(Vorschub von F3)
F3	F3

Wegen der durch die Zwischenpufferung bedingten Druckverzögerung für die FORTRAN-Datensätze ist die korrekte Reihenfolge nicht eingehalten.

Abhilfe: Entweder ändert der Anwender die Voreinstellung RUNOPT OVERPRINT = LASER ab oder die korrekte Ausgabe der FORTRAN-Datensätze wird über die Anweisung zur Pufferleerung (s.o.) gesteuert.

6.3.4 Unterdrücken der STXIT-Fehlerbehandlungsroutine: RUNOPT STXIT

`[*]RUNOPT`	`STXIT = {YES|NO}`

STXIT=YES

Standardmäßig wird bei der FOR1-Initialisierung eine STXIT-Fehlerbehandlungsroutine angemeldet (siehe Anhang A.8.1).

STXIT=NO

Das Anmelden einer STXIT-Fehlermeldungsroutine wird unterdrückt. Dadurch kann bei Sprachverknüpfungen, die nicht in einer Standard-Linkage-Umgebung ablaufen, verhindert werden, daß sich die STXIT-Anmeldungen gegenseitig überschreiben. In ILCS-Umgebungen ist STXIT=NO wirkungslos, da STXITs nicht von FOR1 selbst, sondern nur von der ILCS-Initialisierung angemeldet werden.

6.3.5 Einstellen des Maschinenadreßmodus: RUNOPT START

Der vom Lader ermittelte Adreßmodus kann durch eine Laufzeitoption beim Starten des Programms noch geändert werden:

[*]RUNOPT	START=XS

START=XS
der Maschinenadreßmodus 31 wird eingestellt.

Ein vom Lader eingestellter bzw. ermittelter Maschinenadreßmodus 24 kann durch RUNOPT START=XS nur geändert werden, wenn das betreffende Programm ein XS-Programm ist. (Ab FOR1 Version 2.2A werden grundsätzlich XS-Moduln erzeugt. Bei FOR1-Versionen < 2.2A mußte zur Erzeugung von XS-Moduln mit EXTENDED-SYSTEM=YES übersetzt werden.)

Ein Programm läuft auch mit RUNOPT START = XS im unteren Adreßraum ab, jedoch können dynamische Felder im oberen Adreßraum erzeugt werden.

Hinweise:

Falls ein NXS-Programm oder ein Programm gemischt aus NXS- und XS-Programmeinheiten vorliegt, das im Maschinenadreßmodus 24 ablauffähig ist, dann erfolgt nach Eingabe von RUNOPT START = XS keine Fehlermeldung. In diesem Fall können beim Programmablauf schwer diagnostizierbare Fehler auftreten.

Auf einer NXS-Anlage wird die Laufzeitoption START=XS ignoriert (Ausgabe einer WARNUNG).

6.3.6 Einstellen der Exponenten-Unterlauf-Behandlung: RUNOPT EXPONENT-UNDERFLOW

Mit der Laufzeitoption EXPONENT-UNDERFLOW kann die Einstellung der Programmunterbrechung wegen Exponenten-Unterlaufs (Bit 2 der Programmaske) zur Laufzeit aktualisiert werden. Die Behandlung eines Exponentenunterlaufs wird durch die Compileroption EXPUNDERFLOW (Voreinstellung: NOEXPUNDERFLOW) festgelegt.

`[*]RUNOPT`	`EXPONENT-UNDERFLOW={YES|NO}`

NO Das Programm wird bei Auftreten eines Exponenten-Unterlaufs nicht unterbrochen. Das Datenelement, bei dem ein Exponenten-Unterlauf aufgetreten ist, wird auf 0 gesetzt.

YES Das Programm wird bei Auftreten eines Exponenten-Unterlaufs mit einer Fehlermeldung unterbrochen.

6.4 Interne Prozeduren zur Initialisierung und Beendigung von Programmen

6.4.1 Programminitialisierung

Bei Beginn des Programmablaufs müssen intern eine Reihe von Initialisierungsmaßnahmen durchgeführt werden. Dies geschieht in einer eigenen Initialisierungsroutine.

Die Initialisierungsroutine bewirkt u.a. die Ausgabe einer Startmeldung. Die Startmeldung wird über SYSOUT ausgegeben und hat folgende Form:

```
BS2000 F O R 1 : FORTRAN PROGRAM "name" STARTED ON datum AT zeit
```

name	Name des Hauptprogramms
datum	laufendes Datum in der Form `jjjj-mm-tt`, beispielsweise: 1991-08-30
zeit	Zeitpunkt des Ladevorgangs in der Form hh:mm:ss

Die Startmeldung kann durch das Setzen des Auftragsschalters 4 (/MODIFY-JOB-SWITCHES ON=4) unterdrückt werden.

6.4.2 Programmbeendigung

Die Programmbeendigungsroutine wird vom Objektprogramm am Ende der Ausführung aufgerufen, d.h. bei einer STOP- bzw. END-Anweisung oder bei einem Aufruf des Unterprogramms EXIT.

Der Aufruf des vorgefertigten Unterprogramms EXIT zur Beendigung des Programms kann in das FORTRAN-Quellprogramm geschrieben werden. Er hat diesselbe Wirkung wie eine STOP-Anweisung. Außerdem wird die Programmbeendigungsroutine von der Fehlerbehandlung aufgerufen, wenn ein Fehler aufgetreten ist, der eine Fortführung des Ablaufs unmöglich macht.

Die Programmbeendigungsroutine führt folgende Arbeiten durch:

1. Abschließen der Ein-/Ausgabe-Operationen bei Aufruf in fehlerhafter Ein-/Ausgabe-Anweisung

   Falls die Routine durch die Fehlerbehandlung aktiviert wurde, wird zuerst geprüft, ob es eine nicht ordnungsgemäß abgeschlossene Ein-/Ausgabe-Operation gibt. Solch eine Operation wird dann mit einer Ein-/Ausgabe-Beendigungsroutine abgeschlossen. Vorher muß eventuell der Inhalt des Ein-/Ausgabe-Puffers ausgegeben werden.

Falls jedoch ein Ein-/Ausgabe-Fehler zur Beendigung des Programms geführt hat, kann die Ein-/Ausgabe-Operation nicht ordnungsgemäß abgeschlossen werden, die entsprechende Datei wird aber geschlossen.

2. Aufheben von intern erzeugten TFT-Einträgen.
3. Aufrufen von evtl. angemeldeten Abschlußprozeduren anderssprachiger Programmteile.
4. Schließen der noch geöffneten Dateien.

   Die Programmbeendigungsroutine schließt alle noch geöffneten Dateien durch einen Aufruf der Routine CLOSE.
   Die Routine wird dabei mit dem Parameterwert KEEP aufgerufen, so daß die Dateien nach Ausführung der CLOSE-Routine weiter zur Verfügung stehen. Lediglich die temporären Arbeitsdateien werden mit dem Parameterwert DELETE geschlossen und stehen anschließend nicht mehr zur Verfügung.
   Falls eine Datei nicht geschlossen werden kann, wird folgende Meldung auf SYSOUT ausgegeben:

   ```
   FILE name COULD NOT BE PROPERLY CLOSED
   ```

   name Name der Datei

5. Ausgeben der Endemeldung.

   Am Ende der Routine wird eine Endemeldung über SYSOUT ausgegeben.

   ```
   BS2000 FOR1:FORTRAN PROGRAM "name" ENDED [PROPERLY|BADLY] AT zeit

              CPU-TIME USED: nn.nnn SECONDS

              ELAPSED TIME: nn.nnn SECONDS
   ```

name	Name des Hauptprogramms
zeit	Zeitpunkt der Programmbeendigung
nn.nnn	verbrauchte CPU-TIME bzw. ELAPSED TIME des Objektprogramms

   Die Ausgabe der Endemeldung wird unterdrückt, wenn der Auftragsschalter 4 gesetzt ist (/MODIFY-JOB-SWITCHES ON=4).

6.5 Fehlerbehandlung zur Laufzeit

6.5.1 Aufbau der Fehlermeldungen

Bei Auftreten eines Laufzeitfehlers wird eine Fehlermeldung über SYSOUT ausgegeben.

Die Fehlermeldung hat folgende allgemeine Form:

```
type ERROR DETECTED IN MODULE "modul" AT hh:mm:ss
WHILE EXECUTING STMT statementnr/SEG segnr IN PROGRAM UNIT "unit"

xxnn: fehlermeldungstext
```

type bezeichnet den Fehlertyp. Nach ihrer Schwere und/oder dem Ort ihres Auftretens werden die FOR1-Laufzeitfehler in folgende fünf Typen unterteilt:

FATAL (siehe 6.5.4)
I/O (siehe 6.5.5)
LIBRARY (siehe 6.5.6)
PROGRAM (siehe 6.5.7)
EXECUTION (siehe 6.5.8)

xx kennzeichnet den Fehlerbedingungs-Code, *nn* die sedezimale Fehlernummer. Folgende Fehlerbedingungs-Codes werden unterschieden:

- Ein/Ausgabefehler

  IC Initial Call Errors
  OP Open Errors
  CL Close Errors
  IQ Inquire Errors
  PO Positioning Erros
  IO Record I/O-Errors
  CO Conversion Errors
  FC Formatted Control Errors
  UC Unformatted Control Errors
  VS Value Separator Errors
  NC Namelist Control Errors
  PA Pause Errors

- Bibliotheksprogramm-Fehler

  SH "Short" functions (z.B. ABS, IMAG)
  PO Potenzierungen
  NU Numerische Funktionen
  CH CHARACTER-Funktionen
  VS XS-spezifische Meldungen

- Programmfehler

  PR Programmüberwachung
  UR Unrepairable error
  AR ARITHMOS-Fehler

6.5.2 Programmfortsetzung bei Laufzeitfehlern

Fehler, die während der Laufzeit (d.h. Ablauf des Programms) auftreten, müssen nicht notwendigerweise zum Programmabbruch führen. In manchen Fällen ist eine Programmfortsetzung erwünscht, z.B. um mit korrigierten Werten aufzusetzen oder aber um in einem Ablauf weitere Fehler zu entdecken. Bedenklich ist die Programmfortsetzung z.B., wenn ein nicht behobener Fehler schwere Folgefehler provoziert oder wenn er in einer Schleife die Diagnose erschwert und die Rechenzeit stark erhöht. Der Anwender hat also von Fall zu Fall selbst zu entscheiden.

Bei Fehlern, die zum Programmabbruch führen, wird eine Fehlermeldung in der beschriebenen allgemeinen Form ausgegeben. Anschließend wird durch die Programmbeendigungsroutine eine Endemeldung ausgegeben.

Bei Laufzeitfehlern, die eine Programmfortsetzung erlauben, wird zunächst eine Fehlermeldung der allgemeinen Form ausgegeben. Die anschließenden Meldungen und die Möglichkeiten der Fehlerbehandlung sind von folgenden Bedingungen abhängig:
- Dialog- oder Stapelbetrieb
- Typ des Fehlers
- ERR- und IOSTAT-Parameter bei Ein-/Ausgabe-Anweisungen
- PARAM DEBUG-Kommando
- Testhilfeunterprogramme OVERFL, DVCHK, FIXOV.

Läuft das Programm im Dialogbetrieb, dann kann der Anwender über die Programmfortsetzung im Dialog an der Datensichtstation entscheiden. Das FOR1-Laufzeitsystem fragt ab, ob
- der Programmablauf fortgesetzt werden soll
- die Aufrufhierarchie ausgegeben werden soll
- eine Programmunterbrechung (BREAKPOINT) gesetzt werden soll.

Wünscht der Anwender eine Programmfortsetzung, dann kann er - abhängig vom Typ des Fehlers - weitere Angaben zur Art der Programmfortsetzung machen. In den folgenden Abschnitten wird das Ablaufdiagramm der Fehlerbehandlung und der Dialog zwischen FOR1 und dem Anwender für jeden Fehlertyp dargestellt.

Läuft das Programm im **Stapelbetrieb**, dann kann der Anwender nicht nach Auftreten des Fehlers über eine Programmfortsetzung entscheiden. Im Stapelbetrieb kann der Anwender jedoch vor dem Programmstart durch Absetzen des Kommandos PARAM DEBUG=YES bzw. durch den SDF-Operanden OBJECT-CONTINUATION=YES im Fehlerfall eine Fehlerbehandlung veranlassen. Falls möglich, setzt das Laufzeitsystem dann

den Programmablauf mit intern vordefinierten Werten ("Standardwerten") fort. Bei PARAM DEBUG=NO bzw. dem SDF-Operand OBJECT-CONTINUATION=NO führen Laufzeitfehlern zur Programmbeendigung.

Im **Dialogbetrieb** wird die Fehlermeldung der allgemeinen Form zusätzlich über SYSLST ausgegeben. Im Stapelbetrieb erscheint die Fehlermeldung im Protokoll des ENTER-Jobs, das über SYSOUT ausgegeben wird. Zusätzlich werden die Fehlermeldungen und Informationen zur Fehlerdiagnose über SYSLST ausgegeben. Die Informationen zur Fehlerdiagnose entsprechen den Informationen, die im Dialogbetrieb an der Datensichtstation abgerufen werden können (Ausgabe von Parametern, der Aufrufhierarchie, der Register).

Mit dem ERR-, END oder dem IOSTAT-Parameter in Ein-/Ausgabe-Anweisungen wird im Fehlerfall in die benutzereigene Fehlerbehandlung verzweigt. Bei Ein-/Ausgabe-Fehlern wird das Programm an der im ERR- bzw. END-Parameter angegebenen Anweisungsmarke fortgesetzt.

6.5.3 Überwachen des Programmablaufs durch Jobvariable

Mit Hilfe des Softwareprodukts JV (Jobvariable) kann der Ablauf eines FORTRAN-Programms überwacht werden (siehe "Jobvariablen" [24]). Eine vorher definierte Jobvariable wird bei einem LOGON-, ENTER-JOB oder START-PROGRAM-Kommando als Operand angegeben. In diese Jobvariable werden Informationen über den aktuellen Zustand des Programms ("Zustandsanzeige") und weitere Informationen über den Ablauf des Programms ("Rückkehrcode-Anzeige") eingetragen. Der Anwender kann diese Informationen abfragen und weitere Aufträge und Programme in Abhängigkeit von diesen Informationen steuern. Die Bedeutung der Anzeigen von Jobvariablen wird in Abschnitt 4.9 beschrieben.

Informationen der Zustandsanzeige in den ersten 3 Bytes einer Jobvariablen:

$T␣ Das Programm wurde normal beendet.

$A␣ Das Programm wurde nicht normal beendet. Diese Anzeige wird ebenfalls bei Programmabbruch durch das System gesetzt.

$R␣ Nach dem Starten eines Programms wird die Zustandsanzeige auf "$R" gesetzt.

Informationen des Beendigungscodes im 4. Byte einer Jobvariablen:

0 Das Objektprogramm ist fehlerfrei gelaufen.

2 Das Objektprogramm ist wegen des Auftretens von Fehlern (ERRORS) kontrolliert beendet worden.

Die Programminformation in Bytes 5 bis 7 der Jobvariablen ist nach dem Lauf eines FORTRAN-Programms mit drei Leerzeichen belegt.

6.5.4 Fatale Fehler

Diese Fehler führen zum Programmabbruch, da nach ihrem Auftreten keine sinnvollen Ergebnisse mehr zu erwarten sind bzw. die Programmfortsetzung unmöglich ist.

Fatale Fehler treten z.B. in folgenden Fällen auf:

- Speicherplatz erschöpft (NO DYNAMIC STORAGE AVAILABLE)
- Endlos-Schleife im Programm (PROGRAM LOOP DETECTED)
- Fehlerhafter Aufruf des Unterprogramms ALLOC (WRONG LIMITS IN ALLOCATION CALL)

Tritt ein fataler Fehler auf, so wird das Programm kontrolliert abgebrochen (Endebehandlung).

6.5.5 Ein-/Ausgabe-Fehler

Diese Fehler treten im Zusammenhang mit Ein-/Ausgabe-Anweisungen auf.

Ein-/Ausgabefehler treten z.B. in folgenden Fällen auf:

- Gerätefehler
- Formatfehler
- Datenfehler (Typen)
- DVS-Fehler
- unzulässige Operationen
- Reihenfolgefehler.

Der Anwender kann eine eigene Fehlerbehandlung programmieren, indem er in Ein-/Ausgabe-Anweisungen den ERR-Parameter angibt. Im Fehlerfall wird dann das Programm an der Anweisungsmarke fortgesetzt, die im ERR-Parameter angegeben wurde.

Wird während der Ausführung einer READ-Anweisung oder Positionierungsanweisung das Dateiende erreicht, dann kann der Anwender durch Angabe des END-Parameters einen Programmabbruch verhindern. Bei Erreichen des Dateiendes wird das Programm an der Anweisungsmarke fortgesetzt, die im END-Parameter angegeben wurde.

Ist kein ERR- bzw. END-Parameter, aber ein IOSTAT-Parameter angegeben, dann wird das Programm im Fehlerfall bzw. bei Erreichen des Dateiendes mit der nächsten ausführbaren Anweisung fortgesetzt.
Ist weder ein ERR- noch ein IOSTAT-Parameter angegeben, dann wird im Fehlerfall eine Fehlermeldung ausgegeben und abgefragt, ob das Programm fortgesetzt werden soll.
Ist weder ein END- noch ein IOSTAT-Parameter angegeben, dann wird bei Erreichen des Dateiendes die Meldung "EOF^x80ON^x80UNIT dsetnr (pfadname)" ausgegeben und das Programm abgebrochen.

Die verschiedenen Möglichkeiten sind in folgender Tabelle zusammengefaßt.

angegebene Parameter	Bedingung	
	Fehler	Dateiende
-	Fehlermeldung;Abfrage: Programmfortsetzung?	EOF-Meldung Programmabbruch
ERR	ERR-Anweisungsmarke	EOF-Meldung Programmabbruch
END	Fehlermeldung;Abfrage: Programmfortsetzung?	END-Anweisungsmarke
IOSTAT	nächste ausführbare Anweisung	nächste ausführbare Anweisung
ERR END	ERR-Anweisungsmarke	END-Anweisungsmarke
ERR IOSTAT	ERR-Anweisungsmarke	nächste ausführbare Anweisung
END IOSTAT	nächste ausführbare Anweisung	END-Anweisungsmarke
END/ERR IOSTAT	ERR-Anweisungsmarke	END-Anweisungsmarke

Tab. 6-3: Programmfortsetzung in Abhängigkeit von ERR-, END-, und IOSTAT-Parameter

Die Fehlermeldung beginnt mit einer Fehlernummer in sedezimaler Darstellung, die der IOSTAT-Fehlernummer in dezimaler Darstellung entspricht. Beispielsweise entspricht in der Meldung

```
OP15: FILE COULD NOT BE OPENED
```

die sedezimale Fehlernummer 15 der IOSTAT-Fehlernummer 21.

Das Programm kann bei derjenigen ausführbaren Anweisung aufsetzen, die der verursachenden Ein-/Ausgabe-Anweisung unmittelbar folgt. Voraussetzung dafür ist aber, daß die standardmäßige Voreinstellung COMOPT TESTOPT=(STNR) gilt, so daß der Compiler eine Statement-Tabelle ins Objekt eingeneriert, die das FOR1-Laufzeitsystem abarbeiten kann.

Die Programmfortsetzung ist sowohl im Dialog- als auch im Stapelbetrieb möglich, wie das folgende Ablaufdiagramm zeigt. Der im Dialog mögliche Abfragemechanismus ist anschließend dargestellt.

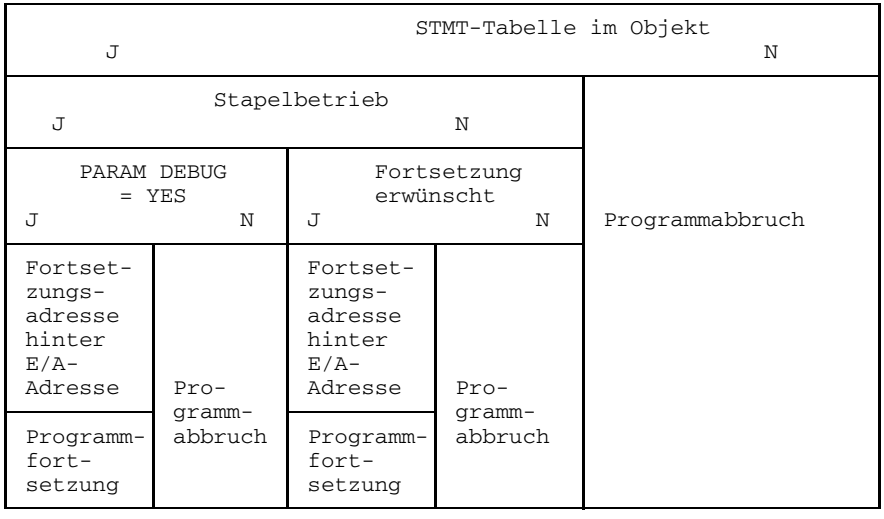


Bild 6-1: Ablaufdiagramm bei Ein-/Ausgabefehlern

Dialog bei Ein-/Ausgabefehlern:

Ausgabe von FOR1	Anwender-Antwort	Reaktion von FOR1
OBJECT CONTINUATION? Y/N	Y	Programmablauf wird hinter fehlerhafter Ein-/Ausgabe-Anweisung fortgesetzt.
	N	Fortsetzung des Dialoges
DO YOU WANT PARAMETERS DISPLAYED? Y/N	Y	Ausgabe von Parametern
DO YOU WISH CALLING SEQUENCE DISPLAYED? Y/N	Y	Ausgabe der Aufrufhierarchie
DO YOU WISH A BREAKPOINT? Y/N	Y	Programmunterbrechung

6.5.6 Fehler bei mathematischen Bibliotheksprogrammen

Diese Fehler treten beim Aufruf von FOR1-INTRINSIC-Funktionen auf, wenn Aktualparameter falsch versorgt werden.

Solche Fehler treten z.B. in folgenden Fällen auf:

- Argument zu groß oder zu klein
- Argument unerlaubterweise negativ
- falscher Argumenttyp
- Division durch Null
- Basis ist Null.

Wiederaufsetzen kann durchaus sinnvoll sein, wenn man Ergebnisse simuliert, welche die zum Fehler führende Funktion hätte produzieren sollen. Zwischen Dialog- und Stapelbetrieb ist zu unterscheiden.

Im Dialog sieht der Anwender die gerufene Funktion und den erwarteten Typ des Ergebnisses, so daß er entsprechend eingeben kann, mit welchem Ergebnis aufgesetzt werden soll.
Im Stapelbetrieb können solche Werte angenommen werden, bei denen die Gefahr von Folgefehlern möglichst gering wird (z.B. 1 oder 1.0).

Das Ablaufdiagramm sowie die im Dialog möglichen Abfragen sind im folgenden dargestellt.

<table>
<tr><td colspan="5">Stapelbetrieb
N J</td></tr>
<tr><td colspan="3">Fortsetzung
J N</td><td colspan="2">PARAM DEBUG=
J YES N</td></tr>
<tr><td colspan="2">Anwender-Antwort:
D für Standardwert
I für INTERRUPT</td><td rowspan="4">Programmabbruch</td><td rowspan="4">Rückkehr mit Standardwerten ins Objekt hinter Aufruf des Bibliotheksprogramms</td><td rowspan="4">Programm-abbruch</td></tr>
<tr><td colspan="2">Anwender-
D Antwort I</td></tr>
<tr><td>Standard-wert in ent-sprechen-de Register setzen</td><td>Wieder-herstel-len der Programm-inhalte bei Pro-gramm-fortset-zung</td></tr>
<tr><td rowspan="2">Rückkehr ins Objekt hinter Aufruf des Bib-liotheks-programms</td><td>BKPT</td></tr>
<tr><td>Rückkehr ins Objekt hinter Aufruf des Bib-liotheks-programms</td></tr>
</table>

Bild 6-2: Ablaufdiagramm bei Bibliotheksfehlern

Dialog bei Bibliotheksfehlern:

Ausgabe von FOR1	Anwender-Antwort	Reaktion von FOR1
OBJECT CONTINUATION? Y/N	N	Fortsetzung des Dialogs bei (1).
	Y	Der Anwender wird aufgefordert, die Art der Fortsetzung anzugeben
DEFAULT VALUE (D) or INTERRUPT (I)	D	Standardwerte werden in entsprechende Register geladen. Ablauffortsetzung hinter CALL
	I	In den Systemmodus verzweigen (BKPT), damit der Anwender mittels AID den gewünschten Wert in entsprechende Register setzen kann
BKPT (1)	AID-Kdos.; /RESUME -PROGRAM	Ablauf wird hinter CALL fortgesetzt
DO YOU WISH CALLING SEQUENCE DISPLAYED? Y/N	Y	Ausgabe der Aufrufhierarchie
DO YOU WISH A BREAKPOINT? Y/N	Y	Programmunterbrechung

Beispiel:

Das folgende Programm enthält ein falsches Argument: negativer Radikand. Beim Ablauf wird ein Bibliotheksfehler gemeldet.

```
/SET-TASKLIB LIB=$FOR1MODLIBS                                          (1)
/DEL-SYS-FILE OMF
/START-PROG $FOR1
%  BLS0500 PROGRAM 'FOR1', VERSION '2.2A00' OF '91-06-05' LOADED.
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1991 ...
 FOR1:    V2.2A00 READY, GIVE COMPILER OPTION
*COMOPT LISTFILE=LIST(OBJECT),END
*      PROGRAM SQ                                                      (2)
*      R=-16
*      X=SQRT(R)
*      WRITE *,X
*      END
*/*                                                                    (3)
 FOR1: NO ERRORS DURING COMPILATION OF P.U. SQ
 END OF  F O R 1  COMPILATION;  CPU TIME USED: 0.191 SEC.
```

```
Auszug aus dem OBJECT-Listing:

****  STATEMENT 3 (ASSIGNMENT) ********
         X = SQRT(R)
         LA    1,264(0,13)
         USING SQ@@@@@@+264,1
         L     15,136(0,12)
         LA    0,1(0,0)
         BALR  14,15                    IF@Q                          (4)
         STE   0,256(0,13)              X                             (5)



/START-PROG FROM-FILE=*MODULE(*OMF)                                   (6)
%  BLS0001 ### DBL VERSION 070 RUNNING
%  BLS0517 MODULE 'SQ' LOADED
 BS2000  F O R 1 : FORTRAN PROGRAM "SQ"
 STARTED ON 1991-08-30 AT 13:57:38
  LIBRARY ERROR DETECTED IN MODULE "IF@Q    " AT 13:57:38             (7)
 WHILE EXECUTING STMT        3/SEG        1  IN PROGRAM UNIT "SQ      "
 NU1A: ARGUMENT X < 0
 OBJECT CONTINUATION ? Y/N
Y
 DEFAULT VALUE "D" OR INTERRUPT "I" ? D/I
I
%  IDA0199 PROGRAM BREAK AT 02C956, AMODE = 24
/%AID CHECK=ALL
/%SET 4 INTO %0E                                                      (8)
 OLD CONTENT:
 -.1600000 E+002
 NEW CONTENT:
 +.4000000 E+001
%  IDA0129 CHANGE? (Y=YES; N=NO)?
Y
/%R
 0.40000000E+01
 BS2000  F O R 1 : FORTRAN PROGRAM "SQ      " ENDED PROPERLY AT 13:58:04
 CPU - TIME USED :       0.0491 SECONDS
 ELAPSED TIME    :      26.2960 SECONDS
```

Erläuterung des Beispiels:

(1) Zuweisen des Laufzeitsystems.
(2) Programmeingabe direkt am Datensichtgerät.
(3) Eingabeende.
(4) Der BALR-Befehl bewirkt den Aufruf der Wurzelfunktion IF@Q.
(5) Das Ergebnis wird mit dem Befehl STE aus dem Gleitpunktregister 0 in das Ergebnisfeld gebracht.
(6) Programmablauf.
(7) Fehlermeldung.
(8) Verändern eines Speicherinhalts (Gleitpunktregister 0; einfach genau).

6.5.7 Programmfehler

Diese Fehler werden bei der Ausführung der Maschinenbefehle erkannt und über STXIT-Routinen dem FOR1-Laufzeitsystem zur weiteren Bearbeitung übergeben. Es handelt sich um Unterbrechungsereignisse der STXIT-Ereignisklassen "Programmüberprüfung" und "nicht behebbarer Programmfehler" (vgl. Handbuch "Makroaufrufe an den Ablaufteil" [26]).

Programmfehler sind z.B.:

- Adressübersetzungsfehler
- illegaler SVC bzw. illegaler Operationscode
- Adressierungsfehler
- Datenfehler
- Exponentenüberlauf /-unterlauf
- Divisionsfehler
- Dezimalüberlauf
- Festpunktüberlauf.

Die Fehlerart bestimmt, wo der Programmablauf wiederaufsetzt. Zwischen arithmetischen Fehlern und übrigen Fehlern wird unterschieden.

Arithmetische Fehler:

Gekennzeichnet sind sie durch die hardwarebedingten Unterbrechungsgewichte X'64' - X'78' (siehe Handbuch "Assemblerbefehle - Beschreibung" [8]). Der Anwender kann im Ergebnisregister des unterbrochenen Maschinenbefehls ein Ergebnis simulieren, das zum Weiterrechnen geeignet ist. Bei einigen Fehlern gibt es die Möglichkeit, durch Einbinden der entsprechenden Testhilfeunterprogramme (OVERFL, FIXOV bzw. DVCHK) einen Programmabbruch zu verhindern und Standardergebnisse einsetzen zu lassen. Sofern diese nicht eingebunden sind, kann das simulierte Rechenergebnis

- entweder im Dialog vom Anwender eingegeben werden (Abfragemechanismus)
- oder im Stapelbetrieb vom Laufzeitsystem vorgegeben werden.

Der Programmablauf wird hinter dem unterbrochenen Maschinenbefehl fortgesetzt.

Übrige Fehler:

Gekennzeichnet sind sie durch die Unterbrechungsgewichte X'48' - X'60'. Hier brauchen keine Ergebnisse bereitgestellt zu werden. Hier wird lediglich unterschieden, ob das Laufzeitsystem oder das FOR1-Objekt unterbricht:

- Unterbrechung im Laufzeitsystem: Hinter dem Aufruf der betreffenden Routine wird im Objekt aufgesetzt.
- Unterbrechung im Objekt: Die unterbrechende FORTRAN-Anweisung wird übergangen und bei der nachfolgenden Anweisung aufgesetzt.

Beim Auftreten von Programmfehlern wird zunächst eine Fehlermeldung der allgemeinen Form ausgegeben. Danach werden die Absolutadresse, die Adresse relativ zum Modulanfang und der Maschinenbefehl ausgegeben, bei dem der Fehler auftrat:

```
PROGRAM COUNT AT INTERRUPT OCCURENCE: absolutadresse CC=n

PROGRAM COUNT RELATIVE TO ENTRYPOINT: relativadresse

STATEMENT CODE: maschinenbefehl
```

Anschließend wird der Inhalt der Mehrzweck- und Gleitpunktregister ausgegeben. Im Dialogbetrieb kann der Anwender dann über die Programmfortsetzung entscheiden.

Das Ablaufdiagramm ist im folgenden dargestellt. Der Abfragemechanismus für den Dialogbetrieb entspricht dem bei Bibliotheksfehlern, jedoch wird der Programmablauf hinter dem fehlerhaften Maschinenbefehl fortgesetzt. Das simulierte Ergebnis der fehlerhaften Operation wird in das Ergebnisregister geladen, dessen Nummer man aus dem Befehlscode entnimmt.

Geschachtelte Fehlerbehandlung ist möglich, so daß in der Fehlerroutine auftretende Fehler behandelt werden können.

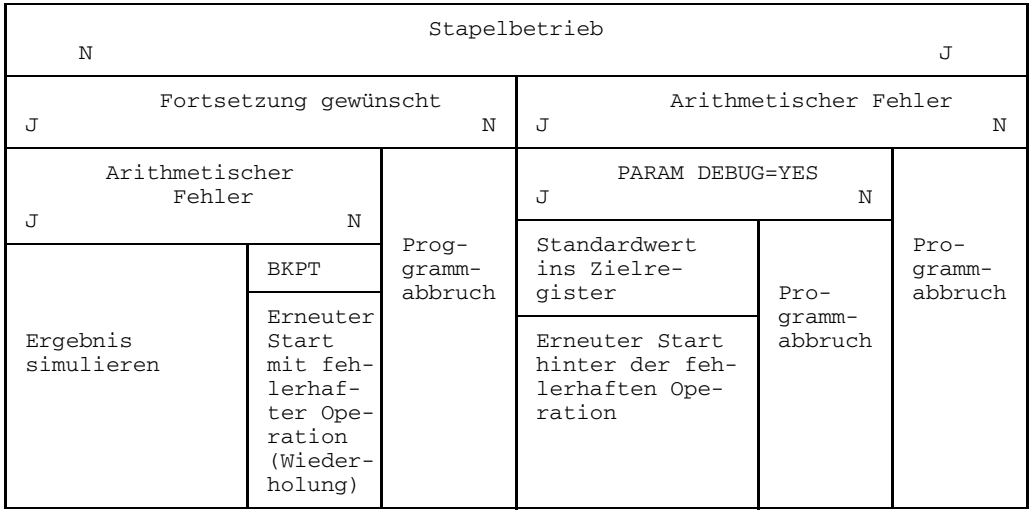


Bild 6-3: Ablaufdiagramm bei Programmfehlern

Ergebnis-Simulation:

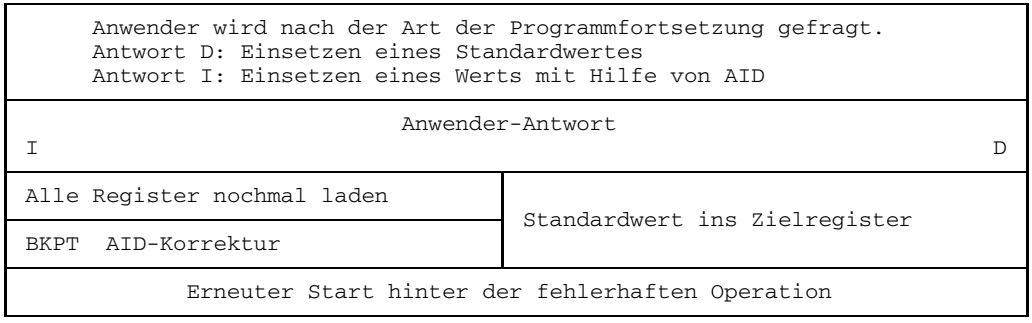


Bild 6-4: Ergebnissimulation bei Programmfehlern

6.5.8 Fehler bei Testoptionen, Testanweisungen, irregulärem Kontrollfluß

Diese Fehler werden in der Fehlermeldung als "EXECUTION ERROR" bezeichnet. Drei verschiedenartige Fehler werden unterschieden.

Fehler als Folge von TESTOPT

Da diese Fehler nur dann gemeldet werden, wenn beim Übersetzen die Testoptionen eingeschaltet waren, kann man die Fehlermeldungen wie Warnungsmeldungen auffassen und im Dialogbetrieb nach Meldungsausgabe mit der Bearbeitung des Programms ganz normal fortfahren. Das Objekt sorgt dafür, daß Register 14 eine verwendbare Rückkehradresse erhält. In der Regel wird als Folge des Fehlers bald ein weiterer Fehler gemeldet werden.
Im Batchbetrieb erfolgt ein Programmabbruch.

Fehler bei der %COUNT-Auswertung

Da die %COUNT-Anweisung ganz am Ende des Programmlaufs ausgewertet wird - also nach abgeschlossenem Produktionslauf - kann man einfach die Auswertung und Listenerzeugung abbrechen und das Objekt mit den weiteren Programmbeendigungsroutinen fortfahren lassen (Schließen der Dateien, Aufruf der Abschlußprozeduren, Laufzeitberechnung).

Fehler durch irregulären Kontrollfluß

Folgende Fälle sind gemeint:

- unzulässiger Wert in DO-Schleife
- unzulässige Referenz in assigned GOTO

- rekursiver Unterprogrammaufruf
- falsche END- oder ERR-Angabe
- falsche Format-Angabe
- zu wiederholende Formatgruppe enthält kein Konvertierungsformat
- Schrittweite in DO-Schleife ist Null
- DO-Kontrollvariable außerhalb der Schleife überschrieben

Bei jedem dieser Fehler muß der Anwender individuell entscheiden, wo er den Programmablauf fortsetzen will. Zwischen Dialog- und Stapelbetrieb wird unterschieden: Im Dialogbetrieb entscheidet der Anwender, ob und wenn ja bei welcher Adresse, bei welcher Marke oder bei welcher FORTRAN-Anweisung er den Ablauf fortsetzen möchte. Im Stapelbetrieb erfolgt Programmabbruch.

Nachfolgend sind das Ablaufdiagramm und der Mechanismus der Dialogabfrage dargestellt.

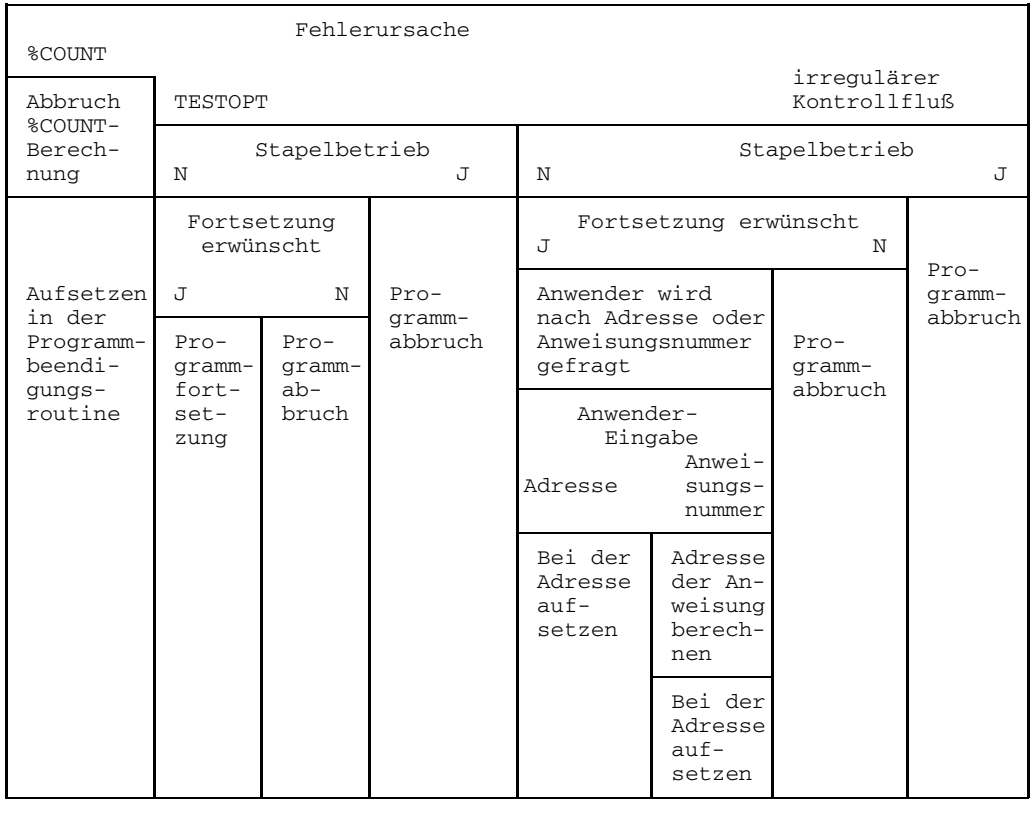


Bild 6-5: Ablaufdiagramm für Fehler mit Testoptionen, Testanweisungen, irregulärem Kontrollfluß

Dialog bei irregulärem Kontrollfluß:

Ausgabe von FOR1	Anwender-Antwort	Reaktion von FOR1
OBJECT CONTINUATION? Y/N	N	Fortsetzung des Dialogs bei (1).
	Y	Der Anwender wird aufgefordert, die Adresse oder Anweisungsnummer anzugeben
GIVE STORAGE ADDRESS (NNNNNN) OR STMT-NR (%NNN) WHERE TO CONTINUE	NNNNNN	Der Programmablauf wird an der angegebenen Adresse fortgesetzt
	%NNN	Die Adresse der angegebenen Anweisung wird mittels Anweisungstabelle ausgerechnet und der Ablauf wird dort fortgesetzt
(1) DO YOU WISH CALLING SEQUENCE DISPLAYED? Y/N	Y	Ausgabe der Aufrufhierarchie
DO YOU WISH A BREAKPOINT? Y/N	Y	Programmunterbrechung

7 Testhilfen

Auch ein fehlerfrei übersetztes FORTRAN-Programm kann möglicherweise noch Fehler aufweisen, die sich erst beim Ablauf des Programms auswirken. Als Testhilfen zum Auffinden solcher Laufzeitfehler stehen dem Anwender zur Verfügung:

- Testoptionen in Form von Compileroptionen,
- Testanweisungen in Form von Compiler-Steueranweisungen,
- Testhilfe-Unterprogramme in Form von vorgefertigten Unterprogrammen,
- die Dialogtesthilfe AID (ab AID-Version 1.0C).

7.1 Steuern der Testhilfen: SDF-Operand TEST-SUPPORT

```
START-FOR1-COMPILER

,TEST-SUPPORT = STD / NO / PARAMETER(...)

   PARAMETER(...)
      | STATEMENT-TABLE = YES / NO
      |
      |,TOOL-SUPPORT = NO / AID
      |
      |,CHECK-CODE = NO / ALL / YES(...)
      |
      |   YES(...)
      |      | PROCEDURE-ARGUMENTS = NO / YES
      |      |,ARRAY-BOUNDS = NO / YES
      |      |,ARRAY-SUBSCRIPTS = NO / YES
      |      |,SUBSTRING-BOUNDS = NO / YES
      |      |,BRANCH-STMTS = NO / YES
      |      |,VARIABLE-ASSIGNMENT = NO / YES
      |      |,USER-DEBUG-STMTS = NO / YES
```

Eine Gegenüberstellung von SDF-Operanden und entsprechenden Compileroptionen enthält Tab. 2-10.

7.2 Übersicht: SDF-Operand TEST-SUPPORT und entsprechende Compileroptionen

In folgender Übersicht wird in der ersten Zeile die Compileroption angegeben, in der zweiten Zeile der entsprechende SDF-Operand, darunter wird die Bedeutung erklärt.

```
TESTOPT=(ALL)
CHECK-CODE=ALL
```

Alle im folgenden genannten Prüfungen werden eingeschaltet.

```
TESTOPT=(STNR)
STATEMENT-TABLE=YES
```

Bei allen Fehlermeldungen wird die entsprechende Quellprogramm-Anweisungsnummer ausgegeben. Die Voreinstellung TESTOPT=(STNR) gilt immer, sofern sie nicht durch NOTESTOPT [=(ALL)] ausgeschaltet wird (siehe Beispiele am Ende dieses Abschnitts).

```
TESTOPT=(ARG)
CHECK-CODE=YES(PROCEDURE-ARGUMENTS=YES)
```

Beim Eingang in ein Unterprogramm werden die übergebenen Parameter nach Anzahl, Typ und Länge geprüft. Ruft ein fremdsprachiges ILCS-Programm ein FOR1-ILCS-Objekt, so wird nur die Anzahl der Parameter verglichen.

```
TESTOPT=(BOUNDS)
CHECK-CODE=YES(ARRAY-BOUNDS=YES)
```

Bei Feldelementen wird geprüft, ob jeder Index innerhalb seiner deklarierten Grenzen liegt (siehe Indexrechnung, Abschnitt 9.3.4).

```
TESTOPT=(SUBSCR)
CHECK-CODE=YES(ARRAY-SUBSCRIPTS=YES)
```

Bei Feldelementen wird geprüft, ob die nach der üblichen Formel berechnete Elementadresse innerhalb des Feldes liegt. Diese Prüfung ist eine Abschwächung der vorigen.

```
TESTOPT=(STRING)
CHECK-CODE=YES(SUBSTRING-BOUNDS=YES)
```

Bei Teilkettenzugriffen wird geprüft, ob die angesprochene Teilkette ganz innerhalb der zugrundeliegenden Variablen liegt.

```
TESTOPT=(CNTRL)
CHECK-CODE=YES(BRANCH-STMTS=YES)
```

Alle gesetzten Sprünge und alle Sprünge in Schleifen hinein werden auf Korrektheit geprüft. Der Compiler prüft, ob die Schleifenvariable eine Wertveränderung erfährt.

```
TESTOPT=(UNDEF)
CHECK-CODE=YES(VARIABLE-ASSIGNMENT=YES)
```

Bei der Benutzung der Werte von Variablen wird geprüft, ob sie definierte Werte haben.

```
TESTOPT=(DEBUG)
CHECK-CODE=YES(USER-DEBUG-STMTS=YES)
```

Die Testanweisungen (siehe Abschnitt 7.4) werden von FOR1 mitübersetzt.

Durch das Präfix NO lassen sich Komplementärmengen spezifizieren (siehe Abschnitt 2.3.1).

7.3 Steuern der Testhilfen durch Compileroption TESTOPT

Mit der Compileroption TESTOPT generiert der Anwender Prüfungen zum Auffinden von Laufzeitfehlern in das Objektprogramm. Auf Grund dieser Prüfungen werden zur Übersetzungs- bzw. Ablaufzeit Fehler erkannt und gemeldet, die im weiteren Ablauf zu falschen Ergebnissen, unerklärlichem Verhalten oder Programmabbruch führen könnten. Solche Fehler werden in der ausgegebenen Meldung als "EXECUTION ERROR" bezeichnet. Die Semantik des Programms wird durch Testoptionen nicht beeinflußt.

Testoptionen steuern ferner die Ausgabe der Anweisungsnummern (STMT) im Falle von Laufzeitfehlern und die Wirksamkeit der Testanweisungen (siehe Abschnitt 7.4).

Testoptionen werden in der TESTOPT-Option spezifiziert.

`[*]COMOPT`	`[NO] TESTOPT [=([testparameter][,...])]`
	
	`testparameter:={ALL|STNR|ARG|BOUNDS|`
	`SUBSCR|STRING|CNTRL|UNDEF|`
	`DEBUG}`

ALL Alle in diesem Abschnitt beschriebenen Aktionen werden durchgeführt

STNR Bei Laufzeitfehlern wird in der Fehlermeldung auch die Anweisungsnummer (STMT) der Anweisung, die zum Fehler führte, ausgegeben (Standard). Keine Belastung des Ablaufs durch zusätzlichen Code.

ARG Bei Aufruf von Funktionen und Unterprogrammen werden Anzahl, Typ und Länge der Aktual- und der Formalparameter auf Übereinstimmung geprüft.

Die Überprüfung ist bei Unterprogrammen, die von anderssprachigen Programmen aufgerufen werden, nicht möglich (außer bei Assembler, wenn die entsprechenden Makros verwendet wurden und bei PLI1).
Ruft ein fremdsprachiges ILCS-Programm ein FOR1-ILCS-Objekt, so wird nur die Anzahl der Parameter verglichen.

Bei der Programmverknüpfung von ALT-, NXS- und XS-Objektprogrammen (siehe Abschnitt A.7) werden folgende Überprüfungen durchgeführt und gegebenenfalls Meldungen zur Laufzeit ausgegeben:

- Bei der Verknüpfung von ALT- mit NXS-Programmen darf kein dynamisches Feld als Parameter übergeben werden;
- einem dynamischen Feld als Aktualparameter muß als Formalparameter ebenfalls ein dynamisches Feld mit der gleichen Dimensionszahl entsprechen;
- Aufrufe von ALT- und NXS-Unterprogrammen dürfen nur im Maschinenadreßmodus 24 erfolgen;

- im XSTONXS-Aufruf darf kein XS-Programm als Unterprogramm stehen;
- einem XS-Unterprogramm darf keine ALT-Parameterliste übergeben werden;
- einem ALT-Unterprogramm darf keine XS-Parameterliste übergeben werden.

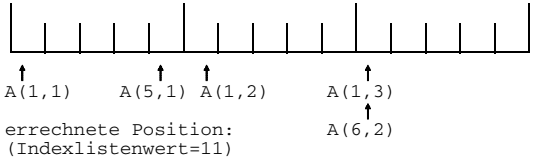
BOUNDS Bei jedem Feldelementnamen wird geprüft, ob die einzelnen Indexwerte innerhalb der Indexgrenzen dieses Feldes liegen. Bei Angabe von BOUNDS werden für die unter SUBSCR beschriebenen Beispiele Fehler gemeldet.
Diese Prüfung ist nur bei ausgeschalteter Optimierung möglich. Ist BOUNDS in Verbindung mit OPTIMIZE>0 angegeben, so schaltet FOR1 SUBSCR ein und gibt eine Meldung aus.

SUBSCR Bei jedem Feldelementnamen wird geprüft, ob die errechnete Position des Feldelements (Indexlistenwert) innerhalb der Grenzen des Feldes liegt (siehe Formel im Abschnitt 9.3.4).

Beispiel:

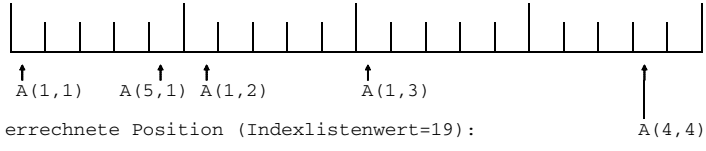
```
DIMENSION A(5,3)
I=6
J=2
A(I,J)= ...
```

Dieses Element liegt innerhalb der Grenzen des Feldes, es wird kein Fehler gemeldet.




```
DIMENSION A(5,3)
I=4
J=4
A(I,J)= ...
```

In diesem Fall liegt das Feldelement außerhalb der Grenzen des Feldes, es wird daher eine Fehlermeldung ausgegeben.



STRING Bei Verwendung von CHARACTER-Teilketten wird überprüft, ob die angegebenen Werte für die Grenzen der Teilkette innerhalb der zugehörigen CHARACTER-Variablen liegt. Außerdem wird gemeldet, wenn der Wert der unteren Grenze der Teilkette größer als der Wert der oberen Grenze ist.

Beispiel:

```
CHARACTER*20 B
I=20
J=22
B(I:J) = ...
```

Es wird ein Fehler gemeldet, weil das Ende der Teilkette außerhalb der Variablen liegt.

CNTRL Sprünge innerhalb einer Programmeinheit werden auf ihre Zulässigkeit geprüft.

Bei Sprüngen in den Bereich von DO-Schleifen generiert der Compiler den Prüfcode nur dann, wenn gleichzeitig ein Sprung aus dem Schleifenbereich heraus vorgesehen ist ("Extended Range" zur Auslagerung von Teilen des Schleifenbereichs). Zur Laufzeit wird dann geprüft, ob der Iterationszähler dieser und von eventuell vorhandenen umfassenden Schleifen größer als Null ist und ob die Laufvariable dieser Schleifen verändert wurde.

Der Compiler prüft, ob der Wert der Schleifenvariable innerhalb des Schleifenbereichs verändert wurde. Falls eine Zuweisung, Mehrfachzuweisung, ASSIGN-, READ-, oder DECODE-Anweisung vorhanden ist, wird eine Warnung (SA 168) ausgegeben.

Bei einer gesetzten Sprunganweisung wird geprüft, ob der entsprechenden Variablen mit einer ASSIGN-Anweisung eine gültige Sprungmarke zugewiesen wurde. Die gleichen Prüfungen werden auch bei gesetzten Marken im ERR- und END-Parameter in Ein-/Ausgabe-Anweisungen und bei gesetztem Format durchgeführt.

UNDEF Bei jeder Verwendung eines Datenelements wird geprüft, ob es schon einen Wert erhalten hat. Damit diese Prüfung durchgeführt werden kann, werden zu Beginn die Datenabschnitte aller Programmeinheiten in den Teilen, die nicht auf Grund von Benutzerangaben initialisiert sind, byteweise mit sedezimal ’80’ initialisiert. Bei Datenelelementen, die beim Ablauf des Programms dieses Bitmuster (X’80’) als Wert aufweisen, wird angenommen, daß sie noch keinen Wert erhalten haben. Beim Zugriff auf den Wert wird daher eine Meldung ausgegeben. Bei INTEGER*1-Datenelementen wird zwar die Initialisierung mit X’80’, nicht jedoch die Prüfung durchgeführt, da es nur 256 mögliche Werte gibt und X’80’ den Wert -128 repräsentiert. Diese Prüfung wird auch nicht bei CHARACTER-Datenelementen mit variabler Länge ausgeführt, da hier die

Länge 0 zulässig ist (siehe "FOR1-Beschreibung" [21]). Dynamische Felder werden ebenfalls nicht auf X'80' überprüft.

Falls ein UNDEF-Testmuster bei der Prüfung erkannt wird, wird der Anwender mit einer Fehlermeldung (zur Laufzeit des Programms) darauf hingewiesen. Der Anwender kann über eine mögliche Fortsetzung des Programmlaufs entscheiden (siehe Abschnitt 6.5.8).
Variablen, bei denen zur Übersetzungszeit bekannt ist, daß sie einen definierten Wert haben (z.B. durch Initialisierung in der DATA-Anweisung), werden zur Laufzeit nicht überprüft. Wird in einem Feld mindestens ein Feldelement definiert, so gilt zur Übersetzungszeit das ganze Feld als definiert. Zur Ablaufzeit werden die nicht-definierten Feldelemente beanstandet.

DEBUG Für die im Quellprogramm angegebenen Testanweisungen (siehe Abschnitt 7.4) werden Befehlsfolgen erzeugt. Bei Nichtangabe dieses Operanden werden die Testanweisungen ignoriert.

Einschränkungen:

– Die Compileroptionen TESTOPT und OPTIMIZE=3,4 schließen sich gegenseitig aus. Die zuletzt angegebene Compileroption wird berücksichtigt.

– Wird TESTOPT zusammen mit PROCEDURE-OPTIMIZATION=YES oder PROCEDURE-OPTIMIZATION=SPECIAL angegeben, dann wird PROCEDURE-OPTIMIZATION auf NO zurückgesetzt.

Beispiele:

Angegebene Option:	Wirkung:
[*]COMOPT TESTOPT=(ALL)	Alle Prüfungen werden durchgeführt.
[*]COMOPT NOTESTOPT=(STRING)	Alle Prüfungen - mit Ausnahme der bei STRING beschriebenen - werden durchgeführt.
[*]COMOPT NOTESTOPT	Es werden überhaupt keine Prüfungen durchgeführt Auch die Ausgabe der Anweisungsnummern bei Laufzeitfehlern wird unterdrückt.
[*]COMOPT NOTESTOPT=(STNR)	Da die Voreinstellung TESTOPT=(STNR) immer gilt, sofern sie nicht durch NOTESTOPT[=(ALL)] ausgesschaltet wird, werden alle Prüfungen einschließlich STNR durchgeführt.

7.4 Testanweisungen (Steuern der Testhilfen durch Anweisungen im Quellprogramm)

Die Testanweisungen werden ins FORTRAN-Quellprogramm geschrieben und bieten eine Hilfe zum Austesten von Programmen. Sie werden bei der Übersetzung in Befehlsfolgen übersetzt, die Informationen über die Struktur und den Ablauf des Programms und über die dynamischen Werte der einzelnen Daten im Programm ausgeben.

Im Gegensatz zu den Testoptionen (siehe Abschnitt 7.4), die für den kompletten Übersetzungsvorgang gelten, können die Testanweisungen an gezielten Stellen im Quellprogramm geschrieben werden. Sie sind nur in der Programmeinheit gültig, in der sie definiert wurden, und zwar für einen vom Anwender bestimmten Bereich.

Die Testanweisungen beginnen mit einem Prozentzeichen.
Vor Testanweisungen dürfen keine Anweisungsmarken stehen.

Voraussetzung für die Übersetzung der Testanweisungen ist die Compileroption COMOPT TESTOPT = (DEBUG). Wird diese Compileroption nicht angegeben, dann werden die Testanweisungen ignoriert.

7.4.1 Übersicht: Testanweisungen

Format	**Bedeutung**
%DISPLAY [(unit)] name [,name]...	Ausgabe der Werte von Variablen und Feldern.
%CHECK { ON / OFF } [(unit)][name[,name]...]	Ausgabe der neuen Werte von Variablen und Feldern bei Zuweisungen zwischen ...ON und ...OFF.
%CALLTRACE { ON / OFF } [(unit)]	Ausgabe der Aufrufinformation rufende/gerufene Programmeinheit, Anweisungsnummer der Aufrufstelle, Aufrufstelle, Eingangspunkt, Parameteradressen und -werte bei jedem Aufruf zwischen ...ON und ...OFF.
%JUMPTRACE { ON / OFF } [(unit)]	Protokollierung der ausgeführten Sprünge zu Anweisungsmarken zwischen ...ON und ...OFF.
%FULLTRACE { ON / OFF } [(unit)]	Protokollierung der Nummern der ausgeführten Anweisungen sowie Ausgabe der neuen Werte von Variablen und Feldelementen bei Zuweisungen zwischen ...ON und ...OFF.
%COUNT { ON / OFF }	Zählung der Ausführungshäufigkeit und des Zeitanteils der Anweisungen zwischen ...ON und ...OFF.

unit Variable oder Konstante
name Name eines Datenelements

Über den Parameterwert *unit* kann die Dateinummer der Datei angegeben werden, auf die die Ausgabe erfolgen soll. Standardmäßig erfolgt die Ausgabe auf SYSLST.

Falls in einer Testanweisung eine Dateinummer angegeben ist, so gilt sie für alle weiteren gleichen Testanweisungen, bis eine weitere gleiche Testanweisung eine neue Dateinummer spezifiziert.

Mit dem Operanden ON wird die Testfunktion eingeschaltet, mit dem Operanden OFF ausgeschaltet.

Wird keine Anweisung zum Ausschalten der Testfunktion gegeben, so wird die Testfunktion am Ende der Programmeinheit ausgeschaltet.

Wird in einer Testanweisung der Operand OFF angegeben und eine Dateinummer, die mit der aktuell gültigen Dateinummer für diese Testanweisung übereinstimmt, dann schaltet die Anweisung die Testfunktion aus und setzt die Ausgabe für diese Testanweisungen auf SYSLST zurück.

Wurde für die angegebene Dateinummer kein FILE-Kommando angegeben, so wird eine Datei mit folgendem Standardnamen erzeugt:

```
DBG.FOR1.stmt.prog.unit[.tsn[.time]]
```

wobei

stmt	Name der Testanweisung in der Form #DISPL, #CHECK, #FULL, #CALL oder #JUMP
prog	Name der Programmeinheit
unit	Dateinummer
tsn	task sequence number, vierstellige Zahl
time	aktuelle Zeit der Dateierstellung in der Form hhmmss

Die Qualifizierungen *tsn* und *time* werden nur durchgeführt, wenn es für die Eindeutigkeit des Eintrags notwendig ist.

7.4.2 %DISPLAY-Anweisung

Mit der %DISPLAY-Anweisung können die Namen und aktuellen Werte von Datenelementen ausgegeben werden.

```
%DISPLAY [(unit)] name [,name]...
```

unit	Variable oder Konstante
name	Name eines Datenelements, d.h. einer Variablen oder eines Feldes

Form der Ausgabe für Variable bzw. Feld:

```
 %DISPLAY/prog/ddddd:
name = wert,

 %DISPLAY/prog/dddddd:
name = wert1, wert2, wert3, ...
```

prog	Name der Programmeinheit
ddddd	Anweisungsnummer der %DISPLAY-Anweisung
name	Name eines Datenelements, das in der %DISPLAY-Anweisung angegeben wurde.

Die Datenelemente werden in der gleichen Reihenfolge ausgegeben, wie sie in der zugehörigen %DISPLAY-Anweisung angeführt wurden. Die Werte der einzelnen Datenelemente werden mit dem Standardformat für den jeweiligen Typ ausgegeben. Die Standardformate sind im Handbuch "FOR1-Beschreibung" [21] beschrieben. Wird ein Feldname spezifiziert, so werden alle Elemente des Feldes ausgegeben.

7.4.3 %CHECK-Anweisung

Mit der %CHECK-Anweisung lassen sich die Wertänderungen von Variablen verfolgen.

```
%CHECK { ON  } [(unit)] [name [,name]...]
       { OFF }
```

unit	Variable oder Konstante
name	Name eines Datenelements, d.h. einer Variablen oder eines Feldes

%CHECK ON (unit) name,name,...name
schaltet die Testfunktion für die angegebenen Datenelemente ein.

%CHECK OFF (unit) name,...name
schaltet die Testfunktion für die angegebenen Datenelemente ab.

%CHECK OFF (unit)
ohne Angabe von Datenelementen schaltet zwar die Prüfung ab, eine nachfolgende %CHECK ON-Anweisung schaltet die Prüfung jedoch für die neu angegegebenen *und* die alten Datenelemente wieder ein.

Die Anweisung %CHECK ON (unit) ohne Angabe von Datenelementen als erste %CHECK-Anweisung ist sinnlos.

Ändert sich der Wert eines spezifizierten Datenelements, so wird der aktuelle Wert des Datenelements am Ende der betreffenden Anweisung ausgegeben.

Position für mögliche Wertänderung	**betroffene Daten**
Linke Seite von Zuweisungsanweisung, Ein-/Ausgabe-Liste in Eingabeanweisung, Speicher-Ein-/Ausgabe-Einheit in Ausgabe-Anweisungen	alle vorkommenden Daten
ENTRY-Anweisung	alle Formalparameter der Eingangs- und COMMON-Daten der Programmeinheit
Funktionsaufruf und CALL-Anweisung	vorkommende Aktualparameter, sofern Variablen oder Feldnamen und alle COMMON-Daten der (rufenden) Programmeinheit

Form der Ausgabe für Variable bzw. Feld:

```
 %CHECK/prog/aaaaa :
name = wert ,

 %CHECK/prog/bbbbb :
name = wert1 , wert2 , wert3 , ...
```

Form der Ausgabe für LABEL-Variable:

```
 %CHECK/prog/aaaaa:
name = 'ZZZZ (LABEL)',
```

prog	Name der Programmeinheit
aaaaa,bbbbb	Anweisungsnummern jener Anweisungen, in denen die Wertänderung stattgefunden hat.
name	Name eines Datenelements, der in der %CHECK-Anweisung angegeben wurde.
ZZZZ	Anweisungsmarke

Die Werte der Datenelemente werden mit dem für den jeweiligen Typ gültigen Standardformat ausgegeben. Die Standardformate sind im Handbuch "FOR1-Beschreibung" [21] dargestellt.

7.4.4 %CALLTRACE-Anweisung

Die %CALLTRACE-Anweisung liefert Informationen über die Aufrufe von Unterprogrammen und Funktionen in einer Programmeinheit.

```
%CALLTRACE  {ON }
            {   } [(unit)]
            {OFF}
```

Für jeden Aufruf innerhalb des Bereichs zwischen %CALLTRACE ON und %CALLTRACE OFF werden die Namen des Eingangspunktes, der Name der zugehörigen Programmeinheit und die übergebenen Parameterwerte ausgegeben.

Werden in einer aufgerufenen Programmeinheit neuerliche Aufrufe durchgeführt, so werden darüber keine Informationen mehr ausgegeben, es sei denn für diese Programmeinheit wurde ebenfalls eine %CALLTRACE-Anweisung angegeben.

Form der Ausgabe:

```
   %CALLTRACE/prog/cccc:      CALLED ENTRY : <eingang          unterprogramm"
 SUBMITTED PARAMETERS :
ARG.   ABSOLUTE   PROGRAM  ARGUMENT   DATA   HEXADECIMAL VALUE                VALUE
NO.    LOCATION   ADDRESS   OFFSET    TYPE
-----  ---------  --------  --------  -----  ---------------------------  ----------
  .        .         .         .        .        .         .
  .        .         .         .        .        .         .
  .        .         .         .        .        .         .
  .        .         .         .        .        .         .
```

prog	Name der aufrufenden Programmeinheit
ccccc	Anweisungsnummer jener Anweisung, in der der Aufruf stattfindet
eingang	Name des Eingangspunktes
unterprogramm	Name der aufgerufenen Programmeinheit

7.4.5 %JUMPTRACE-Anweisung

Die %JUMPTRACE-Anweisung liefert Informationen über Verzweigungen innerhalb einer Programmeinheit.

```
%JUMPTRACE { ON  } [(unit)]
           {     }
           { OFF }
```

Bei jedem Sprung innerhalb einer Programmeinheit werden die Segmentnummer und die Anweisungsnummer der Sprungstelle ausgegeben, sofern diese im Bereich zwischen %JUMPTRACE ON und %JUMPTRACE OFF liegt.

Dies betrifft explizite Sprünge, die durch eine GOTO-Anweisung hervorgerufen werden, ebenso wie implizite, z.B. auf Grund von DO-Anweisungen oder IF-Anweisungen.

Form der Ausgabe:

```
%JUMPTRACE/prog/jjjjj:SEG-NR=SSSSS;STMT-NR=sssss
```

prog	Name der Programmeinheit
jjjjj	Anweisungsnummer der %JUMPTRACE-Anweisung
SSSSS	Segmentnummer der Ansprungstelle
sssss	Anweisungsnummer der Ansprungstelle

7.4.6 %FULLTRACE-Anweisung

Die %FULLTRACE-Anweisung liefert Informationen über die Ausführungsreihenfolge von Anweisungen und die Wertveränderungen aller in den jeweiligen Anweisungen veränderten Variablen

```
%FULLTRACE {ON } [(unit)]
           {   }
           {OFF}
```

Für jede ausgeführte Anweisung und mögliche Wertänderung der Variablen innerhalb des Bereichs zwischen %FULLTRACE ON und %FULLTRACE OFF werden die Segmentnummer, die Anweisungsnummer und der aktuelle Wert des Datenelements am Ende der betreffenden Anweisung angegeben.

Form der Ausgabe:

```
%FULLTRACE/prog/fffff:SEG-NR=SSSSS;STMT-NR=sssss
```

prog	Name der Programmeinheit
fffff	Anweisungsnummer der zugehörigen %FULLTRACE ON-Anweisung
SSSSS	Nummer des Segments
sssss	Anweisungsnummer

Bei Angabe von %FULLTRACE wird stets für alle Wertänderungen ein %CHECK-Protokoll ausgegeben. Die %FULLTRACE-Anweisung sollte nur für kleinere Bereiche verwendet werden, da die Übersetzungszeit, die Größe des erzeugten Objektprogramms, die Laufzeit und die Menge der ausgegebenen Daten merklich angehoben wird. Die mit den beiden folgenden Anweisungen erreichbaren Resultate genügen im allgemeinen, um den Ablauf des Programms verfolgen zu können.

7.4.7 %COUNT-Anweisung

Die %COUNT-Anweisung liefert Informationen über die Häufigkeiten, mit denen die einzelnen Programmteile durchlaufen werden, sowie über den jeweils anteiligen Zeitverbrauch.

```
%COUNT {ON }
       {OFF}
```

Für jedes Segment, dessen Anfang im Bereich zwischen %COUNT ON und %COUNT OFF liegt, wird die Anzahl der Durchläufe durch dieses Segment während des gesamten Objektprogrammablaufs festgestellt.

Außerdem wird die Zeit, die für jede Anweisung in dem festgelegten Bereich benötigt wurde, näherungsweise ermittelt. Für jede Programmeinheit, in der diese Zählungen durchgeführt wurden, werden nach Beendigung des Programmlaufs zwei Statistiken ausgegeben. Die Ausgabe erfolgt nicht auf eine beliebige Dateinummer, wie bei den anderen Testanweisungen, sondern auf SYSLST. Die Ausgabe auf eine andere Einheit läßt sich durch Laufzeitoptionen (siehe Abschnitt 6.3) ändern.

Wurden für mindestens zwei Programmeinheiten eines Programmsystems solche Statistiken ausgegeben, so wird zum Schluß eine Gesamtstatistik über diese Programmeinheiten erstellt.

Form der Ausgabe:

Statistik 1

```
DYNAMIC COUNT PROFILE OF PROGRAM UNIT prog

SEG-NR  STMT-NR  COUNT  PROFILE
SSSSS   sssss      n    ******
  .       .        .      .
  .       .        .      .
  .       .        .      .
```

prog	Name der Programmeinheit
SSSSS	Segmentnummer
sssss	Anweisungsnummer der ersten Anweisung in dem entsprechenden Segment
n	Anzahl der Durchläufe dieses Segments
******	Histogramm, welches das Verhältnis der Durchläufe zwischen den einzelnen Segmenten, für die die Zählung durchgeführt wurde, graphisch darstellt.

Statistik 2

```
DYNAMIC TIME PROFILE OF PROGRAM UNIT prog

SEG-NR  STMT-NR  TIME   PROFILE
SSSSS   sssss     n     ******
  .       .       .        .
  .       .       .        .
  .       .       .        .
```

prog	Name der Programmeinheit
SSSSS	Segmentnummer
sssss	Anweisungsnummer
n	relativer Anteil des Zeitaufwandes aller Ausführungen der betreffenden Anweisungen am Gesamtaufwand aller erfaßten Anweisungen dieser Programmeinheit
******	Histogramm, welches den relativen Anteil graphisch darstellt.

Statistik 3

```
DYNAMIC TIME PROFILE ON ALL COUNTED PROGRAM UNITS
PROGRAM UNIT   TIME    PROFILE
prog             n     ******
 .               .        .
 .               .        .
 .               .        .
```

prog	Name der Programmeinheit
n	relativer Anteil des Zeitaufwandes der betreffenden Programmeinheit am gesamten Zeitaufwand aller Programmeinheiten, für die die entsprechenden %COUNT-Anweisungen gegeben wurden.
******	Histogramm, welches den relativen Anteil graphisch darstellt.

Die durch %COUNT gezählten Durchlaufhäufigkeiten und die Erzeugung der Listen wird erst am Ende des Programmlaufs ausgewertet. Zu diesem Zeitpunkt müssen alle betroffenen Programmeinheiten im Hauptspeicher stehen und dürfen nicht durch andere Programmeinheiten überlagert sein.

Die Angaben über die Zeitanteile der Programmeinheiten sind nur dann voll verwertbar, wenn jeweils alle Anweisungen der betreffenden Programmeinheiten im Zählbereich liegen. Werden von einer Programmeinheit andere Programmeinheiten aufgerufen, so ist der Zeitaufwand für diese aufgerufenen Programmeinheiten in der Zeitangabe für die rufende Programmeinheit nicht berücksichtigt. Wurden in den aufgerufenen Programmeinheiten ebenfalls %COUNT-Anweisungen gegeben, so lassen sich die benötigten Zeiten aus den entsprechenden Zeilen der Statistik 3 entnehmen.

7.4.8 Beispiel: Anwendung von Testanweisungen

Das Programm INV steht in der Datei QUELL.MAT und wird mit den folgenden Anweisungen übersetzt:

```
/START-PROG $FOR1
 %  BLS0500 PROGRAM 'FOR1', VERSION '2.2A00' OF '91-06-05'LOADED.
 %  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1991 ...
  FOR1:    V2.2A00 READY, GIVE COMPILER OPTION
*COMOPT TESTOPT=(DEBUG),SOURCE=QUELL.MAT,END                          (9)
```

SOURCE-LISTING für das Quellprogramm:

```
 ****  SOURCE  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER  FOR1 V2.2A00  DATE = 1991-08-26 TIME = 12:36:48   PAGE  1
                                                PROGRAM UNIT: INV
DO/IF SEG   STMT  I/H LINE        SOURCE-TEXT                                                    COL73-80 RECORD-ID.

      1/1      1         1        PROGRAM INV
        1      2         2        COMMON IR
        1      3         3        %COUNT ON                                                (1)
        1      4         4        %CALLTRACE ON (2)                                        (2)
        1      5         5      1 WRITE (2,10)
        1      6         6        %CHECK ON (2) IR                                         (3)
        1      7         7        READ (1,11) IR
        1      8         8        %CHECK OFF (2)
        1      9         9        IF (IR.GT.3) THEN
  1     2     10        10          GO TO 1
  1     2     11        11        ELSE IF (IR.EQ.0) THEN
  1     3     12        12          GO TO 100
  1     3     13        13        ELSE
  1     4     14        14          CALL DETS ()
  1     4     15        15        END IF
        4     16        16        %CALLTRACE OFF
        4     17        17        %COUNT OFF
        4     18        18     10 FORMAT (' RANG DER MATRIX EINGEBEN')
        4     19        19     11 FORMAT (I1)
        5     20        20    100 END
```

```
 ****  SOURCE  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER  FOR1 V2.2A00  DATE = 1991-08-26 TIME = 12:36:48   PAGE  1
                                                PROGRAM UNIT: DETS
DO/IF SEG   STMT  I/H LINE        SOURCE-TEXT                                                    COL73-80 RECORD-ID.

      1/1      1         1        SUBROUTINE DETS ()
        1      2         2        COMMON IR
        1      3         3        REAL A(:,:)
        1      4         4        DIMENSION B(3,3)
        1      5         5        DIMENSION DET(3,3)
        1      6         6        CALL ALLOC(A,1,IR,1,IR,'ANY')
        1      7         7        %COUNT ON                                                (1)
        1      8         8        WRITE (2,10)
        6      9         9        READ (1,*) ((A(IA,IB),IB=1,IR),IA=1,IR)
        6     10        10        %JUMPTRACE ON (2)                                        (4)
        6     11        11        IF (IR.EQ.2) THEN
  1     7     12        12          SDET=A(1,1)*A(2,2)-A(2,1)*A(1,2)
  1     7     13        13          %DISPLAY (2) SDET                                      (5)
  1     7     14        14          %CHECK ON (2) B                                        (6)
  1     7     15        15          B(1,1)=A(2,2)/SDET
  1     7     16        16          B(2,2)=A(1,1)/SDET
  1     7     17        17          B(1,2)=(-A(1,2))/SDET
  1     7     18        18          B(2,1)=(-A(2,1))/SDET
  1     7     19        19          %CHECK OFF (2)
  1     7     20        20        ELSE
  1     8     21        21          DET(1,1)=A(2,2)*A(3,3)-A(3,2)*A(2,3)
  1     8     22        22          DET(1,2)=A(2,1)*A(3,3)-A(3,1)*A(2,3)
  1     8     23        23          DET(1,3)=A(2,1)*A(3,2)-A(3,1)*A(2,2)
  1     8     24        24          DET(2,1)=A(1,2)*A(3,3)-A(3,2)*A(1,3)
```

```
1     8     25        25        DET(2,2)=A(1,1)*A(3,3)-A(3,1)*A(1,3)
1     8     26        26        DET(2,3)=A(1,1)*A(3,2)-A(3,1)*A(1,2)
1     8     27        27        DET(3,1)=A(1,2)*A(2,3)-A(2,2)*A(1,3)
1     8     28        28        DET(3,2)=A(1,1)*A(2,3)-A(2,1)*A(1,3)
1     8     29        29        DET(3,3)=A(1,1)*A(2,2)-A(2,1)*A(1,2)
1     8     30        30        SDET=-A(1,1)*DET(1,1)+A(1,2)*DET(1,2)-A(1,3)*DET(1,3)
1     8     31        31        %CHECK ON (2) B                                          (7)
1     8     32        32        DO 6 I=1,IR
2    11     33        33        DO 6 J=1,IR
3    12     34        34      6   B(I,J)=(-1)**(I+J+1)*DET(J,I)/SDET
1    12     35        35        %CHECK OFF (2)
1    13     36        36      END IF
     13     37        37      %FULLTRACE ON (2)                                        (8)
     14     38        38      CALL DVCHK (IS)
     14     39        39      IF (IS.NE.1) THEN
1    15     40        40        WRITE (2,11)
1    15     41        41        %FULLTRACE OFF
1    15     42        42        %JUMPTRACE OFF
1    15     43        43        DO 9 K=1,IR
2    19     44        44      9   WRITE (2,*) (B(K,J),J=1,IR)
1    19     45        45     10   FORMAT (' MATRIX ZEILENWEISE EINGEBEN')
1    19     46        46     11   FORMAT (' INVERSE MATRIX:')
1    19     47        47     12   FORMAT (' INVERSE EXISTIERT NICHT')
1    20     48        48        RETURN
1    20     49        49      END IF
     21     50        50      WRITE (2,12)
     21     51        51      %COUNT OFF                                               (1)
     21     52        52      CALL DEALLOC(A)
     21     53        53      RETURN
     21     54        54      END
```

Erläuterung des Beispiels:

(1) Für alle Anweisungen zwischen %COUNT ON und %COUNT OFF werden die Zahl der Ausführungen und die benötigte Rechenzeit auf SYSLST ausgegeben.

(2) Für den Aufruf des Unterprogramms DETS werden die gerufene Programmeinheit, die Anweisungsnummer der Aufrufstelle, Eingangspunkt und Parameter auf der Datenstation ausgegeben.

(3) Die Wertänderung der Variablen IR wird auf der Datenstation protokolliert

(4) Alle Sprünge zu Anweisungen zwischen hier und %JUMPTRACE OFF werden auf der Datenstation protokolliert.

(5) Der Wert der Variablen SDET wird auf der Datenstation ausgegeben.

(6) Die Wertänderung von B wird auf der Datenstation protokolliert.

(7) Die Wertänderung von B wird auf der Datenstation protokolliert.

(8) Bis zur Anweisung %FULLTRACE OFF werden alle ausgeführten Anweisungen auf der Datenstation protokolliert.

(9) Die Option TESTOPT=(DEBUG) bewirkt die Übersetzung der Testanweisungen im Quellprogramm.

Beim Programmablauf werden am Bildschirm z.B. folgende Daten ausgegeben:

```
/START-PROG FROM-FILE=*MODULE(*OMF)
 %  BLS0001 ### DBL VERSION 070 RUNNING
 %  BLS0517 MODULE 'INV' LOADED
 BS2000  F O R 1 : FORTRAN PROGRAM "INV"
 STARTED ON 1991-08-26 AT 15:33:02
 RANG DER MATRIX EINGEBEN
*2
    %CHECK/INV/7 :
   IR = 2 ,
    %CALLTRACE/INV/14 :       CALLED ENTRY : <DETS                DETS"
                              NO PARAMETERS SUBMITTED
 MATRIX ZEILENWEISE EINGEBEN
*1 2
*2 1
    %DISPLAY/DETS/13 :
  SDET = -0.30000000E+01 ,
    %CHECK/DETS/15 :
   B = -0.33333331E+00 , 8*0.00000000E+00 ,
    %CHECK/DETS/16 :
   B = -0.33333331E+00 , 3*0.00000000E+00 , -0.33333331E+00 , 4*0.00000000E+00 ,
    %CHECK/DETS/17 :
    %CHECK/DETS/17 :
   B = -0.33333331E+00 , 2*0.00000000E+00 , 0.66666663E+00 , -0.33333331E+00 , 4*0.00000000E+00 ,
    %CHECK/DETS/18 :
   B = -0.33333331E+00 , 0.66666663E+00 , 0.00000000E+00 , 0.66666663E+00 , -0.33333331E+00 , 4*0.00000000E+00 ,
    %JUMPTRACE/DETS/10 :      SEG-NR = 13    ; STMT-NR = 36
    %FULLTRACE/DETS/37 :      SEG-NR = 14    ; STMT-NR = 38
    %CHECK/DETS/38 :
   IS = 2 ,
    %FULLTRACE/DETS/37 :      SEG-NR = 14    ; STMT-NR = 39
    %FULLTRACE/DETS/37 :      SEG-NR = 15    ; STMT-NR = 40
 INVERSE MATRIX:
 -0.33333331E+00 , 0.66666663E+00
 0.66666663E+00 , -0.33333331E+00
 BS2000  F O R 1 : FORTRAN PROGRAM "INV     " ENDED PROPERLY AT 15:33:17
 CPU - TIME USED :       0.0318 SECONDS
 ELAPSED TIME    :      15.2920 SECONDS
```

Nach SYSLST wird das DYNAMIC COUNT PROFILE der Programmeinheiten INV und DETS ausgegeben:

DYNAMIC COUNT PROFILE OF PROGRAM UNIT DETS

SEG-NR	STMT-NR	COUNT	PROFILE
1	1	1	**********
1	8	1	**********
2	9	2	**********
3	9	2	**********
4	9	4	**********
5	9	2	**********
6	9	1	**********
7	11	1	**********
8	21	0	
9	33	0	
10	34	0	
11	34	0	
12	34	0	
13	36	0	
14	37	1	**********
15	39	1	**********
16	41	2	**********
17	44	2	**********
18	44	4	**********
19	44	2	**********
20	48	1	**********
21	49	0	

DYNAMIC TIME PROFILE OF PROGRAM UNIT DETS

SEG-NR	STMT-NR	TIME	PROFILE
1	1	83	**********
1	8	9	*********
1	9	24	**********
2	9	30	**********
2	9	20	**********
3	9	14	**********
4	9	100	**********
5	9	26	**********
6	9	8	********
6	10	5	*****
7	11	48	**********
7	12	16	**********
7	15	27	**********
7	16	26	**********
7	17	28	**********
7	18	27	**********
7	19	1	*
7	20	15	**********
8	21	0	
8	22	0	
8	23	0	
8	24	0	
8	25	0	
8	26	0	
8	27	0	
8	28	0	
8	29	0	
8	30	0	
8	32	0	
9	33	0	
9	34	0	
10	34	0	
11	34	0	
11	34	0	
12	34	0	
13	36	0	
14	37	31	**********
14	38	19	**********
15	39	30	**********
15	40	11	**********
16	41	44	**********
16	44	16	**********

```
    17       44        12    ************
    18       44        88    ********************************************************************************
    19       44        16    ****************
    19       44        16    ****************
    20       48         8    ********
    21       49         0

                              DYNAMIC COUNT PROFILE OF PROGRAM UNIT INV
                              =========================================

SEG-NR  STMT-NR    COUNT   PROFILE

     1        1         1   ************************************************************************************************************************
     2       10         0
     3       12         0
     4       13         1   ************************************************************************************************************************
                              DYNAMIC TIME  PROFILE OF PROGRAM UNIT INV
                              =========================================

SEG-NR  STMT-NR    TIME    PROFILE

     1        1        35   ************************************************************************************************************************
     1        4         9   *************************
     1        7        21   ***********************************************************
     1        9         5   **************
     2       10         0
     2       11         0
     3       12         0
     4       13        21   ***********************************************************
                              DYNAMIC TIME  PROFILE OF ALL COUNTED PROGRAM UNITS
                              ================================================

PROGRAM UNIT     TIME     PROFILE

        DETS      798    ********************************************************************************************************************
         INV       91    ***********
```

7.5 Testhilfe-Unterprogramme

Testhilfe-Unterprogramme werden im FORTRAN-Quellprogramm wie folgt aufgerufen:

```
CALL name (parameter)
```

Testhilfe-Unterprogramme sind Bestandteil des Laufzeitsystems und erfordern außer dem Aufruf keine FORTRAN-Programmierung.

Testhilfe-Unterprogramme dienen zur

- Kommunikation zwischen Programmeinheiten (Unterprogramme SLITE und SLITET),
- Programmfortsetzung bei Über- oder Unterläufen bzw. Divisionsfehlern (Unterprogramme OVERFLOW, FIXOV, DVCHK),
- Ausgabe von Informationen über den Programmzustand (Unterprogramm DEBUG).

7.5.1 Übersicht: Testhilfe-Unterprogramme

Unterprogramm	**Funktion**
SLITE und SLITET	Ermöglichen das Setzen, Löschen, Abfragen von 4 symbolischen Anzeigen CALL SLITE(i) i: INTEGER-Ausdruck, 0 ≤ i ≤ 4 CALL SLITET(i,j) i: INTEGER-Ausdruck, 1 ≤ i ≤ 4 j: INTEGER-Variable
OVERFL	Erkennt die intern gesetzten Über- und Unterlaufanzeigen nach arithmetischen Operationen mit Gleitkommazahlen CALL OVERFL(j) j: INTEGER-Variable
DVCHK	Erkennt den Überlauf bei einer Division mit Fest- und Gleitkommazahlen (Division durch Null) CALL DVCHK (j) j: INTEGER-Variable
FIXOV	Erkennt den Festpunktüberlauf bei arithmetischen Operationen CALL FIXOV(j) j: INTEGER-Variable
DEBUG	Liefert Informationen über den Programmzustand (aktuelle Anweisungsnummer, Aufrufhierarchie, Parameterlisten), beendet den Programmablauf CALL DEBUG [(v)] v: CHARACTER-Datenelement

Hinweis:

Zum Ausdrucken eines PDUMP siehe auch Dienstprogramm "DUMP" im Handbuch "Dienstprogramme" [17].

7.5.2 SLITE- und SLITET-Unterprogramm

Mit dem Unterprogramm SLITE können Schalter gesetzt und gelöscht werden.
Mit dem Unterprogramm SLITET können Schalter abgefragt und gelöscht werden.

SLITE und SLITET ermöglichen eine Kommunikation zwischen Programmeinheiten innerhalb eines Objektlaufs. Für diese Kommunikation ist ein Byte in der RUN TIME COMMUNICATION AREA vorgesehen, so daß die übliche Kommunikation über COMMON-Bereiche oder durch Übergabe von Parametern entfällt.

Aufruf der Unterprogramme:

```
CALL SLITE(i)
```

i INTEGER-Ausdruck; Wert: $0 \leq i \leq 4$

Wenn i=0, dann werden alle Schalter gelöscht. Ist i=1,2,3 oder 4, dann wird der entsprechende Schalter gesetzt.

```
CALL SLITET(i,j)
```

i INTEGER-Ausdruck; Wert: $1 \leq i \leq 4$
j INTEGER-Variable

Der Schalter i (gleich 1,2,3 oder 4) wird getestet und ggf. gelöscht.
Die Variable j wird gleich 1 gesetzt, wenn i gesetzt war bzw. gleich 2, wenn i nicht gesetzt war.

7.5.3 OVERFL-Unterprogramm

Das Unterprogramm OVERFL
- erkennt die intern gesetzte Überlauf- und Unterlaufanzeige nach arithmetischen Operationen mit Gleitkommazahlen,
- verhindert einen Programmabbruch bei Auftreten eines Exponenentenüberlaufs.

Ohne Verwendung des Unterprogramms OVERFL wird bei einem Exponentenüberlauf bzw. -unterlauf der Programmablauf abgebrochen. Es werden Informationen über den Fehler und die Fehlerstelle ausgegeben.

Wurde jedoch das Unterprogramm OVERFL in den Codemodul mit eingebunden (auf Grund eines Aufrufs im FORTRAN-Quellprogramm), so erfolgt kein Programmabbruch. Dem Resultat der betreffenden Operation wird unter Berücksichtigung des Vorzeichens der größtmögliche bzw. kleinstmögliche Wert zugewiesen.

Aufruf:

```
CALL OVERFL (j)
```

j INTEGER-Variable

Die Überlaufanzeige wird intern gesetzt, wenn ein Exponentenüberlauf auftritt, d.h. wenn das Ergebnis einer arithmetischen Operation mit Gleitkommazahlen größer als 16**(+63) ist (Mantisse≠0).
Die Unterlaufanzeige wird gesetzt, wenn ein Exponentenunterlauf auftritt, d.h. wenn das Ergebnis einer arithmetischen Operation mit Gleitkommazahlen kleiner als 16**(-64) ist (Mantisse≠0).

Der Parameter j wird bei Aufruf des Unterprogramms abhängig von den intern gesetzten Anzeigen folgendermaßen versorgt

j	Überlaufanzeige	Unterlaufanzeige	Löschen der Anzeige
0	gesetzt	nicht gesetzt	Löschen der Überlaufanzeige
1	gesetzt	gesetzt	Löschen der Überlaufanzeige
2	nicht gesetzt	nicht gesetzt	—
3	nicht gesetzt	gesetzt	Löschen der Unterlaufanzeige

Nach dem Setzen der Variablen j löscht das Unterprogramm OVERFL die internen Anzeigen. Wenn sowohl die Überlauf- als auch die Unterlaufanzeige gesetzt sind, dann löscht OVERFL lediglich die interne Überlaufanzeige, so daß ein nachfolgender Aufruf von OVERFL die Unterlaufanzeige prüfen kann.

Bei den Datentypen REAL und COMPLEX mit allen Längen sowie beim Datentyp INTEGER*8 berücksichtigt das Unterprogramm OVERFL Über- bzw. Unterläufe nach allen arithmetischen Operationen.

Bei den Datentypen INTEGER*1 und INTEGER*2 erkennt das Unterprogramm OVERFL bei keiner arithmetischen Operation einen Über- bzw. Unterlauf.

Beim Datentyp INTEGER*4 werden bei einem Überlauf nach einer Multiplikation die führenden Stellen abgeschnitten. Die Integer-Variable j hat nach dem Überlauf den Wert 2, da weder die interne Überlauf- noch die Unterlaufanzeige gesetzt ist. Bei einem Überlauf nach einer Addition oder Subtraktion erfolgt beim Datentyp INTEGER*4 ein Programmabbruch mit einer Fehlermeldung (FIXED POINT OVERFLOW).

Durch Abfrage der Variablen j kann ein durch eine arithmetische Operation verursachter Über- oder Unterlauf überwacht werden. Wird der Wert der Variablen j im FORTRAN-Programm abgefragt, dann sollte der Aufruf CALL OVERFL (j) jeweils unmittelbar hinter der FORTRAN-Anweisung stehen, bei der ein Über- bzw. Unterlauf überwacht werden soll.

Beispiel:

```
            .
            .
            .
      A = 0.72370E+76
      B = 1.E-50
      C = 1.E+30
      J = 5
C
      D = A+0.1E+76
      CALL OVERFL (J)
      IF (J.EQ.0) GO TO 200
C
      E = B/C
      CALL OVERFL (J)
      IF(J.EQ.3) GO TO 300
            .
            .
            .
200   WRITE (20,'(''OVERFLOW : D = '',E20.5,'' J='',I4)')D,J

300   WRITE (20,'(''UNDERFLOW : E = '',E20.5,'' J='',I4)')E,J
```

Ausgabe bei Überlauf: `OVERFLOW: D = 0.72370E+76 J=0`

Ausgabe bei Unterlauf: `UNDERFLOW: E = 0.00000E+00 J=3`

7.5.4 DVCHK-Unterprogramm

Das Unterprogramm DVCHK erkennt den Überlauf bei einer Division mit Festkomma- und Gleitkommazahlen. Ein Überlauf bei einer Division tritt auf, wenn der zweite Operand (Divisor) den Wert 0 hat.

Ohne Verwendung des Unterprogramms DVCHK wird bei einem Divisionsüberlauf der Programmlauf abgebrochen, und Meldungen über den Fehler und die Fehlerstelle werden Divisionsüberlauf der Programmlauf abgebrochen, und Meldungen über den Fehler und die Fehlerstelle werden ausgegeben.

Wurde jedoch das Unterprogramm DVCHK in den Lademodul mit eingebunden (auf Grund der Verwendung im FORTRAN-Quellprogramm), so erfolgt kein Programmabbruch. Das Resultat der betreffenden Operation wird auf den größtmöglichen Wert gesetzt, außer wenn der erste Operand gleich 0 ist, in diesem Fall ist das Ergebnis ebenfalls gleich 0.

Aufruf:

```
CALL DVCHK (j)
```

j INTEGER-Variable der Länge 4

Der Parameter j wird bei Aufruf des Unterprogramms DVCHK in Abhängigkeit der Anzeige für Divisionsüberlauf wie folgt gesetzt:

j	Anzeige für Divisionsüberlauf
1	gesetzt
2	nicht gesetzt

Nach dem Setzen der Variablen j wird die Anzeige für Divisionsüberlauf gelöscht.

Durch Abfrage der Variablen j kann ein durch eine Division verursachter Überlauf überwacht werden. Wird der Wert von j im FORTRAN-Programm abgefragt, dann sollte der Aufruf CALL DVCHK(j) jeweils unmittelbar hinter der Division stehen, bei der ein Überlauf überwacht werden soll.

7.5.5 FIXOV-Unterprogramm

Das Unterprogramm FIXOV erkennt den Festpunktüberlauf bei arithmetischen Operationen mit Ausnahme der Multiplikation (s.u., Hinweis). Zu einem Festpunktüberlauf kommt es, wenn bei arithmetischen Festpunktbefehlen ein Übertrag aus der höchstwertigen Bitstelle auftritt oder wenn bei arithmetischen Linksschiebebefehlen gültige Bits verlorengehen.
Ohne Verwendung des Unterprogramms FIXOV wird bei einem Festpunktüberlauf der Programmlauf abgebrochen. Meldungen über Fehler und die Fehlerstelle werden ausgegeben.
Wurde jedoch das Unterprogramm FIXOV in den Lademodul miteingebunden (z.B. durch Verwendung im FORTRAN-Quellprogramm), so erfolgt kein Programmabbruch. Das Resultat der betreffenden Operation wird auf den größtmöglichen Wert gesetzt. Es wird mit der Operation fortgefahren, die auf die Operation folgt, die den Festpunktüberlauf verursachte.

Aufruf:

```
CALL FIXOV (j)
```

j INTEGER-Variable der Länge 4

Der Parameter j wird bei Aufruf des Unterprogramms FIXOV in Abhängigkeit von der Anzeige für Festpunktüberlauf wie folgt gesetzt:

j	Anzeige für Festpunktüberlauf
1	gesetzt
2	nicht gesetzt

Nach dem Setzen der Variablen j wird die interne Anzeige für Festpunktüberlauf gelöscht.

Durch Abfrage der Variablen j kann ein durch eine arithmetische Operation verursachter Festpunktüberlauf überwacht werden. Wird der Wert der Variablen j im FORTRAN-Programm abgefragt, dann sollte der Aufruf CALL FIXOV (j) jeweils hinter der FORTRAN-Anweisung stehen, bei der ein Festpunktüberlauf überwacht werden soll.

Hinweis:

Das Unterprogramm FIXOV kann einen Festpunktüberlauf nur dann erkennen, wenn vom Prozessor das Unterbrechungsgewicht X’78’ gesetzt wird. Dies geschieht, wenn bei den Festpunktbefehlen A, AR, AH, LCR, LPR, S SR, SH, SLA oder SLDA ein Festpunktüberlauf auftritt. Ein Festpunktüberlauf infolge einer Multiplikation (Befehle M, MR, MH) kann daher nicht erkannt werden.

7.5.6 DEBUG-Unterprogramm

Das Unterprogramm DEBUG liefert Informationen über den Programmzustand.

Aufruf:

```
CALL DEBUG[(v)]
```

v CHARACTER-Datenelement

Bei Ausführung dieser Anweisung wird der Programmablauf beendet. Auf SYSLST werden die ersten 133 Zeichen von v sowie Informationen über den Programmzustand ausgegeben. Dazu gehören insbesondere die aktuelle Anweisungsnummer, die Aufrufhierarchie und die Parameterlisten aller gerade aktiven, d.h. aufgerufenen und noch nicht beendeten Programmeinheiten.

Diese Informationen werden auch bei einem Laufzeitfehler über die Standardfehlerroutine ausgegeben.

7.5.7 Beispiel: Anwendung der Testhilfe-Unterprogramme

```
    PROGRAM INV
    COMMON IR
  1 WRITE (2,10)
    READ (1,11) IR
    IF (IR.GT.3) THEN
      GO TO 1
    ELSE IF (IR.EQ.0) THEN
      GO TO 100
    ELSE
      CALL DETS ()
    END IF
    CALL SLITET (1,IV)                                          (1)
    IF (IV.NE.1) GO TO 100
 10 FORMAT (' RANG DER MATRIX EINGEBEN')
 11 FORMAT (I1)
    CALL DEBUG                                                  (2)
100 END
    SUBROUTINE DETS ()
    COMMON IR
    REAL A(3,3)
    DIMENSION B(3,3)
    DIMENSION DET(3,3)
    WRITE (2,10)
    READ (1,*) ((A(IA,IB),IB=1,IR),IA=1,IR)
    IF (IR.EQ.2) THEN
      SDET=A(1,1)*A(2,2)-A(2,1)*A(1,2)
      B(1,1)=A(2,2)/SDET
      B(2,2)=A(1,1)/SDET
      B(1,2)=(-A(1,2))/SDET
      B(2,1)=(-A(2,1))/SDET
```

```
      ELSE
        DET(1,1)=A(2,2)*A(3,3)-A(3,2)*A(2,3)
        DET(1,2)=A(2,1)*A(3,3)-A(3,1)*A(2,3)
        DET(1,3)=A(2,1)*A(3,2)-A(3,1)*A(2,2)
        DET(2,1)=A(1,2)*A(3,3)-A(3,2)*A(1,3)
        DET(2,2)=A(1,1)*A(3,3)-A(3,1)*A(1,3)
        DET(2,3)=A(1,1)*A(3,2)-A(3,1)*A(1,2)
        DET(3,1)=A(1,2)*A(2,3)-A(2,2)*A(1,3)
        DET(3,2)=A(1,1)*A(2,3)-A(2,1)*A(1,3)
        DET(3,3)=A(1,1)*A(2,2)-A(2,1)*A(1,2)
        SDET=-A(1,1)*DET(1,1)+A(1,2)*DET(1,2)-A(1,3)*DET(1,3)
        DO 6 I=1,IR
        DO 6 J=1,IR
 6      B(I,J)=(-1)**(I+J+1)*DET(J,I)/SDET
      END IF
      CALL DVCHK (IS)                                          (3)
      IF (IS.NE.1) THEN
        CALL OVERFL (IT)                                       (4)
        IF (IT.NE.2) GO TO 30
        WRITE (2,11)
        DO 9 K=1,IR
 9      WRITE (2,*) (B(K,J),J=1,IR)
10      FORMAT (' MATRIX ZEILENWEISE EINGEBEN')
11      FORMAT (' INVERSE MATRIX:')
12      FORMAT (' INVERSE EXISTIERT NICHT')
13      FORMAT (' OVERFLOW')
        RETURN
      END IF
20    WRITE (2,12)
      RETURN
30    WRITE (2,13)
      CALL SLITE (1)                                           (5)
      RETURN
      END
```

Erläuterung des Beispiels:

(1) Der Schalter 1 wird geprüft. Wenn er nicht gesetzt ist, verzweigt das Programm zur Anweisung mit der Sprungmarke 100, andernfalls wird es mit der nächsten Anweisung fortgesetzt.

(2) Das Unterprogramm DEBUG gibt Informationen über den Programmzustand aus und beendet das Programm.

(3) Wenn im Programmablauf eine Division durch Null aufgetreten ist, erhält IS den Wert 1 und es wird die Anweisung mit der Sprungmarke 20 ausgeführt.

(4) Wenn die Über- oder Unterlaufanzeige gesetzt ist, springt das Programm zur Anweisung mit Sprungmarke 30.

(5) Der Schalter 1 wird gesetzt.

7.6 Dialogtesthilfe AID (Advanced Interactive Debugger)

FOR1-Programme sind mit der Dialogtesthilfe AID (ab AID-Version 1.0C) testbar (siehe Handbuch "AID Testen von FORTRAN-Programmen" [3]). Mit AID können auch FOR1-Programme, die im 31-Bit-Raum ablaufen, getestet werden.

7.6.1 Voraussetzung für das Testen mit AID: SYMTEST-Option

Eine Voraussetzung für das symbolische Testen mit AID ist das Erzeugen von LSD-Informationen (LSD=List for Symbolic Debugging) durch den Compiler. Das Erzeugen dieser Information steuert der FOR1-Anwender durch die SYMTEST-Option:

`[*]COMOPT`	`SYMTEST = {NO / MAP / ALL}`

NO Es werden keine LSD-Informationen erzeugt. FOR1-Programme können mit AID nicht symbolisch, sondern nur maschinennah getestet werden.

MAP Mit SYMTEST=MAP ist symbolisches Testen nur eingeschränkt möglich: Programmnamen können angesprochen und Aufrufhierarchien können rückverfolgt werden.

ALL Durch SYMTEST=ALL werden vom Compiler LSD-Informationen erzeugt. FOR1-Programme können symbolisch mit AID getestet werden.

Es wird empfohlen, mit COMOPT OPTIMIZE=NO zu arbeiten (siehe Kapitel 9). Die Angabe SYMTEST=ALL läßt sich zwar mit allen Optimierungsstufen kombinieren, allerdings kann man die Quellprogrammliste eines optimierten Programms nicht mehr als eindeutige Grundlage für das Testen mit AID heranziehen. Die Optimierung kann z.B. die Reihenfolge der Anweisungen verändern, eine Anweisung kann in mehrere Anweisungen aufgeteilt werden oder kann ganz entfallen. Soll ein Programm trotz eingeschalteter Optimierung getestet werden, dann kann eine Decompilerliste (siehe 4.7.9) eine Hilfe sein, die bei OPT=3 oder 4 angefordert werden kann. Die Decompilerliste liefert eine "High level"-Beschreibung des Objektcodes, die die Ablaufverfolgung und das Setzen von Testpunkten mit AID erleichtern soll.

Für das Testen hat der Anwender folgende Möglichkeiten, die LSD-Informationen zu laden. Er kann sie

- zusammen mit dem Programm laden (TEST-OPTIONS-Operand beim Aufruf des DBL bzw. des TSOSLNK und des ELDE)
- erst bei Bedarf nachladen, sofern die zugehörigen Bindemoduln in einer PLAM-Bibliothek stehen (AID-Kommando %SYMLIB).

7.6.2 Leistungsumfang von AID

AID ist ein leistungsstarkes Testsystem zur Fehler-Diagnose, zum Test und für die vorläufige Korrektur von Programm-Fehlern im BS2000.

AID unterstützt nicht nur das symbolische Testen von FORTRAN- (FOR1-) Programmen, sondern auch das symbolische Testen von PL/1- (PLI1-) Programmen, COBOL-Programmen, Assembler-Programmen und C-Programmen, sowie das Testen auf Maschinencode-Ebene aller Programmiersprachen des BS2000.

Beim symbolischen Testen eines FORTRAN-Programms können die symbolischen Namen aus einem FORTRAN-Quellprogramm zur Adressierung verwendet werden. Das Testen auf Maschinencode-Ebene bietet sich dort an, wo das nicht möglich ist.

In AID-Kommandos sind symbolisch ansprechbar:
- ausführbare FORTRAN-Anweisungen, bezeichnet durch die Anweisungsnummer,
- ausführbare FORTRAN-Anweisungen, bezeichnet durch die Anweisungsmarke,
- symbolische Konstanten, Variablen, Felder und Feldelemente
- dynamische Felder,
- Prozeduranfänge.

Symbolische Konstanten, Variablen, Felder und Feldelemente werden mit ihrem im Quellprogramm definierten Namen angesprochen. FORTRAN-Anweisungen werden in der Form S'nnnnn', Anweisungsmarken in der Form L'nnnnn' angesprochen (nnnnn: maximal 5stellige Anweisungsnummer bzw. Anweisungsmarke).

Es stehen folgende AID-Kommandos zur Verfügung:

- Kommandos zur Ablaufüberwachung
  - bestimmter Typen von Anweisungen im Quellprogramm (%CONTROLn)
  - ausgewählter Ereignisse im Programm-Ablauf (%ON)
  - vereinbarter Programm-Adressen (%INSERT)

  Der Anwender kann festlegen, daß AID den Programmablauf an definierten Adressen oder bei Ausführung ausgewählter Anweisungstypen oder beim Eintreten definierter Ereignisse unterbricht und dann Subkommandos ausführt. Ein Subkommando ist ein einzelnes Kommando oder eine Folge von AID- und BS2000-Kommandos. Es wird als Operand eines AID-Kommandos definiert.

- Kommandos zur Ablaufverfolgung und -Protokollierung (%TRACE) sowie zum Überspringen von Anweisungen (%JUMP)

  Mit %TRACE kann der Programmbereich der Ablaufverfolgung ausgewählt werden sowie Anzahl und Typ der Anweisungen, die protokolliert werden.

  Mit %JUMP kann eine Anweisung innerhalb einer Programmeinheit festgelegt werden, an der das Programm nach Beendigung der Kommandoeingabe fortgesetzt werden soll. Mit %JUMP kann die Ablaufreihenfolge geändert werden, indem man z.B. fehlerhafte Anweisungen durch eine AID-Kommandofolge ersetzt und anschließend mit %JUMP die Fortsetzung des Programms bei einer bestimmten Anweisung festlegt. Durch %RESUME oder %TRACE wird der Programmablauf bei der angegebenen Anweisung fortgesetzt. Das %JUMP-Kommando wird nur bei OPTIMIZE=NO unterstützt.

- Kommandos zur Ausgabe und zur Änderung von Speicherinhalten

  – zur Ausgabe der Werte von symbolischen Konstanten, Variablen, Feldelementen und Feldern (%DISPLAY),
  – zum Ändern der Werte von Variablen, Feldelementen und Feldern (%SET),
  – zur Ausgabe von Aufrufhierarchien (%SDUMP %NEST),
  – zum Rückübersetzen von Speicherinhalten in Assembler (%DISASSEMBLE).

  Mit dem %NEST-Operanden des %SDUMP-Kommandos kann man sich anzeigen lassen, auf welcher Stufe der Aufrufhierarchie das Programm unterbrochen wurde und welche Moduln in der CALL-Verschachtelung liegen. In der Aufrufhierarchie werden die Namen von FORTRAN-Unterprogrammen und vom Hauptprogramm ausgegeben.

- Kommandos zur Verwaltung von AID-Eingabedateien (%DUMPFILE, %SYMLIB) und AID-Ausgabe-Medien (%OUTFILE).

  Mit AID kann ein laufendes Programm bearbeitet oder ein Speicherauszug in einer Plattendatei diagnostiziert werden. Innerhalb einer Testsitzung kann zwischen beiden Möglichkeiten gewechselt werden, z.B. um Datenbestände im laufenden Programm mit einem Speicherauszug zu vergleichen.

- Kommandos zum Festlegen von Ausgabe-Datenmengen (%OUT) und globalen Vereinbarungen (%BASE, %AID).

- Die Anwendung von AID wird unterstützt durch die %HELP-Funktion

  – für alle AID-Kommandos und Operanden
  – für die Bedeutung der AID-Meldungen und die möglichen Reaktionen auf AID-Meldungen.

7.6.3 Beispiel: Anwendung der Dialogtesthilfe AID

Source-Listing des Programms INV:

```
  ****  SOURCE  LISTING  ****    SIEMENS-NIXDORF FORTRAN COMPILER    FOR1 V2.2A00 DATE = ...
                                                 PROGRAM UNIT: INV
 DO/IF SEG   STMT  I/H LINE      SOURCE-TEXT
-------------------------------------------------------------------------------
       1/1      1         1     PROGRAM INV
         1      2         2     COMMON IR
         1      3         3    1 WRITE (2,10)
         1      4         4     READ (1,11) IR
         1      5         5     IF (IR.GT.3) THEN
   1     2      6         6       GO TO 1
   1     2      7         7     ELSE IF (IR.EQ.0) THEN
   1     3      8         8       GO TO 100
   1     3      9         9     ELSE
   1     4     10        10       CALL DETS ()
   1     4     11        11     END IF
         4     12        12   10 FORMAT (' RANG DER MATRIX EINGEBEN')
         4     13        13   11 FORMAT (I1)
         5     14        14  100 END

  ****  SOURCE  LISTING  ****    SIEMENS-NIXDORF FORTRAN COMPILER    FOR1 V2.2A00 DATE = ...
                                                 PROGRAM UNIT: DETS
 DO/IF SEG   STMT  I/H LINE      SOURCE-TEXT
-------------------------------------------------------------------------------
       1/1      1         1     SUBROUTINE DETS ()
         1      2         2     COMMON IR
         1      3         3     REAL A (3,3)
         1      4         4     DIMENSION B(3,3)
         1      5         5     DIMENSION DET(3,3)
         1      6         6     WRITE (2,10)
         5      7         7     READ (1,*) ((A(IA,IB),IB=1,IR),IA=1,IR)
         5      8         8     IF (IR.EQ.2) THEN
   1     6      9         9       SDET=A(1,1)*A(2,2)-A(2,1)*A(1,2)
   1     6     10        10       B(1,1)=A(2,2)/SDET
   1     6     11        11       B(2,2)=A(1,1)/SDET
   1     6     12        12       B(1,2)=(-A(1,2))/SDET
   1     6     13        13       B(2,1)=(-A(2,1))/SDET
   1     6     14        14     ELSE
   1     7     15        15       DET(1,1)=A(2,2)*A(3,3)-A(3,2)*A(2,3)
   1     7     16        16       DET(1,2)=A(2,1)*A(3,3)-A(3,1)*A(2,3)
   1     7     17        17       DET(1,3)=A(2,1)*A(3,2)-A(3,1)*A(2,2)
   1     7     18        18       DET(2,1)=A(1,2)*A(3,3)-A(3,2)*A(1,3)
   1     7     19        19       DET(2,2)=A(1,1)*A(3,3)-A(3,1)*A(1,3)
   1     7     20        20       DET(2,3)=A(1,1)*A(3,2)-A(3,1)*A(1,2)
   1     7     21        21       DET(3,1)=A(1,2)*A(2,3)-A(2,2)*A(1,3)
   1     7     22        22       DET(3,2)=A(1,1)*A(2,3)-A(2,1)*A(1,3)
   1     7     23        23       DET(3,3)=A(1,1)*A(2,2)-A(2,1)*A(1,2)
   1     7     24        24       SDET=-A(1,1)*DET(1,1)+A(1,2)*DET(1,2)-A(1,3)*DET(1,3)
   1     8     25        25       DO 6 I=1,IR
   2     8     26        26       DO 6 J=1,IR
   3    10     27        27    6    B(I,J)=(-1)**(I+J+1)*DET(J,I)/SDET
   1    10     28        28     END IF
   1    10               29
        11     29        30     CALL DVCHK (IS)
        11     30        31     IF (IS.NE.2) THEN
   1    12     31        32       WRITE (2,11)
   1    13     32        33       DO 9 K=1,IR
   2    16     33        34    9    WRITE (2,*) (B(K,J),J=1,IR)
   1    16     34        35   10    FORMAT (' MATRIX ZEILENWEISE EINGEBEN')
   1    16     35        36   11    FORMAT (' INVERSE MATRIX:')
   1    16     36        37   12    FORMAT (' INVERSE EXISTIERT NICHT')
   1    17     37        38       RETURN
   1    17     38        39     END IF
*       18     39        40   20 WRITE (2,12)
***** WARNING (SA047) *******                                                  (1)
        18     40        41     RETURN
        18     41        42     END
```

(1) Bei der Übersetzung des Programms INV werden außer einer WARNUNG (UNREFERENCED LABEL) keine Übersetzungsfehler gemeldet.

```
/START-PROG FROM-FILE=*MODULE(LIBRARY=PLAM.LIB,ELEMENT=INV)                    (2)
%  BLS0001 ### DBL VERSION 070 RUNNING
%  BLS0517 MODULE 'INV' LOADED
 BS2000  F O R 1 : FORTRAN PROGRAM "INV"
 STARTED ON 1991-08-25 AT 10:59:06
 RANG DER MATRIX EINGEBEN
*2
 MATRIX ZEILENWEISE EINGEBEN
*1 2
*2 1
 INVERSE EXISTIERT NICHT
 BS2000  F O R 1 : FORTRAN PROGRAM "INV     " ENDED PROPERLY AT 10:59:22
 CPU - TIME USED :       0.0480 SECONDS
 ELAPSED TIME    :      17.2550 SECONDS
```

(2) Das Programm läuft ab, ohne daß ein Ablauffehler gemeldet wird. Das Ergebnis ("INVERSE EXISTIERT NICHT") ist allerdings nicht richtig und läßt auf einen logischen Fehler schließen.

```
/START-PROG $FOR1                                                              (3)
*COMOPT SOURCE=SRC.INV,MODULE-LIBRARY=PLAM.LIB
*COMOPT SYMTEST=ALL
*END
```

(3) INV wird mit den Optionen "SYMTEST=ALL" und "MODULE-LIBRARY=PLAM.LIB" neu übersetzt. Dadurch wird der Bindemodul zusammen mit den LSD-Informationen in eine PLAM-Bibliothek geschrieben.

```
/LOAD-PROG FROM-FILE=*MODULE(LIBRARY=PLAM.LIB,ELEMENT=INV)                      (4)
%  BLS0001 ### DBL VERSION 070 RUNNING
%  BLS0517 MODULE 'INV' LOADED
/%INSERT PROG=DETS.S'13' <%SDUMP;%STOP>                                        (5)
 I378 SYMBOLIC INFORMATION MISSING                                             (6)
/%SYMLIB PLAM.LIB                                                              (7)
```

(4) Das Programm wird mit dem dynamischen Bindelader geladen.

(5) Nach dem Laden wird mit dem AID-Kommando %INSERT ein Testpunkt auf der Anweisung mit der Nummer 13 der Programmeinheit DETS definiert. Vor Ausführung dieser Anweisung soll die Subkommando-Folge "<%SDUMP;%STOP>" ausgeführt werden. "%SDUMP" bewirkt, daß die Werte aller Datenelemente in der aktuellen Aufrufhierarchie entsprechend ihrem vereinbarten Datentyp ausgegeben werden. "%STOP" bewirkt, daß das Programm nach Ausführung von "%SDUMP" anhält, so daß z.B. weitere AID-Kommandos eingegeben werden können.

(6) AID weist das Kommando (5) mit einer Fehlermeldung ab. Beim Aufruf des DBL wurde der Operand TEST-OPTIONS=AID nicht angegeben, so daß keine LSD-Informationen geladen wurden.

(7) Durch das AID-Kommando %SYMLIB wird die PLAM-Bibliothek angegeben, in der die LSD-Informationen stehen. AID lädt diese Informationen nach, sobald ein symbolischer Name angesprochen wird, für den keine LSD-Informationen geladen wurden.

```
/%INSERT PROG=DETS.S'13' <%SDUMP;%STOP>                                          (8)
/%TRACE                                                                          (9)
 BS2000  F O R 1 : FORTRAN PROGRAM "INV"
 STARTED ON 1991-08-24 AT 12:03:29
      3  1                    START    , I-O-ACCESS
 RANG DER MATRIX EINGEBEN
      4                       I-O-ACCESS
*2
      5                       IF
      7                       THEN/ELSE
     10                       THEN/ELSE, CALL
 MATRIX ZEILENWEISE EINGEBEN
*1 2
*2 1
 ** ITN: #'00000041' *** TSN: 3113 ********************************************* (10)
 SRC␣REF:    13  SOURCE: DETS      PROC: DETS  ********************************
 IR              =           2

 A( 1: 3, 1: 3)
 ( 1, 1) +.1000000 E+01  ( 2, 1) +.2000000 E+01  ( 3, 1) +.0000000 E+00
 ( 1, 2) +.2000000 E+01  ( 2, 2) +.1000000 E+01  ( 3, 2) +.0000000 E+00
 ( 1, 3) +.0000000 E+00  ( 2, 3) +.0000000 E+00  ( 3, 3) +.0000000 E+00


 B( 1: 3, 1: 3)
 ( 1, 1) -.3333333 E+00  ( 2, 1) +.0000000 E+00  ( 3, 1) +.0000000 E+00
 ( 1, 2) +.6666666 E+00  ( 2, 2) -.3333333 E+00  ( 3, 2) +.0000000 E+00
 ( 1, 3) +.0000000 E+00  ( 2, 3) +.0000000 E+00  ( 3, 3) +.0000000 E+00


 DET( 1: 3, 1: 3)
 ( 1, 1) +.0000000 E+00  ( 2, 1) +.0000000 E+00  ( 3, 1) +.0000000 E+00
 ( 1, 2) +.0000000 E+00  ( 2, 2) +.0000000 E+00  ( 3, 2) +.0000000 E+00
 ( 1, 3) +.0000000 E+00  ( 2, 3) +.0000000 E+00  ( 3, 3) +.0000000 E+00


 IA              =           3

 IB              =           3

 SDET            = -.3000000 E+01

 I               =           0

 J               =           0

 IS              =           0

 K               =           0

 STOPPED AT SRC␣REF: 13 , SOURCE: DETS , PROC: DETS                              (11)
```

(8) Das %INSERT-Kommando wird nun akzeptiert.

(9) %TRACE startet das Programm und führt zu einem Ablauf mit Ausgabe aller ausführbaren FORTRAN-Anweisungen, die das Programm durchläuft. Ausgegeben werden die Anweisungsnummern, evtl. vorhandene Anweisungsmarken und der Typ der Anweisung. Standardmäßig werden 10 ausführbare Anweisungen durchlaufen und protokolliert.

(10) Vor Ausführung der Anweisung mit der Nummer 13 wird das Subkommando %SDUMP des AID-Kommandos (8) ausgeführt: die Werte aller Datenelemente an dieser Stelle des Programms werden entsprechend ihrem Datentyp ausgegeben. B(2,1) hat noch den Wert 0, da die Anweisung 13 noch nicht ausgeführt wurde.

(11) Durch das Subkommando %STOP in (8) wurde das Programm in den STOP-Zustand versetzt. Nach der Ausgabe der STOP-Meldung können neue Kommandos eingegeben werden.

```
/%INSERT S'28' <%DISPLAY B(2,1);%STOP>                                          (12)
/%TRACE                                                                         (13)
 SRC␣REF:    29 SOURCE: DETS      PROC: DETS  **********************************
 B( 2, 1)        = +.6666666 E+00                                               (14)

 STOPPED AT SRC␣REF: 29 , SOURCE: DETS , PROC: DETS
```

(12) Vor Ausführung der Anweisung 28 soll der Wert des Feldelements B(2,1) ausgegeben werden. Die Qualifikation "PROG=DETS." im %INSERT-Kommando (8) ist nun nicht mehr notwendig, da DETS die aktuelle Programmeinheit ist.

(13) Durch %TRACE läuft das Programm mit Protokollierung des Ablaufs weiter.

(14) Der Wert des Feldelements B(2,1) wurde ebenfalls richtig berechnet.

```
/%TRACE                                                                         (15)
     29                      CALL
     30                      IF
     39  20                  I-O-ACCESS, LABEL                                  (16)
 INVERSE EXISTIERT NICHT
     40                      END
 BS2000  F O R 1 : FORTRAN PROGRAM "INV     " ENDED PROPERLY AT 16:56:51
 CPU - TIME USED :       0.7277 SECONDS
 ELAPSED TIME    :     141.7410 SECONDS
```

(15) %TRACE setzt den Programmablauf an der Unterbrechungsstelle fort.

(16) Nach der IF-Anweisung 30 wird fälschlicherweise zur Anweisung 39 gesprungen und der Text "INVERSE EXISTIERT NICHT" ausgegeben. Die IF-Anweisung muß richtig "IF (IS.NE.1)" heißen, da das Testhilfe-Unterprogramm bei Divisionsüberlauf den Wert 1 und sonst den Wert 2 hat.

```
/LOAD-PROG FROM-FILE=*MODULE(LIBRARY=PLAM.LIB,ELEMENT=INV)                    (17)
%  BLS0001 ### DBL VERSION 070 RUNNING
%  BLS0517 MODULE 'INV' LOADED
/%INSERT PROG=DETS.S'30' <%SDUMP %NEST;%DISPLAY IS;%STOP>                    (18)
/%RESUME                                                                     (19)
 BS2000  F O R 1 : FORTRAN PROGRAM "INV"
 STARTED ON 1991-08-24 AT 16:00:22
 RANG DER MATRIX EINGEBEN
*2
 MATRIX ZEILENWEISE EINGEBEN
*1 2
*2 1
 ** ITN: #'000000C7' *** TSN: 3704 ******************************************
 SRC␣REF:    30 SOURCE: DETS      PROC: DETS  ********************************
 SRC␣REF:    30 SOURCE: INV       PROC: INV   ********************************
 IS             =           2                                                 (20)

 STOPPED AT SRC␣REF: 30 , SOURCE: DETS , PROC: DETS
/%AID CHECK=ALL                                                              (21)
/%SET 1 INTO IS;%RESUME                                                      (22)
 OLD CONTENT:                                                                (23)
           2
 NEW CONTENT:
           1
%  IDA0129 CHANGE? (Y=YES;N=NO)?
Y
 INVERSE MATRIX:                                                             (24)
 -0.33333331E+00 , 0.66666663E+00
 0.66666663E+00 , -0.33333331E+00
 BS2000  F O R 1 : FORTRAN PROGRAM "INV     " ENDED PROPERLY AT 16:15:04
 CPU - TIME USED :       0.3266 SECONDS
 ELAPSED TIME    :     862.3140 SECONDS
```

(17) INV wird erneut geladen.

(18) Auf die Anweisung 30 wird ein Haltepunkt gesetzt. Das Subkommando %SDUMP %NEST bewirkt die Ausgabe der aktuellen Aufrufhierarchie.

(19) %RESUME startet das Programm.

(20) Ausführung des Kommandos %DISPLAY.

(21) Mit dem %SET-Kommando läßt sich der Speicherinhalt eines Datenelements verändern. Wurde vorher das Kommando %AID CHECK=ALL gegeben, dann führt AID vor der Ausführung eines %SET-Kommandos einen Änderungsdialog.

(22) Mit dem %SET-Kommando wird der Wert des Datenelements IS in 1 geändert. Anschließend soll der Programmablauf durch %RESUME fortgesetzt werden.

(23) AID führt vor Ausführung des %SET-Kommandos einen Änderungsdialog.

(24) Nach Änderung des Wertes von IS läuft das Programm korrekt ab.

8 Dateiverarbeitung

Mit den in FORTRAN definierten Sprachmitteln zur Ein-/Ausgabe können Daten auf externe Dateien ausgegeben und von dort gelesen werden. Hierzu müssen die FORTRAN-Ein-/Ausgabekonzepte mit denen des Datenverwaltungssystems (DVS) in Beziehung gesetzt werden.

Die Ein-/Ausgabe-Anweisungen sind im Handbuch "FOR1-Beschreibung" [21] dargestellt. Die verschiedenen Funktionen des DVS und die Kommandos, die für die Dateiverarbeitung nötig sind, werden in den Handbüchern "DVS - Einführung und Kommandoschnittstelle" [18], "DVS - Assemblerschnittstelle" [19] und "Benutzerkommandos (SDF-Format)" [12] beschrieben.

In den Abschnitten 8.1 und 8.2 werden einige wichtige Eigenschaften und Bearbeitungsmöglichkeiten von BS2000-Dateien dargestellt.
Abschnitt 8.3 beschreibt, wie BS2000-Dateien mit FOR1-Programmen verbunden werden. In Abschnitt 8.4 schließlich werden die Zusammenhänge zwischen FORTRAN-Datensätzen und DVS-Datensätzen erläutert.

8.1 BS2000-Systemdateien

Das BS2000 verwendet logische Systemdateien zur Kommando- und Dateneingabe bzw. zur Ausgabe von Daten oder Meldungen des Betriebssystems (siehe Handbuch "Benutzerkommandos (SDF-Format)" [12]).

Man unterscheidet die logischen Eingabedateien des Betriebssystems mit den Standarddateinamen

SYSCMD, SYSDTA, SYSIPT

und die logischen Ausgabedateien mit den Standarddateinamen

SYSOUT, SYSLST, SYSOPT.

Systemdateien werden für jede Task automatisch eingerichtet und müssen vom Anwender nicht eigens vereinbart werden.

Für Systemdateien ist meist eine Zuordnung zu bestimmten Ein-/Ausgabegeräten oder Dateien, die Primärzuweisung, vorgegeben. Der Anwender kann diese Zuordnung mit Hilfe von Kommandos verändern, insbesondere den Systemdateien katalogisierte

Dateien zuordnen (mit den Kommandos ASSIGN-SYSDTA, ASSIGN-SYSIPT, ASSIGN-SYSOUT, ASSIGN-SYSLST, ASSIGN-SYSOPT).

Folgende Tabelle zeigt die vorgegebenen Zuordnungen der Systemdateien:

Systemdatei	Primärzuweisungen	
	Dialogbetrieb	Stapelbetrieb
SYSCMD	Datensichtstation	SPOOLIN- oder ENTER-Datei
SYSDTA		
SYSIPT	keine Primärzuweisung	
SYSOUT	Datensichtstation	SPOOLOUT-Datei (EAM-Datei), die bei Taskende auf Drucker ausgegeben und anschließend gelöscht wird
SYSLST SYSLSTnn	SPOOLOUT-Datei (EAM-Datei), die bei Taskende auf Drucker ausgegeben und anschließend gelöscht wird	
SYSOPT	SPOOLOUT-Datei (EAM-Datei), die bei Taskende auf Diskette oder Kartenstanzer ausgegeben und anschließend gelöscht wird.	

Tab. 8-1: Primärzuweisungen der Systemdateien

Standardmäßig liest FOR1 Quellprogramme, Änderungszeilen und Compileroptionen über die Systemdatei SYSDTA und gibt die erzeugten Listen auf die Systemdatei SYSLST aus.

Den Systemdateien SYSDTA, SYSOUT, SYSLST, SYSIPT und SYSOPT sind in FORTRAN bestimmte Ein-/Ausgabe-Einheiten mittels Voreinstellung zugeordnet (siehe 8.3.3.1).

8.2 BS2000-Anwenderdateien

Im Gegensatz zu Systemdateien, die für jede Task ohne vorhergehende Vereinbarung eingerichtet werden, kann der Anwender bei Anwenderdateien die Eigenschaften der Datei selbst vereinbaren. Der Anwender kann u.a. folgende Dateieigenschaften festlegen:

- den Dateinamen (siehe u.a. "DVS - Einführung und Kommandoschnittstelle" [18])
- den Dateikettungsnamen (8.3.1)
- die Zugriffsmethode (8.2.1)
- das Satzformat (8.2.2)
- die Satzlänge (8.2.2)

Kommandos zum Festlegen der Dateieigenschaften

Die Eigenschaften einer Datei können mit den Kommandos CREATE-FILE, MODIFY-FILE-ATTRIBUTES und SET-FILE-LINK vereinbart werden.

Das Kommando CREATE-FILE erstellt einen Eintrag im Benutzerkatalog und weist der Datei Speicherplatz zu. Die im CREATE-FILE-Kommando angeführten Datei-Eigenschaften werden in den Katalog übernommen. Die restlichen Dateimerkmale, z.B. Zugriffsmethode oder Satzlänge, werden erst dann in den Katalog eingetragen, wenn die Datei nach dem ersten Öffnen geschlossen wird.

Mit dem Kommando MODIFY-FILE-ATTRIBUTES können Katalog-Eigenschaften bereits im Katalog verzeichneter Dateien geändert werden. Informationen über die Katalog-Eigenschaften von Dateien kann man mit dem Kommando SHOW-FILE-ATTRIBUTES anfordern.

Mit dem Kommando SET-FILE-LINK wird ein Dateikettungsname vereinbart; die im Kommando angegebenen Datei-Eigenschaften werden in die TFT (Task File Table) eingetragen (siehe 8.3.1). Das SET-FILE-LINK-Kommando erzeugt nur dann einen neuen Katalogeintrag, wenn zum im FILE-NAME-Operanden genannten Dateinamen noch kein Katalogeintrag existiert.

Permanente und temporäre Dateien

Anwenderdateien werden unterschieden in permanente und temporäre Dateien. Permanente Dateien sind vom Anwender mit den Zugriffsmethoden SAM, ISAM, UPAM oder BTAM erstellte Dateien, die auch nach Beendigung der Task, in der sie erstellt und katalogisiert wurden, erhalten bleiben.

Temporäre Dateien sind vom Anwender mit den Zugriffsmethoden SAM, ISAM, UPAM oder BTAM erstellte Dateien, die an die erzeugende Task gebunden sind. Sie werden im Rahmen der LOGOFF-Behandlung automatisch gelöscht. Temporäre Dateien

unterscheiden sich in der Namensgebung von permanenten Dateien (durch Präfix "#" oder "@"). Der Begriff "temporäre Datei" bezieht sich nur auf Dateien nach obiger Definition und nicht auf andere taskabhängige Dateiarten wie Systemdateien oder EAM-Dateien. EAM-Dateien und Systemdateien werden im Unterschied zu temporären Dateien nicht im Benutzerkatalog geführt und unterliegen nicht der Pubspace-Kontrolle.

8.2.1 Zugriffsmethoden des DVS

Eine ausführliche Beschreibung der Zugriffsmethoden befindet sich in den Handbüchern "DVS - Einführung und Kommandoschnittstelle" [18] und "DVS - Assemblerschnittstelle" [19]. Im folgenden wird nur eine kurze Übersicht gegeben.

Für den Zugriff auf Dateien verwenden FOR1-Programme die logischen Zugriffsmethoden des Datenverwaltungssystems. Die logischen Zugriffsmethoden des DVS übertragen Daten zwischen einer Datei und dem Adreßraum eines aktiven Programms. Durch die Zugriffsmethode wird die Dateiorganisation und die Art des Zugriffs auf die Datensätze festgelegt.

Das FOR1-Laufzeitsystem unterstützt folgende Zugriffsmethoden:

- die allgemein nutzbaren Zugriffsmethoden ISAM, SAM, BTAM
- die Spezialzugriffsmethode EAM für auftragsabhängige Dateien.

Die Zugriffsmethode kann durch den Operanden ACCESS-METHOD des Kommandos SET-FILE-LINK festgelegt werden (siehe 8.3.1). Voreingestellt ist ACCESS-METHOD= BY-PROGRAM. Ein FOR1-Programm eröffnet standardmäßig Dateien mit Zugriffsmethode ISAM.

ISAM (Indexed Sequential Access Method)
Die index-sequentielle Zugriffsmethode bietet die Möglichkeit, Sätze sequentiell und nicht-sequentiell zu verarbeiten. Sätze können hier auf Grund ihrer logischen Folge in der Datei eingefügt werden. Für das nicht-sequentielle Verarbeiten von Sätzen hat jeder Satz einen Schlüssel. Dieser befindet sich bei allen Sätzen einer Datei jeweils an der gleichen Position. Standardmäßig steht der Schlüssel vor dem Datenteil des Satzes und ist 8 byte lang.

SAM (Sequential Access Method)
Die sequentielle Zugriffsmethode bietet die Möglichkeit, ab einer definierten Stelle Sätze sequentiell zu verarbeiten. Das Aktualisieren und Rückschreiben von Sätzen in eine SAM-Datei wird vom FOR1 nicht unterstützt.

BTAM (Basic Tape Access Method)
Die Bandzugriffsmethode bietet die Möglichkeit, Sätze einer sequentiell organisierten Banddatei zu speichern und Blöcke einer sequentiell organisierten Banddatei wiederaufzufinden.

EAM (Evanescent Access Method)
Mit der Zugriffsmethode EAM kann auf Arbeitsdateien zugegriffen werden, die nur für die Dauer des Programmablaufs zur Verfügung stehen. FOR1 legt EAM-Dateien an, wenn in der OPEN-Anweisung der Operand STATUS='SCRATCH' angegeben wurde.

Bei allen Zugriffsmethoden werden die Dateieigenschaften festgelegt, wenn eine Datei erzeugt wird, d.h. wenn sie als Ausgabedatei eröffnet wird. Die Dateieigenschaften werden in den Katalogeintrag übernommen und gelten für alle folgenden Vereinbarungen.

8.2.2 Satzformat und Satzlänge

Das DVS unterscheidet drei verschiedene *Satzformate*:

- Satzformat FIXED, feste Länge
- Satzformat VARIABLE, variable Länge
- Satzformat UNDEFINED, undefinierte Länge.

Eine Datei kann nur aus Sätzen mit gleichem Satzformat bestehen. Bei Satzformat FIXED sind alle Sätze einer Datei gleich lang. Bei Satzformat VARIABLE und Satzformat UNDEFINED können die Sätze verschieden lang sein.

Tab. 8-2 zeigt die zulässigen Satzformate und Gerätetypen für die verschiedenen Zugriffsmethoden.

Zugriffs-methode	Satzformate F	V	U	Gerätetypen	Dateien anderer Zugriffsmethoden als Eingabe zulässig
SAM	X	X	X	Platte, Band	
ISAM	X	X		Platte	
BTAM	X	X [1)]	X	Band	SAM
EAM	X			Platte	

[1)] wird als Satzformat U behandelt

Tab. 8-2: Zugriffsmethoden, Satzformate und Gerätetypen

Festlegen des Satzformats

Das Satzformat wird im SET-FILE-LINK-Kommando mit dem Operanden RECORD-FORMAT festgelegt. Voreingestellt ist BY-PROGRAM. Standardmäßig legen FOR1-Programme Dateien mit variablem Satzformat an.

Festlegen der Satzlänge

Die *Länge eines Satzes* (RECSIZE) kann mit dem Operanden RECORD-SIZE im SET-FILE-LINK-Kommando festgelegt werden. Die Angabe im RECORD-SIZE-Operanden bezieht sich auf die Länge des FORTRAN-Datensatzes zuzüglich evtl. vorhandener Verwaltungsinformation (siehe 8.4).

Satzlängenfeld

Die Information über die Länge des einzelnen Satzes wird bei Satzformat VARIABLE durch ein zusätzliches Informationsfeld am Beginn des Satzes, das Satzlängenfeld (SLF), angegeben. Das 4 byte lange Satzlängenfeld enthält in den beiden höherwertigen Bytes die Länge des Satzes (einschließlich Satzlängenfeld). In den beiden restlichen Bytes steht ein Zeiger auf den nächsten Satz. Bei Satzformat UNDEFINED erfolgt die Angabe der Länge über ein Register.

8.2.3 Datenblock und Puffer

Die Zugriffsmethoden des DVS übertragen Daten zwischen Datei und Adreßraum des Auftrags.
Die Übertragung der Daten erfolgt nicht satzweise. Das DVS stellt Sätze zu Datenblöcken zusammen und überträgt diese Datenblöcke zwischen Peripheriespeicher und Hauptspeicher. Unter einem *Datenblock* oder *logischem Block* versteht man die Einheit, die das DVS bei einem Dateizugriff zwischen peripherem Speicher und Hauptspeicher überträgt. Der Teil des Hauptspeichers, der einen Datenblock aufnimmt oder dessen Inhalt zum Peripheriespeicher übertragen wird, heißt *Puffer*. Der Puffer gehört zum Adreßraum des Programms, das die Ein-/Ausgabeoperation veranlaßt hat.

Die Größe eines logischen Blocks wird in PAM-Seiten beschrieben. Eine *PAM-Seite* (auch Standardblock, Halbseite oder im Gegensatz zum logischen Block physi(kali)scher Block genannt) ist eine Speichereinheit, die 2048 byte umfaßt und je nach Dateiformat einen PAM-Schlüssel enthalten kann.

Die Größe des logischen Blocks und damit des Puffers wird durch den BUFFER-LENGTH-Operanden des Kommandos SET-FILE-LINK festgelegt. Ein logischer Block kann folgende Größen annehmen:

- bei Plattendateien einen Standardblock (2048 byte) oder ein ganzzahliges Vielfaches eines Standardblocks (maximal 16 PAM-Seiten)
- bei Banddateien darüberhinaus als Nichtstandardblock eine Länge bis maximal 32767 byte.

ISAM und EAM arbeiten mit Standardblöcken. SAM und BTAM können sowohl Standard- als auch Nicht-Standardblöcke verarbeiten (bei Magnetbanddateien). Bei Plattendateien arbeitet SAM mit Standardblöcken.

Der für Daten nutzbare Bereich eines logischen Blocks ergibt sich aus der Größe des logischen Blocks (festgelegt durch den BUFFER-LENGTH-Operanden) abzüglich evtl. vorhandener Verwaltungsinformation. Die Größe der Verwaltungsinformation unterscheidet sich nach Zugriffsart und Dateiformat (vgl. Tab. 8-3 u. Tab. 8-4). Bei der Festlegung des logischen Blocks muß der FOR1-Anwender beachten, daß der für Daten nutzbare Bereich eines Datenblocks

- mindestens so lang wie der längste Satz der Datei sein muß
- möglichst genau ein Vielfaches der Satzlänge betragen sollte, um eine optimale Nutzung des Datenbereichs zu erreichen.

Beispiel: Optimale Blockgröße (Dateiformat NK-SAM, variable Satzlänge)

Dateivereinbarung:

```
/SET-FILE-LINK LINK-NAME=DSET12, FILE-NAME=DAT, ACCESS-METHOD=SAM,
RECORD-FORMAT=FIXED(RECORD-SIZE=1500)
```

Diese Kombination von BUFFER-LENGTH und RECORD-SIZE wäre ungünstig: Voreingestellt ist eine Blocklänge von 2048 byte. Davon sind 2044 byte für DVS-Sätze nutzbar (vgl. Tab. 8-4). Jeder Block kann nur einen Satz aufnehmen, jeweils 544 byte gehen verloren.
Günstiger wären Blöcke, die drei PAM-Seiten lang sind: BUFFER-LENGTH= STD(SIZE=3). Bei dieser Blockgröße (6144 byte, davon nutzbar 6140) kann jeder Block 4 Sätze aufnehmen, verloren gehen nur 140 byte.

8.2.4 Keybehaftetes und keyloses Dateiformat

Ab der BS2000-Version 10.0 wird die keylose Plattenperipherie unterstützt. Ab dieser Version unterscheidet man zwischen zwei Dateiformaten:

- dem bisherigen keybehafteten Format (im folgenden auch K-Format genannt)
- dem neu eingeführten keylosen Format (im folgenden auch NK-Format genannt).

Das keylose Dateiformat für ISAM-Dateien wird bereits vom Betriebssystem V9.5 unterstützt.

Auf keybehafteten Platten kann sowohl mit Dateien im K-Format, als auch mit Dateien im NK-Format gearbeitet werden.
Auf keylosen Platten dagegen kann nur mit Dateien im NK-Format gearbeitet werden.

Das Dateiformat wird durch den Operanden BLOCK-CONTROL-INFO des SET-FILE-LINK-Kommandos gesteuert:

BLOCK-CONTROL-INFO=PAMKEY (Dateiverarbeitung im keybehafteten Format)
BLOCK-CONTROL-INFO=WITHIN-DATA-BLOCK (Dateiverarbeitung im keylosen Format)

Die Verwaltungsinformationen, die beim K-Format in einem separaten PAM-Key untergebracht sind, werden beim NK-Format in den Datenbereich verlagert. Die Verlagerung der PAM-Key-Informationen in den Datenbereich verkürzt den für die Datensätze zur Verfügung stehenden Bereich. Dies hat eine **Veränderung der optimalen und maximalen Satzlängen** beim keylosen Format zur Folge, die der FOR1-Anwender beachten muß.

Keybehaftete und keylose ISAM-Dateien

Sollen ISAM-Dateien im K-Format, die die maximale Satzlänge ausnützen, bei gleicher Blocklänge auf NK-Format umgestellt werden, so legt das DVS Überlaufblöcke an.

Die Bildung von Überlaufblöcken bringt folgende Probleme mit sich:

- die Überlaufblöcke erhöhen den Platzbedarf auf der Platte und damit die Zahl der Ein-/Ausgaben während der Dateibearbeitung
- der ISAM-Schlüssel darf in keinem Fall in einem Überlaufblock liegen.

Überlaufblöcke können vermieden werden, wenn man dafür sorgt, daß der längste Satz der Datei nicht länger ist als der bei NK-ISAM-Dateien nutzbare Bereich eines logischen Blockes.

In folgender Tabelle wird dargestellt, wie man bei ISAM-Dateien errechnen kann, wieviel Platz pro logischem Block für DVS-Sätze zur Verfügung steht.

Dateiformat	**Satzformat**	**nutzbarer Bereich**
K-ISAM	VARIABLE	Blockgröße
	FIXED	Blockgröße - (s*4) s = Anzahl der Sätze pro logischem Block
NK-ISAM	VARIABLE	Blockgröße - (n*16) - 12 - (s*2) (auf nächste durch 4 teilbare Zahl abgerundet) n = Blockungsfaktor s = Anzahl der Sätze pro logischem Block
	FIXED	Blockgröße - (n*16) - 12 - (s*2) - (s*4) (auf nächste durch 4 teilbare Zahl abgerundet) n = Blockungsfaktor s = Anzahl der Sätze pro logischem Block

Tab. 8-3: Nutzbarer Block-Bereich bei ISAM-Dateien

Zur Erläuterung der Formeln:
Bei RECORD-FORMAT=FIXED ist sowohl bei K- als auch bei NK-ISAM-Dateien pro Satz ein 4 byte langes Satzlängenfeld zwar vorhanden, wird aber nicht zur Blockgröße gerechnet. Deshalb müssen in diesen Fällen pro Satz jeweils 4 byte abgezogen werden.
Bei NK-ISAM-Dateien enthält jede PAM-Seite eines logischen Blocks jeweils 16 byte Verwaltungsinformation. Der logische Block enthält zusätzlich weitere 12 byte Verwaltungsinformation und pro Satz einen 2 byte langen Satzpointer.

Beispiel: maximale Satzlänge einer NK-ISAM-Datei (feste Satzlänge)

Dateivereinbarung:

```
/SET-FILE-LINK LINK-NAME=DSETnn, FILE-NAME=datei, RECORD-FORMAT=FIXED,
 BUFFER-LENGTH=STD(SIZE=2), BLOCK-CONTROL-INFO=WITHIN-DATA-BLOCK
```

maximale Satzlänge (nach Formel in Tab. 8-3):
4096 - (2*16) - 12 - 1*2 - 1*4 = 4046, abgerundet auf die nächste durch vier teilbare Zahl: 4044 (byte).

Keybehaftete und keylose SAM-Dateien

Bei SAM-Dateien gibt es keine Überlaufblöcke. Deshalb können SAM-Dateien im K-Format, die die maximale Satzlänge ausnützen, nicht in NK-SAM-Dateien umgewandelt werden. FOR1-Programme, die mit solchen für K-SAM-Dateien maximalen Satzlängen arbeiten, sind mit NK-SAM-Dateien nicht mehr ablauffähig.

In folgender Tabelle wird dargestellt, wieviel Platz bei SAM-Dateien pro logischem Block für DVS-Sätze zur Verfügung steht.

Dateiformat	**Satzformat**	**nutzbarer Bereich**
K-SAM	VARIABLE	Blockgröße - 4
	FIXED / UNDEFINED	Blockgröße
NK-SAM	VARIABLE / FIXED UNDEFINED	Blockgröße - 16

Tab. 8-4: Nutzbarer Bereich bei SAM-Dateien

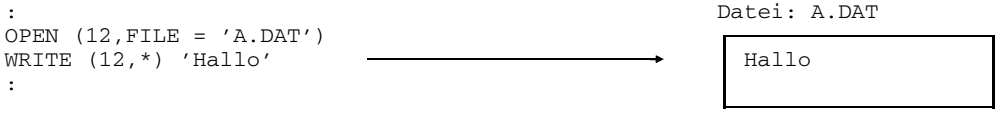
Der Abzug von 4 byte bei K-SAM-Dateien mit variabler Satzlänge resultiert daraus, daß die logischen Blöcke solcher Dateien ein Blocklängenfeld dieser Länge enthalten, das nicht zur BLKSIZE gerechnet wird.

8.3 Verbindung von BS2000-Dateien und FOR1-Programmen

Soll ein FOR1-Programm Dateien verarbeiten, so muß eine Verbindung zwischen Dateien und Programm hergestellt werden. Als Grundlage hierfür dienen die FORTRAN-Ein-/Ausgabe-Einheiten. Die Verbindung kann auf verschiedene Arten hergestellt werden:

- direkt, ohne Dateikettungsnamen
  FOR1-Programme können auf BS2000-Dateien direkt über die FORTRAN-Ein-/Ausgabe-Einheiten zugreifen, ohne daß Dateikettungsnamen vereinbart werden müssen. Durch Angabe des Dateinamens im FILE-Operanden der FORTRAN-Anweisung OPEN kann man festlegen, auf welche Anwenderdatei jeweils zugegriffen werden soll.

  Beispiel:



  Da die OPEN-Anweisung jedoch keine Parameter zur Bestimmung von Dateiformat, Satzformat, Satzgröße, Blockgröße und OPEN-Modus hat, können diese Datei-Eigenschaften bei direkter Verbindung nicht explizit festgelegt werden. Deshalb ist es empfehlenswert, Ein-/Ausgabe-Dateien grundsätzlich über Dateikettungsnamen anzusprechen (siehe unten).

  BS2000-Systemdateien sind bereits per Voreinstellung mit bestimmten Ein-/Ausgabe-Einheiten verbunden. Systemdateien werden immer direkt angesprochen, d.h. eine Verwendung von Dateikettungsnamen ist hier nicht möglich. Die Anweisungen OPEN und CLOSE entfallen.

- über Dateikettungsnamen
  In Entsprechung zu den FORTRAN-Dateinummern stehen die Dateikettungsnamen DSET00, DSET01,..., DSET99 zur Verfügung. Dateikettungsnamen werden mit dem BS2000-Kommando SET-FILE-LINK vereinbart (siehe folgender Abschnitt).

8.3.1 Vereinbaren von Dateikettungsnamen: BS2000-Kommando SET-FILE-LINK

Mit dem Kommando SET-FILE-LINK kann man BS2000-Anwenderdateien über einen Dateikettungsnamen mit FOR1-Programmen verbinden. Diese "indirekte" Verbindung hat den Vorteil, daß sie variabel ist: Ohne Änderung des Quellcodes kann für jeden Programmablauf festgelegt werden, welche Dateien dem Programm zugeordnet werden sollen. Außerdem können mit dem SET-FILE-LINK-Kommando jeweils die Eigenschaften der zu verbindenden Dateien festgelegt werden. Eine ausführliche Beschreibung des SET-FILE-LINK-Kommandos enthält das Handbuch "Benutzerkommandos (SDF-Format)" [12].

FOR1-spezifische Dateikettungsnamen

Dateikettungsnamen, die Dateien mit FOR1-Programmen verbinden, beziehen sich jeweils auf eine bestimmte Ein-/Ausgabe-Einheit. Sie werden wie folgt gebildet:

```
LINK-NAME = DSETnn
```

nn ist dabei die Nummer einer Ein-/Ausgabe-Einheit. Es stehen somit für die Verbindung von Dateien mit FOR1-Programmen die Dateikettungsnamen DSET00, DSET01, DSET02, ..., DSET99 zur Vefügung.

Beispiel:

```
/SET-FILE-LINK LINK-NAME=DSET12, FILE-NAME=B.DAT, ACCESS-METHOD=SAM
```

TFT-Eintrag

Unter dem im LINK-NAME-Operanden des SET-FILE-LINK-Kommandos angegebenen Dateikettungsnamen erstellt das DVS einen Eintrag in der Task File Table. Die Task File Table (TFT) ist eine temporäre Tabelle, die für jede Task automatisch eingerichtet wird. In den TFT-Eintrag werden alle im SET-FILE-LINK-Kommando spezifizierten Dateieigenschaften übernommen. Besteht zum angegebenen Dateikettungsnamen bereits ein Eintrag in der TFT, so wird dieser überschrieben. Über den Inhalt der TFT kann man sich mit dem Kommando SHOW-FILE-LINK informieren. TFT-Einträge können mit dem Kommando REMOVE-FILE-LINK gelöscht oder mit dem CHANGE-FILE-LINK-Kommando geändert werden.

Zugriff auf Dateien über den TFT-Eintrag

Versucht ein FOR1-Programm über eine bestimmte Ein-/Ausgabe-Einheit eine Datei zu eröffnen, so prüft das DVS, ob unter dem entsprechenden Dateikettungsnamen ein Eintrag in der TFT existiert. Falls ein solcher Eintrag gefunden wird, werden alle folgenden Ein-/Ausgabe-Anweisungen, die diese Einheit ansprechen, auf die verbundene Datei bezogen - solange der entsprechende TFT-Eintrag besteht.

Die Zuordnung über Dateikettungsnamen hat Vorrang gegenüber der direkten Zuordnung, d.h. falls ein entsprechender TFT-Eintrag besteht, erfolgt die Zuordnung auch dann über den Dateikettungsnamen, wenn im FILE-Operanden der OPEN-Anweisung der Namen einer anderen Datei genannt wird.

Beispiel:

```
:                                    TFT:
:                                   LINK=DSET12              Datei B.DAT
OPEN (12, FILE='A.DAT')             FILE=B.DAT
WRITE (12,*) 'Hallo'  ------------------------------------->  Hallo
```

Ändern der TFT während des Programmablaufs

Ein TFT-Eintrag kann während des Programmablaufs geändert, gelöscht oder überschrieben werden (mit den Kommandos CHANGE-FILE-LINK REMOVE-FILE-LINK bzw. SET-FILE-LINK). Während der Kommando-Eingabe muß die betreffende Datei geschlossen sein.

Es gibt zwei Möglichkeiten, die TFT während des Programmablaufs zu ändern:

– Unterbrechen des Ablaufs mit der FORTRAN-Anweisung PAUSE,
Absetzen des gewünschten Kommandos auf Betriebssystem-Ebene und
Fortsetzen des Programmablaufs mit dem Kommando R[ESUME].

Beispiel:

```
 CLOSE (UNIT=unit1,...)
 PAUSE [ausdruck]
/SET-FILE-LINK LINK-NAME=DSETunit2, FILE-NAME=datei,...
/R
 OPEN (UNIT=unit2,...)
```

– Absetzen des gewünschten Kommandos ohne Programmunterbrechung mit Hilfe der FPOOL-Funktion FCMD (siehe Abschnitt 12.2.5).

Beispiel:

```
COMOPT FPOOL = FOR1.FPOOL
            .
            .
      COMMAND='/SET-FILE-LINK LINK-NAME=DSETunit2, FILE-NAME=datei, ...'
      CLOSE (UNIT=unit1,...)
      CALL FCMD(RETCODE, COMMAND, ANTWORT, 'Y', SCRATCH)
      OPEN (UNIT=unit2,...)
```

8.3.2 Festlegung der Dateimerkmale durch das DVS

Beim Eröffnen einer Datei bezieht das DVS Informationen über Dateimerkmale aus drei verschiedenen Quellen:

- dem Katalogeintrag (falls vorhanden)
- dem TFT-Eintrag (falls vorhanden)
- expliziten und impliziten Vereinbarungen im Programm

Dabei haben Angaben aus dem TFT-Eintrag, d.h. durch ein SET-FILE-LINK-Kommando festgelgte Spezifikationen, Vorrang vor Vereinbarungen im Programm. Aus dem Katalogeintrag werden nur Dateimerkmale übernommen, die weder durch das Programm noch durch den TFT-Eintrag festgelegt sind, oder die im SET-FILE-LINK-Kommando mit BY-CATALOG spezifziert wurden.

Enthalten die verschiedenen Informationsquellen widersprüchliche Angaben, so kann es zu Unvereinbarkeiten kommen:

Ein Ablauffehler tritt beispielsweise dann auf, wenn die Angaben zur Datei-Organisationsform (FCBTYPE) im Katalog und in der TFT nicht übereinstimmen, oder wenn sie nicht zu den im Programm getroffenen Vereinbarungen passen. Wird etwa im Programm der ACCESS-Operand der OPEN-Anweisung mit 'DIRECT' spezifiziert, so darf weder im Katalogeintrag noch im TFT-Eintrag ein anderer FCBTYPE als ISAM verzeichnet sein. (Der im TFT-Eintrag verzeichnete FCBTYPE wird durch den Operanden ACCESS-METHOD des SET-FILE-LINK-Kommandos festgelegt.)

8.3.3 FORTRAN Ein-/Ausgabe-Einheiten

FORTRAN Ein-/Ausgabe-Anweisungen stellen den Bezug zu Dateien über logische Ein-/Ausgabe-Einheiten her.

Die logischen Ein-/Ausgabe-Einheiten für externe Dateien sind die Dateinummern 0 - 99. Dateinummern können auf verschiedene Weise zugeordnet werden:

- durch Voreinstellung (bei BS2000-Systemdateien)
- durch die OPEN-Anweisung (bei Anwenderdateien)
- durch implizites OPEN (falls es zur entsprechenden Dateinummer weder eine Voreinstellung noch eine OPEN-Anweisung gibt)

Neben den externen Dateien gibt es in FORTRAN interne Dateien (siehe Handbuch "FOR1-Beschreibung" [21]). Interne Dateien sind FORTRAN Datenelemente vom Typ CHARACTER. Ein-/Ausgabe-Einheiten für interne Dateien sind keine Dateinummern, sondern die Namen der Datenelemente. Interne Dateien müssen nicht eigens zugeordnet werden, somit gibt es für sie keine OPEN-Anweisung.

8.3.3.1 Standard-Zuordnung von BS2000-Systemdateien

Die Systemdateien SYSDTA, SYSOUT, SYSLST, SYSOPT und SYSIPT sind per Voreinstellung mit bestimmten Ein-/Ausgabe-Einheiten verbunden. Im FORTRAN-Programm muß in Ein-/Ausgabe-Anweisungen nur die jeweilige Dateinummer angegeben werden.

Tab. 8-5 zeigt die Zuordnung von Systemdateien zu Ein-/Ausgabe-Einheiten:

Datei-nummer	Datei-name	verwen-deter Makro	Satz-format	Länge des Daten-teils	Länge des DVS-Satzes
1	SYSDTA	RDATA	V	2024	2028
2	SYSOUT	WROUT	V	2024	2028
5	SYSDTA	RDATA	V	2024	2028
6	SYSLST	WRLST	V	133	137
7	SYSOPT	WRTOT	F	80	80
8	SYSIPT	RDCRD	F	80	80
97	SYSDTA	RDATA	V	2024	2028
98	SYSOPT	WRTOT	F	80	80
99	SYSLST	WRLST	V	133	137

Tab. 8-5: Standardzuordnung von BS2000-Systemdateien

Auf die Ein-/Ausgabe-Einheiten 1, 5, 8 und 97 darf nur lesend, auf die Ein-/Ausgabe-Einheiten 2, 6, 7, 98 und 99 darf nur schreibend zugegriffen werden.

Die voreingestellten Zuordnungen kann der Anwender durch die Laufzeitoptionen SUBSTITUTE, ADD und DELETE ändern (siehe 6.3.2).

8.3.3.2 OPEN-Anweisung

Mit der FORTRAN-Anweisung OPEN kann man
- eine katalogisierte Datei mit einer Ein-/Ausgabe-Einheit verbinden
- eine noch nicht existierende Datei katalogisieren und mit einer Ein-/Ausgabe-Einheit verbinden
- die FORTRAN-Zugriffsart und weitere Eigenschaften festlegen oder ändern.

Die FORTRAN-Anweisung DEFINE FILE wirkt wie eine eingeschränkte OPEN-Anweisung für Dateien mit Direktzugriff.

Die OPEN-Anweisung wird ausführlich im Handbuch "FOR1-Beschreibung" [21] dargestellt. Im folgenden werden das Format der OPEN-Anweisung dargestellt und die Operanden RECL, STATUS und FILE erläutert.

```
OPEN    ([ACCESS=caccess] [,ASSOVAR=^xc0iassovar|cassovar^xd0] [,BLANK=cblank]
          [,ERR=ierr] [,FILE=cfile] [,FORM=cform] [,IOSTAT=iostat]
          [,MAXREC={imaxrec|cmaxrec}] [,RECL=irecl] [,STATUS=cstatus]
          [,UNIT=] iunit )
```

Der erste Buchstabe der einzelnen Parameter deutet den Typ an

i INTEGER
c CHARACTER

RECL

Der Operand RECL beschreibt die Länge eines FORTRAN-Satzes in byte. Die Angabe RECL=irecl wird nur bei ACCESS='DIRECT,...' ausgewertet. Die Ein-/Ausgabe eines längeren Satzes verursacht einen Fehler.

Fehlt die Angabe des RECL-Parameters, dann gilt:

- wenn im Programmablauf schon ein OPEN oder implizites OPEN auf diese Datei ausgeführt wurde, behält RECL den darin festgelegten Wert;
- ansonsten erhält RECL den für RECORD-SIZE eingetragenen bzw. voreingestellten Wert minus Länge der entsprechenden Verwaltungsinformation (siehe Tab. 8-6 und 8-7 in Abschnitt 8.4.2).

Hinweis:

Bei Sätzen einer ISAM-Datei zählt der Schlüssel zu Beginn eines Satzes nicht zum Datenteil. Steht jedoch der Schlüssel nicht am Beginn des Satzes, so gehört er zum Datenteil des Satzes und muß mitgerechnet werden (siehe 8.4.2).

STATUS

Im Operand STATUS wird angegeben, ob die Datei bereits existiert bzw. ob sie als auftragsabhängige Arbeitsdatei (FCBTYPE=EAM) angelegt werden soll. Maßgebend für die Angabe im Operanden STATUS ist, ob der entsprechende Eintrag im Systemkatalog vorhanden ist, nicht jedoch die physikalische Existenz der Datei. Z.B. kann ein Katalogeintrag durch das CREATE-FILE-Kommando erzeugt werden, die Datei existiert jedoch physikalisch erst nach einem OPEN-/CLOSE-Vorgang.

Für den Operanden STATUS können folgende Werte spezifiziert werden:

```
        ⎡'OLD'    ⎤
        ⎢'NEW'    ⎥
STATUS = ⎨'UNKNOWN'⎬
        ⎢'SCRATCH'⎥
        ⎣cstatus  ⎦
```

'OLD' Für die angesprochene Datei existiert bereits ein Katalogeintrag.

Ist 'OLD' spezifiziert und existiert kein Eintrag im Systemkatalog, dann tritt ein Ablauffehler auf.

'NEW' Es existiert kein Eintrag im Systemkatalog, es wird eine ISAM-Datei mit dem angegebenen Dateinamen erstellt. Ist 'NEW' spezifiziert und existiert unter dem angegebenen Dateinamen bereits ein Eintrag im Systemkatalog, dann tritt ein Ablauffehler auf.

'SCRATCH' Es wird eine prozeßabhängige Arbeitsdatei (EAM-Datei) erstellt, die bei Programmende gelöscht wird. Für diese Datei angegebene SET-FILE-LINK-Kommandos werden nicht ausgewertet.

'UNKNOWN' Wenn der FILE-Operand angegeben ist und die dort angegebene Datei schon existiert, oder für die angegebene Ein-/Ausgabe-Einheit auch kein SET-FILE-LINK-Kommando in Kraft, dann erhält der STATUS-Operand den Wert 'OLD'.
Ist kein FILE-Operand angegeben und ist für die angegebene Ein-/Ausgabe-Einheit auch kein SET-FILE-LINK-Kommando in Kraft, dann erhält der STATUS-Operand den Wert 'NEW' und es wird eine Datei mit einem Standard-Dateinamen (siehe FILE-Operand) angelegt. Nach Ausführung der OPEN-Anweisung wird der STATUS-Operand auf 'OLD' gesetzt.

cstatus CHARACTER-Ausdruck, der zum Zeitpunkt der OPEN-Anweisung den Wert 'OLD', 'NEW', 'SCRATCH' oder 'UNKNOWN' haben muß.

Wird STATUS='OLD' oder 'NEW' spezifiziert, so muß auch der Operand FILE spezifiziert werden.

FILE

Im Operanden FILE wird der Name der Datei spezifiziert. Existiert keine Datei mit dem angegebenen Dateinamen, dann wird eine Datei mit diesem Namen erzeugt, falls nicht STATUS='OLD' angegeben wurde. Wurde für eine Datei kein Name angegeben (fehlender FILE-Operand, FILE='' oder FILE=' '), und wurde für die angegebene Ein-/Ausgabe-Einheit weder ein SET-FILE-LINK-Kommando noch eine vorhergehende OPEN-Anweisung mit FILE='name' angegeben, dann wird eine Datei mit folgendem Standard-Dateinamen erzeugt:

```
UNIT.FOR1.prog.unit [.tsn[.time]]
```

prog	Name der Programmeinheit
unit	Dateinummer
tsn	Task sequence number, vierstellige Zahl
time	Aktuelle Zeit der Dateierstellung in der Form hhmmss.

Die letzten beiden Qualifizierungen werden nur durchgeführt, wenn es für die Eindeutigkeit des Eintrags im Systemkatalog notwendig ist.

Wird für eine Datei weder eine OPEN- noch eine DEFINE FILE-Anweisung verwendet, so wird sie bei der ersten Ein-/Ausgabe-Anweisung auf diese Datei implizit geöffnet. Die Operandenwerte des impliziten OPEN hängen von den Operandenwerten dieser ersten Ein-/Ausgabe-Anweisung ab (siehe Handbuch "FOR1-Beschreibung" [21]).

8.4 Abbildung der FORTRAN-Sätze in das DVS

8.4.1 FORTRAN-Satz und DVS-Satz

Unter einem **FORTRAN-Satz** wird im folgenden ein durch FORTRAN-Sprachelemente beschriebener Datensatz verstanden. Ein FORTRAN-Satz entspricht dem Datenteil eines oder mehrerer DVS-Sätze. Die Länge eines FORTRAN-Satzes kann bei ACCESS= 'DIRECT' im RECL-Operand der OPEN-Anweisung oder in der DEFINE FILE-Anweisung (Angabe als Stellungsparameter) angegeben werden.

Ein **DVS-Satz** ist ein Satz, mit dem das DVS arbeitet. Er kann zusätzlich zum FORTRAN-Satz Verwaltungsinformationen enthalten:

- das Satzlängenfeld bei variablem Format
- den Satzschlüssel bei ISAM-Dateien
- das GREEN CONTROL WORD (GCW) bei formatfreier Ein-/Ausgabe.

Die Länge bzw. bei variabler Satzlänge die maximale Satzlänge eines DVS-Satzes wird durch den Operand RECORD-SIZE des SET-FILE-LINK-Kommandos angegeben.

Für den FOR1-Anwender ist das Verhältnis von FORTRAN-Satz zu DVS-Satz wichtig, um

- ausgehend von der Länge des FORTRAN-Satzes den RECORD-SIZE-Operanden des SET-FILE-LINK-Kommandos zu bedienen
- ausgehend von der Länge des DVS-Satzes eine möglichst günstige Größe des logischen Blocks zu wählen.

Satzlängenfeld

Bei Satzformat V steht am Anfang des Satzes das 4 byte lange Satzlängenfeld (siehe 8.2.2).

Satzschlüssel

Standardmäßig steht der Schlüssel vor dem Datenteil des Satzes und ist 8 byte lang.

Befindet sich der Schlüssel am Beginn eines Satzes, so zählt er für FORTRAN nicht zum Datenteil des Satzes und wird bei der Ein-/Ausgabe nicht übertragen. Beginnt jedoch der Schlüssel nicht am Anfang eines Satzes, so zählt er zum Datenteil eines Satzes und wird bei der Ein-/Ausgabe übertragen. Die Angabe im RECL-Operanden schließt in diesem Fall die Schlüssellänge mit ein. Ein innerhalb der Daten liegender Satzschlüssel muß bei der Ein-/Ausgabe berücksichtigt werden: Entweder sieht man in der Ein-/Ausgabeliste eine Dummy-Variable für dem Satzschlüssel vor, oder man überspringt ihn, indem man in der Formatangabe den Tabulator-Formatschlüssel X verwendet. Berücksichtigt man einen innerhalb der Daten liegenden Statzschlüssel nicht, so werden z.B. bei der Ausgabe Ausgabedaten durch den Schlüssel überschrieben.

Bei sequentieller Ausgabe auf eine ISAM-Datei erhält der erste Satz den Schlüsselwert 1 und nach jedem Schreiben eines Satzes wird der laufende Schlüsselwert um 1 erhöht.

GREEN CONTROL WORD

Ein unformatierter Satz kann beliebig lang sein. Das DVS kann nur Sätze mit einer Länge von maximal 32 Kbyte verarbeiten. Um die Abbildung von beliebig langen FORTRAN-Sätzen in das DVS zu ermöglichen, wird ein GREEN CONTROL WORD benötigt. Das GREEN CONTROL WORD (GCW) enthält Verwaltungsinformationen über den Aufbau von Datensätzen bei formatfreier Ein-/Ausgabe. Das GCW ist 4 byte lang und steht unmittelbar vor dem Datenteil eines Teilsatzes.

Wird bei unformatierter Ein-/Ausgabe ein FORTRAN-Datensatz auf mehrere DVS-Sätze verteilt, so werden diese zu Paketen zusammengefaßt. Ein Paket kann aus bis zu 255 DVS-Sätzen bestehen. Da die Länge eines unformatierten FORTRAN-Satzes unbegrenzt ist, können pro FORTRAN-Satz beliebig viele Pakete gebildet werden.

Aufbau des GREEN CONTROL WORD:

Byte 0

0	Teilsatz ist nicht letzter Satz im Paket
Anzahl der Teilsätze im Paket	Teilsatz ist letzter Satz im Paket

Byte 1

0	Teilsatz ist nicht letzter Satz im Paket (Byte 0 = 0) oder der FORTRAN-Satz besteht nur aus einem Paket
1	Teilsatz ist letzter Satz im ersten Paket
2	Teilsatz ist letzter Satz im letzten Paket
3	Teilsatz ist letzter Satz und Paket ist nicht erstes und nicht letztes Paket

Byte 2, Byte 3

Byte 2 und 3 enthalten die Anzahl der Datenbytes im Teilsatz.

Beispiel:

ISAM-Datei mit RECORD-SIZE=120. Es wird ein FORTRAN-Satz mit 700 Teilsätzen geschrieben.

ISAM-Key	G C W	FORTRAN-Daten	
	00000068		1. Teilsatz, 1. Paket
	00000068		2. Teilsatz, 1. Paket
	. . .		
	FF010068		letzter (255.) Teilsatz, 1. Paket
	00000068		1. Teilsatz, 2. Paket
	. . .		
	FF030068		letzter (255.) Teilsatz, 2. Paket
	00000068		1. Teilsatz, 3. Paket
	. . .		
	BE020068		letzter (190.) Teilsatz, 3. Paket

8.4.2 Übersicht: Verhältnis DVS-Satz/FORTRAN-Satz

Tabelle 8-6 und 8-7 zeigen, wie sich die Länge eines DVS-Satzes aus der Länge der FORTRAN-Daten zuzüglich evtl. vorhandener Verwaltungsinformation ergibt, abhängig von Zugriffsart, Satzformat und Ein-/Ausgabeformat. Die Verwaltungsinformation kann sich aus folgenden Bestandteilen zusammensetzen:

KEY ISAM-Schlüssel, standardmäßig 8 byte
SLF Satzlängenfeld, 4 byte
GCW GREEN CONTROL WORD, 4 byte

DAT bezeichnet die Länge des FORTRAN-Datensatzes. Bei ACCESS= 'DIRECT' in der OPEN-Anweisung entspricht DAT der Angabe im RECL-Parameter.

Für die Zugriffsart ISAM gilt die Darstellung des Satzaufbaus nur für KEY-POSITION=1 bei RECORD-FORMAT=FIXED bzw. KEY-POSITION=5 bei RECORD-FORMAT= VARIABLE.

DVS-Sätze bei formormatgebundener Ein-/Ausgabe

ZUGRIFFS-ART	SATZFORMAT F	V	U
S A M	DAT	SLF + DAT	DAT
I S A M	KEY + DAT	SLF + KEY + DAT	-
B T A M	DAT	SLF + DAT	DAT
E A M	2048 byte DAT DAT ≤ 2048 byte	-	-
SYSTEM-DATEIEN	BEI IPT UND LST: DAT	BEI DTA,OUT UND LST: SLF + DAT	-

Tab. 8-6: Aufbau und Länge des DVS-Satzes bei formatgebundener Ein-/Ausgabe

DVS-Sätze bei formatfreier Ein-/Ausgabe

ZUGRIFFS-ART	SATZFORMAT F	V	U
S A M	GCW + DAT	SLF + GCW + DAT	GCW + DAT
I S A M	KEY + GCW + DAT	SLF + KEY + GCW + DAT	-
B T A M	GCW + DAT	SLF + GCW + DAT	GCW + DAT
E A M	2048 byte GCW + DAT GCW+DAT≤2048 byte	-	-
SYSTEM-DATEIEN	BEI IPT UND LST: GCW + DAT	BEI DTA,OUT UND LST: SLF + GCW + DAT	-

Tab. 8-7: Aufbau und Länge des DVS-Satzes bei formatfreier Ein-/Ausgabe

8.4.3 Beispiele: FORTRAN-/DVS-Satz

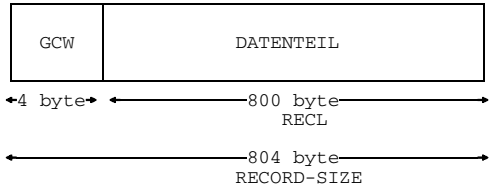
Beispiel 1: FORTRAN-/DVS-Satz bei SAM-Datei

```
/CREATE-FILE FILE-NAME=DAT, SUPPORT=PUBLIC-DISK(SPACE=RELATIVE(PRIMARY-
 ALLOCATION=4))
/SET-FILE-LINK LINK-NAME=DSET17, FILE-NAME=DAT, ACCESS-METHOD=SAM,
 RECORD-FORMAT=FIXED(RECORD-SIZE=804), BUFFER-LENGTH=STD(SIZE=2)
```

Ein-/Ausgabe-Anweisung auf diese Datei:

```
.
.
.
INTEGER*8 FELD(100)
OPEN(UNIT=17,FORM='UNFORMATTED')
WRITE(17)FELD
```

Ein Satz der Datei hat folgendes Aussehen:



Im SET-FILE-LINK-Kommando wird RECORD-SIZE=804 angegeben, da sich die Satzlänge bei formatfreier Ein-/Ausgabe aus dem Datenteil von 800 byte und den 4 byte des Green Control Word zusammensetzt.

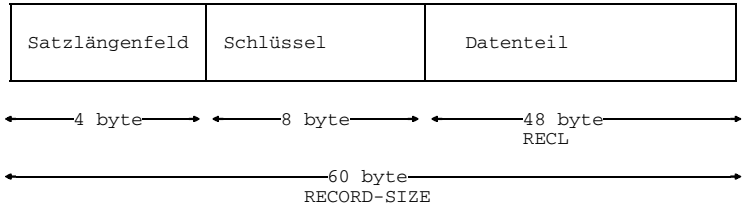
Beispiel 2: FORTRAN-/DVS-Satz bei ISAM-Datei

```
/SET-FILE-LINK LINK-NAME=DSET20, FILE-NAME=DAT1, ACCESS-METHOD=ISAM,
 RECORD-FORMAT=VARIABLE(RECORD-SIZE=60)
```

Ein-/Ausgabe-Anweisungen auf diese Datei können folgendes Aussehen haben:

```
      sequentieller Zugriff                  Direktzugriff

      OPEN(UNIT=20,ACCESS='SEQUENTIAL',      OPEN(UNIT=20,ACCESS='DIRECT',
     *     STATUS='OLD',FILE='DAT1')        *     RECL=48,STATUS='OLD',
                                            *     FILE='DAT1')
      WRITE(20,100)I,J,K,L                   READ (20,REC=16,FMT=100)I,J,K,L
100   FORMAT (4I12)                    100   FORMAT (4I12)
                .                                .
                .                                .
                .                                .
```

Da RECORD-FORMAT=VARIABLE tritt bei jedem Satz zu Datenteil und Schlüssel noch ein Satzlängenfeld. Ein Satz hat in diesem Beispiel folgenden Aufbau:



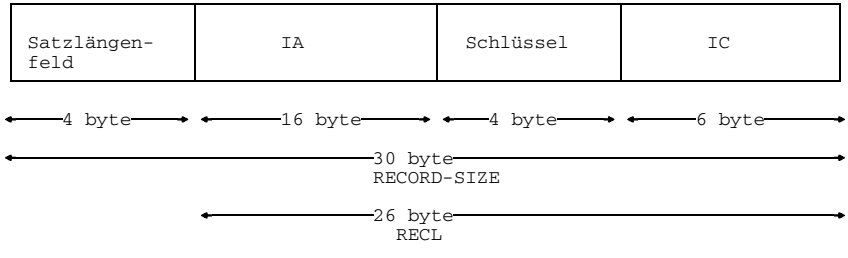
Beispiel 3: FORTRAN-/DVS-Satz bei ISAM-Datei mit Schlüssel im Datenteil

```
/SET-FILE-LINK LINK-NAME=DSET21, ACCESS-METHOD=ISAM(KEY-LENGTH=4,
 KEY-POSITION=21), RECORD-FORMAT=VARIABLE(RECORD-SIZE=30)
```

Ein-/Ausgabe-Anweisungen auf diese Datei können folgendes Aussehen haben:

```
      sequentieller Zugriff                  Direktzugriff

      OPEN (UNIT=21,ACCESS='SEQUENTIAL',     OPEN (UNIT=21,RECL=26,ACCESS=
     *      STATUS='OLD',                   *      'DIRECT',STATUS='OLD',
     *      FILE='DAT2')                    *      FILE='DAT2')
      READ(21,100)IA,IB,IC                   WRITE(21,REC=18,FMT=100)IA,IB,IC
100   FORMAT(I16,A4,I6)                100   FORMAT(I16,I4,I6)

      Nach Ausführung der                Der durch IB angegebene Wert wird
      READ-Anweisung ent-                durch die binäre Darstellung
      hält die Variable IB               des Schlüsselwertes 18 über-
      den Wert des Schlüssels.           schrieben.
```

Ein Satz hat folgendes Aussehen:



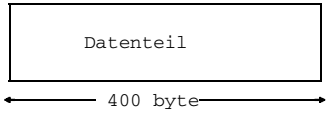
Beispiel 4: FORTRAN-/DVS-Satz bei BTAM-Datei

```
/CREATE-FILE FILE-NAME=DAT2, SUPPORT=TAPE(VOLUME=datenträger, DEVICE-TYPE=
 gerätetyp)
/SET-FILE-LINK LINK-NAME=DSET50, FILE-NAME=DAT, ACCESS-METHOD=BTAM,
 RECORD-FORMAT=UNDEFINED, BUFFER-LENGTH=400
```

Ein-/Ausgabe-Anweisungen auf diese Datei können folgendes Aussehen haben:

```
    INTEGER*8 A(50)
    OPEN (UNIT = 50)
    WRITE (50,100)A
100 FORMAT (50 I8)
```

Ein Satz der Datei hat folgendes Aussehen:



9 Optimierung

Die Optimierung des FOR1-Compilersystems ist auf eine Beschleunigung der Objektprogramme ausgerichtet. Unter Optimierung werden im folgenden alle Maßnahmen verstanden, die mit Hilfe zusätzlicher Analysen und zusätzlicher Transformationen eine kürzere Laufzeit der Objektprogramme bewirken.

Der FOR1-Programmierer hat grundsätzlich zwei Möglichkeiten zur Optimierung:

- manuelle Optimierung, d.h. Verkürzen der Laufzeit durch effizienten Programmierstil (Abschnitt 9.1);
- Optimierungsmaßnahmen des FOR1, die durch den SDF-Operand OPTIMIZATION des Kommandos START-FOR1-COMPILER (Abschnitt 9.2.1) oder durch die Compileroption OPTIMIZE (Abschnitte 9.2.2) gesteuert werden.

Die Optimierung betrifft in erster Linie häufig auszuführende Programmteile (Schleifen) und langsame Programmteile (Ein-/Ausgabe).

Folgende Optimierungsmaßnahmen werden häufig vom FOR1-Compiler durchgeführt:

- Berechnung konstanter arithmetischer Ausdrücke während der Übersetzung,
- Optimierung von logischen Ausdrücken,
- Wiederverwendung gemeinsamer Teilausdrücke,
- Optimierung von Indexberechnungen,
- Schleifenoptimierung,
- Eliminieren von überflüssigem Code,
- Registeroptimierung.

Beim Aufruf von Unterprogrammen und Funktionen können ebenfalls Optimierungen durchgeführt werden, die mit der Compileroption PROCEDURE-OPTIMIZATION (siehe Abschnitt 9.2.3) gesteuert werden.

Der Anwender kann eine der Optimierungsstufen 0 bis 4 bzw. NO auswählen. Der Programmbereich, über den optimiert wird, unterscheidet sich nach der gewählten Optimierungsstufe. Bei den Optimierungsstufen 1 und 2 wird über Basisblöcke und DO-Schleifen optimiert. Ein Basisblock ist eine maximale unverzweigte Befehlsfolge, die genau einen Eingangspunkt und einen Ausgangspunkt hat. Bei den Optimierungsstufen 3 und 4 wird dagegen über ganze Schleifen jeglicher Art optimiert.

Die Optimierungsstufen des FOR1 sind in Abschnitt 9.2 ausführlich beschrieben.

9.1 Manuelle Optimierung

Die Optimierung von Programmen von Hand ist ein wirkungsvolles Mittel, die Laufzeit unabhängig von einer Übersetzung herabzusetzen. Der Anwender kann einige Maßnahmen, die die FOR1-Optimierung durchführt, bereits bei der Erstellung des Programms berücksichtigen, wie z.B. die Wiederverwendung bereits berechneter (Teil-) Ausdrücke oder die Herausnahme schleifeninvarianter Teile aus dem Schleifenbereich. Diese Optimierungen werden an dieser Stelle nicht beschrieben. Die hier angeführten Optimierungsmaßnahmen stellen eine Ergänzung der FOR1-Optimierung dar.

Verzweigungen

Für logische IF-Anweisungen und Block-IF-Anweisungen wird der günstigste Objektcode erzeugt, wenn der Testausdruck eine LOGICAL*1-Variable ist. Bei arithmetischen IF-Anweisungen ist eine INTEGER*2-Variable am günstigsten.

Unterprogramme

Wenn es nur um die Effizienz geht, können FUNCTION- oder SUBROUTINE-Unterprogramme für häufig ausgeführte, aber kleine Aufgaben ungeeignet sein, wenn der Aufwand für Aufruf und Rücksprung relativ hoch im Verhältnis zu den eigentlichen Aufgaben ist. Es gibt zwei Alternativen dafür:

- Code als %INCLUDE-Element realisieren, d.h. Ausprogrammieren der betreffenden Anweisungen statt Aufruf eines Unterprogramms
- Verwendung einer Formelfunktion (wenn möglich).

Die Übergabe von Parametern über COMMON-Bereiche ist effizienter als über Parameterlisten, allerdings nicht mit der Testoption ARG zu erfassen.

Arithmetische Ausdrücke

Für kleine ganzzahlige Exponenten ist A*A*A . . . günstiger als A**I, weil iterative Multiplikationen wesentlich schneller sind als eine Potenzierung. Aus ähnlichem Grund ist A+A+A . . . für kleine N schneller als A*N. Durch Weiterrechnen mit Zwischenergebnissen kann eine noch schnellere Ausführungszeit erreicht werden.

Beispielsweise kann die Potenzierung `X=A**8` in folgende Anweisungen zerlegt werden:

```
X1=A*A
X2=X1*X1
X =X2*X2
```

Die Auflösung in Multiplikationen wird auch durch die Optimierung des FOR1 durchgeführt.

Da die Multiplikation kürzere Ausführungszeit hat als die Division, sollte so weit wie möglich Division durch Multiplikation ersetzt werden.

Beispiel:

`A/(B*C*D)` statt `(((A/B)/C)/D)`
`SQRT(X)` ist effizienter als `X**.5`

Konversionen

Konversionen zwischen verschiedenen Datentypen können mehr Zeit kosten als die eigentliche Operation und sollten daher nach Möglichkeit vermieden werden. Dies kann durch Umordnen von Operanden in arithmetischen Ausdrücken geschehen.

Beispiel:

Anweisungsfolge:

```
     INTEGER I1,I2,I3
     REAL R1,R2,R3
     A=I1+R1+I2-R2-I3+R3
```

besser:

```
     A=(I1+I2-I3)+(R1-R2+R3)
```

DO-Schleifen

Gleichstrukturierte aufeinanderfolgende DO-Schleifen sollen zusammengelegt werden.

Beispiel:

```
     DO 1 I=1,10
1    A(I)=B(I)+C(I)
     DO 2 I=1,10
2    AA(I)=F(I)
```

Diese Anweisungsfolge kann folgendermaßen umgeformt werden:

```
     DO 1 I=1,10
     A(I)=B(I)+C(I)
1    AA(I)=F(I)
```

Diese Schleife ist schneller als die ursprünglichen zwei Schleifen, weil der Aufwand für die Steuerung der Schleife nur einmal auftritt.

Mehrdimensionale Felder

Bei Verwendung von mehrdimensionalen Feldern kann in manchen Fällen durch Überlagerung mit eindimensionalen Feldern eine schnellere Verarbeitung erreicht werden.

Beispiel:

```
      REAL A (10,50,50)
      DO 1 I=1,50
         :
      DO 1 J=1,50
      DO 1 K=1,10,2
1     A(K,J,I)=A(K,J,I)+B
      X=A(L,M,N)
```

Diese Anweisungsfolge kann folgendermaßen umgeformt werden:

```
      REAL A(10,50,50),AA(25000)
      EQUIVALENCE(A,AA)
         :
      DO 1 I=1,25000,2
1     AA(I)=AA(I)+B
      X=A(L,M,N)
```

Ein-/Ausgabe

Die formatfreie Ein-/Ausgabe ist die schnellste Art der Ein-/Ausgabe. Zwischenergebnisse, die nur vom selben Programm wieder gelesen werden, sollten daher immer formatfrei ausgegeben werden. Außerdem belegen formatfrei angegebene Daten meist weniger Platz auf dem externen Speicher.

Die Anzahl der Ausführungen von Ein-/Ausgabe-Anweisungen soll möglichst gering gehalten werden.
Ungünstig ist beispielsweise folgender Programmteil, die WRITE-Anweisung wird zehnmal ausgeführt:

```
    REAL A(10),B(10)
    DO 1 I=1,10
1   WRITE(21) A(I),B(I)
```

Besser ist folgende Anweisungsfolge, da hier die WRITE-Anweisung nur einmal ausgeführt wird:

```
    REAL A(10),B(10)
    WRITE(21) (A(I),B(I),I=1,10)
```

Noch besser sind folgende Anweisungsfolgen, da hier die Elemente von A und B nicht einzeln, sondern in zusammenhängenden Stücken verarbeitet werden:

```
    REAL A(10),B(10)
    WRITE(21) (A(I),I=1,10),(B(I),I=1,10)
```

bzw.

```
    REAL A(10),B(10)
    WRITE(21) A,B
```

oder:

```
    REAL A(10),B(10)
    WRITE(21) A(1)...A(10),B(1)...B(10)
```

Programmierung für virtuellen Speicher

Der virtuelle Speicher bietet mit seinem großen Adreßraum in vielen Fällen die Möglichkeit, auch größere Dateien im Speicher anstatt in externen Dateien zu halten. Dies erspart eventuell sehr viele Ausführungen von Ein-/Ausgabe-Anweisungen und vereinfacht die Programmierung, denn der Zugriff zu den Daten geschieht direkt bzw. assoziativ (Variablenamen) anstatt über Ein-/Ausgabe-Anweisungen, und man braucht keine Programmteile zur Verwaltung der Dateien oder zur Fehlerbehandlung.

Die virtuelle Speichertechnik hat aber noch weitere Aspekte, die für die Laufzeit von Programmen und den Durchsatz des Systems bedeutsam sind. Im Betriebsystem BS2000 hat jede Task einen eigenen Adreßraum von (je nach Systemgenierung) einigen Megabyte. Dieser Adreßraum ist in "Seiten" (pages) zu je 4096 byte unterteilt. Das System sorgt dafür, daß die Seiten, auf die ein Prozeß gerade zugreift, auch tatsächlich im Hauptspeicher der Zentraleinheit vorhanden sind. Die unbenützten Seiten werden dagegen möglichst aus dem Hauptspeicher ferngehalten bzw. auf einen externen Speicher, den sogenannten "Seitenwechselspeicher" ausgelagert. Das folgende Schema veranschaulicht dieses Prinzip, die Seiten A1, A3 bzw. B3 sind ausgelagert:

Bild 9-1: Schema der virtuellen Speicherung

Will nun ein Prozeß auf eine Seite seines Adreßraums zugreifen, die nicht im Hauptspeicher vorhanden ist, so wird der Prozeß unterbrochen und das System muß die Seite vom Seitenwechselspeicher in den Hauptspeicher einlesen. Falls kein Platz im Hauptspeicher frei ist, muß eine andere Seite aus dem Hauptspeicher ausgelagert werden. Zugriffe auf nicht im Hauptspeicher vorhandenen Seiten sind also mit Unterbrechungen und - wenn auch bevorzugten und relativ schnellen - Ein-/Ausgabe-Operationen verbunden.

Um Laufzeit einzusparen, sollte man sein Programm so einrichten, daß die benötigten Seiten möglichst immer im Hauptspeicher vorhanden sind. Die Grundregeln hierfür sind:

- Man vermeide es, unnötig neue Seiten anzusprechen, indem man die Speicherbereiche, in denen man gerade arbeitet, möglichst nicht verläßt.
- Anweisungen, die im Ablauf aufeinanderfolgen, bzw. Daten, auf die aufeinanderfolgend zugegriffen wird, sollten im (Objekt-)Programm auch räumlich nahe beieinander angeordnet sein.

Es gibt eine Reihe von Möglichkeiten, dies im FORTRAN-Programm zu berücksichtigen:

Bei Algorithmen sollten "Weit"-Sprünge über große Bereiche vermieden werden.

Teile von Algorithmen, die nur selten ausgeführt werden, z.B. Fehlerroutinen, sollten aus dem normalerweise durchlaufenden Kontext herausgenommen werden und an separate Stellen plaziert werden.

Teile von Algorithmen, die regelmäßig ausgeführt werden, sollten "in-line" programmiert werden, d.h. nicht durch Aufruf von Unterprogrammen.

In Unterprogrammen kann der Zugriff zu den globalen Datenelementen (Übergabe in COMMON-Bereichen oder durch Adreßübergabe) zu Seitenwechsel führen, so daß sich eine Wertübergabe empfiehlt.

Bei der Verarbeitung großer Datenaggregate sollte auf die physikalische Anordnung der Daten im Speicher Rücksicht genommen werden.

Beispielsweise ist folgender Programmteil ungünstig, da FOR1 Felder spaltenweise ablegt:

```
      REAL*4 A (1000,100)
        .
        .
        .
      DO 1 I = 1,1000
      DO 1 J = 1,100
1     A(I,J) = A(I,J) + I*J
```

Bei einer Seitengröße von 4096 byte liegt das angesprochene Element in der Anweisung 1 fast für jede Iteration der inneren DO-Schleife in einer anderen Seite und verursacht daher mit hoher Wahrscheinlichkeit einen sehr häufigen Seitenwechsel. Im ungünstigsten Fall gibt es fast 100000 Seitenwechsel.

Günstiger ist folgende Anordnung der DO-Schleifen:

```
      DO 1 J = 1,100
      DO 1 I = 1,1000
1     A(I,J) = A(I,J) + I*J
```

Die Elemente von A werden in der Reihenfolge ihrer Anordnung im virtuellen Adreßraum angesprochen. Im ungünstigsten Fall gibt es etwa 100 Seitenwechsel, eine Verbesserung, die in der Größenordnung von vielen Minuten liegen kann.

Allgemein ist es günstiger, wenn bei Schachtelungen von DO-Schleifen der von den inneren Schleifen angesprochene Speicherbereich möglichst klein ist, da auf Grund der häufigen Ausführung der inneren Schleifen die dabei notwendigen Seitenwechsel ebenfalls sehr häufig wären.

Beim Binden können durch explizite Angaben der INCLUDE-Steueranweisung im zeitlichen Ablauf benachbarte Programmeinheiten räumlich ebenfalls beieinander angeordnet werden. Dabei wird es i.a. zweckmäßig sein, die häufigsten Programmeinheiten nicht an den Anfang zu sortieren, sondern eher in die "Mitte".

Verbesserung der Übersetzungszeit

Die Reihenfolge der Spezifikationsanweisungen im FORTRAN-Quellprogramm kann die Übersetzungszeit eines Programms beeinflussen. Bei einer großen Anzahl von Spezifikationsanweisungen können die erzielten Einsparungen von Bedeutung sein.

Es ist günstig, alle gleichartigen Spezifikationsanweisungen nebeneinander anzuordnen. Folgende Anweisungen sollten jeweils zusammengefaßt werden:

IMPLICIT-Anweisungen
INTEGER-, REAL-, LOGICAL-Anweisungen
DIMENSION-, CHARACTER-Anweisungen
PARAMETER-Anweisungen
COMMON-Anweisungen
EQUIVALENCE-Anweisungen
DATA-Anweisungen

Die Reihenfolge der einzelnen Gruppen hat keinen Einfluß auf die Übersetzungszeit.

PARAMETER-Anweisungen, die auf andere symbolische Konstanten zurückgreifen, sollten hinter den PARAMETER-Anweisungen für diese symbolischen Konstanten stehen.

Ungünstig ist beispielsweise:

```
PARAMETER(A=B+5)
PARAMETER(B=7)
```

Günstiger:

```
PARAMETER(B=7)
PARAMETER(A=B+5)
```

DATA-Anweisungen, die eine implizite DO-Schleife verwenden, sollten am Ende der DATA-Anweisungsgruppe stehen.

9.2 Steuern der Optimierung

Für die verschiedenen Phasen der Programmentwicklung bietet FOR1 jeweils geeignete Optimierungsstufen.

In der Testphase kommt es darauf an, möglichst schnell und bequem Fehler in einem Programm zu finden. Dazu eignet sich die Optimierungsstufe NO. Bei dieser Optimierungsstufe wird das Programm direkt Anweisung für Anweisung in Maschinencode umgesetzt. Der Compiler benötigt dadurch für die Übersetzung nur wenig Zeit. Außerdem kann das so entstandene Objekt optimal mit der symbolischen Testhilfe AID untersucht werden.

Ist das Programm ausgetestet und soll eingesetzt werden, sollte es möglichst schnell laufen und wenig Speicherplatz verbrauchen. Deshalb ist es das Ziel der Hochoptimierung, die mit Optimierungsstufe 3 eingeschaltet werden kann, einen Objektcode zu erzeugen, der auch von einem Assembler-Programmierer nicht wesentlich effizienter erstellt werden könnte. Dazu müssen umfangreiche Analysen am Programm durchgeführt werden, was sich durch eine deutlich höhere Übersetzungszeit bemerkbar macht.

Einen Kompromiß zwischen den Optimierungsstufen NO und 3 stellt die voreingestellte Optimierungsstufe 1 dar. Hier werden nur begrenzt Analysen durchgeführt und dadurch weniger Optimierungsmaßnahmen vorgenommen. Durch eine etwas höhere Übersetzungszeit wird die Objektlaufzeit etwas verbessert.

Bei den bisher genannten Optimierungsstufen NO, 1 und 3 werden nur solche Transformationen am Programm vorgenommen, die sein Verhalten nach außen nicht verändern. Die mit diesen Optimierungsstufen erzeugten Objekte liefern immer exakt die gleichen Ergebnisse wie nicht optimierte Objekte.

Die Objektlaufzeit kann u.U. durch den Einsatz der Optimierungsstufe 4 und die Optimierungsparameter REORDER, PARAMETER-SIDEEFFECT und FUNCTION-SIDEEFFECT noch weiter verkürzt werden als mit Stufe 3. Da es hierbei in seltenen Fällen aber zu einer Verschlechterung der Objektlaufzeit bzw. zu abweichenden Ergebnissen kommen kann, sollte der Anwender diese weitergehenden Optimierungsmaßnahmen nur dann verwenden, wenn er deren Wirkungsweise kennt und beurteilen kann, ob sie für sein Programm geeignet sind. (Das gleiche gilt für die Optimierungsstufe 2.) Die Optimierungsmaßnahmen des FOR1 sind deshalb in Kapitel 9.3 ausführlich beschrieben.

9.2.1 SDF-Operand OPTIMIZATION

```
START-FOR1-COMPILER

,OPTIMIZATION = NO / LOW / MEDIUM(...) / HIGH(...)

   MEDIUM(...)
      |  CONDITIONAL-LOOPS = IGNORED / RISK-OPTIMIZED
      | ,OPTIMIZE-PROCEDURES = NO / YES / SPECIAL

   HIGH(...)
      |  CONDITIONAL-LOOPS = IGNORED / RISK-OPTIMIZED
      | ,OPTIMIZE-PROCEDURES = NO / YES / SPECIAL
      | ,OPTIMIZATION-HINTS = STD / PARAMETER(...)
      |    PARAMETER(...)
      |      |  REORDER-EXPRESSIONS = YES / NO
      |      | ,FUNCTION-SIDEEFFECTS = YES / NO
      |      | ,ARGUMENT-SIDEEFFECTS = NO / YES
```

Eine Gegenüberstellung von SDF-Operanden und entsprechenden Compileroptionen enthält Tab. 2-11.

9.2.2 Compileroption OPTIMIZE

Die Compileroption COMOPT OPTIMIZE steuert die Optimierung. Die Optimierungsstufen NO, 0, 1, 2, 3 oder 4 können ausgewählt werden. Standardmäßig ist die Optimierungsstufe 1 eingestellt.

Die mit OPTIMIZE gewählte Optimierungsstufe beeinflußt die Voreinstellungen der Compileroptionen SAVE-CONSTANT (siehe Abschnitt 4.1.2.7) und PROCEDURE-OPTIMIZATION (siehe Abschnitt 9.2.3):

bei OPTIMIZE=NO, 0, 1, 2:
- SAVE-CONSTANT=YES ist voreingestellt
- bei PROCEDURE-OPTIMIZATION wird die Voreinstellung PROCEDURE-OPTIMIZATION=STD als PROCEDURE-OPTIMIZATION=NO interpretiert

bei OPTIMIZE=3, 4:
- SAVE-CONSTANT=NO ist voreingestellt
- bei PROCEDURE-OPTIMIZATION wird die Voreinstellung PROCEDURE-OPTIMIZATION=STD als PROCEDURE-OPTIMIZATION=YES interpretiert (außer wenn *explizit* LINKAGE=STD angegeben wird)

PROCEDURE-OPTIMIZATION=YES ist unverträglich zu LINKAGE=STD (siehe 4.2.2.6). Deshalb wird bei Eingabe von OPTIMIZE=3,4 die Voreinstellung LINKAGE=STD in LINKGAGE=FOR1-SPECIFIC umgewandelt und folgende Warnung ausgegeben:

```
MA43 LINKAGE=FOR1-SPECIFIC EXPECTED
```

Es gibt jedoch zwei Möglichkeiten, trotz OPTIMIZE=3,4 ILCS-Moduln zu erzeugen:

- LINKAGE=STD wird *explizit* angegeben. PROCEDURE-OPTIMIZATION=STD wird dann trotz OPTIMIZE=3,4 als PROCEDURE-OPTIMIZATION=NO interpretiert, worauf folgende Warnung aufmerksam macht:

  ```
  MA42 PROC-OPT=NO BECAUSE LINK=STD
  ```

- Angabe von PROCEDURE-OPTIMIZATION=NO *vor* Eingabe von OPTIMIZE=3,4

Die Optimierung bewirkt Veränderungen und Umstellungen des Codes. Werte von Variablen lassen sich unter Umständen nicht mehr anhand des Quellprogramms finden, bzw. läßt sich die Ausführung von Anweisungen nicht mehr verfolgen, wenn ein Programm mit der Dialogtesthilfe AID getestet wird. Deshalb ist es während der Testphase eines Programms sinnvoll, die Optimierung mit OPTIMIZE=NO auszuschalten.

Soll ein Programm mit einer Dialogtesthilfe getestet werden, obwohl es optimiert und damit nicht mehr definiert testbar ist, dann kann eine Decompilerliste (siehe Abschnitt 4.7.9) eine Hilfe sein. Eine Decompilerliste kann angefordert werden, wenn mit den Optimierungsstufen 3 oder 4 optimiert wurde. Die Decompilerliste zeigt die durch die Optimierung verursachten Veränderungen gegenüber dem ursprünglichen Quellprogramm auf. Dadurch wird die Ablaufverfolgung und das Setzen von Testpunkten auf der Ebene des Quellprogramms erleichtert.

[*]COMOPT	OPT[IMIZE]=NO

schaltet alle Optimierungsmaßnahmen aus. OPTIMIZE=NO ist beim Testen mit AID sinnvoll, um Codeveränderungen zu verhindern. Bei OPT=NO befinden sich an Statementgrenzen keine Variablen im Register. Dadurch können durch AID die Variablen eindeutig geändert werden. Die AID-Anweisung %JUMP (Überspringen von Anweisungen) wird nur bei OPT=NO unterstützt.

[*]COMOPT	OPT[IMIZE]=0

schaltet die meisten Optimierungsmaßnahmen aus. Optimiert werden
- logische Ausdrücke,
- arithmetische Ausdrücke, deren Operanden alle konstant sind,
- die Verwendung der Register.

Der Bereich für diese Optimierungen ist die ganze Programmeinheit. Die Übersetzungsphase "Globale Optimierung" (siehe Anhang 2) wird bei OPTIMIZE=0 nicht aktiviert.

[*]COMOPT	OPT[IMIZE]=1

Alle durch OPTIMIZE=0 aktivierten Optimierungen werden durchgeführt. Zusätzlich werden optimiert:
- die Teile arithmetischer Ausdrücke, die aus konstanten Operanden bestehen
- gemeinsame arithmetische Ausdrücke
- Indexberechnungen
- Basisblöcke von expliziten DO-Schleifen, die pro Schleifendurchlauf genau einmal ausgeführt werden. Schleifeninvariante Berechnungen werden vor die Schleife verlagert. Multiplikationen einer INTEGER-Schleifenvariablen mit einer pseudokonstanten Größe (siehe Abschnitt 9.3.5) werden durch Additionen ersetzt.
- Exponentiationen mit Exponent 2 oder 3 werden durch Multiplikationen ersetzt.

[*]COMOPT	OPT[IMIZE]=2

Alle durch OPTIMIZE=1 aktivierten Optimierungen werden zusätzlich auch für bedingt ausgeführte Schleifenteile durchgeführt.

`[*]COMOPT`	`OPT[IMIZE]=` { `3` | `(3,parameter[,...])` }

`parameter:=` { `FUNC[TION-SIDEEFFECT]=` { `YES` \| `NO` } \| `PARAM[ETER-SIDEEFFECT]=` { `YES` \| `NO` } \| `REORDER=` { `YES` \| `NO` } }

Alle durch OPTIMIZE=1 aktivierten Optimierungen werden in größerem Umfang und Bereich durchgeführt. Wird *parameter* nicht angegeben, dann gelten die Voreinstellungen FUNCTION-SIDEEFFECT=YES, PARAMETER-SIDEEFFECT=NO und REORDER=NO. Durch Ändern dieser Voreinstellungen kann die Compilezeit wesentlich verkürzt werden.

Bei OPTIMIZE=3 werden zusätzlich folgende Optimierungen durchgeführt:

- Neben expliziten DO-Schleifen werden auch implizite Schleifen optimiert, wie durch IF-Anweisungen, GOTO-Anweisungen oder ERR- und END-Parameter in Ein-/Ausgabe-Anweisungen gebildete Schleifen. Es werden nur Schleifen optimiert, die nicht mehr als einen Eingang haben, und nur die Teile dieser Schleifen, die bei jedem Schleifendurchlauf genau einmal ausgeführt werden.
- "Überflüssiger" Code, d.h. Anweisungen, deren Ergebnisse nicht mehr benötigt werden, wird eliminiert. Meist entsteht überflüssiger Code als Folge von Optimierungsmaßnahmen. Beispielsweise kann durch die Schleifenoptimierung die Fortschaltung einer Schleifenvariable unnötig werden. Die Schleifenvariable wird auf ihren Endwert gesetzt oder ganz eliminiert.
- Exponentiationen mit konstanten ganzzahligen Exponenten werden in eine Folge von Multiplikationen aufgelöst. Diese Zerlegung ist bis zu einem Exponenten von 120 günstiger als der Aufruf einer Laufzeitroutine. Durch diese Optimierung können bei großen Exponenten Rundungsunterschiede auftreten.
- Bei formatfreien Ein-/Ausgabe-Anweisungen werden Zugriffe auf benachbarte Feldelemente durch Zugriffe auf Feld-Bereiche ersetzt. Um diese Optimierung zu ermöglichen, muß der Zugriff auf die Feldelemente in der Reihenfolge erfolgen, in der die Elemente im Speicher angeordnet sind (siehe Handbuch "FOR1-Beschreibung" [21]).

  Beispiel:
  `READ(X) ((A(I,J),I=1,100),J=1,100)` wird umgewandelt in
  `READ(X) A(1,1): A(100,100)`

Sind die Feldelemente nicht aufsteigend benachbart angeordnet, dann erfolgt eine Warnung: `UNFAVOURABLE INCREMENTATION PREVENTS ARRAY OPTIMIZATION.`
Wird die Reihenfolge der Feldelemente umgestellt, dann kann die Optimierung des Zugriffs durchgeführt werden.

– Isolierte konstante Operanden in arithmetischen Ausdrücken werden zur Übersetzungszeit in den resultierenden Datentyp konvertiert. Sonderfälle, z.B. Multiplikation mit 1 oder 0 oder Addition bzw. Subtraktion von 0, werden erkannt und entsprechend vereinfacht. Wird bei der Auswertung konstanter Ausdrücke eine Division durch Null entdeckt, so erfolgt eine Warnung:
`DIVISION BY ZERO IGNORED`
Die Division wird erst zur Laufzeit durchgeführt.

– INTRINSIC- und normale Funktionen werden vor der Schleife berechnet, wenn ihre Parameter pseudokonstant (siehe Abschnitt 9.3.5) sind.

FUNC[TION-SIDEEFFECT]=NO
: Alle Funktionen werden als normal angenommen.
Bei normalen Funktionen werden Optimierungsmaßnahmen über Funktionsaufrufe hinweg durchgeführt. Für eine als normal definierte Funktion werden folgende Eigenschaften angenommen:

  – Die Funktion verändert keinen ihrer Parameter, sie gibt nur den errechneten Funktionswert an die aufrufende Programmeinheit zurück. Die Funktion darf keine COMMON-Daten benutzen oder verändern.

  – Die Funktion liefert bei jedem Aufruf mit gleichen Parameterwerten den gleichen Funktionswert.

  – Die Funktion führt keine Ein-/Ausgabe durch und ruft keine SUBROUTINE-Unterprogramme oder abnormalen Funktionen auf.

  Der Compiler überprüft nicht, ob eine als "normal" deklarierte Funktion die genannten Eigenschaften wirklich besitzt. Wird eine Funktion als "normal" deklariert, obwohl sie die Eigenschaften einer normalen Funktion nicht besitzt, dann können durch die Optimierung Fehler auftreten.

  Die Angabe FUNCTION-SIDEEFFECT=NO entspricht in ihrer Wirkung der FORTRAN-Anweisung ABNORMAL ohne Parameter (siehe Handbuch "FOR1-Beschreibung" [21]).
  Wurden einzelne Funktionen durch die ABNORMAL-Anweisung als abnormal deklariert, dann hebt FUNCTION-SIDEEFEECT=NO die Wirkung der ABNORMAL-Anweisung wieder auf.

FUNC[TION-SIDEEFFECT]=<u>YES</u>
: Alle Funktionen werden als abnormal angenommen.

PARAM[ETER-SIDEEFFECT]=NO
: Es wird angenommen, daß Formalparameter eines Unterprogramms nicht mit anderen Formalparametern oder COMMON-Variablen assoziiert sind.

PARAM[ETER-SIDEEFFECT]=YES
: Es wird angenommen, daß Formalparameter eines Unterprogramms mit anderen Formalparametern oder COMMON-Variablen assoziiert sind. Bei Veränderung eines Formalparameters oder einer COMMON-Variablen wird eine Veränderung aller Formalparameter und COMMON-Variablen angenommen, so daß viele Optimierungen nicht durchgeführt werden.

REORDER=NO
: Gleichrangige kommutative Operationen werden von links nach rechts durchgeführt, wie es vom Standard ANS FORTRAN 77 vorgeschrieben ist (siehe Handbuch "FOR1-Beschreibung" [21]).

REORDER=YES
: Gleichrangige kommutative Operationen können durch die Optimierung vertauscht werden. Diese Vertauschung kann Auswirkungen auf den Überlauf von Zwischenergebnissen haben. Bei der Berechnung von REAL- und COMPLEX-Größen können durch die Vertauschung Rundungsunterschiede auftreten.

  Beispiel:
  In dem Ausdruck A = 3. + B + 4. werden durch Vertauschung zunächst die Konstanten 3 und 4 zusammengefaßt: A = 7.+ B

`[*]COMOPT`	`OPT[IMIZE]=`{`4` | `(4,parameter[,...]`}

Alle durch OPTIMIZE=3 aktivierten Optimierungen werden zusätzlich auch für bedingt ausgeführte Schleifenteile durchgeführt. Die Angaben FUNCTION-SIDEEFFECT, PARAMETER-SIDEEFFECT und REORDER haben dieselbe Bedeutung wie bei OPTIMIZE=3.

9.2.3 Compileroption PROCEDURE-OPTIMIZATION

Die Compileroption PROCEDURE-OPTIMIZATION steuert Optimierungen beim Aufruf von Prozeduren, d.h. Unterprogrammen und Funktionen. Diese Optimierungen bewirken eine Laufzeitverbesserung durch eine abgestufte Verkürzung des ENTRY-/EXIT-Codes beim Aufruf von Prozeduren. Die Laufzeitverbesserung fällt besonders bei Programmen mit sehr vielen kurzen Prozeduren ins Gewicht.

Die Wirkung der PROCEDURE-Option unterliegt einigen Einschränkungen, die mit dem Grad der gewünschten Optimierung zunehmen.

Die Prozedur-Optimierung schränkt die Möglichkeiten für die Fehlerdiagnose je nach Grad der Optimierung ein. Insbesondere kann bei eingeschalteter Prozedur-Optimierung die Aufrufhierarchie bei Programmabbruch oft nicht mehr zurückverfolgt werden.

Um den Vorrang der Fehlersuche zu gewährleisten, wird eine Prozedur-Optimierung nur durchgeführt, wenn keine Testoptionen außer dem Standardwert TESTOPT=(STNR) angegeben werden. Wird eine andere Testoption außer dem Standardwert gesetzt, dann wird die Prozedur-Optimierung ausgeschaltet.

Sollen bei der Übersetzung ILCS-Moduln erzeugt werden, dann muß mit PROCEDURE-OPTIMIZATION=NO gearbeitet werden (siehe Abschnitt 4.2.2.6, LINKAGE-Option).

`[*]COMOPT`	`PROCEDURE[-OPTIMIZATION] = { STD | NO | YES | SPECIAL[-ATTEMPTS] }`

STD Voreinstellung. Die Prozedur-Optimierung wird durch die in der OPTIMIZE-Option angegebene Optimierungsstufe festgelegt:

bei OPT=NO,0,1,2 PROCEDURE-OPTIMIZATION=NO
bei OPT=3,4 PROCEDURE-OPTIMIZATION=YES

Falls der Anwender explizit LINKAGE=STD setzt, wird PROCEDURE-OPTIMIZATION=STD bei allen Optimierungsstufen als PROCEDURE-OPTIMIZATION=NO interpretiert. Es wird folgende Meldung ausgegeben:

```
MA42 PROC-OPT=NO BECAUSE LINK=STD
```

NO Es wird keine Verkürzung des ENTRY-/EXIT-Codes vorgenommen. Ist eine Testoption außer TESTOPT=(STNR) eingeschaltet, so wird PROCEDURE=NO gesetzt und eine Warnung ausgegeben.

Sollen bei der Übersetzung ILCS-Moduln erzeugt werden, so muß mit PROCEDURE-OPTIMIZATION=NO gearbeitet werden (siehe Abschnitt 4.2.2.6, LINKAGE-Option).

YES Die Prozeduraufrufe werden in Abhängigkeit von den jeweiligen Bedingungen der ENTRY-Umgebung optimiert. Dadurch werden je Unterprogrammaufruf beim ENTRY und beim RETURN Maschinenbefehle eingespart. Diese Einsparung wird erreicht durch stufenweisen Verzicht auf

- Verketten der Saveareas
- Rekursivitätsprüfung
- Kopieren der Argumente
- Sprung über Adreßkonstante
- Bereitstellen separater Register für Konstanten- und Datenadressierung
- Rückkopieren der Argumente
- Restaurieren der alten Basisadressen für Konstanten und Daten
- Rekursionsmarkierung
- BGFOR-kompatible Rücksprungsmarkierung

Werden ENTRY- oder RETURN ausdruck-Anweisungen verwendet, dann ist die erreichbare Code-Verkürzung geringer.

Bei PROCEDURE=YES werden die wichtigsten INTRINSIC-Funktionen durch Aufruf von optimierten Laufzeitsystemfunktionen berechnet. Die ENTRY-Namen der optimierten Laufzeitsystemfunktionen werden aus den ENTRY-Namen der nicht-optimierten Funktionen und einem $-Zeichen als letztem Zeichen gebildet.

INTRINSIC-Funktion	ENTRY-Name	ENTRY-Name der optimierten Funktion
SIN	IF@S	IF@S$
COS	IF@C	IF@C$
ATAN	IF@AT	IF@AT$
SQRT	IF@Q	IF@Q$
EXP	IF@E	IF@E$
ALOG	IF@AL	IF@AL$
DSIN	IF@DS	IF@DS$
DCOS	IF@DC	IF@DC$
DATAN	IF@DAT	IF@DAT$
DSQRT	IF@DQ	IF@DQ$
DEXP	IF@DE	IF@DE$
DLOG	IF@DL	IF@DL$

Tab. 9-1: ENTRY-Namen von optimierten Funktionen

Die optimierten Laufzeitsystemfunktionen können nur aufgerufen werden, wenn keine Formelfunktionen mit dem Namen von INTRINSIC-Funktionen verwendet werden.

Die optimierten Laufzeitfunktionen für die INTRINSIC-Funktionen DSIN, DCOS, DATAN, DSQRT, DEXP und DLOG werden nur von Objekten angesprochen, die mit einer FOR1-Version ≤ 2.1 erstellt wurden. Ab FOR1-Version 2.2 erstellte Objekte verwenden die hochgenauen mathematischen Routinen.

SPECIAL-[ATTEMPTS]

Weitergehende Verkürzung des ENTRY-/EXIT-Codes gegenüber PROCEDURE=YES. Dies führt vor allem für den Fall sehr vieler kurzer Prozeduren zu einer Verbesserung der Laufzeit.

Eine Verbesserung gegenüber der Optimierung bei PROCEDURE=YES tritt nur bei Prozeduren mit folgenden Einschränkungen auf:

- Die gesamte Länge des Prozedur-Objektcodes darf 4096 byte nicht überschreiten. Der Compiler schätzt die zu erwartende Länge des Objektcodes ab. Dabei werden 12 byte für jede Verwendung einer Variablen angenommen. Ab einer geschätzten Länge von etwa 4000 byte wird die Optimierung PROCEDURE=SPECIAL nicht durchgeführt. Überschreitet die tatsächliche Länge des Objektcodes entgegen der Abschätzung 4096 byte, so wird die entgegen der Abschätzung 4096 byte, so wird die Übersetzung mit einer Fehlermeldung abgebrochen.
- Die Prozedur darf keine Nebeneingänge, d.h. keine ENTRY-Anweisungen mit unterschiedlichen Argumenten enthalten. Funktionen müssen bei allen Entries denselben Typ des berechneten Funktionswerts besitzen.
- Die Prozedur darf keine weitere Prozedur aufrufen.
- In einem Unterprogramm darf keine mathematische Funktion aufgerufen werden, es dürfen keine Potenzierungen, Rechnungen mit komplexen Zahlen, Vergleichsoperationen, vierfach genaue Gleitpunktdivisionen oder Compiler-Steueranweisungen ausgeführt werden.
- In der Prozedur dürfen keine Ein-/Ausgabe-Operationen durchgeführt werden.

Erfüllt eine Prozedur diese Bedingungen nicht, dann wird von PROCEDURE= SPECIAL auf PROCEDURE=YES umgeschaltet, ohne daß eine Meldung erfolgt.

Führt die ENTRY-/EXIT-Verkürzung bei PROCEDURE=SPECIAL zu einem Fehler, obwohl das Programm ohne Prozedur-Optimierung fehlerfrei ist, dann wird der Fehler in der Diagnoseliste gemeldet. Das Programm kann dann mit PROCEDURE= YES statt mit PROCEDURE=SPECIAL neu übersetzt und fehlerfrei optimiert werden.

9.3 Optimierungsmaßnahmen des FOR1

9.3.1 Berechnen konstanter Ausdrücke zur Übersetzungszeit

Ausdrücke, deren Operanden wertmäßig bekannt sind, werden zur Übersetzungszeit berechnet. Dies verlagert die Ausführung von Befehlen vom Objektlauf in die Übersetzungszeit und verkürzt die Laufzeit des Programms.

Die Berechnungen zur Übersetzungszeit umfassen alle arithmetischen, logischen und Vergleichsoperationen.

Es werden nicht nur Ausdrücke mit explizit angegebenen Konstanten ausgewertet, sondern auch solche Ausdrücke, in denen Variablen auftreten, deren Werte während der Übersetzung bereits bekannt sind.

Beispiel:

Original	Wirkung der Optimierung
`I = 17/2` `J = I + 1`	`I = 8` `J = 9`

Ausdrücke mit ausschließlich explizit angegebenen Konstanten werden auch bei der Optimierungsstufe 0 bereits zur Übersetzungszeit ausgewertet. Die erste Anweisung des obigen Beispiels wird also auf jeden Fall umgeformt, während die zweite Anweisung nur bei eingeschalteter Optimierung umgeformt wird.

Falls bei der Auswertung eines Ausdrucks, in dem Variablen vorkommen, ein Fehler auftreten würde, wird die Auswertung nicht durchgeführt und eine Warnung ausgegeben. Bei einem fehlerhaften Ausdruck mit lauter expliziten Konstanten wird mit 1 weitergerechnet.

Für "versteckte" Wertzuweisungen an Variablen (READ-Anweisung, Parameter im Unterprogrammaufruf) wird immer angenommen, daß die betreffende Variable ihren Wert ändert.

Eine Wertverfolgung von COMMON-Variablen wird bei den Optimierungsstufen 1 bis 4 nur solange durchgeführt, bis ein Unterprogramm aufgerufen wird. Bei OPT=3/4 werden bis zu einem erneuten Aufruf des Unterprogramms dann erneut Informationen über COMMON-Variablen gesammelt.
Ausdrücke, in denen äquivalent gesetzte Variablen (durch die EQUIVALENCE-Anweisung) auftreten, werden bei OPT=1/2 nicht ausgewertet.

INTRINSIC-Funktionen mit konstanten Parametern werden bei keiner Optimierungsstufe zur Übersetzungszeit berechnet.

9.3.2 Optimieren logischer Ausdrücke

Logische Ausdrücke in der logischen IF-Anweisung werden in eine Folge von bedingten Sprüngen aufgelöst. Sobald der Boolesche Wert des gesamten Ausdrucks feststeht, wird verzweigt und der Rest des Ausdrucks wird nicht mehr behandelt.
Logische Ausdrücke werden auch bei Optimierungsstufe 0 optimiert.

Beispiel:

Original	Wirkung der Optimierung

`IF(A.LT.B.OR.C.GT.F(0).OR.X.EQ.Y)`
`GOTO 10`
	`IF(A.LT.B) GOTO 10`
	`IF(C.GT.F(0))GOTO 10`
	`IF(X.EQ.Y)GOTO 10`

`IF(A.EQ.B.AND.C.GT.D)X=Y`	`IF(A.NE.B)GOTO 10`
	`IF(C.LE.D)GOTO 10`
	`X = Y`
	`10 CONTINUE`

Der Anwender kann durch die Reihenfolge der einzelnen Operanden die Zeit für die Auswertung eines Ausdrucks beeinflussen. Ist A öfter wahr als B, dann ist es günstiger, einen Ausdruck in der Form A.OR.B zu schreiben, anstatt B.OR.A. Analog ist für diesen Fall B.AND.A günstiger als A.AND.B. Wenn die Auswertung eines logischen Ausdrucks nicht alle Operanden erfaßt, entfällt eventuell die Ausführung darin enthaltener FUNCTION-Unterprogrammaufrufe. Im ersten Beispiel wird F(O) nicht ausgeführt, wenn A.LT.B bereits .TRUE. liefert.

9.3.3 Erkennen gemeinsamer Teilausdrücke

Die bei der Übersetzung erzeugten Befehlsfolgen werden dadurch verbessert, daß mehrmals auftretende (Teil-)Ausdrücke nur einmal berechnet werden und bei Bedarf auf den bereits vorhandenen Wert zurückgegriffen wird.

Beispiel:

Original	Wirkung der Optimierung
X=(A+B)*C Y=(A+B)*D	%T1=A+B X=%T1*C Y=%T1*D

Die Optimierung gemeinsamer Teilausdrücke erstreckt sich auf die arithmetischen Operatoren +,-,*,/, bei OPT=3/4 auch auf die arithmetische Operation ** und INTRINSIC-Funktionen. Das Distributivgesetz wird gemäß dem Sprachstandard nicht angewendet.

Teilausdrücke werden bei jeder Optimierungsstufe nur dann als gemeinsam erkannt, wenn sie intern genau gleich dargestellt werden. Die interne Darstellung richtet sich nach der Reihenfolge, in der Ausdrücke ausgewertet werden. Die Reihenfolge ist durch Klammern und durch die Priorität der Operatoren in FORTRAN festgelegt.

Beispiel:

```
X=A+B*C*D
Y=E+B*C
```

In der ersten Anweisung wird der Teilausdruck "B*C*D" und in der zweiten Anweisung der Teilausdruck "B*C" intern als Einheit dargestellt. Da die interne Darstellung beider Teilausdrücke nicht übereinstimmt, wird nicht optimiert.
Wird "B*C" explizit geklammert, dann wird in beiden Anweisungen intern derselbe Teilausdruck B*C dargestellt:

Original	Wirkung der Optimierung
X=A+(B*C)*D Y=E+B *C	%T1=B*C X=A+%T1*D Y=E+%T1

Die gemeinsamen Teilausdrücke beziehen sich auf Werte und nicht auf Variablennamen. Falls eine Variable vor dem wiederholten Auftreten des gleichen Ausdrucks ihren Wert ändert, so kann der bereits berechnete Wert nicht mehr verwendet werden.

Beispielsweise wird folgende Befehlsfolge nicht optimiert, da sich der Wert von A geändert hat:

```
X=A+B
A=A/3
Y=A+B
```

Bezüglich der Verfolgung des Wertes einer Variablen gelten folgende Regeln:

- Für "versteckte" Wertzuweisungen an Variablen (READ-Anweisung, Parameter in Unterprogrammaufruf) wird immer angenommen, daß die betreffende Variable ihren Wert ändert.
- COMMON-Variablen werden bei OPT=1/2 nur solange berücksichtigt, bis ein Aufruf eines SUBROUTINE-Unterprogramms oder eines ABNORMAL FUNCTION-Unterprogramms auftritt. Bei OPT=3/4 werden COMMON-Variablen im Bereich zwischen zwei Unterprogrammaufrufen berücksichtigt.
- Der Bereich der Wertverfolgung erstreckt sich bei OPT=1/2 über einen Basisblock, bei OPT=3/4 über eine Schleife.

Äquivalent gesetzte Größen

Enthält ein Ausdruck äquivalent gesetzte Größen (durch die EQUIVALENCE-Anweisung), dann wird dieser Ausdruck bei OPT = 1/2 nicht optimiert. Bei OPT = 3/4 wird nur optimiert, wenn keine der äquivalent gesetzten Größen ihren Wert zwischenzeitlich ändert.

Beispiel:

```
EQUIVALENCE (A,C)
X = A+B               (1)
   .
   .
   .
Y = A+B               (2)
```

Bei OPT = 1/2 wird der gemeinsame Teilausdruck A+B nicht erkannt, da A eine mit C äquivalent gesetzte Größe ist und äquivalent gesetzte Größen bei diesen Optimierungsstufen nicht ausgewertet werden.
Bei OPT = 3/4 wird der gemeinsame Teilausdruck A+B optimiert, wenn weder A noch B noch die mit A äquivalent gesetzte Größe C zwischen (1) und (2) ihren Wert ändern.

Überlagerte Variablen

Teilweise überlagerte Variablen werden bei der Erkennung gemeinsamer Teilausdrücke nicht berücksichtigt. **Vollständig** überlagerte Variablen können erfaßt werden. Eine Wertänderung einer Variablen bewirkt aber auch die Wertänderung der überlagerten Variablen.

Im folgenden Beispiel kann jedoch optimiert werden:

Beispiel:

Original	Wirkung der Optimierung

`X = (A+B)*D`
`C = A`
`Y = C+B`
	`%T1 = A+B`
	`X  = %T1*D`
	`C  = A`
	`Y  = %T1`

9.3.4 Indexrechnung

Damit der Compiler ein Feldelement ansprechen kann, muß er aus den angegebenen Indexwerten die Adresse des Feldelements bestimmen. Dazu expandiert der Compiler die Indexliste in eine Folge von arithmetischen Operationen.

Diese bei Indexexpansion erzeugte Arithmetik wird folgendermaßen optimiert:

- bei OPTIMIZE=0 wird nicht optimiert
- bei OPTIMIZE=1/2 wird optimiert, z.B. durch Auswerten gemeinsamer Teilausdrükke, durch Ausnützen des Distributivgesetzes, durch Zurückführen von Multiplikationen auf Additionen und durch Verlagerung von Befehlscode (siehe Abschnitt 9.3.5).
- bei OPTIMIZE=3/4 werden alle durch OPTIMIZE=1/2 aktivierten Optimierungsmaßnahmen durchgeführt. Zusätzlich wird versucht, die Adreßberechnung in drei Teile zu zerlegen:
  - einen konstanten Teil, der zur Compilezeit berechnet wird,
  - einen schleifeninvarianten Teil, der vor die Schleife verlagert wird
  - einen variablen Teil, der in der Schleife verbleibt.

Die Adresse A_s des Feldelements $A(s_1, ..., s_n)$ eines Feldes $A (p_1:q_1,p_2:q_2, ..., p_n:q_n)$ bezeichnet das erste Byte von $A (s_1,...,s_n)$. Man berechnet diese Adresse wie folgt:

Seien

$$d_k = q_k - p_k + 1 \qquad (k=1,...,n)$$

die Größen der einzelnen Dimensionen und

$$m_0 = 1,\ m_1 = d_1\ ,...,\ m_i = d_1 * d_2 *...* d_i \qquad (i=1,...,n)$$

die Größen der i-dimensionalen Untermatrizen.

Dann erhält man den Indexlistenwert I_s von A $(s_1,...,s_n)$ aus

$$I_s = (s_n-p_n)\ m_{n-1} + ... + (s_2-p_2)\ m_1 + (s_1-p_1)\ m_0 + 1,$$

und es gilt:

$$A_s = A + l(I_s-1),$$

wobei A die Adresse des Feldes und
l die Elementlänge bezeichnet.

Also

$$A_s = A + l(s_n-p_n)\ m_{n-1} + ... + l(s_2-p_2)\ m_1 + l(s_1-p_1)\ m_0$$

$$= A - l(p_1m_0 + ... + p_nm_{n-1}) + l(s_1m_0 + ... + s_nm_{n-1})$$

Insbesondere gilt:

$$A_0 = A(0, ...,0) = A-l(p_1m_0 + ... + p_nm_{n-1})$$

und daher

$$A_s = A_0 + l(s_1m_0 + ... + s_nm_{n-1})$$

Diese Formel bietet die besten Chancen für das Auftreten gemeinsamer Teilausdrücke und damit für die Optimierung. Sie wird daher in FOR1 zur Indexexpansion verwendet.

Die Adresse A_0 und die Multiplikatoren m_i werden bei konstanten Indexgrenzen zur Übersetzungszeit, sonst zur Laufzeit am Anfang der Programmeinheit berechnet.

Beispiel:

```
Feld                          m0 = 1
INTEGER*8 A(3,7,5)            m1 = 3
                              m2 = 21
```

Original		Wirkung der Indexexpansion	
	DO 1 I = M1,M2		DO 1 I = M1,M2
	DO 1 J = N1,N2		DO 1 J = N1,N2
	DO 1 K = L1,L2		DO 1 K = L1,L2
	.		%T1 = S1(I)
	.		%T2 = S2(J)*3
	.		%T3 = %T1+%T2
1	A(S1(I),S2(J),S3(K))=		%T4 = S3(K)*21
	F(I,J,K)		%T5 = %T3+%T4
			%T6 = %T5*8
		1	A (%T6) = F(I,J,K)

%T6 ist die Distanz zur Adresse, die für A(0,0,0) berechnet wurde.

9.3.5 Schleifenoptimierung

Der Bereich einer Schleife wird im allgemeinen mehrfach durchlaufen; eine Optimierung dieser Programmteile ist daher besonders wirksam.

Die Optimierungsstufen 1 und 3 führen Optimierungsmaßnahmen über den gesamten Schleifenbereich nur für "ideale" Schleifen durch. "Ideale" Schleifen sind Schleifen, deren Bereiche keine Verzweigungen aufweisen, d.h.

- keine GO TO-Anweisung,
- kein END-, ERR-Parameter in Ein-/Ausgabe-Anweisungen,
- keine IF-Anweisungen,
- keine Sprungmarkenparameter bei Unterprogrammaufruf.

Nicht ideale Schleifen haben Verzweigungen innerhalb des Schleifenbereichs. Der Schleifenbereich besteht aus mehreren Basisblöcken. Bei den Optimierungsstufen 1 und 3 wird die Schleifenoptimierung nur für jene Basisblöcke durchgeführt, bei denen zur Übersetzungszeit erkennbar ist, daß sie pro Schleifendurchlauf genau einmal ausgeführt werden. Bei der Optimierungsstufe 2 werden die Optimierungsmaßnahmen der Stufe 1, bei der Optimierungsstufe 4 werden die Optimierungssmaßnahmen der Stufe 3 auch in bedingt ausführbaren Schleifenteilen durchgeführt. Optimierungen in bedingt ausführbaren Schleifenteilen bedeuten in den meisten Fällen einen weiteren Laufzeitgewinn, können jedoch in speziellen Fällen zu Laufzeitverlusten oder ungewollten Unterbrechungen führen.

Beispiel 1:

```
   LOGICAL BREAK
   DO 10 I = 1,100
   IF(BREAK) GOTO 10
   A(K,I)=A(K,I)+B(L,I)
10 CONTINUE
```

Mit OPTIMIZE = 2/4 kommt in der Anweisung A(K,I)... lineare Indexfortschaltung zur Wirkung:

- Hat die logische Variable BREAK den Wert .FALSE., ergibt sich eine bedeutende Laufzeitverbeserung.
- Hat aber die Variable BREAK den Wert .TRUE., werden unnötige Hilfsberechnungen für die Indexfortschaltung ausgeführt, die mit OPTIMIZE = 0/1 oder OPTIMIZE = 3 entfallen würden, und es ergibt sich eine Laufzeitverschlechterung.

Beispiel 2:

```
   DO 10 I = 1,100
10 IF(A.GT.O)B = SQRT(A)
```

Bei den Optimierungsstufen 3 oder 4 werden INTRINSIC-Funktionen bei schleifeninvarianten Parametern vor der Schleife berechnet. Bei OPTIMIZE=4 wird die Schleifenoptimierung auch in dem bedingt ausgeführten Schleifenteil durchgeführt: SQRT(A) wird außerhalb der Schleife berechnet. Im Fall eines negativen Arguments A erfolgt eine Fehlerunterbrechung.

Schleifeninvarianz

Um den Begriff "schleifeninvariant" zu erklären, muß zuerst der Begriff "pseudokonstant" erklärt werden. Ein Datenelement ist pseudokonstant innerhalb einer Schleife, wenn folgende Bedingungen erfüllt sind:

- der Wert des Datenelements wird im Schleifenbereich nach einer ersten Zuweisung nicht mehr verändert oder es erfolgt keine Zuweisung.
- die erste Zuweisung, falls vorhanden, erfolgt vor der ersten Benützung des Wertes und es wird eine pseudokonstante Größe oder ein Ausdruck aus lauter pseudokonstanten Größen zugewiesen. Diese Zuweisung darf nur in einem Teil geschehen, der pro Schleifendurchlauf genau einmal ausgeführt wird.

Konstanten sind demnach auch pseudokonstant.

Daten, die in einem Schleifenbereich nicht pseudokonstant sind, sind auch in allen umfassenden Schleifen nicht pseudokonstant.

Beispiel:

```
  DO 1 I = 1,10
  IF(I.GT.3)M = 3       J   ist pseudokonstant, da der Wert von J in der
  %T1 = J                   Schleife nicht verändert wird
  %T2 = K               K   ist nicht pseudokonstant, da der Wert nach der
  K  = 5                    Benutzung zugewiesen wird
  L  = 5                L   ist pseudokonstant, der Wert wird vor der
  %T3 = L                   Benutzung zugewiesen
  %T4 = M               M   ist nicht pseudokonstant, da die Wertzuweisung
  N  = F(I)                 in einem bedingt ausgeführten Teil erfolgt
  %T5 = N               N   ist nicht pseudokonstant, da ein nicht
1 CONTINUE                  pseudokonstanter Wert zugewiesen wird
```

Eine Operation im Schleifenbereich ist schleifeninvariant, wenn folgende Bedingungen zutreffen:

bei einer arithmetischen Operation:	Alle Operanden sind pseudokonstant
bei einer Funktion:	keine Abnormal-FUNCTION; alle Parameter sind pseudokonstant
bei einer Wertzuweisung:	die zugewiesene Größe ist pseudokonstant; die Variable, der etwas zugewiesen wird, wird vorher nicht benutzt.

Operationen, die diese Bedingungen erfüllen, werden aus dem Bereich der Schleife nach außen verlagert.

Hinweis:

Da im Normalfall außer INTRINSIC-Funktionen alle Funktionen abnormal sind, sollte Gebrauch von der ABNORMAL-Anweisung gemacht werden (siehe Handbuch "FOR1-Beschreibung" [21]).

Verlagerung von Befehlscode

Schleifeninvariante Teile werden aus dem Schleifenbereich herausgezogen, um die Durchlaufzahlen dieser Teile zu verringern.

Beispiel:

STMT	Original	Wirkung der Optimierung
6	DO 1 I = 1,9,2	%T1 = F(K)
7	1 A(I) = F(K)+I**2	DO 1 I = 1,9,2
		1 A(I) = %T1+I**2

In der Decompilerliste (siehe Abschnitt 4.7.9) wird die Wirkung der Optimierung folgendermaßen dargestellt:

```
      ***** STATEMENT 6 (DO) *****************
    6       I=1
      ***** STATEMENT 7 (MOVED STMT) *********
    7       %T00010154=F(K)
      ***** STATEMENT 7 (MOVED STMT) *********
    7       %T00010330=4
    7       %I1=5
      ***** STATEMENT 7 (ASSIGNMENT) *********
    7   L3  CONTINUE
    7   1   %T00010198=I*I
    7       A(%T00010330/4)=%T00010154+%T00010198
      ***** STATEMENT 7 (INCR STMT) **********
    7       %T00010330=%T00010330+8
      ***** STATEMENT 7 (DOEND) **************
    7       I=I+2
    7       %I1 = %I1- 1
    7       IF (%I1 .NE. 0) GOTO L3
```

F darf keine abnormale Funktion sein, d.h. die Compileroption COMOPT OPT=(3,FUNCTION-SIDEEFFECT=NO) oder die FORTRAN-Anweisung ABNORMAL ohne Parameter muß angegeben werden.

Die herausgezogenen Teile werden vor Abarbeitung der Schleife ausgeführt. Ist zur Übersetzungszeit noch nicht bekannt, ob diese Schleife mindestens einmal durchgeführt wird, so dürfen die verlagerten Teile nur in Abhängigkeit vom Iterationszähler ausgeführt werden.

Der Iterationszähler kontrolliert die richtige Abarbeitung der Schleife. Bild 9-2 zeigt den Ablauf einer DO-Schleife (siehe Handbuch "FOR1-Beschreibung" [21]).

Bei einer Schachtelung von mehreren Schleifen beginnt der Optimierungsprozeß bei der innersten Schleife. Falls die herausgezogenen Teile unbedingt ausgeführt werden, können sie in den Optimierungsprozeß für die äußeren Schleifen miteinbezogen werden. Dadurch können schleifeninvariante Teile über mehrere geschachtelte Schleifen immer weiter nach außen verlagert werden.

Das Bild steht in dieser Online-PDF nicht mehr zur Verfügung.

Bild 9-2: Ablauf einer DO-Schleife bei Verlagerung von Befehlscode

Zurückführen auf weniger zeitaufwendige Operationen

In Schleifenbereichen ersetzt die Optimierung aufwendige durch weniger aufwendige Operationen: Exponentiationen werden auf Multiplikationen, Multiplikationen auf Additionen zurückgeführt.

Bei OPT=1 werden Multiplikationen, bei denen das Produkt aus einer Schleifenvariable und einer pseudokonstanten Größe gebildet wird, durch Additionen ersetzt. Bei OPT=3 werden auch Multiplikationen, bei denen programmierte Iterationsvariablen als Faktor auftreten, durch Additionen ersetzt (siehe Abschnitt 9.4.2).

Iterationsvariablen

Unter Iterationsvariablen versteht man Variablen, deren Wert bei jedem Durchlauf des Schleifenbereichs um einen pseudokonstanten Wert geändert wird.

Die Laufvariable einer Schleife ist also stets eine Iterationsvariable. Das Zurückführen auf eine Addition geschieht durch ein Aufspalten der Operation in einen Initialisierungsteil und in einen Inkrementierungsteil.

Der Initialisierungsteil ist schleifeninvariant und wird vor die Schleife gestellt (siehe oben, Verlagerung von Befehlscode), der Inkrementierungsteil wird in einen eigenen Inkrementierungsblock gestellt. Bild 9-3 zeigt den Ablauf einer DO-Schleife bei Zurückführung von Multiplikationen auf Additionen (siehe Handbuch "FOR1-Beschreibung" [21]).

Das Bild steht in dieser Online-PDF nicht mehr zur Verfügung.

Bild 9-3: Ablauf einer DO-Schleife bei Zurückführung von Multiplikationen auf Additionen

Beispiel:

```
  PROGRAM OPT
  ABNORMAL
  INTEGER I,A,B,X,F
  READ (*,*)B
  DO 1 I = 1,9,2
  A=B*I
1 X=F(A)
  WRITE (*,*) X,I
  END
```

F ist durch die ABNORMAL-Anweisung ohne Parameter als normale Funktion definiert. Wenn F eine abnormale Funktion wäre, dann würde die Multiplikation B*I bei OPT=1/2 nicht auf eine Addition zurückgeführt. Dies wäre nur bei OPT=3/4 der Fall. Die Schleifeninitialisierung I=1 im Beispiel wird nur bei den Optimierungsstufen 3 und 4 eliminiert.

Im Decompilerlisting wird (mit OPT=3) die Wirkung der Optimierung auf die Schleife sichtbar:

```
    | ***** STATEMENT 5 (DO) *****************
  5 | L5    I=1
    | ***** STATEMENT 6 (MOVED STMT) *********
  6 |       %T00010220=B
  6 |       %T00010264=2*B
  6 |       I=11
  6 |       %I1=5
    | ***** STATEMENT 6 (ASSIGNMENT) *********
  6 | L3    A=%T00010220
    | ***** STATEMENT 7 (ASSIGNMENT) *********
  7 | 1     X=F(A)
    | ***** STATEMENT 6 (INCR STMT) **********
  6 |       %T00010220=%T00010220+%T00010264
    | ***** STATEMENT 7 (DOEND) **************
  7 |       %I1 = %I1- 1
  7 |       IF (%I1 .NE. 0) GOTO L3
```

Bild 9-4: Ablauf der DO-Schleife vor und nach der Optimierung (OPT=1)

Eliminierung "unnötiger" Fortschaltungen von Iterationsvariablen

Tritt eine Iterationsvariable innerhalb eines Schleifenbereichs außer in der Iterationsanweisung nicht mehr auf, so ist ihre laufende Fortschaltung nicht notwendig. Die laufende Fortschaltung der Iterationsvariablen kann weggelassen werden, wenn man der Iterationsvariablen vor Abarbeitung der Schleife ihren Endwert zuweist und wenn die Iterationsvariable nach der Schleife nicht mehr verwendet wird. Der "Endwert" der Iterationsvariablen ist jener Wert, den sie bei der Einzelfortschaltung nach Abarbeitung der Schleife aufgewiesen hätte.

Für Schleifen, die einen Sprung aus dem Schleifenbereich heraus aufweisen, kann dieser "Endwert" nicht zugewiesen und die Einzelfortschaltung nur dann eliminiert werden, wenn die Iterationsvariable nach der Schleife nicht mehr verwendet wird.

"Unnötige" Fortschaltungen entstehen häufig durch das Zurückführen von Multiplikationen auf Additionen.

Beispiel:
Im Beispiel in Bild 9-4 ist die Anweisung I = I+2 gestrichen und durch die "Endwert"-Zuweisung I = 11 ersetzt.

9.3.6 Globale Registerzuordnung

Grundgedanke der globalen Optimierung ist die Einsparung von Speicherzugriffen. Häufig verwendete Variablen und vom Compiler erzeugte temporäre Hilfsgrößen sollen in Registern belassen werden.

Bei eingeschalteter Optimierung werden für die einzelnen Größen Untersuchungen über die Anzahl der Zugriffe durchgeführt. Außerdem wird der Zeitgewinn bewertet, den man erhält, wenn eine spezielle Größe in einem Register steht. Diese Informationen werden dann bei der Registerzuordnung ausgewertet.

9.4 Beispiele zur Optimierung

9.4.1 Wirkung der Optimierung auf eine Programmschleife

```
  DO     SEG     STMT     I     LINE            SOURCE-TEXT
-------------------------------------------------------------------
        1/1        1              1          PROGRAM OPT
          2        2              2          DO 1 I=1,5
   1      2        3              3          L=7
   1      2        4              4          M=M+N*L
   1      2        5              5          K=I*3+L*4
   1      3        6              6        1 N=N*7+K
          3        7              7          END
```

Die Ausschnitte aus den Objektlisten in Bild 9-5 zeigen die Wirkung von OPTIMIZE=0 und OPTIMIZE=1 auf die obige Programmschleife:

(1) Verlagern der konstanten Zuweisung aus dem Iterationsbereich nach (9).

(2) Ersetzen der Variablen L durch die Konstante 7 (13).

(3) Zurückführen der Multiplikation auf eine Addition (12).

(4) Berechnen des konstanten Ausdrucks (L*4) zur Übersetzungszeit

(5) Berechnen des Anfangswerts von K zur Übersetzungszeit (10).

(6) Wiederverwenden des gemeinsamen Teilausdrucks N*7 aus Anweisung 4.

(7) Eliminieren der Fortschaltung der Iterationsvariablen. Zuweisen des Endwertes außerhalb der Schleife (8).

(11) Register-Ladepunkte außerhalb der Schleife.

(13),(14) Verwenden von Maschinenbefehlen vom Typ RR.

(15) Register-Speicherpunkte außerhalb der Schleife.

Das Bild steht in dieser Online-PDF nicht mehr zur Verfügung.

Bild 9-5: Wirkung der Optimierung auf eine Programmschleife

9.4.2 Unterschiede zwischen den Optimierungsstufen 1 und 3

Original	vor der Optimierung	nach OPTIMIZE=1	nach OPTIMIZE=3
`  SUBROUTINE OPTIM (A,` `1END,INCR)`			
`  INTEGER A(100),END,` `1INCR,IV`			

`  IV=1`	`IV=1`
	`%V1=END`
	`J=2`
	`%I1=%V1-1`
	`IF %I1≤0 GOTO L13` } (1)
		`IV=1`
		`%V1=END` (3)
		`J=2`
		`%I1=END-1`
		`IF %I1≤0 GOTO L13`
			`J=2`
			`%I1=END-1`
			`IF %I1≤0 GOTO L13`
			`%T4=4`
			`%T5=INCR*4` } (6)

`  DO 10 J=2,END`	`L3  %T3=IV`	`L3  %T3=IV`	
`  A (IV)=J`	`%T1=%T3*4` `A(%T1)=J`	`%T1=IV*4` (4) `A(%T1)=J`	`L3  A(%T4)=J`
`  IV=IV+INCR`	`IV=IV+INCR` `J=J+1`	`IV=IV+INCR` (5) `J=J+1`	`%T4=%T4+%T5` (7) `J=J+1`

`10 CONTINUE`	`%I1=%I1-1` (2)
	`IF(%I1.NE.0)`
	`.GOTO L3`
		`%I1=%I1-1`
		`IF(%I1.NE.0)`
		`.GOTO L3`
			`%I1=%I1-1`
			`IF(%I1.NE.0)`
			`.GOTO L3`

`  END`	`L13 END`	`L13 END`	`L13 END`

Tab. 9-2: Unterschiede zwischen OPTIMIZE=1 und OPTIMIZE=3

Am Beispiel des Unterprogramms OPTIM werden einige Unterschiede in der Wirkung der Optimierungsstufen 1 und 3 deutlich. Größen mit einem %-Zeichen als erstem Zeichen beziehen sich auf die interne Darstellung. Im Initialisierungsteil (1) der Schleife werden die Schleifenparameter ausgewertet und der Iterationszähler %I1 getestet. Bei (2) wird der Iterationszähler um 1 erniedrigt und falls %I1>0 wird nach L3 verzweigt.

Ersetzen von aufwendigen Operationen durch weniger aufwendige

Die Optimierungsstufen 1 und 3 unterscheiden sich in der Behandlung von Iterationsvariablen. Iterationsvariablen sind Integer-Variablen, die ihren Wert in einer Schleife linear ändern. Iterationsvariablen sind im vorliegenden Beispiel die Schleifendurchlaufvariable J und die programmierte Iterationsvariable IV.

Mit OPT=1 werden nur Schleifendurchlaufvariablen als Iterationsvariablen erkannt. Deshalb werden nur solche Multiplikationen vereinfacht, in denen Schleifendurchlaufvariablen als Faktor auftreten. Durch die Indexexpansion (vgl. Abschnitt 9.3.4) entsteht die Multiplikation (4) mit der programmierten Iterationsvariablen IV. Diese Multiplikation wird bei OPT=1 nicht vereinfacht.

Mit OPT=3 wird dagegen auch die Multiplikation (4) mit der programmierten Iterationsvariablen IV vereinfacht. Die schleifeninvarianten Operationen (6) werden vor die Schleife gestellt; die Multplikation (4) wird auf eine Addition (7) zurückgeführt. Durch diese Vereinfachung wird der Term (5) überflüssig und durch die "dead code elimination" entfernt.

Berechnen konstanter Ausdrücke zur Übersetzungszeit

Nach der Optimierung der Schleife besteht der Programmteil vor der Schleife aus dem ersten Basisblock (1) und dem zweiten Basisblock

```
%T4=IV*4
%T5=INCR*4
```

Dieser Programmteil vor der Schleife wird mit OPT=3 weiter optimiert. Im ersten und im zweiten Basisblock hat IV den Wert 1. MIT OPT=1 wird die Optimierung nur innerhalb jedes Basisblocks durchgeführt. Bei der Optimierung des zweiten Basisblocks wird deshalb nicht berücksichtigt, daß IV den Wert 1 hat: Es wird nicht optimiert.

Mit OPT=3 werden dagegen die Werte über die ganze Schleife verfolgt. Dadurch kann `%T4=IV*4` zu `%T4=4` vereinfacht werden.

Eliminieren von überflüssigem Code

In beiden Optimierungsstufen 1 und 3 kann durch die Technik der Wertverfolgung im Ausdruck

```
%I1=%V1-1
```

%V1 durch END ersetzt werden:

```
%I1=END-1
```

Die Anweisung (3) %V1=END wird dadurch überflüssig. Diese überflüssige Anweisung wird jedoch nur mit OPT=3 durch die Optimierungstechnik der "dead code elimination" entfernt.

10 Programmierhinweise

10.1 Hinweise zu einzelnen FOR1-Sprachelementen

Siehe auch Handbuch "FOR1-Beschreibung" [21].

WAIT-Anweisung

Die WAIT-Anweisung ist so realisiert, daß lediglich die angegebenen Parameter mit den entsprechenden Werten versorgt werden. Die WAIT-Anweisung bewirkt aber keine Suspendierung des Prozesses, da bereits bei den asynchronen READ- und WRITE- Anweisungen die Beendigung der Datenübertragung abgewartet wird.

SAVE-Anweisung

Die SAVE-Anweisung wird akzeptiert, hat aber keine zusätzliche Wirkung. Bei FOR1 behalten grundsätzlich alle Datenelemente eines FUNCTION- oder SUBROUTINE-Unterprogramms nach Verlassen des Unterprogramms ihren Wert.

FIND-Anweisung

Die FIND-Anweisung wird insofern bedient, als die angegebenen Parameter mit den entsprechenden Werten versorgt und die vorgesehenen Prüfungen durchgeführt werden. Die eigentliche Positionierung der angesprochenen Datei findet jedoch stets in den READ- oder WRITE- Anweisungen statt.

ENCODE-, DECODE-Anweisungen

Die Anweisungen werden aus Kompatibilitätsgründen weiterhin bedient. Es wird jedoch empfohlen, die von FOR1 zur Verfügung gestellten Sprachmittel der Ein-/Ausgabe auf interne Dateien für die Datenübertragung im Speicher zu verwenden.

"Gefährliche" Sprachelemente

Im folgenden wird auf Sprachelemente und Konstruktionen hingewiesen, die nicht ohne schwerwiegende Gründe angewendet werden sollten. Sie vermindern die Transparenz der Programmlogik und, da sie erfahrungsgemäß häufig die Quelle versteckter Programmierfehler sind, die Sicherheit eines Programms. Außerdem beeinträchtigen oder vermindern sie die Optimierung von Teilen des Programms (siehe Kap. 9).

- Erweiterter Bereich von DO-Schleifen
  Der erweiterte Bereich kann durch den Ablauf eines Unterprogramms oder durch Ausprogrammierung der betreffenden Anweisungen im Schleifenbereich ersetzt werden. Falls der erweiterte Bereich dazu kopiert werden müßte, macht man aus ihm zweckmäßigerweise ein %INCLUDE-Element.
- Doppelbenutzung von ASSIGN-Variablen
  ASSIGN-Variablen sollen nur für ihren eigentlichen Zweck als Träger von Anweisungsnummern verwendet werden und nicht auch für arithmetische Operationen. Aus der Mischanwendung resultierende Fehler sind i.a. schwer zu finden.
- Überlagerung von Datenelementen unterschiedlicher Länge.

10.2 Vom Fortran90-Compiler nicht mehr unterstützte FOR1-Erweiterungen

Fast alle FOR1-Erweiterungen werden auch vom Fortran90-Compiler unterstützt. Es gibt jedoch einige wenige Ausnahmen. Der Anwender hat die Möglichkeit, durch Angabe der Compileroption FORTRAN90-CHECK = YES (siehe 4.1.2.8) ein Quellprogramm daraufhin prüfen zu lassen, ob es FOR1-Erweiterungen enthält, die vom Fortran90-Compiler nicht mehr unterstützt werden.

Folgender Abschnitt enthält eine Aufstellung dieser zum Fortran90-Compiler inkompatiblen Sprachmittel. Außerdem ist jeweils die Meldung angegeben, mit der beim Fortran90-Check ein Auftreten solcher Sprachelemente markiert wird.

Zyklische Bereichsangabe bei der IMPLICIT-Anweisung

Im FOR1-Sprachumfang ist es erlaubt, in einer IMPLICIT-Anweisung einen Buchstabenbereich zyklisch zu definieren. Möglich ist beispielsweise:

```
IMPLICIT INTEGER*2 (Y-D)
```

Dies wird durch den Fortran90-Compiler nicht mehr unterstützt werden. Beim Fortran90-Check markiert der Compiler IMPLICIT-Anweisungen mit zyklischen Bereichsangaben mit der Meldung:

```
FA300  FORTRAN90 DEVIATION:  CYCLIC RANGE-SPECIFICATION
```

Komplexe Ausdrücke als Untergrenze, Obergrenze oder Schrittweite in Schleifen:

Im FOR1-Sprachumfang sind Ausdrücke vom Datentyp COMPLEX als Untergrenze, Obergrenze und Schrittweite in DO-Schleifen und impliziten DO-Schleifen zulässig. Der Fortran90-Compiler wird dies nicht mehr unterstützen. Werden komplexe Ausdrücke zu solchen Zwecken verwendet, markiert der Compiler beim Fortran90-Check die entsprechenden Stellen mit einer der folgenden Warnungen:

```
SA250  FORTRAN90 DEVIATION:  LOWER BOUND OF TYPE COMPLEX
SA251  FORTRAN90 DEVIATION:  UPPER BOUND OF TYPE COMPLEX
SA252  FORTRAN90 DEVIATION:  STEP SIZE OF TYPE COMPLEX
```

Schachtelung von logischen IF-Anweisungen

Der FOR1-Sprachumfang erlaubt geschachtelte logische IF-Anweisungen, d.h. eine logische IF-Anweisung kann wiederum eine logische IF-Anweisung enthalten.
Dies wird vom Fortran90-Compiler nicht mehr unterstützt werden. Beim Fortran90-Check wird eine entsprechende Konstruktion mit folgender Meldung markiert:

```
FA301  FORTRAN90 DEVIATION:  NESTED BOOLEAN IF-STATEMENTS
```

Freie Anweisungsreihenfolge

Der FOR1-Compiler erlaubt eine vollkommen freie Abfolge von Deklarationen und ausführbaren Anweisungen. Für den Fortran90-Compiler gilt dies nur noch eingeschränkt: Deklarationen müssen vor der ersten Verwendung erfolgen. Da es für den FOR1-Compiler sehr aufwendig wäre, dies jeweils zu überprüfen, wird beim Fortran90-Check eine Warnung bereits dann ausgegeben, wenn im Programm einer Deklarationsanweisung ausführbare Anweisungen vorangehen:

```
FA302  FORTRAN90 DEVIATION:  POSSIBLE USE BEFORE DECLARATION
```

Temporäres Ändern von Quellprogrammen

Das temporäre Ändern von Quellprogrammen mit der Compileroption UPD wird vom Fortran90-Compiler nicht mehr unterstützt werden. Die Verwendung der UPD-Option wird beim Fortran90-Check mit folgender Warnung markiert:

```
MA34  FORTRAN90 DEVIATION: UPD
```

Im Programm auftretende *DELETE-Anweisungen werden dagegen nicht markiert.

Fehlende END-Anweisung

Nach dem FOR1-Sprachumfang ist es erlaubt, die END-Anweisung am Ende einer Programmeinheit wegzulassen. Der Fortran90-Compiler wird dies nur noch am Ende der gesamten Quellprogrammdatei tolerieren. Fehlt eine END-Anweisung, gibt der Compiler beim Fortran90-Check folgende Meldung aus:

```
FA304  FORTRAN90 DEVIATION:  END-STATEMENT MISSING
```

RETURN-Anweisungen im Hauptprogramm

Der FOR1 ersetzt RETURN-Anweisungen in einem Hauptprogramm durch STOP-Anweisungen. Der Fortran90-Compiler wird dies nicht mehr unterstützen. Tritt in einem Hauptprogramm eine RETURN-Anweisungen auf, wird beim Fortran90-Check folgende Meldung ausgegeben:

```
FA305  FORTRAN90 DEVIATION: RETURN-STATEMENT IN MAIN-PROGRAM
```

Endemarkierung bei LINE-END-Kommentaren

Auch der Fortran90-Compiler wird LINE-END-Kommentare unterstützen. Allerdings muß dabei beachtet werden, daß der Fortran90-Zeichensatz vier Zeichen mehr umfaßt als der FOR1-Zeichensatz. Diese Zeichen sind:

- Anführungszeichen (")
- Semikolon (;)
- Größer-als (>)
- Kleiner-als (<)

Hat man eines dieser Zeichen als Endemarkierung angegeben, so wird beim Fortran90-Check folgende Meldung ausgebeben:

```
MA36  FORTRAN90 DEVIATION:  FORTRAN90 CHARACTER AS LINEEND
```

FPOOL

Der Fortran90-Compiler wird FPOOL nicht mehr unterstützen, da für die Überprüfung von Aufrufschnittstellen in Fortran90 unmittelbar Sprachmittel zur Verfügung stehen. Die %FPOOL-Anweisungen werden zwar syntaktisch akzeptiert, aber ihre Semantik wird ignoriert. Treten in einem Quellprogramm %FPOOL-Anweisungen auf, so werden sie beim Fortran90-Check mit einer der folgenden Meldungen markiert:

```
FA308  FORTRAN90 DEVIATION:  %FPOOL COMPILER DIRECTED STATEMENT
                             WILL BE IGNORED
FA309  FORTRAN90 DEVIATION:  %NOFPOOL COMPILER DIRECTED STATEMENT
                             WILL BE IGNORED
```

Bei Verwendung der Compileroption FPOOL erfolgt die Warnung:

```
MA35  FORTRAN90 DEVIATION:  FPOOL
```

Die Funktionen des zentralen FPOOL können auch vom Fortran90-Compiler noch genutzt werden, jedoch ohne Schnittstellenüberprüfung durch FPOOL.

Testanweisungen

Im FOR1-Sprachumfang gibt es Testanweisungen, die im Quellprogramm angegeben werden.
Im BS2000 steht die symbolische Testhilfe AID zur Verfügung. AID umfaßt alle wesentlichen Funktionen der Testanweisungen und macht diese somit entbehrlich. Der Fortran90-Compiler wird Testanweisungen ignorieren. Werden in einem Quellprogramm Testanweisungen verwendet, so gibt der Compiler beim Fortran90-Check eine der folgenden Meldungen aus:

```
FA310  FORTRAN90 DEVIATION:  %CALLTRACE COMPILER DIRECTED STATEMENT
                             WILL BE IGNORED
FA311  FORTRAN90 DEVIATION:  %CHECK COMPILER DIRECTED STATEMENT
                             WILL BE IGNORED
FA312  FORTRAN90 DEVIATION:  %COUNT COMPILER DIRECTED STATEMENT
                             WILL BE IGNORED
FA313  FORTRAN90 DEVIATION:  %DISPLAY COMPILER DIRECTED STATEMENT
                             WILL BE IGNORED
FA314  FORTRAN90 DEVIATION:  %FULLTRACE COMPILER DIRECTED STATEMENT
                             WILL BE IGNORED
FA315  FORTRAN90 DEVIATION:  %JMPTRACE COMPILER DIRECTED STATEMENT
                             WILL BE IGNORED
```

Parameter der ENCODE- und DECODE-Anweisung

ENCODE- und DECODE-Anweisungen haben im FOR1-Sprachumfang folgendes Format:

```
ENCODE (intsatzlänge, format, charname [,intname])   bzw.
DECODE (intsatzlänge, format, charname [,intname])
```

Der Parameter *intname* ist nur aus Kompatibilitätsgründen vorhanden und wird vom FOR1-Compiler ignoriert. Der Fortran90-Compiler wird diesen Parameter auch syntaktisch nicht mehr akzeptieren. Tritt der Parameter in einem Quellprogramm auf, so wird beim Fortran90-Check folgende Meldung ausgegeben:

```
SA253  FORTRAN90 DEVIATION:  FORTH PARAMETER OF ENCODE/DECODE
```

Überlappungen bei der CHARACTER-Zuweisung

Während es nach ANS FORTRAN 77 unzulässig ist, daß sich bei einer CHARACTER-Wertzuweisung linke und rechte Seite überlappen, ist dies sowohl nach dem Fortran90-Standard als auch beim FOR1-Compiler möglich. Die Zuweisung wird aber jeweils unterschiedlich vollzogen.

Beispiel:

```
CHARACTER*5 CHAR
CHAR      = 'ABCDE'
CHAR(2:5) = CHAR(1:4)
```

Nach dem Fortran90-Standard muß CHAR nach der Zuweisung den Wert 'AABCD' enthalten. Bei FOR1 hat sie jedoch den Wert 'AAAAA'.

Wenn bei einer CHARACTER-Zuweisung eine Überlappung der beiden Operanden möglich ist, wird beim Fortran90-Check folgende Meldung ausgegeben:

```
SA254 FORTRAN90 DEVIATION:  DIFFERENT SEMANTICS BY OVERLAPPING FROM
                            SOURCE AND TARGET
```

10.3 Hochgenaue mathematische INTRINSIC-Funktionen

Ab der FOR1-Version 2.2A werden verbesserte Routinen für folgende INTRINSIC-Funktionen vom Typ DOUBLE PRECISION (bzw. REAL*8) angeboten:

- DSIN
- DLOG
- DSQRT
- DTAN
- DASIN
- DLOG2
- DCOS
- DEXP
- DATAN
- DCOTAN
- DACOS
- DLOG10

Die verbesserten Routinen kommen selbstverständlich auch dann zum Einsatz, wenn die generischen Namen obiger Funktionen mit Argumenten vom Datentyp DOUBLE PRECISION (bzw. REAL*8) angewendet werden.

Die neuen Routinen sind hochgenau, d.h.:

a) zwischen dem errechneten Funktionsergebnis und dem exakten Ergebnis existiert keine darstellbare doppelt-genaue Gleitpunktzahl. (Es kann aber eine doppelt-genaue Gleitpunktzahl geben, die *näher* am exakten Ergebnis liegt.)

b) exakt darstellbare Funktionswerte werden exakt berechnet.

Trotz dieser erheblich gesteigerten Genauigkeit bieten die neuen Routinen eine mindestens ebensogute Performance wie die entsprechenden alten FOR1-Routinen.

Die neuen Routinen werden über dieselbe Schnittstelle angesprochen wie die bisherigen.

Die neuen Routinen haben jedoch intern andere Namen wie die entsprechenden alten Routinen. Bei Programmsystemen, die neben neuen FOR1-V2.2A-Programmen auch FOR1-Programme enthalten, die mit einer FOR1-Version < 2.2A übersetzt wurden, kann es vorkommen, daß ein Unterprogramm mit der alten, ein anderes mit der neuen Routine rechnet.

Da die neuen Routinen andere Funktionswerte liefern als die alten FOR1-Routinen, kann es zu Inkompatibilitäten kommen - beispielsweise bei Testpaketen, deren Korrektheit anhand von Funktionsergebnissen kontrolliert wird. Deshalb wird, falls das Programm Aufrufe der neuen Routinen enthält, beim Programmstart folgende Meldung ausgegeben:

```
IMPROVED MATHEMATICAL ACCURACY
```

Bei der Versorgung von Jobvariablen, die den Ablauf überwachen, wird diese Meldung nicht berücksichtigt.

10.4 Gleit- und Fixpunktarithmetik

Gleitpunktarithmetik

Die Abfrage auf Gleichheit von Werten hat bei Datenelementen vom Typ REAL und COMPLEX eventuell nicht den gewünschten Effekt, weil die Fortpflanzung von Rundungsfehlern in verschiedenen Rechenvorgängen zu verschiedenen Approximationen desselben arithmetischen Resultats führen kann, so daß Bit-Gleichheit nicht mehr gegeben sein muß. Es empfiehlt sich, einen Nicht-Standard-Vergleichsoperator EQ mittels einer Formelfunktion zu definieren, etwa

```
LOGICAL*1 EQ
PARAMETER(EPSILON = 0.000001)
EQ(A,B)=ABS(A-B).LE.EPSILON
```

Fixpunktarithmetik

Ein Überlauf wird nur nach Additionen oder Subtraktionen mit Datenelementen vom Typ INTEGER*4 erkannt (Fehlermeldung: FIXED POINT OVERFLOW). Überläufe nach Multiplikationen mit INTEGER*4-Datenelementen, Über- bzw. Unterläufe nach arithmetischen Operationen mit INTEGER*1- und INTEGER*2-Datenelementen werden nicht erkannt. Die höherwertigen Stellen gehen verloren und es wird mit falschen Werten weitergerechnet. Das Vorzeichen-Bit eines übergelaufenen Datenelements wird überschrieben. Die Verzweigung einer arithmetischen IF-Anweisung, in denen der Wert dieses Datenelements getestet wird, liefert keine Rückschlüsse mehr auf den Wert des Datenelements.

10.5 Ausrichtung der Datenelemente

Bei FOR1 werden die Datenelemente ihrem Typ entsprechend auf Byte-, Halbwort-, Wort- oder Doppelwortgrenzen ausgerichtet. Tabelle 10-1 zeigt die Regeln für die Ausrichtung der einzelnen Datentypen.

Datentyp	Ausrichtung auf
INTEGER*1	Byte
INTEGER*2	Halbwort
INTEGER*4	Wort
INTEGER*8	Doppelwort
REAL*4	Wort
REAL*8	Doppelwort
REAL*16	Doppelwort
COMPLEX*8	Wort
COMPLEX*16	Doppelwort
COMPLEX*32	Doppelwort
CHARACTER*N	Byte
LOGICAL*1	Byte
LOGICAL*4	Wort

Tab. 10-1: Ausrichtung der Datenelemente

Bei Datenelementen vom Typ COMPLEX bezieht sich die Ausrichtung einzeln auf den Realteil und den Imaginärteil.

Bei Feldern wird das erste Element des Feldes den Regeln entsprechend ausgerichtet.

Bei COMMON-Bereichen und Bereichen, die durch Überlagerung mit der EQUIVALENCE-Anweisung gebildet sind, muß der Anwender diese Ausrichtung der Datenelemente beachten. Der Beginn eines COMMON-Bereichs oder eines überlagerten Bereichs ist immer auf eine Doppelwortgrenze ausgerichtet. Bei COMMON-Bereichen können durch Ausrichtung der einzelnen Datenelemente des COMMON-Bereichs "Lükken" entstehen.

Das Auftreten von Lücken in COMMON-Bereichen kann durch Anordnen der Datenelemente nach absteigender Längenspezifikation verhindert werden (vgl. Beispiel 1). Da CHARACTER-Datenelemente auf Byte-Grenze ausgerichtet werden, empfiehlt es sich, diese ans Ende des COMMON-Bereichs zu setzen.

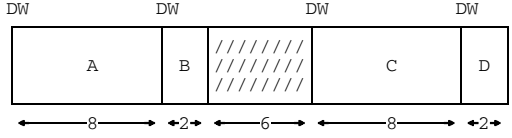
Bei der Überlagerung von Speicherbereichen (EQUIVALENCE-Anweisung) können Ausrichtungsfehler auftreten (vgl. Beispiel 2).

Beispiel 1:

a) COMMON-Bereich mit Lücke:

```
REAL*8 A,C
INTEGER*2 B,D
COMMON/LIST/A, B, C, D
```

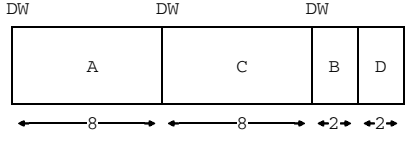
Anordnung im COMMON-Bereich (DW = Doppelwortgrenze):



b) COMMON-Bereich mit verbesserter Anordnung:

```
REAL*8 A,C
INTEGER*2 B,D
COMMON/LIST/A, C, B, D
```

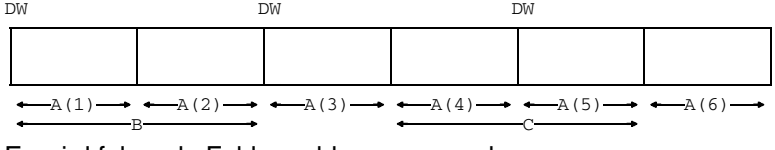
Anordnung im COMMON-Bereich (DW = Doppelwortgrenze):



Beispiel 2:

```
REAL*4 A(6)
REAL*8 B,C
EQUIVALENCE (A(1),B),(A(4),C)
```

Diese Anwendungsfolge liefert einen Ausrichtungsfehler, da C nicht auf Doppelwortgrenze (DW) ausgerichtet werden kann:



Es wird folgende Fehlermeldung ausgegeben:

```
ERROR  (SA124)  ENTITY C MISALIGNED DUE TO EQUIVALENCE
```

10.6 Dynamische Speicherplatzbeschaffung für Felder

Ab der FOR1-Version 2.0A kann der Anwender den erweiterten Adreßraum von XS-Anlagen (XS für eXtended System) nutzen, der eine Verarbeitung großer Datenmengen durch FORTRAN-Programme ermöglicht.

Auf diesen Anlagen können dynamische Felder verwendet werden. Bei dynamischen Feldern erfolgt die Speicherplatzzuweisung erst zur Laufzeit des Programms. Der Anwender muß für ein bestimmtes Feld den benötigten Speicherplatz durch Aufruf des Unterprogramms ALLOC anfordern. Der bereitgestellte Speicherplatz wird explizit durch Aufruf des Unterprogramms DEALLOC oder automatisch bei Programmbeendigung zurückgegeben.

ALLOC und DEALLOC sind vorgefertigte Unterprogramme im Laufzeitsystem, die mit CALL aufgerufen werden.

Die bisher verfügbaren Unterprogramme zur dynamischen Speicherverwaltung DYNARA und DYNAST werden vom Compiler ab der FOR1-Version 2.0A nicht mehr unterstützt. Programme, die Aufrufe von DYNARA oder DYNAST enthalten, laufen mit einem Laufzeitsystem ≥2.0A noch ab.

Um den erweiterten Adreßraum oberhalb 16 Megabyte nutzen zu können, muß ein Programm auf einer XS-fähigen Anlage ablaufen. Die dynamische Speicherplatzzuweisung durch die Unterprogramme ALLOC und DEALLOC ist jedoch auch für Benutzer von nicht-XS-fähigen Anlagen sinnvoll. Nicht dynamisch angelegte Felder werden nämlich beim Laden des Lademoduls angelegt und erst wieder beim Entladen freigegeben. Der Speicherplatz eines dynamisch angelegten Feldes wird dagegen nur zwischen den Aufrufen von ALLOC und DEALLOC (bzw. dem Programmende) benötigt, d.h. durch diese Unterprogramme kann auch der Benutzer einer nicht-XS-fähigen Anlage den Adreßraum entlasten.

Eine Voraussetzung für die Nutzung des erweiterten Adreßraums oberhalb 16 Megabyte ist, daß der betreffende Bindemodul ein XS-Modul ist. (Ab FOR1 Version 2.2A werden grundsätzlich XS-Moduln erzeugt. Bei FOR1-Versionen < 2.2A mußte zur Erzeugung von XS-Moduln mit EXTENDED-SYSTEM=YES übersetzt werden.)

Die Eigenschaften eines Programms werden vom Binder-/Ladersystem ausgewertet bzw. können noch vom Binder-/Ladersystem sowie durch die Laufzeitoption START=XS geändert werden. Das Zusammenwirken der möglichen Angaben, nach denen entweder ein XS-Programm oberhalb 16 Megabyte oder ein XS-Programm bzw. NXS (Nicht-XS)-Programm unterhalb 16 Megabyte abläuft, wird in Anhang A.7 dargestellt.

Für die dynamische Speicherplatzbeschaffung stehen folgende Sprachmittel zur Verfügung:

- das Format (:[,:] [,...])
  für die Indexgrenzenliste eines dynamisch angelegten Feldes;
- das Unterprogramm ALLOC, um Speicherplatz für ein dynamisch angelegtes Feld anzufordern;
- das Unterprogramm DEALLOC, um diesen Speicherplatz wieder freizugeben;
- das Unterprogramm GETSHAPE, um die aktuellen Indexgrenzen eines dynamisch angelegten Feldes abzufragen.

10.6.1 Vereinbaren von dynamischen Feldern

Dynamisch angelegte Felder müssen analog zu statischen Feldern im FORTRAN-Programm vereinbart werden. Die bei der Vereinbarung noch offene Ober- und Untergrenze einer Dimension wird in der Indexgrenzenliste nur in der Form eines Doppelpunktes angegeben. Ein dynamisch angelegtes Feld wird somit in einer Typ-, DIMENSION- oder COMMON-Anweisung in folgender Form vereinbart:

```
feldname (:[,:] [,...])
```

Die Anzahl der Doppelpunkte in der Indexgrenzenliste ist gleich der Anzahl der Dimensionen des Feldes. Es sind maximal 7 Dimensionen zulässig.

10.6.2 Zuweisen von Speicherplatz (CALL ALLOC)

Vor der ersten Bezugnahme auf ein dynamisch angelegtes Feld (bzw. Feldelement) muß der benötigte Speicherplatz durch einen Aufruf des Unterprogramms ALLOC zugewiesen werden:

```
CALL ALLOC (feldname,u1,o1[,u2,o2][,...][,un,on][,{'NXS'}])
                                                  {'ANY'}
```

feldname
: Name des in der Typ-, DIMENSION- oder COMMON-Anweisung als dynamisch vereinbarten Feldes.

u_i,o_i arithmetische Ausdrücke vom Typ INTEGER*4. u_i ist der kleinste, o_i der größte Index der i-ten Dimension ($1 \le i \le 7$) des dynamischen Feldes *feldname*. Die Anzahl der Dimensionen n ($1 \le n \le 7$) muß gleich der Anzahl der Dimensionen in der zugehörigen Typ-, DIMENSION- oder COMMON-Anweisung sein.

'NXS' Der Speicherplatz für das Feld feldname wird unterhalb 16 Megabyte angelegt.

'ANY' Der Speicherplatz für das Feld *feldname* wird in Abhängigkeit vom aktuellen Maschinenadreßmodus (siehe Anhang A.7) angelegt:

- Ist der aktuelle Maschinenadreßmodus 24, dann wird der Speicherplatz für das dynamisch angelegte Feld unterhalb 16 Megabyte angelegt.
- Ist der aktuelle Maschinenadreßmodus 31, dann wird versucht, den Speicherplatz für das dynamisch angelegte Feld oberhalb 16 Megabyte anzulegen. Falls das nicht gelingt, wird der Speicherplatz unterhalb 16 Megabyte angelegt.

Verhalten im Fehlerfall

Anzahl und Typ der Parameter im Aufruf von ALLOC werden zur Übersetzungszeit auf ihre Übereinstimmung mit der Vereinbarung als dynamisches Feld geprüft.

Folgt auf einen Aufruf des Unterprogramms ALLOC ein zweiter Aufruf mit denselben Aktualparametern, dann wird dieser zweite Aufruf ignoriert. In allen anderen Fehlerfällen tritt ein Laufzeitfehler auf (FATAL ERROR).

Wird ein Feld angesprochen, das als dynamisch vereinbart wurde, dem aber kein Speicherplatz durch CALL ALLOC zugewiesen wurde, dann erfolgt ein undefinierter Ablauf. Solche Felder können nur erkannt werden, wenn die Testoption TESTOPT=(BOUNDS) gesetzt wurde.

10.6.3 Freigeben des Speicherplatzes (CALL DEALLOC)

Der Speicherplatz für ein dynamisch angelegtes Feld feldname wird durch den Aufruf des Unterprogramms DEALLOC wieder freigegeben:

```
CALL DEALLOC (feldname)
```

feldname

Name des in der Typ-, DIMENSION- oder COMMON-Anweisung als dynamisch vereinbarten Feldes.

Verhalten im Fehlerfall

Wird in einer Programmeinheit ein durch den Aufruf von ALLOC dynamisch angelegtes Feld nicht durch den Aufruf von DEALLOC freigegeben, dann wird der Speicherplatz erst bei Programmbeendigung freigegeben. Wurde der Speicherplatz eines dynamisch angelegten Feldes noch nicht zugewiesen oder bereits durch den Aufruf von DEALLOC freigegeben, dann wird der Aufruf von DEALLOC ignoriert (Ausgabe einer LIBRARY WARNING). In allen anderen Fällen tritt ein Laufzeitfehler auf (FATAL ERROR).

10.6.4 Abfragen der Indexgrenzen (CALL GETSHAPE)

Um die Indexgrenzen eines dynamisch angelegten Feldes feldname abzufragen, steht das Unterprogramm GETSHAPE zur Verfügung. Die Abfrage der aktuellen Indexgrenzen ist z.B. in Unterprogrammen mit dynamisch angelegten Feldern als Formalparameter sinnvoll.

CALL GETSHAPE (feldname,u_1,o_1[,u_2,o_2][,...][,u_n,o_n])

---

feldname
: Name des in der Typ-, DIMENSION- oder COMMON-Anweisung als dynamisch vereinbarten Feldes.

u_i,o_i
: INTEGER*4-Variable. u_i erhält als Wert den kleinsten, o_i den größten Index der i-ten Dimension ($1 \le i \le 7$) des dynamischen Feldes *feldname*. Die Anzahl n ($1 \le n \le 7$) der Dimensionen muß gleich der in derselben Programmeinheit für das dynamische Feld vereinbarten Anzahl sein.

Verhalten im Fehlerfall

Laufzeit-Fehlermeldungen (LIBRARY ERRORs) werden in folgenden Fällen ausgegeben:

- *feldname* ist in der aktuellen Programmeinheit nicht als dynamisch angelegtes Feld vereinbart;
- dem dynamischen Feld *feldname* wurde noch kein Speicherplatz durch das Unterprogramm ALLOC zugewiesen;
- die Anzahl n der abgefragten Untergrenzen u_i bzw. Obergrenzen o_i stimmt nicht mit der in der Programmeinheit vereinbarten Anzahl von Dimensionen überein.

10.6.5 Einschränkungen bei der Programmierung mit dynamisch angelegten Feldern

Initialisierung
Ein dynamisch angelegtes Feld kann nicht durch eine Typanweisung oder eine DATA-Anweisung initialisiert werden. Die Initialisierung muß nach dem Aufruf des Unterprogramms ALLOC durch Wertzuweisungen oder Eingabe-Anweisungen erfolgen.

Überlagerung (EQUIVALENCE-Anweisung)
Ein dynamisch angelegtes Feld kann nicht durch die EQUIVALENCE-Anweisung mit einem anderen Datenelement derselben Programmeinheit überlagert werden. Bei einer Überlagerung eines dynamisch angelegten Feldes durch EQUIVALENCE erfolgt eine Fehlermeldung zur Übersetzungszeit.

Überlagerung (COMMON-Anweisung)
Bei dynamisch angelegten Feldern in nicht initialisierten COMMON-Bereichen wird nur der Speicherplatz für einen Felddeskriptor reserviert. Der Speicherplatz für das dynamisch angelegte Feld wird erst durch den Aufruf des Unterprogramms ALLOC angelegt.

Bei dynamisch angelegten Feldern in COMMON-Bereichen muß der Anwender folgende Einschränkung beachten:

Ein dynamisches Feld in einem COMMON-Bereich darf in anderen Programmeinheiten nur mit einer Indexgrenzenliste der Form (:[,:][,...]) vereinbart werden. Dimensionsanzahl und Typ des dynamischen Feldes müssen in allen Programmeinheiten übereinstimmen, in denen das Feld angesprochen wird.

Feldelemente als Aktualparameter
Ist ein Aktualparameter ein Feldelement, dem als Formalparameter ein Feld entspricht, dann darf dieses Feld kein dynamisches Feld sein.

Testhilfen
Dynamische Felder können mit der Testhilfe AID (ab V1.0C) angesprochen werden.

Dynamische Felder als Formalparameter
Einem dynamischen Feld als Formalparameter darf als Aktualparameter nur ein dynamisches Feld vom gleichen Datentyp mit der gleichen Dimensionsanzahl zugeordnet werden.

11 Programmverknüpfungen

Ein Programmsystem besteht aus einem Hauptprogramm (das Programm, das auf Systemebene aufgerufen wird) und einem oder mehreren Unterprogrammen, die sowohl in der Sprache des Hauptprogramms als auch in anderen Programmiersprachen geschrieben sein können.

Für die hierzu erforderlichen Programmverknüpfungen gibt es ab der FOR1-Version 2.2A zwei unterschiedliche Konzepte:

- Verknüpfung entsprechend den bisherigen Konventionen
- Verknüpfung entsprechend der Programm-Kommunikationsschnittstelle ILCS (= Inter Language Communication Services).

Über die Option LINKAGE={STD|FOR1-SPECIFIC} kann der Anwender festlegen, auf welche Art die Verknüpfung erfolgen soll (siehe 4.2.2.6). Wird mit dem voreingestellten Wert STD übersetzt, so erfolgt die Verknüpfung gemäß ILCS.

Aufbau des Kapitels

In Abschnitt 11.1 wird die neue Programm-Kommunikationsschnittstelle ILCS vorgestellt.

Abschnitt 11.2 informiert über die Kompatibilität bei Verknüpfung von Programmen, die mit unterschiedlichen FOR1-Versionen erzeugt wurden, sowie über die Kompatibilität bei der Verknüpfung verschiedensprachiger Programme.

Der Abschnitt 11.3 beschreibt Ablauf und Konventionen der Programmverknüpfung.

In Abschnitt 11.4 wird dargestellt, was der Anwender beim Binden von Programmsystemen beachten muß, die zwar FOR1-Unterprogramme, jedoch kein FOR1-Hauptprogramm enthalten.

Die Abschnitte 11.5 - 11.7 beschreiben die Verknüpfung von FOR1-Programmen mit COBOL-Programmen (11.5), PLI1-Programmen (11.6) und C-Programmen (11.7). In diesen Abschnitten wird aufgelistet, welche Parametertypen neben den von ILCS allgemein garantierten jeweils möglich sind.

FOR1-Programme können auch mit RPG3-Programmen und Pascal-XT-Programmen (ab Pascal-XT V2.2A) über ILCS verknüpft werden. Diese Sprachverknüpfungen werden jedoch im vorliegenden Handbuch nicht gesondert beschrieben.

Für die Verknüpfung von FOR1- und Assembler-Programmen stellt FOR1 Verknüpfungsmakros zur Verfügung. Da mit diesen Makros eine ILCS-Verknüpfung jedoch nur eingeschränkt möglich ist, werden sie nicht in diesem Kapitel, sondern im Anhang (A.9) beschrieben. Uneingeschränkte ILCS-Verknüpfung von FOR1- und Assembler-Programmen wird durch die von ASSEMBH ab Version 1.1A zur Verfügung gestellten Verknüpfungsmakros ermöglicht. Diese werden im Handbuch "ASSEMBH - Beschreibung" [10] dargestellt.

11.1 Die Programm-Kommunikationsschnittstelle ILCS

Von ILCS werden die wesentlichen Funktionen der Kommunikation zwischen den Programmen einer Ablaufeinheit und zwischen Ablaufeinheit und Betriebssystem sprachübergreifend vereinheitlicht und vereinfacht.

ILCS ist eine Kombination aus Software und Schnittstellen-Konvention:
Es beinhaltet zum einen Laufzeitroutinen, die in einer PLAM-Bibliothek zusammengefaßt sind, zum anderen garantiert ILCS auch die den "Standard-Linkage-Konventionen im BS2000" entsprechende Kommunikationsschnittstelle; d.h. jeder von einem ILCS-fähigen Compiler erzeugte Objektmodul ist entsprechend den Standard-Linkage-Konventionen für die Verknüpfung mit gleich- und verschiedensprachigen Programmen vorbereitet.

Die Bibliothek der ILCS-Laufzeitroutinen wird mit jedem ILCS-fähigen Compiler - als gleichsam zusätzliches Laufzeitsystem - ausgeliefert.

Im einzelnen bietet ILCS folgende Funktionen:

- multilaterale Konvention zur Verknüpfung verschiedensprachiger Programme
- einheitliche Richtlinien zur Ereignisbehandlung
- Speicherverwaltung (Stack- und Heap-Speicher)
- Behandlung der Programmaske
- Verarbeitung nicht lokaler Sprünge

11.1.1 Initialisierung des Programmsystems

Die Initialisierung eines Programmsystems verläuft in zwei Stufen:

- Zuerst veranlaßt das Hauptprogramm den Aufruf der ILCS-Initialisierungsroutine. Bei FOR1-Programmsystemen bewirkt, falls das Hauptprogrammm selbst kein ILCS-Objekt ist, auch das Vorhandensein von FOR1-ILCS-Unterprogrammen den Aufruf der ILCS-Initialisierungsroutine (siehe Beispiele, Programmsytem C).

  Beispiele:

  Programmsystem A:

  ```
  FOR1-ILCS-Hauptprogramm              ⟶   Aufruf von ILCS
  COBOL-ILCS-Unterprogramm
  ```

  Programmsystem B:

  ```
  COBOL-Hauptprogramm (Nicht-ILCS)     ⟶   Kein Aufruf von ILCS
  FOR1-ILCS-Unterprogramm
  ```

  Programmsystem C:

  ```
  FOR1-Hauptprogramm (Nicht-ILCS)      ⟶   Aufruf von ILCS
  FOR1-ILCS-Unterprogramm
  ```

- Die aufgerufene ILCS-Initalisierungsroutine ruft dann ihrerseits alle nötigen sprachspezifischen Initialisierungen auf, so daß vor Ausführung der ersten Programmanweisung die für das gesamte Programmsystem benötigten Sprachumgebungen eingerichtet sind.

11.1.2 ILCS-Umgebung

Programmsysteme, in denen die ILCS-Initialisierungsroutine aufgerufen und somit ILCS aktiviert wird, laufen in der ILCS-Umgebung ab. In ILCS-Umgebungen wird die Ereignisbehandlung gemäß der ILCS-Konvention durchgeführt.
Programmsysteme, die keinen ILCS-Modul enthalten, laufen nach den bisherigen Konventionen ab.

11.1.3 Prosys Common Data Area (PCD)

Zur internen Steuerung der Programmverknüpfung in ILCS-Umgebungen gibt es neben den Sicherstellungsbereichen (Save Areas) der einzelnen Programme einen zentralen Datenbereich PCD, der Programmen aller Programmiersprachen zur Verfügung steht. Er hat eine Größe von 4096 byte.
Der erste Teil der PCD enthält die von ILCS verwendeten Datenbereiche, u.a. auch das Feld "Programmaske" (in Byte 148), das mit dem Wert X'0C' vorbelegt ist. Der zweite Teil der PCD enthält die jeweils 128 byte großen Programmiersprachen-Bereiche, die den Laufzeitsystemen der verschiedenen Sprachen zur Verfügung stehen.

11.1.4 Programmasken-Behandlung durch ILCS

Die Programmaske für den Programmablauf wird im Wege der Initialisierung auf den Wert des PCD-Feldes "Programmaske" (vorbelegt mit X'0C') gesetzt. Wird sie während des Programmablaufs verändert, muß sie vor dem nächsten Programmaufruf bzw. -rücksprung auf den Wert des PCD-Feldes "Programmaske" zurückgesetzt werden.

11.1.5 Parameterübergabe in ILCS-Programmsystemen

Die Semantik der Datentypen weist bei den durch ILCS verknüpfbaren Programmiersprachen starke Unterschiede auf. Im folgenden werden jene Datentypen aufgeführt, die in den einzelnen Programmiersprachen eine gleiche Datendarstellung besitzen und daher problemlos als Parameter übergeben werden können. Bei der Verwendung anderer Datentypen als Parameter ist die genaue Kenntnis der jeweiligen Datenablage erforderlich, um den korrekten Programmablauf sicherzustellen.

Compiler	Datentypen			
	Binär Wort	Gleitpunkt Wort	Gleitpunkt Doppelwort	String
FOR1	INTEGER*4	REAL*4	REAL*8	CHARACTER*i (feste Länge)
COBOL85	PIC S9(i) COMP SYNCHRONIZED 5<=i<=9	COMP-1	COMP-2	USAGE DISPLAY
Pascal-XT	long_integer	short_real	long_real	packed array [<range>]of char
PLI1	BIN FIXED(31) ALIGNED	BIN FLOAT(21) DEC FLOAT(6)	BIN FLOAT(53) DEC FLOAT(16)	CHAR(i) NONVARYING
C	long	float	double	char <var> [<size>]
Columbus-Assembler	F	E	D	C
RPG3	binäres Feld mit 0 Dezimal-stellen	—	—	alphanum. Feld (feste Länge)

Tab. 11-1: Datentypen, die zwischen verschiedensprachigen ILCS-Programmen problemlos ausgetauscht werden können

Die Daten müssen immer ausgerichtet abgelegt werden; d.h. 32-Bit-Ganzzahlen in binärer Darstellung liegen auf Wortgrenze, Gleitpunktzahlen auf Wort- bzw. Doppelwortgrenze, Zeichenketten auf Bytegrenze. Die Längen von Zeichenketten sind konstant und dem gerufenen Programm bekannt.

Es werden immer die Adressen der Daten übergeben, nicht die Werte selbst. Da auch bei der FOR1-Wertübergabe ("call by value") intern Adressen übergeben werden, ändert sich auch bei dieser Übergabeart nichts.

Das aufrufende Programm legt eine Liste der übergebenen Adressen an. Die Anzahl der Parameter wird in Register 0, die Adresse der Liste in Register 1 übergeben (siehe 11.3.3 "Registerkonventionen").

Von ILCS nicht allgemein garantierte FOR1-Parametertypen

Folgende FOR1-Parametertypen sind bei der Verbindung von verschiedensprachigen ILCS-Programmen nicht allgemein garantiert:

– Parameter mit FOR1-Datentypen, die nicht in Tabelle 11-1 aufgeführt sind:

INTEGER *1	COMPLEX*16
INTEGER *2	COMPLEX*32
INTEGER *8	LOGICAL*1
REAL*16	LOGICAL*4
COMPLEX*8	CHARCTER*(n,V)

– NAMELIST-Listen, Anweisungsmarken, EXTERNAL-Unterprogramme, mehrdimensionale Felder

Da jedoch die Parameterübergabe des FOR1 unverändert bleibt, können auch die von ILCS nicht allgemein garantierten Parametertypen an fremdsprachige Programme übergeben werden. Es gelten die bisherigen Einschränkungen.

Felder als Parameter bei der Verknüpfung verschiedensprachiger ILCS-Programme

Bei der Verknüpfung verschiedensprachiger Programme ist eine Übergabe von Feldern mit festen Grenzen möglich, da bei solchen Feldern nur die Anfangsadresse übergeben wird und kein Feld-Deskriptor nötig ist.

Die Übergabe von eindimensionalen Feldern mit festen Grenzen wird von ILCS garantiert, falls die Felder einen ILCS-kompatiblen Datentyp haben (vgl. Tab. 11-1).

Auch mehrdimensionale Felder sind wie bisher bei der Verknüpfung verschiedensprachiger Programme als Parameter möglich. Da jedoch mehrdimensionale Felder in den verschiedenen Sprachen nicht einheitlich aufgebaut sind, und die maximal zulässige Dimensionszahl unterschiedlich ist, wird die Übergabe mehrdimensionaler Felder von ILCS nicht garantiert.

11.1.6 Hinweise zum Binden von ILCS-Programmsystemen

Neben den sprachspezifischen Laufzeitbibliotheken wird die Bibliothek SYSLNK.ILCS benötigt, die die ILCS-Laufzeitroutinen enthält.

Statisches Binden

Beim Statischen Binden genügt es wie bisher, mit der RESOLVE-Anweisung des TSOSLNK die FOR1-Laufzeitbibliothek zuzuweisen. Die ILCS-Initialisierungsroutine IT0INITS ist darin enthalten.

Da jedoch die sprachspezifischen Laufzeitbibliotheken jeweils unterschiedliche Versionen der ILCS-Initialisierungsroutine enthalten können, empfiehlt es sich, bei Sprachverknüpfungen die ILCS-Initialisierungsroutine aus der ILCS-Bibliothek SYSLNK.ILCS zu verwenden. Diese hat immer die aktuellste Version.

Auf die Initialisierungsroutine aus der SYSLNK.ILCS-Bibliothek wird zugegriffen, wenn:

(a) diese Routine mit einer INCLUDE-Anweisung explizit eingebunden wird (INCLUDE IT0INITS, $TSOS.SYSLNK.ILCS), oder

(b) die SYSLNK.ILCS-Bibliothek mit der letzten RESOLVE-Anweisung zugewiesen wird (RESOLVE ,$TSOS.SYSLNK.ILCS).

Dynamisches Binden

Enthält das Programmsystem ausschließlich gleichsprachige Programme, genügt es beim dynamischen Binden wie bisher, die FOR1-Laufzeitbibliothek mit SET-TASKLIB zuzuweisen.

Programmsysteme, die verschiedensprachige Programme enthalten, können mit dem DBL (ab BS2000-Version 10.0) dynamisch gebunden werden, da beim DBL im RUN-MODE=ADVANCED mehr als eine Bibliothek zugewiesen werden kann (über die LINK-Namen BLSLIBnn, $00 \leq nn \leq 99$).

Um sicherzustellen, daß die aktuellste ILCS-Initialisierungsroutine verwendet wird, sollte neben den sprachspezifischen Laufzeitsystemen auch die ILCS-Bibliothek SYSLNK.ILCS zuzuweisen werden. Hierbei muß darauf geachtet werden, daß der LINK-Name der ILCS-Bibliothek eine niedrigere Nummer erhält als die LINK-Namen der sprachspezifischen Bibliotheken, da der DBL die zugewiesenen Bibliotheken nach aufsteigenden Nummern durchsucht.

11.2 Kompatibilität

11.2.1 Begriffserläuterungen

ALT-Programme
Programme, die mit einer FOR1-Version ≤ 1.6A erzeugt wurden.

NXS-Programme
Programme, die mit einem FOR1-Compiler der Versionen 2.0A/2.1A mit der Option EXTENDED-SYSTEM=NO erzeugt wurden.

XS-Programme
Programme, die mit einem FOR1-Compiler ab Version 2.2A erzeugt wurden (ab V2.2A werden ausschließlich XS-Moduln generiert) bzw.
FOR1-Programme, die mit einem FOR1-Compiler der Versionen 2.0A/2.1A mit der Option EXTENDED-SYSTEM=YES übersetzt wurden.

ILCS-Programme
FOR1-Programme, die mit einer FOR1-Compiler ab Version 2.2A mit der Option LINKAGE=STD übersetzt wurden. Da ab Version 2.2A ausschließlich XS-Moduln erzeugt werden, sind ILCS-Programme grundsätzlich XS-Programme.

11.2.2 Kompatibilität bei der Verknüpfung von FOR1-Programmen

NXS-Programme, XS-(Nicht-ILCS)-Programme und ILCS-Programme lassen sich problemlos miteinander verknüpfen.

ALT-Programme dagegen lassen sich (außer mit anderen ALT-Programmen) nur mit NXS-Programmen verknüpfen: Eine unmittelbare Verknüpfung mit XS-Programmen - und somit mit ILCS-Programmen - ist nicht möglich.

11.2.3 Kompatibilität bei der Verknüpfung verschiedensprachiger Programme

In ILCS-Umgebungen sollten grundsätzlich alle an Sprachübergängen beteiligten Programme ILCS-Programme sein. Nur so kann korrekte Fehlerbehandlung und Parameterübergabe garantiert werden.

In Nicht-ILCS-Umgebungen können FOR1-ILCS-Programme und fremdsprachige Nicht-ILCS-Programme über die alte Fremdsprachenschnittstelle kombiniert werden - jedoch mit folgenden Einschränkungen:

- C-Programme, die mit älteren C-Compilerversionen (<V2.0) erstellt wurden, gehen davon aus, daß beim Aufruf parameterloser INTEGER-Funktionen das Register R1 erhalten bleibt. ILCS-INTEGER-Funktionen geben jedoch den Funktionswert sowohl in R0 als auch in R1 zurück.
- Bei Assembler-Programmen, die mit Hilfe der FOR1-Makros die FOR1-Schnittstelle simulieren, sollten beim Makro IFART (bzw. IFARTO) die Parameterwerte FIRST=1 und LAST=12 gewählt werden.

11.3 Unterprogrammanschluß: Ablauf und Konventionen

Nach dem Aufruf eines Unterprogramms erfolgen eine Reihe von Maßnahmen, die vom rufenden und vom gerufenen Programm durchgeführt werden.

Bei der Sprachverknüpfung von ILCS-Programmen in ILCS-Umgebung sowie beim Aufruf FORTRAN - FORTRAN werden diese Maßnahmen ohne Zutun des Anwenders korrekt ausgeführt.

Bei der Verknüpfung von FORTRAN-Programmen mit fremdsprachigen Nicht-ILCS-Programmen muß der Anwender durch den Aufruf vorgefertigter Makros bzw. Routinen dafür sorgen, daß die Verknüpfung korrekt abläuft.

11.3.1 Ablauf der Programmverknüpfung

Nach dem Aufruf eines Unterprogramms werden folgende Maßnahmen durchgeführt:

Zeitpunkt	Maßnahmen des aufrufenden Programms	Maßnahmen des aufgerufenen Programms
vor dem Ansprung	- Bereitstellen eines Datenbereichs, in dem die Parameteradressen und Informationen über die Parameter übergeben werden (Parameteradreßliste) - Bereitstellen eines Sicherstellungsbereichs (Save Area), der zur Abspeicherung der Registerinhalte dient - Bereitstellen der Ansprungadresse und der Rücksprungadresse - Ansprung des Unterprogramms	
nach dem Ansprung		- Abspeichern der Registerinhalte in die vom aufrufenden Programm bereitgestellte Save Area - Verketten der Save Areas, um die Aufrufhierarchie verfolgen zu können - nur bei ILCS: Ablegen der Adresse der Save Area des aufgerufenen Programms in die PCD (Prosys Common Data Area); Sichern des alten PCD-Inhalts - Setzen eines Indikators, der anzeigt, daß das Unterprogramm gerade aktiv ist
vor dem Rücksprung		- nur bei ILCS: Rücksetzen der Adresse der Save Area in der PCD - ggf. Setzen der DO-Schleifen auf inaktiv - ggf. Bereitstellen des Funktionswertes - Ablegen des Rückkehrcodes - Rücksetzen der alten Registerinhalte mit Ausnahme von R0 und R1 (vorgesehen für Funktionswerte) - Rücksprung
nach dem Rücksprung	- bei berechnetem Rücksprung: Auswerten des Rückkehrcodes	

Tab. 11-2: Maßnahmen des aufrufenden und aufgerufenen Programms

11.3.2 Aufbau des Sicherstellungsbereichs (Save Area)

Der Sicherstellungsbereich (Save Area) ist ein Zwischenspeicherbereich, in den bei Unterprogrammaufruf die Registerinhalte abgespeichert werden. Der Sicherstellungsbereich ist am Beginn des Datenabschnitts einer Programmeinheit angeordnet. Das rufende Programm versorgt vor dem Ansprung des Unterprogramms Register 13 mit der Adresse des Sicherstellungsbereichs. Register 13 wird im Sicherstellungsbereich des aufgerufenen Programms gesichert.

Ein Sicherstellungsbereich hat folgendes Format:

Byte	Inhalt
1-4	1. Byte: 1. Bit: Aktivitätsbit (1: Programm aktiv, 0: Programm inaktiv) 2.-7. Bit: reserviert 8. Bit = im allgemeinen 0 2. Byte: bei ILCS: Version = X'01'; sonst reserviert 3. und 4. Byte: bei ILCS X'FEFF'; sonst reserviert
5-8	enthält die Anfangsadresse des Sicherstellungsbereiches des aufrufenden Programms. Im **ersten** aufrufenden Programm ist der Inhalt dieses Feldes -1.
9-12	enthält ggf. die Anfangsadresse des nächsten (geketteten) Sicherstellungsbereiches.
13-16	Inhalt von Register 14
17-20	Inhalt von Register 15
21-24	Inhalt von Register 0
25-28	Inhalt von Register 1
29-32	Inhalt von Register 2
.	
.	
69-72	Inhalt von Register 12
73-76	Adresse der Runtime Communication Area (RTCA)
77-80	bei ILCS: Adresse der PCD; sonst reserviert
81-84	bei ILCS: Adresse der EHL (Event Handler List). Ist keine EHL definiert erhält das Feld den Wert -1; sonst: reserviert
85-88	reserviert

Tab. 11-3: Aufbau des Sicherstellungsbereichs (Save Area)

Das erste Bit des Sicherstellungsbereichs ist ein Indikatorbit, das den Wert 1 hat, falls die Programmeinheit gerade aktiv ist, und den Wert 0 hat, falls die Programmeinheit inaktiv ist. Aktive Programmeinheiten sind jene Einheiten, die aufgerufen, aber noch nicht abgeschlossen sind.

Verkettung von Sicherstellungsbereichen

Durch eine Vorwärts- und Rückwärtsverkettung von Sicherstellungsbereichen der aktiven Programmeinheiten kann die Aufrufhierarchie bei Laufzeitfehlern ausgegeben werden. In ILCS-Umgebungen wird die Aufrufhierarchie nur bis zu einer Sprachgrenze wiedergegeben.

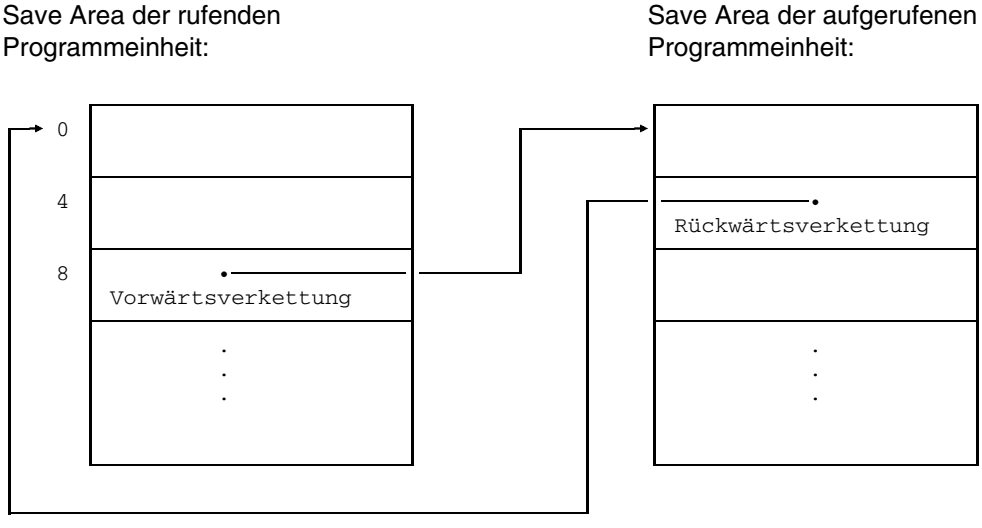


Bild 11-1: Verkettung der Sicherstellungsbereiche

Vorwärtsverkettung: Wort 2 des Sicherstellungsbereichs der rufenden Programmeinheit enthält die Anfangsadresse des Sicherstellungsbereichs des aufgerufenen Programms.

Rückwärtsverkettung: Wort 1 des Sicherstellungsbereichs der aufgerufenen Programmeinheit enthält die Anfangsadresse des Sicherstellungsbereichs des aufrufenden Programms. Im ersten aufrufenden Programm ist der Inhalt dieses Worts 0.

11.3.3 Registerkonventionen

Registerversorgung bei Aufruf

Die nachfolgende Tabelle gibt eine Übersicht über die Registerversorgung, die das aufrufende Programm vor dem Ansprung des aufgerufenen Programms durchführt.

Register-nummer	Inhalt
0	Anzahl der Parameter
1	Anfangsadresse der Parameteradreßliste
2 - 12	Programmdaten
13	Anfangsadresse des Sicherstellungsbereiches des aufrufenden Programms
14	Adresse des Rückkehrpunktes ins aufrufende Programm
15	Adresse des Einsprungpunktes im aufgerufenen Programm.
PM	Programmaske: Wert aus PCD-Feld "Programmaske" (bei ILCS)

Tab. 11-4: Registerversorgung bei Unterprogrammaufruf

Das rufende Programm versorgt diese Register vor dem Ansprung des aufgerufenen Programms. Die Registerinhalte der Mehrzweckregister werden in der vom aufrufenden Programm bereitgestellten Save Area abgespeichert außer der Inhalt von R13, der in der Save Area des aufgerufenen Programms gespeichert wird (Rückwärtsverkettung).

Rückkehrcode bei Anweisungsmarken-Parametern

Anweisungsmarken-Parameter der Form "{&|*}anweisungsmarke" werden nicht in der Parameteradreßliste übergeben, sondern es wird ein Rückkehrcode in Register 1 abgelegt:

- bei einem einfachen Rücksprung (RETURN-Anweisung) wird 0 abgelegt;
- bei einem berechneten Rücksprung (RETURN ausdruck) wird der Wert von *ausdruck* abgelegt.

Registerversorgung bei Rückkehr ins aufrufende Programm

Die nachfolgende Tabelle gibt eine Übersicht über die Registerversorgung, die das aufgerufene Programm beim Rücksprung ins aufrufende Programm durchführt. Gleitpunktregister werden nicht restauriert.

Register-nummer	Inhalt
0 - 1	Returnwerte von Funktionen oder undefiniert
2 - 14	wie bei Aufruf-Versorgung
15	undefiniert
PM	Programmaske: Wert aus PCD-Feld "Programmaske" (bei ILCS)

Tab. 11-5: Registerversorgung bei Rückkehr ins aufrufende Programm

Übergabe eines Funktionswertes

Bei Rückkehr aus einem FUNCTION-Unterprogramm wird der Funktionswert in folgenden Registern abgelegt:

Typ des Funktionswertes	Register
LOGICAL*1	R0
LOGICAL*4	R0
INTEGER*1	R0
INTEGER*2	R0
INTEGER*4	R0
INTEGER*8	F0
REAL*4	F0
REAL*8	F0
REAL*16	F0,F2
COMPLEX*8	F0,F2
COMPLEX*16	F0,F2
COMPLEX*32	F0,F2,F4,F6
CHARACTER*{n|(n,v)}	R1

Tab. 11-6: Registerkonventionen für verschiedene Typen von Funktionen

ILCS-Funktionen vom Datentyp INTEGER*{1|2|4} legen den Funktionswert nicht nur in R0 ab, sondern zusätzlich auch in R1.

In R1 wird die Adresse des Deskriptors für ein Datenelement vom Typ CHARACTER abgelegt. Bei Datenelementen vom Typ LOGICAL*1, INTEGER*1 und INTEGER*2 wird der Wert rechtsbündig im Register R0 abgelegt. Sollen in einem Assembler-Unterprogramm solche von FOR1 berechnete Funktionswerte aus Register R0 verwendet werden, dann muß entsprechend der Befehl "STC R0, <adresse>" bzw. "STH R0, <adresse>" gegeben werden.

11.3.4 Parameteradreßlisten

Beim Aufruf eines FUNCTION- oder SUBROUTINE-Unterprogramms kann man über eine Parameterliste dem Unterprogramm Informationen übergeben. Dieser Parameterliste entspricht intern die Parameteradreßliste.

Die Parameteradreßliste enthält:

- die Adressen der zu übergebenden Datenelemente
- die Adressen von Deskriptoren der zu übergebenden Datenelemente
- Informationen über Typ und Länge der Datenelemente (Typ-Indikator) sowie über weitere Eigenschaften wie Konstante, Variable, Feld, usw. (Art-Indikator).

Die Anzahl der übergebenen Parameter wird in Register 0, die Adresse der Parameteradreßliste wird in Register 1 abgelegt. Beide Informationen werden vom rufenden Programm in den Sicherstellungsbereich übernommen.

Die Adressen der Parameter und die Informationen über Typ- und Arteigenschaften werden für alle Parameter erzeugt. Enthält ein Unterprogramm keine Parameter, so wird keine Parameteradreßliste erzeugt.

Aufbau der Parameteradreßliste

Bild 11-2 zeigt den Aufbau der Parameteradreßliste für eine ungerade Zahl von Parametern.

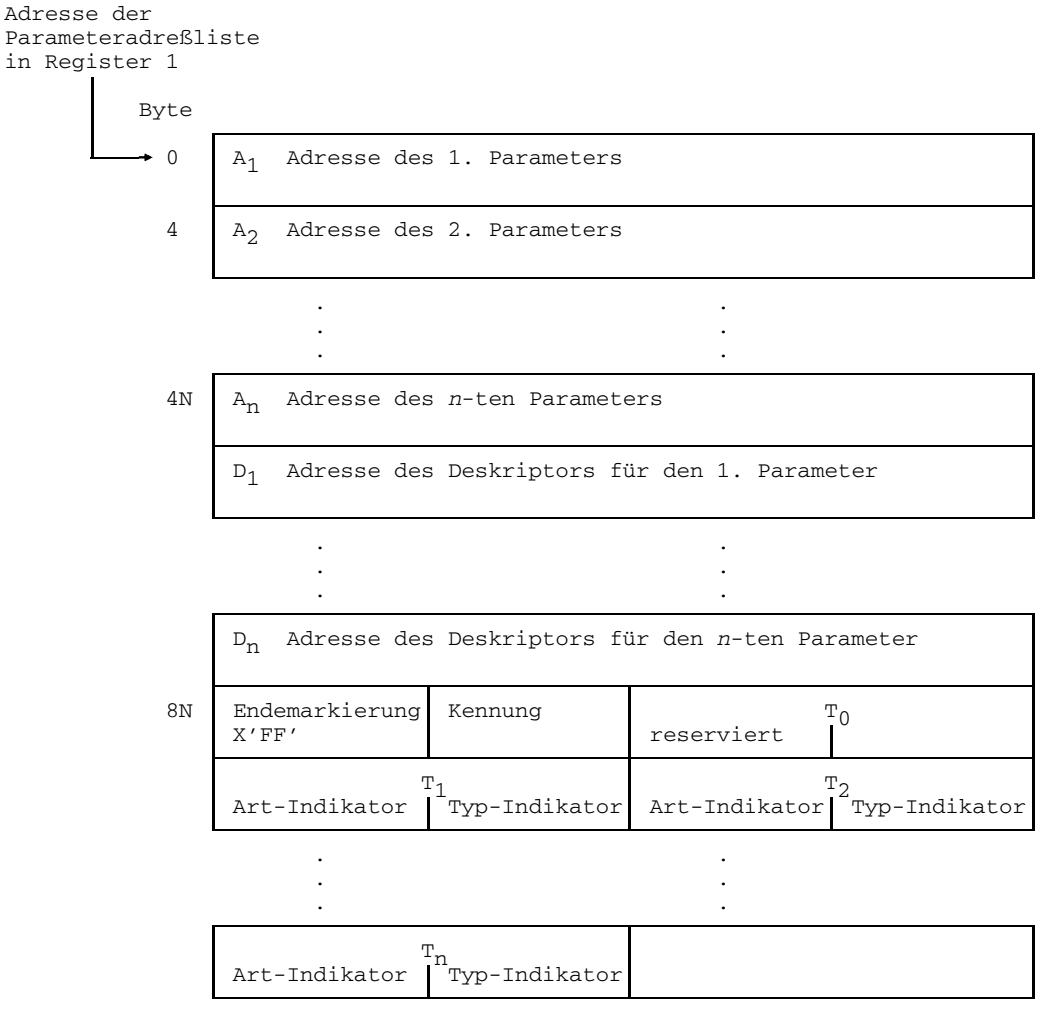


Bild 11-2: Aufbau der Parameteradreßliste

Der Begriff "reserviert" in diesem Kapitel bedeutet, daß der Inhalt eines derartigen Feldes nicht verändert werden darf (z.B. durch Assembler-Programme). Erzeugt ein Nicht-FOR1-Programm eine Parameteradreßliste zur Übergabe an ein FOR1-Programm, dann müssen reservierte Felder mit binären Nullen vorgelöscht werden.

A_i Adresse des *i*-ten Parameters. Das höchstwertige Bit hat den Wert 0. Bei Parametern vom Typ INTEGER und LOGICAL mit einer Länge kleiner 4 byte wird eine modifizierte Adresse übergeben: bei INTEGER*1 und LOGICAL*1 wird "Adresse-3", bei INTEGER*2 wird "Adresse -2" übergeben.

D_i Adresse des Dekriptors des *i*-ten Parameters. Das höchstwertige Bit hat den Wert 0. Falls der Parameter keinen Deskriptor benötigt, ist das zugehörige Wort reserviert.

Kennung

Bit 0 bis 5	Bit 0 bis 5 dieses Bytes enthalten ein Versionskennzeichen für die Parameteradreßliste. Das Kennzeichen für FOR1-Versionen ≥ V2.0A ist B'000000'.
Bit 6	0: keine Deskriptoren vorhanden 1: Deskriptoren vorhanden
Bit 7	0: keine Art- und Typ-Information vorhanden 1: Art- und Typ-Information vorhanden

T_0 Das erste Byte dieses Halbworts ist reserviert, das zweite Byte enthält:
- beim Aufruf eines FUNCTION-Unterprogramms den Typ des zurückgelieferten Funktionswerts
- beim Aufruf einer SUBROUTINE den Typeintrag NIL (Bitmuster 10000)

T_i $(1 \leq i \leq n)$

Das höherwertige Byte dieses Halbworts ist der Art-Indikator, das niederwertige Byte der Typ-Indikator des entsprechenden Parameters.

Werte des Art-Indikators einer Parameteradreßliste:

Bitmuster	Wert	Bedeutung des Art-Indikators
0000	0	temporäre Hilfsvariable für Aktualparameter-Ausdrücke
0001	1	Konstante
0010	2	Variable
0011	3	in einer EXTERNAL-Anweisung genanntes Unterprogramm
0100	4	Feld
0101	5	Feldelement (Deskriptor e. Feldelements o. e. Feldes)
0110	6	Teilkette
0111	7	FOR1-spezifische INTRINSIC-Funktion
1000	8	reserviert
1001	9	NAMELIST-Name
1010	10	reserviert
1011	11	dynamisches Feld
1100	12	direkter Wert

Tab. 11-7: Werte des Art-Indikators

Werte des Typ-Indikators einer Parameterliste:

Bitmuster	Wert	Bedeutung des Typ-Indikators
00000	0	LOGICAL*1
00001	1	LOGICAL*4
00010	2	INTEGER*1
00011	3	INTEGER*2
00100	4	INTEGER*4
00101	5	INTEGER*8
00110	6	REAL*4
00111	7	REAL*8
01000	8	REAL*16
01001	9	COMPLEX*8
01010	10	COMPLEX*16
01011	11	COMPLEX*32
01100	12	CHARACTER fester Länge, Hollerith
01101	13	CHARACTER variierender Länge
01110	14	in einer EXTERNAL-Anweisung genanntes Unterprogramm
01111	15	NAMELIST-Name
10000	16	NIL (wird nur im Feld T_0 gesetzt, falls es sich um ein SUBROUTINE- und kein FUNCTION-Unterprogramm handelt)

Tab. 11-8: Werte des Typ-Indikators einer Parameteradreßliste

Hollerith-Datenelemente werden als CHARACTER-Feld übergeben, dessen Elementanzahl gleich der Länge des Hollerith-Datenelements und dessen Elementlänge gleich 1 ist.

11.3.5 Deskriptoren

Wird bei einem Unterprogrammaufruf mindestens ein Deskriptor benötigt, so wird für jeden der n Parameter Speicherplatz für einen Deskriptorverweis angelegt. Für einfache Variablen sind keine Deskriptoren notwendig. Deskriptoren werden bei Feldern, Feldelementen und Zeichenketten als Aktualparameter benötigt, und zwar

- bei Feldern, denen als Formalparameter ein Feld mit freier Indexobergrenze (*) in der obersten Dimension entspricht;
- bei Feldelementen, denen als Formalparameter ein Feld mit freier Indexobergrenze (*) in der obersten Dimension entspricht;
- bei CHARACTER-Variablen variabler Länge als Formalparameter;
- bei CHARACTER-Variablen der Länge (*) als Formalparameter (die Länge wird vom Aktualparameter übernommen);
- bei Feldern vom Typ CHARACTER, denen als Formalparameter ein Feld von CHARACTER-Variablen der Länge (*) entspricht.

Je nach Art des Aktualparameters werden verschiedene Deskriptoren erzeugt:

- bei Feldern und Feldelementen ein Feld-Deskriptor (Array Descriptor, ADS)
- bei Feldelement-Parametern:
  Zusätzlich zum Feld-Deskriptor (ADS) wird ein Feldelement-Deskriptor (EDS) erzeugt, falls der Feldelement-Parameter eine CHARACTER-Teilkette ist.
- bei Zeichenketten ein Zeichenketten-Deskriptor (String Descriptor, SDS)

Wird ein dynamisches Feld an ein Unterprogramm übergeben, dann wird der ganze Deskriptor in den Eingangscode des Unterprogramms kopiert.

Aufbau der Deskriptoren

Zeichenketten-Deskriptor (SDS)

Byte		
0	reserviert	
4	reserviert	
8	reserviert	
12	maximale Länge	aktuelle Länge

Feld-Deskriptor (ADS)

Byte		
0	reserviert	
4	Adresse des 1. Bytes hinter dem Feldende	
8	reserviert	
12	reserviert	Länge eines Feldelements

Deskriptor für ein Feldelement vom Typ CHARACTER-Teilkette (EDS)

Byte		
0	reserviert	
4	Adresse des 1. Bytes hinter dem Feldende	
8	reserviert	
12	reserviert	aktuelle Länge eines Feldelements

11.4 Binden von Programmsystemen ohne FOR1-Hauptprogramm

Beim Binden von Programmsystemen, die FOR1-Unterprogramme, jedoch kein FOR1-Hauptprogramm enthalten, muß folgendes beachtet werden:

In jeder generierten END-Anweisung eines FOR1-Unterprogramms steht ein Externverweis auf die Startadresse des zugehörigen FOR1-Hauptprogramms mit dem Entry IF@@MPI (Main Program Initializer).

Wenn FOR1-Unterprogramme ohne ein FOR1-Hauptprogramm gebunden werden sollen, meldet der Binder dementsprechend einen unbefriedigten FOR1-Externverweis IF@@MPI.

Ein Binden ohne FOR1-Hauptprogramm kann erreicht werden

- durch die Anweisungen BIND oder CONTINUE des TSOSLNK. Diese Anweisungen lösen sofortiges Binden aus, auch wenn nicht alle Externverweise befriedigt werden können.
- durch die LET-Anweisung des TSOSLNK oder den Operand LET=Y in der PROGRAM-Anweisung, die ebenfalls Binden auslösen, auch wenn nicht alle Externverweise befriedigt werden können.
- durch das Eintragen eines Dummy-Moduls IF@@MPI in die Anwenderbibliothek. Dieser Dummy-Modul kann z.B. durch die Assemblierung folgender Anweisungen erzeugt werden:

```
IF@@MPI CSECT
IF@@MPI AMODE ANY
IF@@MPI RMODE ANY
        END
```

Falls in der Anwenderbibliothek ein Dummy-Modul eingetragen ist, muß ein FOR1-Hauptprogramm explizit mit INCLUDE eingebunden werden, da der Binder beim automatischen Suchen auch den Dummy-Modul einbinden könnte.

11.5 Verknüpfung von FOR1- mit COBOL-Programmen

Unter einem COBOL-Programm wird im folgenden ein mit dem Compiler COBOL85 (siehe Handbuch "COBOL85 - Benutzerhandbuch" [15]) übersetztes COBOL-Programm verstanden.

Zusätzlich zulässige Parametertypen

Bei der Verknüpfung von FOR1- mit COBOL-Programmen sind neben den von ILCS allgemein garantierten Parametertypen (vgl. Tab. 11-1) zusätzlich Parameter des folgenden Datentyps zulässig:

COBOL	FOR1
COMP-2 SYNC	INTEGER*8

Tab. 11-9: COBOL-FOR1-Verknüpfung: zusätzlich zulässiger Parametertyp

Einschränkungen

- Das FOR1-Programm darf keine Testoptionen enthalten.
- Es ist nicht möglich, mit COBOL- und FOR1-Komponenten verzahnt auf derselben Datei zu arbeiten, da die Laufzeitsysteme keine Informationen austauschen.

11.5.1 FOR1-Programm ruft COBOL-Unterprogramm

Für den Aufruf von COBOL-Unterprogrammen aus FOR1-Programmen sind keine Vorkehrungen notwendig.

Aufruf: CALL unterprog (par_1,...,par_n)

11.5.2 COBOL-Programm ruft FOR1-Unterprogramm

In ILCS-Umgebungen sind für den Aufruf eines FOR1-Unterprogramms aus einem COBOL-Programm keine Vorkehrungen notwendig.

Beispiel für den Aufruf eines FOR1-Unterprogramms:

```
CALL "FOR1SUB" USING AP1,AP2,AP3
```

Beispiel: COBOL-Programm ruft FOR1-Unterprogramm

COBOL-Programm:

```
       ID DIVISION
       PROGRAM-ID.     COBFOR.
       ENVIRONMENT DIVISION.
       DATA DIVISION.
       WORKING-STORAGE SECTION
       01  TABELLE-1.
           02  INTEGER1-4
           02  INTEGER1-8                      COMP-2 SYNC.
           02  REAL1-4                         COMP-1 SYNC.
           02  REAL1-8                         COMP-2 SYNC.
           02  LOGICAL1-4          PIC S9(5)   COMP   SYNC.
           02  LOGICAL1-RED        REDEFINES LOGICAL1-4.
               03  LOG1            PIC X.
               03  FILLER          PIC X(3).
           02  CHARACTER1          PIC X(15).
      *
       01  TABELLE-2.
           02  INTEGER2-8                      COMP-2 SYNC.
           02  INTEGER2-4          PIC S9(5)   COMP   SYNC.
           02  REAL2-8                         COMP-2 SYNC.
           02  REAL2-4                         COMP-1 SYNC.
           02  LOGICAL2-4          PIC S9(5)   COMP   SYNC.
           02  LOGICAL2-RED        REDEFINES LOGICAL2-4.
               03  LOG2            PIC X.
               03  FILLER          PIC X(3).
           02  CHARACTER2          PIC X(15).
      *
      *
       77  LOGICAL1                PIC X(5).
       77  LOGICAL2                PIC X(5).
      *
      *
       01  AUSGABE-1               PIC X(25)  VALUE IS
               " COBOL:    INTEGER*4:".
       01  AUSGABE-2               PIC X(25)  VALUE IS
               " COBOL:    INTEGER*8:".
       01  AUSGABE-3               PIC X(25)  VALUE IS
               " COBOL:    REAL*4:".
       01  AUSGABE-4               PIC X(25)  VALUE IS
               " COBOL:    REAL*8:".
       01  AUSGABE-5               PIC X(25)  VALUE IS
               " COBOL:    LOGICAL*4:".
       01  AUSGABE-6               PIC X(25)  VALUE IS
               " COBOL:    CHARACTER*15:".
      *
      *
      *
       PROCEDURE DIVISION.
       ANF.
           MOVE 314            TO INTEGER1-4        INTEGER2-4.
           MOVE 3141592654535  TO INTEGER1-8        INTEGER2-8.
           MOVE 3.141592654    TO REAL1-4           REAL2-4.
     MOVE 3141592654.533 TO REAL1-8           REAL2-8.
```

```
MOVE LOW-VALUES     TO LOG1              LOG2.
       MOVE "ABCDEFGHIJKL" TO CHARACTER1 CHARACTER2.
       PERFORM AUSGABE.
  *
   FORTRAN.
       CALL "UFOR"        USING    TABELLE-1  TABELLE-2.
       CALL "UFOR" USING
          INTEGER1-4  INTEGER1-8  REAL1-4  REAL1-8  LOGICAL1-4
          CHARACTER1
          INTEGER2-8  INTEGER2-4  REAL2-8  REAL2-4  LOGICAL2-4
          CHARACTER2.
   AUSGABE.
   IF LOG1 EQUAL LOW-VALUES THEN
           MOVE "FALSE" TO LOGICAL1
       ELSE
           MOVE "TRUE " TO LOGICAL1.
       IF LOG2 EQUAL LOW-VALUES THEN
           MOVE "FALSE" TO LOGICAL2
       ELSE
           MOVE "TRUE " TO LOGICAL2.
  *
       DISPLAY AUSGABE-1 INTEGER1-4 " ; " INTEGER2-4 UPON TERMINAL.
       DISPLAY AUSGABE-2 INTEGER1-8 " ; " INTEGER2-8 UPON TERMINAL.
       DISPLAY AUSGABE-3 REAL1-4    " ; " REAL2-4    UPON TERMINAL.
       DISPLAY AUSGABE-4 REAL1-8    " ; " REAL2-8    UPON TERMINAL.
       DISPLAY AUSGABE-5 LOGICAL1   " ; " LOGICAL2   UPON TERMINAL.
       DISPLAY AUSGABE-6 CHARACTER1 " ; "
                         CHARACTER2 UPON TERMINAL.
   ENDE.
       STOP RUN.
```

FOR1-Unterprogramm:

```
      SUBROUTINE UFOR (I4,I8,R4,R8,L4,C1,K8,K4,S8,S4,M4,C2)
C
      INTEGER*4   I4,K4
      INTEGER*8   I8,K8
      REAL*4      R4,S4
      REAL*8      R8,S8
      LOGICAL*4   L4,M4
      CHARACTER*15 C1,C2
C
      WRITE(2,10) I4,K4
      WRITE(2,20) I8,K8
      WRITE(2,30) R4,S4
      WRITE(2,40) R8,S8
      WRITE(2,50) L4,M4
      WRITE(2,60) C1,C2
C
      I4 = I4 * 2
      K4 = K4 * 2
      I8 = I8 * 2
      K8 = K8 * 2
      R4 = R4 * 2.
      S4 = S4 * 2.
      R8 = R8 * 2.
      S8 = S8 * 2.
```

```
      L4 = .NOT. L4
      M4 = .NOT. M4
      C1='MNOPQRSTUVWX'
      C2='YZ!"@$%&/()>'
C
      WRITE(2,10) I4,K4
      WRITE(2,20) I8,K8
      WRITE(2,30) R4,S4
      WRITE(2,40) R8,S8
      WRITE(2,50) L4,M4
      WRITE(2,60) C1,C2
C
 10   FORMAT (1X,'FORTRAN:   INTEGER*4:  ',I18,' ; ',I18)
 20   FORMAT (1X,'FORTRAN:   INTEGER*8:  ',I18,' ; ',I18)
 30   FORMAT (1X,'FORTRAN:   REAL*4   :  ',G18.8,' ; ',G18.8)
 40   FORMAT (1X,'FORTRAN:   REAL*8   :  ',G18.8,' ; ',G18.8)
 50   FORMAT (1X,'FORTRAN:   LOGICAL*4:  ',L18,' ; ',L18)
 60   FORMAT (1X,'FORTRAN:   CHARACTER*15:  ',A,' ; ',A)
C
      RETURN
      END
```

Ablauf:

Bei Ablauf des Programms COBFOR werden folgende Daten ausgegeben (das Programm COBFOR befindet sich in einer Datei gleichen Namens):

```
/START-PROG COBFOR
%  BLS0500 PROGRAM 'COBFOR' VERSION '' OF '91-07-30' LOADED
 COBOL:    INTEGER*4:    00314 ; 00314
 COBOL:    INTEGER*8:    +.314159265453500E+13 ; +.314159265453500E+13
 COBOL:    REAL*4:       +.314159E+01 ; +.314159E+01
 COBOL:    REAL*8:       +.314159265453300E+10 ; +.314159265453300E+10
 COBOL:    LOGICAL*4:    FALSE ; FALSE
 COBOL:    CHARACTER*15: ABCDEFGHIJKL    ; ABCDEFGHIJKL
FORTRAN:   INTEGER*4:                314 ;                314
FORTRAN:   INTEGER*8:       3141592654535 ;      3141592654535
FORTRAN:   REAL*4   :       3.1415930     ;      3.1415930
FORTRAN:   REAL*8   :      0.31415927D+10 ;     0.31415927D+10
FORTRAN:   LOGICAL*4:                  F ;                  F
FORTRAN:   CHARACTER*15:  ABCDEFGHIJKL    ; ABCDEFGHIJKL
FORTRAN:   INTEGER*4:                628 ;                628
FORTRAN:   INTEGER*8:       6283185309070 ;      6283185309070
FORTRAN:   REAL*4   :       6.2831860     ;      6.2831860
FORTRAN:   REAL*8   :      0.62831853D+10 ;     0.62831853D+10
FORTRAN:   LOGICAL*4:                  T ;                  T
FORTRAN:   CHARACTER*15:  MNOPQRSTUVWX    ; YZ!"@$%&/()>
 COBOL:    INTEGER*4:    00628 ; 00628
 COBOL:    INTEGER*8:    +.628318530907000E+13 ; +.628318530907000E+13
 COBOL:    REAL*4:       +.628319E+01 ; +.628319E+01
 COBOL:    REAL*8:       +.628318530906600E+10 ; +.628318530906600E+10
 COBOL:    LOGICAL*4:    TRUE  ; TRUE
 COBOL:    CHARACTER*15: MNOPQRSTUVWX    ; YZ!"@$%&/()>
```

11.6 Verknüpfung von FOR1- mit PLI1-Programmen

Die Sprachverknüpfung von FOR1- mit PLI1-Programmen erfolgt mittels Parametern und EXTERNAL-Daten (siehe Handbuch "PLI1-Benutzerhandbuch" [37]). Es können bis zu 255 Parameter übergeben werden.

Zusätzlich zulässige Parametertypen

Neben den von ILCS allgemein garantierten Parametertypen (vgl. Tab. 11-1) sind bei der Verknüpfung FOR1-PL1 Parameter der folgenden Datentypen zulässig:

FOR1	**PLI1** (nur ALIGNED-Daten)	
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 variable Länge	CHARACTER (n) VARYING	

Tab. 11-10: PLI1-FOR1-Verknüpfung: Zusätzlich zulässige Parametertypen

Besonderheiten

1. Ist eine Prozedur EXTERNAL, muß sie auch im PLI1-Programm eine Prozedur sein.
2. PLI1-Felder (DIMENSION), soweit sie einen verbundenen Speicher besitzen, können an FOR1-Felder übergeben werden. Sind die Elemente Zeichenfolgen, so müssen sie das Attribut NONVARYING haben. Felder können nicht als Funktionswert zurückgeben werden.

   Mehrdimensionale Felder in FOR1 werden im Gegensatz zu den Feldern in PLI1, die zeilenweise abgelegt werden, spaltenweise abgelegt.
   Durch eine Index-Überlagerung eines Feldes kann man nun erreichen, daß in PLI1 z.B. mit B (i, j, k) auf dasselbe Element zugegriffen wird wie in FOR1 mit A (i, j, k).

3. Vereinbarungen mit * sind an zulässigen Stellen auch bei der Parameter-Übergabe an FOR1 erlaubt. Es wird dann dafür gesorgt, daß aktuelle Werte in die zu übergebenden Datenbeschreibungen eingesetzt werden.
4. Ein COMMON-Block in FOR1 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.

11.6.1 FOR1-Programm ruft PLI1-Unterprogramm

In ILCS-Umgebungen sind für den Aufruf eines PLI1-Programms aus einem FOR1-Programm keine Vorkehrungen notwendig:

– Anweisung in FOR1:

```
CALL name (par1,...,parn)
```

– Anweisung in PLI1:

```
name: {PROCEDURE} (par1,...,parn) OPTIONS (ILCS);
      {         }
      {ENTRY    }
```

Beispiel: FOR1-Programm ruft PLI1-Unterprogramm

Ein von einem FOR1-Programm gerufenes PLI1-Unterprogramm UPROG könnte folgenden Aufbau haben:

```
UPROG:    PROC (A)    OPTIONS (ILCS);

       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;
```

Im rufenden FOR1-Programm wird das zu übergebende Feld mit `DIMENSION A (6,5,4)` deklariert.

DEFINED-Variablen mit iSUB-Angabe können **nicht** bei GET DATA und PUT DATA verwendet werden.

11.6.2 PLI1-Programm ruft FOR1-Unterprogramm

In ILCS-Umgebungen sind beim Aufruf eines FOR1-Unterprogramms von einem PLI1-Hauptprogramm keine Vorkehrungen notwendig:

– Anweisungen in PLI1:

```
DCL forupro ENTRY OPTIONS (ILCS);
CALL forupro (par1,...parn);
```

`forupro` Name des FOR1-Unterprogramms

– Anweisungen in FOR1:

```
{SUBROUTINE forupro (par1,...parn)}
{                                 }
{FUNCTION forupro (par1,...parn)  }
```

Beispiel: PLI1-Hauptprogramm ruft FOR1-Unterprogramm

Ein PLI1-Programm, das ein FOR1-Unterprogramm aufruft, könnte folgenden Aufbau haben:

```
PLIROUT:              PROC OPTIONS(MAIN);

       DCL    FORUP ENTRY (DIM(4,5,6)...) OPTIONS(ILCS);
       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;
```

Im aufgerufenen FOR1-Programm wird das zu übergebene Feld mit `DIMENSION A (6,5,4)` deklariert.

DEFINED-Variablen mit iSUB-Angabe können **nicht** bei GET DATA und PUT DATA verwendet werden.

11.7 Verknüpfung von FOR1- mit C-Programmen

Die Sprachverknüpfung von FOR1- mit C-Programmen bezieht sich auf FOR1-Programme, die mit einem FOR1-Compiler ab Version V2.0A übersetzt wurden.

Zusätzlich zulässige Parametertypen

Neben den von ILCS allgemein garantierten Parametertypen (vgl. Tab. 11-1) sind bei der Verknüpfung von FOR1-Programmen mit C-Programmen zusätzlich Parameter mit den folgenden Datentypen zulässig:

FOR1	C
INTEGER*8	double
LOGICAL*1	char
LOGICAL*4	int
COMPLEX*8	struct {float; float;}
COMPLEX*16	struct {double; double;}
Feld [1)]	Array

1) C-Arrays sind zeilenweise, FORTRAN-Felder dagegen spaltenweise organisiert. Diese unterschiedliche interne Organisation muß bei Verwendung mehrdimensionaler Arrays/Felder beachtet werden; wurde z.B. im C-Programm ein Array mit "typ feld [2] [3]" deklariert, dann muß im FOR1-Programm eine Dimensionsangabe mit "FELD (3,2)" erfolgen (siehe Beispiel in Abschnitt 11.7.1).

Tab. 11-11: FOR1-C-Verknüpfung: Zusätzliche zulässigen Parametertypen

Hinweise zum Binden

Beim Binden eines C-Hauptprogramms mit FOR1-Unterprogrammen sollte bei der PROG-Anweisung der Parameter LET=Y oder die Anweisung BIND verwendet werden.

Bei der Benutzung von mathematischen Funktionen ist folgendes zu beachten: Mathematische Funktionen (ATAN, ASIN etc.) gibt es sowohl im C- als auch im FOR1-Laufzeitsystem. Sie unterscheiden sich in der Parameterversorgung und im Datentyp. Der jeweils nicht benutzte Eingang muß mit einer EXCLUDE-Anweisung ausgeschlossen werden. Soll jedoch derselbe Eingangsname sowohl im C- als auch im FOR1-Teil benutzt werden, muß der jeweilige C- bzw. FOR1-Teil zu einem Großmodul mit der zugehörigen Laufzeitfunktion vorgebunden werden.

11.7.1 C-Programm ruft FOR1-Unterprogramm

Dieser Abschnitt beschreibt den Anschluß von externen FOR1-Unterprogrammen an C-Programme.
Das vom C-Programm aufgerufene FOR1-Unterprogramm kann ein FUNCTION- oder ein SUBROUTINE-Unterprogramm sein.

Parameterübergabe

In FOR1 werden in der Parameterliste stets die Adressen der Parameter übergeben, auch bei gewünschter Wertübergabe. Die Unterscheidung des Übergabemodus "call by reference" oder "call by value" findet erst in dem aufgerufenen FOR1-Unterprogramm statt und ist abhängig von der Art der dort vereinbarten Formalparameter.

In C werden mit Ausnahme von Arrays (Vektoren) und Pointern stets die Werte der Parameter in die Parameterliste eingetragen.
Bei Aufruf eines FOR1-Unterprogramms müssen deshalb in C als Aktualparameter die Adressen der zu übergebenden Datenelemente angegeben werden (z.B. mit Adreßoperator &: &par). Technisch gesehen wird die Adresse des gewünschten Parameters als Wert übergeben.
Arraynamen können direkt als Parameter angegeben werden, da der Wert eines Arrays per Definition seine Adresse ist.

In FOR1 werden beim Übergabemodus "call by value" die aktuellen Werte der Parameter am Ende des FOR1-Unterprogramms an die Aktualparameter zurücküberwiesen (siehe FOR1 Beschreibung [21]). Das hat zur Folge, daß auch bei diesem Übergabemodus die Aktualparameter nach der Rückkehr aus dem FOR1-Unterprogramm geändert sein können.

Bei der Verknüpfung von C mit FOR1 darf generell nicht das FOR1-Sprachmittel für berechnete Rücksprünge (RETURN ausdruck) verwendet werden.

Einschränkungen bei der Parameterübergabe

Neben dem unterschiedlichen Übergabemodus gibt es weitere Unterschiede im Aufbau der Parameterliste in C und FOR1.

FOR1 übergibt und erwartet bei Zeichenketten, Feldern und Feldelementen in bestimmten Fällen zusätzlich zu den Adressen der eigentlichen Datenelemente die Adressen von Deskriptoren (siehe Abschnitt 11.3.5). Außerdem werden Art- und Typ-Indikatoren übergeben.

Aufgrund dieser Unterschiede ergeben sich für das FOR1-Unterprogramm folgende Einschränkungen:

- keine DEBUG-Ausgabe (CALL DEBUG oder Fehlerfall),
- keine Felder als Formalparameter, deren Indexgrenze vom rufenden Programm versorgt wird (Zeichen * als obere Indexgrenze),
- keine Formalparameter im Zusammenhang mit den TESTOPT-Operanden ARG, BOUNDS, SUBSCR,
- keine CHARACTER-Teilketten als Formalparameter.

Wenn das FOR1-Unterprogramm die o.g. Bedingungen erfüllt, kann es problemlos aufgerufen werden, da FOR1 dann keine Typkennzeichen und keine Aggregatdaten-Deskriptoren benötigt.

Beispiel: C-Programm ruft FOR1-Unterprogramm

C-Programm CMAIN

```
#include <stdio.h>

main()
{
 char cfeld [2] [3];
 int  sz,sz1,sz2;                    /* Schleifenzaehler */
 for (sz = 0;sz <= 2;++sz) {
    cfeld [0] [sz] = 'A';
    cfeld [1] [sz] = 'B';
}for (sz1 = 0; sz1 <= 1; ++sz1)      /* ausgeben cfeld */
   for (sz2 = 0; sz2 <= 2; ++sz2)
     printf("C:    cfeld(%d,%d) = %c\n", sz1, sz2, cfeld [sz1] [sz2] );
 forsub(cfeld);                      /* Aufruf FOR1-Unterprogramm forsub */
 for (sz1 = 0; sz1 <= 1; ++sz1)      /* ausgeben cfeld */
   for (sz2 = 0; sz2 <= 2; ++sz2)
     printf("C:    cfeld(%d,%d) = %c\n", sz1, sz2, cfeld [sz1] [sz2] );
}
```

FOR1-Programm FORUP

```
C     FORTRAN-UNTERPROGRAMM

      SUBROUTINE FORSUB(CFELD)
      CHARACTER*1  CFELD(3,2)
      INTEGER*4    I,J
* AUSGEBEN CFELD
      DO 1 I=1,2
       DO 1 J=1,3
1        WRITE (2,10) J, I, CFELD(J,I)
* VERSORGEN CFELD MIT NEUEN WERTEN
      DO 2 I=1,3
           CFELD(I,1) = 'C'
2          CFELD(I,2) = 'D'
* AUSGEBEN CFELD
      DO 3 I=1,2
       DO 3 J=1,3
3        WRITE (2,10) J, I, CFELD(J,I)
10    FORMAT (' FOR1: CFELD(',I1,',',I1,') = ',A)
      RETURN
      END
```

Ablaufprotokoll zum Übersetzen, Binden und Programmablauf

```
(IN)      START-PROG $FOR1
(OUT)     %  BLS0500 PROGRAM 'FOR1', VERSION '2.2A00' OF '91-06-05' LOADED
(OUT)     %  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG ...
(OUT)      FOR1:   V2.2A00 READY, GIVE COMPILER OPTION
(IN)      COMOPT SRC=FORUP,MODULE-LIBRARY=PLAM.MODFOR1,END
(OUT)      FOR1: COMPILER NOT PRELOADED (BAD LOAD PERFORMANCE)
(OUT)      FOR1: NO ERRORS DURING COMPILATION OF P.U. FORSUB
(OUT)      END OF  F O R 1  COMPILATION;  CPU TIME USED: 0.611 SEC.
```

```
(IN)      START-PROG $C
(OUT)     %  BLS0500 PROGRAM 'C', VERSION '2.0A' OF '91-05-27' LOADED
(OUT)     %  CCM9992  BEGIN C  V2.0A00
(OUT)     %  CCM9993  Copyright (C) Siemens Nixdorf Informationssysteme AG 1991.
(OUT)     %  CCM9994  All rights reserved.
(IN)      COMPILE SOU=CMAIN,MODULE-LIBRARY=PLAM.MODC
(IN)      END
(OUT)     %  CCM9995  NOTES: 0 WARNINGS: 0 ERRORS: 0
(OUT)     %  CCM9997  MODULES GENERATED
(OUT)     %  CCM9998  END   C TIME USED = 4.1822

(IN)      START-PROG $TSOSLNK
(OUT)     %  BLS0500 PROGRAM 'TSOSLNK', VERSION '21.0D17' OF '91-04-25' LOADED
(IN)      PROG CFOR1,FILENAM=C.FORUP,LOADPT=*XS,LET=Y
(IN)      INCLUDE (CMAIN#,CMAIN@),PLAM.MODC
(IN)      RESOLVE ,PLAM.MODFOR1
(IN)      RESOLVE ,$CLIB
(IN)      RESOLVE ,$FOR1MODLIBS
(IN)      RESOLVE ,$TSOS.SYSLNK.ILCS
(IN)      BIND
(OUT)      UNRESOLVED EXTRNS:
(OUT)      IF@@MPI
(OUT)     %  LNK0055 PROGRAM BOUND IN SPITE OF UNRESOLVED EXTERN'S
(OUT)     %  LNK0062 THE PHASE CAN BE LOADED ON XS SYSTEM ONLY
(OUT)     %  LNK0503 PROGRAM FILE 'C.FORUP' WRITTEN
(OUT)     %  LNK0504     18 PAM PAGES USED. TSOSLNK RUN FINISHED

(IN)      SET-TASKLIB $FOR1MODLIBS
(IN)      START-PROG C.FORUP
(OUT)     %  BLS0500 PROGRAM 'CFOR1', VERSION ' ' OF '91-07-30' LOADED
(OUT)     C:    cfeld(0,0) = A
(OUT)     C:    cfeld(0,1) = A
(OUT)     C:    cfeld(0,2) = A
(OUT)     C:    cfeld(1,0) = B
(OUT)     C:    cfeld(1,1) = B
(OUT)     C:    cfeld(1,2) = B
(OUT)     FOR1: CFELD(1,1) = A
(OUT)     FOR1: CFELD(2,1) = A
(OUT)     FOR1: CFELD(3,1) = A
(OUT)     FOR1: CFELD(1,2) = B
(OUT)     FOR1: CFELD(2,2) = B
(OUT)     FOR1: CFELD(3,2) = B
(OUT)     FOR1: CFELD(1,1) = C
(OUT)     FOR1: CFELD(2,1) = C
(OUT)     FOR1: CFELD(3,1) = C
(OUT)     FOR1: CFELD(1,2) = D
(OUT)     FOR1: CFELD(2,2) = D
(OUT)     FOR1: CFELD(3,2) = D
(OUT)     C:    cfeld(0,0) = C
(OUT)     C:    cfeld(0,1) = C
(OUT)     C:    cfeld(0,2) = C
(OUT)     C:    cfeld(1,0) = D
(OUT)     C:    cfeld(1,1) = D
(OUT)     C:    cfeld(1,2) = D
(OUT)     %  CCM0998  used CPU-time 0.0280 seconds
```

Der Hinweis auf den unbefriedigten Externverweis IF@@MPI entsteht aufgrund des Bindens ohne FOR1-Hauptprogramm und kann in diesem Fall ignoriert werden.

11.7.2 FOR1-Programm ruft C-Funktion

Dieser Abschnitt beschreibt den Anschluß von C-Funktionen an FOR1-Programme.

C-Funktionen, die entsprechend ihrem Datentyp einen Funktionswert liefern, können in FOR1 sowohl innerhalb von Ausdrücken als auch mit der CALL-Anweisung aufgerufen werden.

C-Funktionen vom Typ "void" sollten nur mit der CALL-Anweisung aufgerufen werden.

Aufruf einer main-Funktion

Der Aufruf einer main-Funktion ist möglich. Als Einsprungadresse ist dann MAIN zu verwenden.
Wenn mehrere main-Funktionen existieren, kann die Auswahl der gewünschten main-Funktion durch explizites Einbinden des entsprechenden Objektmoduls sichergestellt werden.

Die Umlenkung der Standard-Ein-/Ausgabe-Dateien sowie Parameterübergaben an die main-Funktion sind nicht möglich.

Parameterübergabe

In FOR1 werden in der Parameterliste stets die Adressen der Parameter eingetragen. Die Formalparameter der C-Funktion sind deshalb als Zeiger auf die zu übergebenden Datenelemente zu definieren (<typ> *par); Arraynamen können direkt angegeben werden, da der Wert eines Arrays per Definition seine Adresse ist.

Bei der Verknüpfung von C mit FOR1 darf generell nicht das FOR1-Sprachmittel für berechnete Rücksprünge (RETURN ausdruck) verwendet werden.

Beispiel: FOR1-Programm ruft C-Funktion

FOR1-Programm FORHAUPT

```
C     FORTRAN-HAUPTPROGRAMM

      PROGRAM HAUPT
      INTEGER*4 X, Y, Z
      INTEGER*4 ADR
      X = 5
      Y = 4
      WRITE(2,100) X, Y, Z
100   FORMAT('  X = ', I4, '  Y = ', I4, ' Z = ', I4)
      CALL CSUB(X, Y, Z)
      WRITE(2,200) Z
200   FORMAT('  SUMME = ', I4)
      STOP
      END
```

C-Programm CUP

```
void
csub(a, b, c)
int *a, *b, *c;
{
  printf("c-program: a = %d, b = %d, c = %d\n", *a, *b, *c);
  *c = *a + *b;
  printf("c-program: summe = %d\n", *c);
}
```

Ablaufprotokoll zum Übersetzen, Binden und Programmablauf

```
(IN)      START-PROG $FOR1
(OUT)     %  BLS0500 PROGRAM 'FOR1', VERSION '2.2A00' OF '91-06-05' LOADED
(OUT)     %  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG ...
(OUT)      FOR1:   V2.2A00 READY, GIVE COMPILER OPTION
(IN)      COMOPT SRC=FORHAUPT,MODULE-LIBRARY=PLAM.MODFOR1,END
(OUT)      FOR1: COMPILER NOT PRELOADED (BAD LOAD PERFORMANCE)
(OUT)      FOR1: NO ERRORS DURING COMPILATION OF P.U. HAUPT
(OUT)      END OF  F O R 1  COMPILATION;  CPU TIME USED: 0.598 SEC.

(IN)      START-PROG $C
(OUT)     %  BLS0500 PROGRAM 'C', VERSION '2.0A' OF '91-05-27' LOADED
(OUT)     %  CCM9992  BEGIN C  V2.0A00
(OUT)     %  CCM9993  Copyright (C) Siemens Nixdorf Informationssysteme AG 1991.
(OUT)     %  CCM9994  All rights reserved.
(IN)      COMPILE SOU=CUP,MOD-LIB=PLAM.MODC
(IN)      END
(OUT)     %  CCM9995  NOTES: 0 WARNINGS: 0 ERRORS: 0
(OUT)     %  CCM9997  MODULES GENERATED
(OUT)     %  CCM9998  END   C TIME USED = 3.8963
```

```
(IN)     START-PROG $TSOSLNK
(OUT)    %  BLS0500 PROGRAM 'TSOSLNK', VERSION '21.0D17' OF '91-04-25' LOADED
(IN)     PROG PROG,FILENAM=C.FORHAUPT
(IN)     INCLUDE HAUPT,PLAM.MODFOR1
(IN)     INCLUDE (CUP#,CUP@),PLAM.MODC
(IN)     RESOLVE ,$CLIB
(IN)     RESOLVE ,$FOR1MODLIBS
(IN)     RESOLVE ,$TSOS.SYSLNK.ILCS
(IN)     BIND
(OUT)    %  LNK0500 PROGRAM BOUND
(OUT)    %  LNK0503 PROGRAM FILE 'C.FORHAUPT' WRITTEN
(OUT)    %  LNK0504     17 PAM PAGES USED. TSOSLNK RUN FINISHED

(IN)     SET-TASKLIB $FOR1MODLIBS

(IN)     START-PROG C.FORHAUPT
(OUT)    %  BLS0500 PROGRAM 'PROG', VERSION ' ' OF '91-07-30' LOADED
(OUT)    BS2000  F O R 1  :  FORTRAN PROGRAM "HAUPT"
(OUT)    STARTED ON 1991-07-30 AT 13:47:15
(OUT)     X =    5  Y =    4 Z =    0
(OUT)    c-program: a = 5, b = 4, c = 0
(OUT)    c-program: summe = 9
(OUT)     SUMME =    9
(OUT)    STOP AT STMT 11 IN HAUPT
(OUT)    BS2000  F O R 1 : FORTRAN PROGRAM "HAUPT   " ENDED PROPERLY AT 13:47:16
(OUT)    CPU - TIME USED :       0.0091 SECONDS
(OUT)    ELAPSED TIME    :       0.0150 SECONDS
```

11.7.3 Gemeinsame Dateiverarbeitung

Standard-Ein-/Ausgabedateien

Die Standard-Ein-/Ausgabedateien können sowohl im C- als auch im FOR1-Teil des Programms angesprochen werden.

Nicht-Standarddateien

Gemeinsam bearbeitete Dateien müssen sowohl im C- als auch im FOR1-Teil eröffnet werden. Ihre Abarbeitung ist intern über verschiedene FCB's realisiert.

Da die Verarbeitung einer gemeinsamen Datei über getrennte FCB's erfolgt, ist ein verzahntes Einlesen auf C- und FOR1-Seite nicht möglich. Sämtliche Zeichen der Datei werden dem C-Teil und dem FOR1-Teil geliefert.

12 Funktionenpool FPOOL

Das FPOOL-Konzept ermöglicht eine Ausweitung der bisher nur für INTRINSIC-Funktionen vorgenommenen Überprüfung von Aufrufschnittstellen.
Enthält ein Quellprogramm beispielsweise die Anweisungen

```
CHARACTER *5   X
Y = SIN(X)
```

dann gibt FOR1 eine Fehlermeldung aus, da im Aufruf der Funktion SIN ein Parameter vom Typ CHARACTER nicht zugelassen ist. Parameter in Aufrufen von benutzereigenen Unterprogrammen dagegen (CALL...) wurden bislang nicht vom Compiler überprüft.

Über den Zugriff auf FPOOL-Dateien, die Informationen über Aufrufschnittstellen enthalten, ist es nunmehr dem Compiler möglich, Schnittstellen zu Unterprogrammen (SUBROUTINEs und FUNCTIONs) in seine Fehleranalyse einzubeziehen. Bei FUNCTIONs werden nur die Parameter, nicht jedoch Typ und Länge der FUNCTION überprüft.

Da die FPOOL-Datei im Gegensatz zur INTRINSIC-Tabelle, anhand derer die Aufrufe von INTRINSIC-Funktionen überprüft werden, compilerunabhängig ist, bietet das FPOOL-Konzept die Möglichkeit, durch die Einrichtung eines zentralen FPOOLs Funktionen mit zu INTRINSIC-Funktionen analogen Eigenschaften zur Verfügung zu stellen, ohne daß diese im FOR1-Compiler oder im FOR1-Laufzeitsystem implementiert sein müßten.

Insbesondere kann der Anwender private FPOOL's einrichten und auf diese Weise Aufrufschnittstellen von eigenen Unterprogrammen durch den Compiler überprüfen lassen (vgl. Abschnitt 12.3).

Ein Funktionenpool ist in zwei Dateien realisiert: einer Objektmodulbibliothek, aus der FPOOL-Routinen in den Lademodul eingebunden werden, und der eigentlichen FPOOL-Datei, die die Beschreibung der Aufrufschnittstellen für diese Routinen enthält.

Eine Aufrufschnittstelle ist gegeben durch Anzahl, Folge, Typ, Länge, Ausrichtung, Dimension, Übergabe- und Rückkehr-Art von Parametern.

Die Schnittstellenbeschreibung in einer FPOOL-Datei kann außerdem Sprachsperren enthalten. Im Rahmen von Sprachverknüpfungen sind solche Sperren sinnvoll, wenn zwischen zwei Sprachen Unverträglichkeiten, z.B. im Aufruf- und Rückkehrmechanismus, vorliegen.

Der Fortran90-Compiler wird die Schnittstellenprüfung durch FPOOL nicht mehr unterstützen, da hierfür in Fortran90 unmittelbar Sprachmittel zur Verfügung stehen. Die Funktionen des zentralen FPOOL können aber auch vom Fortran90-Compiler noch genutzt werden, jedoch ohne Schnittstellenüberprüfung durch FPOOL.

Beispiel:

In dem Unterprogramm DIALOG sollen die Parameter X und Y verwendet werden, wobei X vom Typ INTEGER * 1, Y vom Typ CHARACTER * 10 sein soll. Bei der Compilierung eines Quellprogramms mit den folgenden Anweisungen

```
REAL * 4          X         (1)
CHARACTER * 10    Y
CALL DIALOG   (X,Y)         (2)
```

gäbe der FOR1-Compiler ohne FPOOL-Bearbeitung keine Fehlermeldung aus. Mit Hilfe des FPOOL-Anschlusses kann dagegen in der Anweisung (2) geprüft werden, ob DIALOG zu einem vom Anwender angegebenen FPOOL gehört. Falls ja, werden die Schnittstellen anhand der FPOOL-Datei überprüft.
Steht in dieser FPOOL-Datei, daß der 1. Parameter vom Typ INTEGER * 1 sein muß, so gibt FOR1 wegen der Anweisung (1) eine Fehlermeldung (SEVERE) aus.

12.1 Steuern der FPOOL-Bearbeitung

Die Einbeziehung von FPOOL-Dateien zur Überprüfung von Aufrufschnittstellen von Unterprogrammen wird über den SDF-Operand FPOOL-LIBRARY oder über die Compiler-Option FPOOL gesteuert.

12.1.1 SDF-Operand FPOOL-LIBRARY

```
START-FOR1-COMPILER

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

Eine Gegenüberstellung von SDF-Operanden und entsprechenden Compileroptionen enthält Tab. 2-4.

12.1.2 Compileroption FPOOL

*COMOPT	FPOOL [={fpoolname / ([fpoolname[,fpoolname]...])}]

fpoolname Name einer FPOOL-Datei

Nur wenn diese Option gesetzt ist, führt der Compiler eine FPOOL-Bearbeitung durch.

Wenn die COMOPT FPOOL eine Liste von FPOOL-Dateien enthält, wird durch deren Reihenfolge von links nach rechts eine Suchhierarchie aufgestellt: FOR1 sucht den Aufrufnamen eines Unterprogramms in der zuerst genannten FPOOL-Datei; wird er darin nicht gefunden, so setzt FOR1 die Suche in der nachfolgend genannten FPOOL-Datei fort usw..

Ist in der COMOPT FPOOL keine oder nur eine leere Liste von FPOOL-Dateien angegeben, so erwartet FOR1 die Angabe der zu berücksichtigenden FPOOL-Dateien über eine %FPOOL-Anweisung im Quellprogramm.

Die FPOOL-Option gilt für einen Übersetzungslauf und damit für alle davon betroffenen Programmeinheiten; sie kann jedoch durch die Anweisungen %FPOOL und %NOFPOOL modifiziert werden.

12.1.3 %FPOOL-Anweisungen im Quellprogramm

%FPOOL-Anweisung

Nur wenn die FPOOL-Option gesetzt ist, werden die folgenden Anweisungen im Quellprogramm vom FOR1-Compiler bearbeitet.

```
%FPOOL      fpoolname [(uproname[,uproname]...)],...
```

fpoolname Name einer FPOOL-Datei

uproname Aufrufname eines Unterprogramms, dessen Schnittstellen in der FPOOL-Datei *fpoolname* beschrieben sind.

Eine %FPOOL-Anweisung gilt nur in der jeweiligen Programmeinheit.

Durch die %FPOOL-Anweisung wird eine gegebenenfalls durch die FPOOL-Option aufgestellte Suchhierarchie verändert:
Wird zu *fpoolname* eine Liste mit Aufrufnamen angegeben, so werden die zu den Aufrufnamen gehörenden Schnittstellenbeschreibungen nur in der Datei *fpoolname* gesucht. Kommt ein in der %FPOOL-Anweisung genannter Unterprogrammname *uproname* nicht in der FPOOL-Datei *fpoolname* vor, dann wird eine Warnung ausgegeben. In diesem Fall werden keine weiteren Dateien durchsucht und es wird keine Schnittstellenüberprüfung durchgeführt. Kommt ein Unterprogrammname *uproname* in zwei oder mehr %FPOOL-Anweisungen vor, so wird zuerst die Datei durchsucht, die in der ersten %FPOOL-Anweisung steht, dann die Datei, die in der zweiten %FPOOL-Anweisung steht, usw. .

Wird *fpoolname* ohne eine Liste von Aufrufnamen angegeben, so wird diese FPOOL-Datei einer durch die FPOOL-Option bestimmten Suchhierarchie vorangestellt.

Die folgende Übersicht verdeutlicht die Suchreihenfolge des FOR1-Compilers:

Eine Programmeinheit enthalte den Aufruf

```
CALL  funktion (A,B).
```

Falls COMOPT FPOOL gesetzt ist, sucht FOR1 die zu *funktion* gehörende Schnittstellenbeschreibung folgendermaßen:

1) FOR1 prüft, ob *funktion* in einer Liste von Aufrufnamen in einer %FPOOL-Anweisung dieser Programmeinheit vorkommt.
   Falls ja, wird nur in der entsprechenden FPOOL-Datei *fpoolname* gesucht.
2) Gehört *funktion* zu keiner Liste von Aufrufnamen, durchsucht FOR1 die FPOOL-Dateien, die in %FPOOL-Anweisungen dieser Programmeinheit ohne eine Liste von Aufrufnamen angegeben wurden.

3) Wird *funktion* in keiner dieser FPOOL-Dateien gefunden, werden die in der FPOOL-Option genannten FPOOL-Dateien durchsucht.

In der Optionenliste werden nur die explizit in der FPOOL-Option (und nicht die in der %FPOOL-Anweisung) genannten Namen der FPOOL-Dateien aufgeführt.

%NOFPOOL-Anweisung

```
%NOFPOOL (uproname[,uproname]...)
```

uproname Aufrufname eines Unterprogramms

Eine %NOFPOOL-Anweisung gilt nur für die jeweilige Programmeinheit.

Die in der %NOFPOOL-Anweisung genannten Aufrufnamen werden von der FPOOL-Bearbeitung ausgeschlossen.
Verwendet der Anwender Aufrufnamen, die in einer der angegebenen FPOOL-Dateien vorkommen, für andere Unterprogramme, so sollte der entsprechende Aufruf von der FPOOL-Bearbeitung ausgeschlossen werden, da der Compiler sonst die entsprechenden FPOOL-Einträge zur Überprüfung der Aufrufschnittstelle heranzieht.

Falls Unterprogramme verwendet werden, die zu keiner der angegebenen FPOOL-Dateien gehören, kann mit Hilfe dieser Anweisung ein vergebliches Durchsuchen aller angegebenen FPOOL-Dateien vermieden werden.

Die Abbildung auf der folgenden Seite veranschaulicht die FPOOL-Berabeitung des FOR1-Compilers.

Das Bild steht in dieser Online-PDF nicht mehr zur Verfügung.

Bild 12-1: FPOOL-Realisierung im Zusammenhang mit der Übersetzung und dem Binden eines FOR1-Programms

12.2 Der zentrale FPOOL

Der zentrale FPOOL besteht aus der Objektmodulbibliothek FOR1.FPOOLLIB und der zugehörigen Datei der Schnittstellenbeschreibungen FOR1.FPOOL.

Im folgenden werden die in der FOR1.FPOOLLIB vorhandenen Funktionen in alphabetischer Reihenfolge dargestellt.
Die Beschreibung ist jeweils in folgende Abschnitte unterteilt:

- Aufrufname (im Titel)
- Generic-Name
- Connect-Name
- Schnittstellenbeschreibung
- Realisierung
- Aufrufbeispiel

Alle Funktionen werden mit einer CALL-Anweisung aufgerufen.

Wurde die Option COMOPT FPOOL beim Übersetzen angegeben, dann kann eine FPOOL-Funktion mit dem Aufrufnamen oder dem Generic-Namen aufgerufen werden. Der Aufrufname ist der Name, mit dem die Funktion im FPOOL gespeichert ist und mit dem die Schnittstellenüberprüfung vorgenommen wird.

Bei Verwendung eines Generic-Namens erkennt FOR1 den korrekten Aufrufnamen anhand der übergebenen Parameter (eine Zusammenstellung der Generic- und Aufrufnamen befindet sich in Tabelle 12-1 in Abschnitt 12.2.18).

FOR1 setzt den Aufrufnamen im Anschlußcode auf den in der FPOOL-Datei definierten Connect-Namen um. Dieser Connect-Name ist identisch mit dem Entrynamen des entsprechenden Objektmoduls.

Die für die Realisierung verwendeten Makro-Aufrufe sind in "Makroaufrufe an den Ablaufteil" [26] beschrieben.

Hinweis:

Sollen FPOOL-Funktionen verwendet werden, die Parameter des Datentyps INTEGER*1 haben, so muß bei der Übersetzung COMOPT FPOOL angegeben werden, da solche Parameter bei FPOOL-Funktionen anders übergeben werden als bei FOR1-Programmen.

12.2.1 FPOOL-Funktion ACCOUNTNR

ACCOUNTNR

Funktion: Anfordern der Taskabrechnungsnummer
Generic: TMODE
Connect: TMODACC

Schnittstelle

Parameterzahl: 2

1. Parameter
   Typ: CHARACTER*8
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf linksbündig die Abrechnungsnummer, ggf. mit Leerzeichen aufgefüllt.

2. Parameter
   Typ: CHARACTER*45
   Verwendung: SCRATCH
   Bedeutung: Arbeitsbereich der Funktion, Inhalt vor und nach Aufruf unterliegt keiner Regelung.

Realisierung

verwendeter System-Makro: TMODE
mehrfachbenutzbar

Beispiel:

```
Parameter:
   CHARACTER*8  ABRECHNUNGSNR
   CHARACTER*45 SCRATCH

spezifischer Aufruf:
   CALL ACCOUNTNR (ABRECHNUNGSNR,SCRATCH)

generischer Aufruf:
   CALL TMODE (ABRECHNUNGSNR,SCRATCH)
```

12.2.2 FPOOL-Funktion DIALOG

Bei Verwendung dieser Funktion muß mit COMOPT=FPOOL übersetzt werden, da Parameter des Datentyps INTEGER*1 bei FPOOL-Funktionen anders übergeben werden als bei FOR1-Programmen.

```
DIALOG
```

Funktion: Anfordern des Tasktyps
Generic: TMODE
Connect: DIALOG

Schnittstelle

Parameterzahl: 2

1. Parameter
   Typ: INTEGER*1
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den Tasktyp in Zahlenform. Dabei haben die einzelnen Werte folgende Bedeutung:

0	Stapelauftrag	
2	Dialogstation	8103
4	Datensichtgerät	8150
17	TRANSDATA	8418,8415
21	Datensichtgerät	8151
22	Datensichtgerät	8152
23	Schreibstation	8110
24	Datenstation	8161/54
25	Datenstation	8161/64
26	Datenstation	8161/80
44	Datenstation	8162
45	Datenstation	8160/80
53	Datenstation	9750

2. Parameter
   Typ: CHARACTER*10
   Verwendung: SCRATCH
   Bedeutung: Arbeitsbereich der Funktion, Inhalt vor und nach Aufruf unterliegt keiner Regelung.

Realisierung

verwendeter System-Makro: TMODE
mehrfachbenutzbar

Beispiel:

```
Parameter:
   INTEGER*1    TASKTYP
   CHARACTER*10 SCRATCH

spezifischer Aufruf:
   CALL DIALOG (TASKTYP,SCRATCH)

generischer Aufruf:
   CALL TMODE (TASKTYP,SCRATCH)
```

12.2.3 FPOOL-Funktion ELIMCHR

ELIMCHR

Funktion: Löschen eines Datensatzes durch Angabe des CHARACTER-Schlüssels
Generic: IDELETE
Connect: ELIMCHR

Schnittstelle

Parameterzahl: 3

1. Parameter
   Typ: INTEGER [1)]
   Verwendung: IN
   Bedeutung: Nummer der Ein-/Ausgabe-Einheit, mit der die Datei verknüpft sein muß

2. Parameter
   Typ: CHARACTER*n [2)]
   Verwendung: IN
   Bedeutung: CHARACTER-ISAM-Schlüssel des zu löschenden Satzes

3. Parameter
   Typ: INTEGER*4
   Verwendung: OUT
   Bedeutung: IOSTAT-Returncode, der über das Ergebnis des Delete-Aufrufes Auskunft gibt

[1)] beliebige Länge, jedoch muß bei Aufruf über den Connectnamen ohne Angabe von COMOPT FPOOL=FOR1.FPOOL der erste Parameter vom Typ INTEGER*4 sein.

[2)] Länge des zweiten Parameters (siehe "FOR1-Beschreibung" [21]), CHARACTER-ISAM-Schlüssel

Realisierung

verwendeter System-Makro: ELIM
mehrfachbenutzbar

Bei der Verwendung des 2.Parameters ist zu beachten, daß die Datei dem ISAM-Schlüssel entsprechend mit `ACCESS= 'DIRECT, CHARACTER'` eröffnet sein muß.

Beispiel:

```
        INTEGER*4   UNIT/20/
        CHARACTER*8 CKEY
        INTEGER*4   RETCODE

        OPEN (UNIT, ACCESS='DIRECT,C',FILE='DATEI')
        CKEY='AAAAAAAA'

spezifischer Aufruf:
        CALL ELIMCHR (UNIT,CKEY,RETCODE)

generischer Aufruf:
        CALL IDELETE (20,'AAAAAAAA',RETCODE)
```

12.2.4 FPOOL-Funktion ELIMINT

```
ELIMINT
```

Funktion: Löschen eines Datensatzes durch Angabe des INTEGER-Schlüssels
Generic: IDELETE
Connect: ELIMINT

Schnittstelle

Parameterzahl: 3

1. Parameter
   Typ: INTEGER [1)]
   Verwendung: IN
   Bedeutung: Nummer der Ein-/Ausgabe-Einheit, mit der die Datei verknüpft sein muß

2. Parameter
   Typ: INTEGER [1)]
   Verwendung: IN
   Bedeutung: INTEGER-ISAM-Schlüssel des zu löschenden Satzes

3. Parameter
   Typ: INTEGER*4
   Verwendung: OUT
   Bedeutung: IOSTAT-Returncode, der über das Ergebnis des Delete-Aufrufes Auskunft gibt

[1)] INTEGER beliebiger Länge, jedoch muß bei Aufruf über den Connectnamen ohne Angabe von COMOPT FPOOL=FOR1.FPOOL der Datentyp INTEGER*4 verwendet werden.

Realisierung

verwendeter System-Makro: ELIM
mehrfachbenutzbar

Bei der Verwendung des 2.Parameters ist zu beachten, daß die Datei dem ISAM-Schlüssel entsprechend mit `ACCESS= 'DIRECT,[,I]'` eröffnet sein muß.

Beispiel:

```
        INTEGER*4   UNIT/20/
        INTEGER*4   IKEY
        INTEGER*4   RETCODE

        OPEN (UNIT, ACCESS='DIRECT',FILE='SAMMLUNG')
        IKEY=5

spezifischer Aufruf:
        CALL ELIMINT (20,10*5+3, RETCODE)

generischer Aufruf:
        CALL IDELETE (UNIT,10*IKEY+3, RETCODE)
```

12.2.5 FPOOL-Funktion FCMD

FCMD

Funktion: Absetzen von BS2000-Kommandos
Generic: -
Connect: FP@CMD

Schnittstelle
Parameterzahl: 5

1. Parameter
   Typ: CHARACTER*1
   Verwendung: OUT
   Bedeutung: Enthält nach Aufruf den Fehler-Code (des SVC 88). Dabei haben die einzelnen Werte folgende Bedeutung:

0	BS2000-Kommando erfolgreich ausgeführt
1	Speicherplatzmangel
2	Speicheradressen ungültig
3	Systemmeldung wurde abgeschnitten
4	Fehler bei Ausführung des BS2000-Kommandos (z.B. BS2000-Kommando enthält ungültiges Format)
5	BS2000-Kommando entspricht nicht der BS2000-Syntax

2. Parameter
   Typ: CHARACTER*512
   Verwendung: IN
   Bedeutung: BS2000-Kommando (Länge max. 512)
   (z.B. /SET-FILE-LINK LINK-NAME=DSET20, FILE-NAME=DATEI.20, ACCESS-METHOD=SAM)

3. Parameter
   Typ: CHARACTER*1024
   Verwendung: OUT
   Bedeutung: Enthält nach Aufruf die Systemmeldung.

   Aufbau:

1 - 2	Länge des 3. Parameters
3 - 4	leer
5 - 1024	Systemmeldung

4. Parameter
   Typ: CHARACTER*1
   Verwendung: IN
   Bedeutung: Steuerung der Ausgabe der Systemmeldung.
   Mögliche Angaben Y/N

   Y Systemmeldungen werden nach SYSOUT und in das Ausgabefeld (3.Parameter) ausgegeben.
   N Systemmeldungen werden nur in das Ausgabefeld ausgegeben.

   Dieser Parameter hat nur eine Auswirkung bei einer BS2000 Version ≥ 8.0, für Versionen < 8.0 wird bei einer Angabe von N Y angenommen.

5. Parameter
   Typ: CHARACTER*1552
   Verwendung: SCRATCH
   Bedeutung: Arbeitsbereich der Funktion, Inhalt vor und nach Aufruf unterliegt keiner Regelung.

Realisierung

verwendeter System-Makro: CMD
mehrfachbenutzbar

Beispiel:

```
Parameter:

    CHARACTER CPAR1  * 1  /'0'/
    CHARACTER CPAR2  * 512      1)
    CHARACTER CPAR3  * 1024
    INTEGER   CPAR31 * 2
    CHARACTER CPAR32 * 2
    CHARACTER CPAR33 * 1020
    EQUIVALENCE (CPAR3,CPAR31),(CPAR3(3:4),CPAR32),(CPAR3(5:),CPAR33)
    CHARACTER CPAR4  * 1
    CHARACTER CPAR5  * 1552

    1) ACHTUNG:  (512,V) nicht erlaubt !


spezifischer Aufruf:

    CPAR2='/SET-FILE-LINK LINK-NAME=DSET20,FILE-NAME=DATEI.20,
   *       ACCESS-METHOD=SAM'
    CPAR4='Y' 2)
   [%FPOOL FOR1.FPOOL(FCMD)]
    CALL FCMD (CPAR1,CPAR2,CPAR3,CPAR4,CPAR5)

    2) Systemmeldung auf SYSOUT gewünscht

generischer Aufruf:
```

12.2.6 FPOOL-Funktion GDATECHAR

GDATECHAR

Funktion: Anfordern des Tagesdatums in Zeichenform
Generic: GDATE
Connect: GDATCHR

Schnittstelle

Parameterzahl: 5

1. Parameter
   Typ: CHARACTER*2
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den aktuellen Monat in Zeichenform, z.B. ergibt sich '09' für ein Tagesdatum 20.9.81.

2. Parameter
   Typ: CHARACTER*2
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den aktuellen Tag in Zeichenform, z.B. ergibt sich ’20’ für ein Tagesdatum 20.9.81.

3. Parameter
   Typ: CHARACTER*2
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf das aktuelle Jahr in Zeichenform, z.B. ergibt sich ’81’ für ein Tagesdatum 20.9.81.

4. Parameter
   Typ: CHARACTER*3
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den aktuellen Tag des Jahres in Zeichenform, z.B. ergibt sich ’263’ für ein Tagesdatum 20.9.81.

5. Parameter
   Typ: CHARACTER*12
   Verwendung: SCRATCH
   Bedeutung: Arbeitsbereich der Funktion, Inhalt vor und nach Aufruf unterliegt keiner Regelung.

Realisierung

verwendeter System-Makro: GDATE
mehrfachbenutzbar

Beispiel:

```
Parameter:
   CHARACTER*2  TAG,MONAT,JAHR
   CHARACTER*3  TAGDESJAHRES
   CHARACTER*12 SCRATCH

spezifischer Aufruf:
   CALL GDATECHAR (MONAT,TAG,JAHR,TAGDESJAHRES,SCRATCH)

generischer Aufruf:
   CALL GDATE (MONAT,TAG,JAHR,TAGDESJAHRES,SCRATCH)
```

12.2.7 FPOOL-Funktion GDATEINT

```
GDATEINT
```

Funktion: Anfordern des Tagesdatums in Zahlenform
Generic: GDATE
Connect: GDATINT

Schnittstelle

Parameterzahl: 5

1. Parameter
   Typ: INTEGER*4
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den aktuellen Monat in Zahlenform, z.B. ergibt sich die Zahl 9 für ein Tagesdatum 20.9.81.

2. Parameter
   Typ: INTEGER*4
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den aktuellen Tag in Zahlenform, z.B. ergibt sich die Zahl 20 für ein Tagesdatum 20.9.81.

3. Parameter
   Typ: INTEGER*4
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf das aktuelle Jahr in Zahlenform, z.B. ergibt sich die Zahl 81 für ein Tagesdatum 20.9.81.

4. Parameter
   Typ: INTEGER*4
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den aktuellen Tag des Jahres in Zahlenform, z.B. ergibt sich die Zahl 263 für ein Tagesdatum 20.9.81.

5. Parameter
   Typ: CHARACTER*31
   Verwendung: SCRATCH
   Bedeutung: Arbeitsbereich der Funktion, Inhalt vor und nach Aufruf unterliegt keiner Regelung.

Realisierung

verwendeter System-Makro: GDATE
mehrfachbenutzbar

Beispiel:

```
Parameter:
   INTEGER*4  TAG,MONAT,JAHR
   INTEGER*4  TAGDESJAHRES
   CHARACTER*31 SCRATCH

spezifischer Aufruf:
   CALL GDATEINT (MONAT,TAG,JAHR,TAGDESJAHRES,SCRATCH)

generischer Aufruf:
   CALL GDATE (MONAT,TAG,JAHR,TAGDESJAHRES,SCRATCH)
```

12.2.8 FPOOL-Funktion GEPRTCHAR

GEPRTCHAR

Funktion: Anfordern der bisher durch die Task verbrauchte CPU-Zeit in Zeichenform
Generic: GEPRT
Connect: GPRTCHR

Schnittstelle

Parameterzahl: 5

1. Parameter
   Typ: CHARACTER*2
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den Stundenanteil der CPU-Zeit in Zeichenform.
2. Parameter
   Typ: CHARACTER*2
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den Minutenanteil der CPU-Zeit in Zeichenform.
3. Parameter
   Typ: CHARACTER*2
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den Sekundenanteil der CPU-Zeit in Zeichenform.

4. Parameter
   Typ: CHARACTER*4
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den Sekundenbruchteil der CPU-Zeit in Zeichenform; Maßeinheit: Zehntausendstel-Sekunden.

5. Parameter
   Typ: CHARACTER*49
   Verwendung: SCRATCH
   Bedeutung: Arbeitsbereich der Funktion, Inhalt vor und nach Aufruf unterliegt keiner Regelung.

Realisierung

verwendeter System-Makro: TMODE
mehrfachbenutzbar

Beispiel:

```
Parameter:
   CHARACTER*2  STD,MIN,SEC
   CHARACTER*4  ZTDL
   CHARACTER*49 SCRATCH

spezifischer Aufruf:
   CALL GEPRTCHAR (STD,MIN,SEC,ZTDL,SCRATCH)

generischer Aufruf:
   CALL GEPRT (STD,MIN,SEC,ZTDL,SCRATCH)
```

12.2.9 FPOOL-Funktion GEPRTINT

```
GEPRTINT
```

Funktion: Anfordern der bisher durch die Task verbrauchte CPU-Zeit in Zahlenform
Generic: GEPRT
Connect: GPRTINT

Schnittstelle

Parameterzahl: 5

1. Parameter
   Typ: INTEGER*4
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den Stundenanteil der CPU-Zeit in Zahlenform

2. Parameter
   Typ: INTEGER*4
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den Minutenanteil der CPU-Zeit in Zahlenform

3. Parameter
   Typ: INTEGER*4
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den Sekundenanteil der CPU-Zeit in Zahlenform

4. Parameter
   Typ: INTEGER*4
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den Sekundenbruchteil der CPU-Zeit in Zahlenform; Maßeinheit: Zehntausendstel-Sekunden

5. Parameter
   Typ: CHARACTER*45
   Verwendung: SCRATCH
   Bedeutung: Arbeitsbereich der Funktion, Inhalt vor und nach Aufruf unterliegt keiner Regelung

Realisierung

verwendeter System-Makro: TMODE
mehrfachbenutzbar

Beispiel:

```
Parameter:
   INTEGER*4    STD,MIN,SEC,ZTDL
   CHARACTER*45 SCRATCH

spezifischer Aufruf:
   CALL GEPRTINT (STD,MIN,SEC,ZTDL,SCRATCH)

generischer Aufruf:
   CALL GEPRT (STD,MIN,SEC,ZTDL,SCRATCH)
```

12.2.10 FPOOL-Funktion GETDATE

GETDATE

Funktion: Anfordern des Tagesdatums in Zeichenform im ISO4-Format
Generic: -
CONNECT: FP@GTDAT

Schnittstelle

Parameterzahl: 2

1. Parameter
   Typ: CHARACTER*10
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf das aktuelle Datum in Zeichenform im ISO4-Format: `YYYY-MM-DD`.
2. Parameter
   Typ: CHARACTER*3
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den aktuellen Tag des Jahres.

Realisierung

verwendeter System-Makro: GDATE
mehrfachbenutzbar

Beispiel:

```
Parameter:
   CHARACTER*10  DATUM
   CHARACTER*3   TAG

spezifischer Aufruf:
   CALL GETDATE (DATUM,TAG)

generischer Aufruf:
```

12.2.11 FPOOL-Funktion GETMEMMAPLONG

```
GETMEMMAPLONG
```

Funktion: Anfordern des Memory Map, aus dem zu ersehen ist, welche Seiten des Klasse-5- und des Klasse-6-Speichers frei oder durch die Task belegt sind; es wird ein Benutzeradreßraum bis zu 8 Megabyte berücksichtigt.
Generic: GETMEMORYMAP
Connect: GTMAPL

Schnittstelle

Parameterzahl: 5

1. Parameter
   Typ: INTEGER*4
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf die Nummer der letzten Seite des Klasse-6-Speichers.

2. Parameter
   Typ: INTEGER*4
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf die Nummer der letzten Seite des Klasse-5-Speichers.

3. Parameter
   Typ: LOGICAL*1, DIMENSION 2048
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf eine Tabelle der Speicherseitenbelegung. Der Index eines Tabellenelements entspricht der Nummer einer Speicherseite. Ein Tabellenelement hat den Wert .TRUE., wenn die entsprechende Speicherseite von der Task belegt ist, andernfalls den Wert .FALSE..

4. Parameter
   Typ: CHARACTER*1
   Verwendung: OUT
   Bedeutung: Error-Code entsprechend der Rückinformation des Makros GTMAP: X'00' → in Ordnung, X'04',X'08' → Fehler

5. Parameter
   Typ: CHARACTER*260
   Verwendung: SCRATCH
   Bedeutung: Arbeitsbereich der Funktion, Inhalt vor und nach Aufruf unterliegt keiner Regelung.

Realisierung

verwendeter System-Makro: GTMAP
mehrfachbenutzbar

Beispiel:

```
Parameter:
   INTEGER*4     MAXPAGECL6,MAXPAGECL5
   LOGICAL*1     MEMTABLE
   DIMENSION     MEMTABLE(2048)
   CHARACTER*1   ERRCODE
   CHARACTER*260 SCRATCH

spezifischer Aufruf:
   CALL GETMEMMAPLONG (MAXPAGECL6,MAXPAGECL5,MEMTABLE,ERRCODE,SCRATCH)

generischer Aufruf:
   CALL GETMEMORYMAP (MAXPAGECL6,MAXPAGECL5,MEMTABLE,ERRCODE,SCRATCH)
```

12.2.12 FPOOL-Funktion GETMEMMAPSHORT

```
GETMEMMAPSHORT
```

Funktion: Anfordern des Memory Map, aus dem zu ersehen ist, welche Seiten des Klasse-6-Speichers frei oder durch die Task belegt sind; es wird ein Benutzeradreßraum bis zu einem Megabyte berücksichtigt.
Generic: GETMEMORYMAP
Connect: GTMAPS

Schnittstelle

Parameterzahl: 4

1. Parameter
   Typ: INTEGER*4
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf die Nummer der letzten Seite des Klasse-6-Speichers.

2. Parameter
   Typ: LOGICAL*1, DIMENSION 256
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf eine Tabelle der Speicherseitenbelegung. Der Index eines Tabellenelements entspricht der Nummer einer Speicherseite. Ein Tabellenelement hat den Wert .TRUE., wenn die entsprechende Speicherseite von der Task belegt ist, andernfalls den Wert .FALSE..

3. Parameter
   Typ: CHARACTER*1
   Verwendung: OUT
   Bedeutung: Error-Code entsprechend der Rückinformation des Makros GTMAP: X'00' → in Ordnung, X'04',X'08' → Fehler

4. Parameter
   Typ: CHARACTER*34
   Verwendung: SCRATCH
   Bedeutung: Arbeitsbereich der Funktion, Inhalt vor und nach Aufruf unterliegt keiner Regelung.

Realisierung

verwendeter System-Makro: GTMAP
mehrfachbenutzbar

Beispiel:

```
Parameter:
   INTEGER*4     MAXPAGECL6
   LOGICAL*1     MEMTABLE
   DIMENSION     MEMTABLE(256)
   CHARACTER*1   ERRCODE
   CHARACTER*34  SCRATCH

spezifischer Aufruf:
   CALL GETMEMMAPSHORT (MAXPAGECL6,MEMTABLE,ERRCODE,SCRATCH)

generischer Aufruf:
   CALL GETMEMORYMAP (MAXPAGECL6,MEMTABLE,ERRCODE,SCRATCH)
```

12.2.13 FPOOL-Funktion GETODCHAR

```
GETODCHAR
```

Funktion: Anfordern der Uhrzeit in Zeichenform
Generic: GETOD
Connect: GTODCHR

Schnittstelle
Parameterzahl: 4

1. Parameter
   Typ: CHARACTER*2
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den Stundenanteil der Uhrzeit in Zeichenform.

2. Parameter
   Typ: CHARACTER*2
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den Minutenanteil der Uhrzeit in Zeichenform.

3. Parameter
   Typ: CHARACTER*2
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den Sekundenanteil der Uhrzeit in Zeichenform.

4. Parameter
   Typ: CHARACTER*6
   Verwendung: SCRATCH
   Bedeutung: Arbeitsbereich der Funktion, Inhalt vor und nach Aufruf unterliegt keiner Regelung.

Realisierung

verwendeter System-Makro: GDATE
mehrfachbenutzbar

Beispiel:

```
Parameter:
   CHARACTER*2  STD,MIN,SEC
   CHARACTER*6  SCRATCH

spezifischer Aufruf:
   CALL GETODCHAR (STD,MIN,SEC,SCRATCH)

generischer Aufruf:
   CALL GETOD (STD,MIN,SEC,SCRATCH)
```

12.2.14 FPOOL-Funktion GETODINT

```
GETODINT
```

Funktion: Anfordern der Uhrzeit in Zahlenform
Generic: GETOD
Connect: GTODINT

Schnittstelle

Parameterzahl: 4

1. Parameter
   Typ: INTEGER*4
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den Stundenanteil der Uhrzeit in Zahlenform.

2. Parameter
   Typ: INTEGER*4
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den Minutenanteil der Uhrzeit in Zahlenform.

3. Parameter
   Typ: INTEGER*4
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den Sekundenanteil der Uhrzeit in Zahlenform.

4 . Parameter
   Typ: CHARACTER*23
   Verwendung: SCRATCH
   Bedeutung: Arbeitsbereich der Funktion, Inhalt vor und nach Aufruf unterliegt keiner Regelung.

Realisierung

verwendeter System-Makro: GDATE
mehrfachbenutzbar

Beispiel:

```
Parameter:
   INTEGER*4    STD,MIN,SEC
   CHARACTER*23 SCRATCH

spezifischer Aufruf:
   CALL GETODINT (STD,MIN,SEC,SCRATCH)

generischer Aufruf:
   CALL GETOD (STD,MIN,SEC,SCRATCH)
```

12.2.15 FPOOL-Funktion TASKANDUSERID

```
TASKANDUSERID
```

Funktion: Anfordern der Task Sequence Number (TSN) und der Benutzerkennung des LOGON-Kommandos in Zeichenform
Generic: TMODE
Connect: TMODTSN

Schnittstelle

Parameterzahl: 3

1. Parameter
   Typ: CHARACTER*4
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf die vierstellige Task Sequence Number ggf. mit führenden Nullen in Zeichenform.

2. Parameter
   Typ: CHARACTER*8
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf die Benutzerkennung aus dem LOGON-Kommando in Zeichenform.

3. Parameter
   Typ: CHARACTER*45
   Verwendung: SCRATCH
   Bedeutung: Arbeitsbereich der Funktion, Inhalt vor und nach Aufruf unterliegt keiner Regelung.

Realisierung

verwendeter System-Makro: TMODE
mehrfachbenutzbar

Beispiel:

```
Parameter:
   CHARACTER*4  PROZESSNR
   CHARACTER*8  KENNUNG
   CHARACTER*45 SCRATCH

spezifischer Aufruf:
   CALL TASKANDUSERID (PROZESSNR,KENNUNG,SCRATCH)

generischer Aufruf:
   CALL TMODE (PROZESSNR,KENNUNG,SCRATCH)
```

12.2.16 FPOOL-Funktion TMODEALL

Bei Verwendung dieser Funktion muß mit COMOPT=FPOOL übersetzt werden, da Parameter des Datentyps INTEGER*1 bei FPOOL-Funktionen anders übergeben werden als bei FOR1-Programmen.

```
TMODEALL
```

Funktion: Anfordern des Tasktyps, der Task Sequence Number (TSN), der Benutzerkennung aus dem LOGON-Kommando und der Taskabrechnungsnummer
Generic: TMODE
Connect: TMODALL

Schnittstelle

Parameterzahl: 5

1. Parameter
   Typ: INTEGER*1
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf den Tasktyp in Zahlenform. Dabei haben die einzelnen Werte folgende Bedeutung:

0	Stapelauftrag	
2	Dialogstation	8103
4	Datensichtgerät	8150
17	TRANSDATA	8418,8415
21	Datensichtgerät	8151
22	Datensichtgerät	8152
23	Schreibstation	8110
24	Datenstation	8161/54
25	Datenstation	8161/64
26	Datenstation	8161/80
44	Datenstation	8162
45	Datenstation	8160/80
53	Datenstation	9750

2. Parameter
   Typ: CHARACTER*4
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf die vierstellige Task Sequence Number ggf. mit führenden Nullen in Zeichenform.

3. Parameter
   Typ: CHARACTER*8
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf die Benutzerkennung aus dem LOGON-Kommando in Zeichenform.

4. Parameter
   Typ: CHARACTER*8
   Verwendung: OUT
   Bedeutung: enthält nach Aufruf linksbündig die Abrechnungsnummer, ggf. mit Leerzeichen aufgefüllt.

5. Parameter
   Typ: CHARACTER*45
   Verwendung: SCRATCH
   Bedeutung: Arbeitsbereich der Funktion, Inhalt vor und nach Aufruf unterliegt keiner Regelung.

Realisierung

verwendeter System-Makro: TMODE
mehrfachbenutzbar

Beispiel:

```
Parameter:
   INTEGER*1    PROZESSTYP
   CHARACTER*4  PROZESSNR
   CHARACTER*8  KENNUNG,ABRECHNUNGSNR
   CHARACTER*45 SCRATCH

spezifischer Aufruf:
   CALL TMODEALL (PROZESSTYP,PROZESSNR,KENNUNG,ABRECHNUNGSNR,SCRATCH)

generischer Aufruf:
   CALL TMODE (PROZESSTYP,PROZESSNR,KENNUNG,ABRECHNUNGSNR,SCRATCH)
```

12.2.17 FPOOL-Funktion MEMOMAP

```
MEMOMAP
```

Funktion: Anfordern von Informationen über Größe und Belegung des Klasse-6-Speichers oder des Memory Pools im Klasse-6-Speicher. Diese Informationen können bei einem Aufruf im 31-Bit-Adressierungsmodus oder bei einem Klasse-6-Speicher mit mehr als 8 Megabyte Speicherplatz nur durch MEMOMAP (ab BS2000 V9.0) erfragt werden.

Generic: -

Connect: FP@MINF

Schnittstelle

Parameterzahl: 6

1. Parameter
   - Typ: INTEGER*4
   - Verwendung: IN
   - Bedeutung:
     - 1 Informationen über den Klasse-6-Speicher erwünscht
     - 2 Informationen über den Memory Pool erwünscht

2. Parameter
   - Typ: INTEGER*4
   - Verwendung: IN
   - Bedeutung:
     - 1 Die virtuelle Seitennummer der ersten Speicherseite und die Anzahl der Speicherseiten des Klasse-6-Speichers bzw. des Memory Pools werden ausgegeben.
     - 2 Eine Tabelle über die Belegung der Speicherseiten wird ausgegeben.

3. Parameter
   - Typ: Feld vom Typ INTEGER*4
   - Dimension: 4
   - Verwendung: IN/OUT
   - Bedeutung: Die Bedeutung des 3. Parameters (im folgenden TAB genannt) hängt von den Werten des 1. und 2. Parameters ab:

(a) 1. Parameter=1, 2. Parameter=1 (Klasse-6-Speicherbelegung)

Der 3. Parameter ist in diesem Fall nur Ausgabe-Parameter.

TAB(1) enthält die virtuelle Seitennummer der 1. Speicherseite unterhalb 16 Megabyte

TAB(2) enthält die Anzahl der Speicherseiten unterhalb 16 Megabyte.

TAB(3)
- 0 kein Klasse-6-Speicher oberhalb 16 Megabyte vorhanden
- >0 virtuelle Seitennummer der 1. Speicherseite oberhalb 16 Megabyte

TAB(4)
- 0 kein Klasse-6-Speicher oberhalb 16 Megabyte vorhanden
- >0 Anzahl der Speicherseiten oberhalb 16 Megabyte

(b) 1. Parameter=2, 2. Parameter=1 (Memory-Pool-Belegung)

	Eingabe	Ausgabe
TAB(1)	virtuelle Seitennummer einer beliebigen Seite des gewünschten Memory-Pools. Mit dieser Angabe wird der Memory-Pool identifiziert.	virtuelle Seitennummer der 1. Seite des Memory-Pools
TAB(2)	nicht verwendet	Anzahl der Seiten des Memory Pools
TAB(3)	nicht verwendet	nicht verändert
TAB(4)	nicht verwendet	nicht verändert

(c) 1. Parameter=1, 2. Parameter=2 (Belegungstabelle des Klasse-6-Speichers)

	Eingabe	Ausgabe
TAB(1)	virtuelle Seitennummer der 1. Seite des Bereichs, für den eine Belegungstabelle gewünscht wird. Die angegebene Seitennummer muß ein ganzes Vielfaches von 16 sein.	nicht verändert
TAB(2)	Anzahl der Speicherseiten, für die eine Belegungstabelle gewünscht wird.	Anzahl der Speicherseiten, für die die Belegungstabelle tatsächlich beschrieben wird.
TAB(3)	nicht verwendet	nicht verändert
TAB(4)	nicht verwendet	nicht verändert

(d) 1. Parameter=2, 2. Parameter=2 (Belegungstabelle des Memory Pools)

	Eingabe	Ausgabe
TAB(1)	virtuelle Seitennummer der 1. Seite des Bereichs, für den eine Belegungstabelle gewünscht wird. Die angegebene Seitennummer muß innerhalb des gewünschten Memory-Pools liegen und ein Vielfaches von 16 sein.	nicht verändert
TAB(2)	Anzahl der Speicherseiten, für die eine Belegungstabelle gewünscht wird.	Anzahl der Speicherseiten, für die die Belegungstabelle tatsächlich beschrieben wird.
TAB(3)	nicht verwendet	nicht verändert
TAB(4)	nicht verwendet	nicht verändert

Falls der 1. Parameter=1 gewählt wird, werden alle Speicherseiten eines Memory-Pools als belegt gekennzeichnet. Falls der 1. Parameter=2 gewählt wird, werden nur die Speicherseiten als belegt gekennzeichnet, die mit REQMP angefordert waren.

4. Parameter
   Typ: INTEGER*4
   Verwendung: IN
   Bedeutung: Der 4. Parameter legt die Größe der angeforderten Tabelle der Speicherseitenbelegung fest (siehe 5. Parameter).

   Zulässige Werte: 1 ≤ 4. Parameter ≤ 256 Der Parameter wird auch mit PTSIZE abgekürzt.

5. Parameter
   Typ: Feld vom Typ LOGICAL*1
   Dimension (PTSIZE*2048)
   Verwendung: OUT
   Bedeutung: Der 5. Parameter enthält nach Aufruf eine Tabelle der Speicherbelegung. Die Dimension des 5. Parameters soll das 2048-fache des 4. Parameters sein. Der Index eines Tabellenelements entspricht der Nummer einer Speicherseite - (Nummer der in TAB(1) angegebenen Seite -1). Ein Tabellenelement hat den Wert .TRUE., wenn die entsprechende Speicherseite von der Task belegt ist, andernfalls den Wert .FALSE..

6. Parameter
   Typ: INTEGER*4
   Verwendung: OUT
   Bedeutung: Der 6. Parameter enthält den Rückkehrcode des Makros MINF:
   X'00' Funktion ausgeführt
   X'01' ungültige Eingabe für den 1. Parameter
   X'02' ungültige Eingabe für den 2. Parameter
   X'04' Operandenfehler
   X'08' ungültige virtuelle Seitennummer
   X'0C' Adressenfehler

Realisierung

verwendeter System-Makro: MINF
mehrfachbenutzbar

Beispiel:

```
Parameter:
   INTEGER*4     INFORM1, INFORM2, PAGETAB(4),
  *              PTSIZE, ERRCODE
   LOGICAL*1     MEMTABLE
   PARAMETER     (PTSIZE=10)
   INFORM1 = 1
   INFORM2 = 1
   DIMENSION     MEMTABLE(PTSIZE*2048)

spezifischer Aufruf:
   CALL MEMOMAP (INFORM1,INFORM2,PAGETAB,PTSIZE,MEMTABLE,ERRCODE)

generischer Aufruf:
```

12.2.18 Übersicht: Generic-, Call- und Connect-Namen

Im Abschnitt zum zentralen FPOOL wurde die Verwendung von Generic-Namen für FPOOL-Unterprogramme angesprochen. Aus der folgenden Übersicht kann - abhängig von Zahl, Typ und Stellung der Parameter - die Verknüpfung zwischen Generic-Namen und dem spezifischen Unterprogrammnamen sowie dem Connect-Namen ersehen werden.

Generic-Name	Parameter Zahl	Parameter Nr: Typ	Spezifisches Unterprogramm (Aufrufname)	Connect-Name	Modul-Name
-	5	1: CHARACTER*1 2: CHARACTER*512 3: CHARACTER*1024 4: CHARACTER*1 5: CHARACTER*1552	FCMD	FP@CMD	FP@CMD
GDATE	5	1: CHARACTER*2 2: CHARACTER*2 3: CHARACTER*2 4: CHARACTER*3 5: CHARACTER*12	GDATECHAR	GDATCHR	FP@GDCHR
	5	1: INTEGER*4 2: INTEGER*4 3: INTEGER*4 4: INTEGER*4 5: CHARACTER*31	GDATEINT	GDATINT	FP@GDINT
GEPRT	5	1: CHARACTER*2 2: CHARACTER*2 3: CHARACTER*2 4: CHARACTER*4 5: CHARACTER*49	GEPRTCHAR	GPRTCHR	FP@GPCHR
	5	1: INTEGER*4 2: INTEGER*4 3: INTEGER*4 4: INTEGER*4 5: CHARACTER*45	GEPRTINT	GPRTINT	FP@GPINT
-	2	1: CHARACTER*10 2: CHARACTER*3	GETDATE	FP@GTDAT	FP@GTDAT
GETMEMORYMAP	5	1: INTEGER*4 2: INTEGER*4 3: LOGICAL*1(2048) 4: CHARACTER*1 5: CHARACTER*260	GETMEMMAPLONG	GTMAPL	FP@GTAPL
	4	1: INTEGER*4 2: LOGICAL*1(256) 3: CHARACTER*1 4: CHARACTER*34	GETMEMMAPSHORT	GTMAPS	FP@GTAPS
GETOD	4	1: CHARACTER*2 2: CHARACTER*2 3: CHARACTER*2 4: CHARACTER*6	GETODCHAR	GTODCHR	FP@GTCHR
	4	1: INTEGER*4 2: INTEGER*4 3: INTEGER*4 4: CHARACTER*23	GETODINT	GTODINT	FP@GTINT

Fortsetzung▷

Fortsetzung

Generic-Name	Parameter Zahl	Parameter Nr: Typ	Spezifisches Unterprogramm (Aufrufname)	Connect-Name	Modul-Name
IDELETE	3	1: INTEGER 2: CHARACTER*n 3: INTEGER*4	ELIMCHR	ELIMCHR	FP@ELM
	3	1: INTEGER 2: INTEGER 3: INTEGER*4	ELIMINT	ELIMINT	FP@ELM
-	6	1: INTEGER*4 2: INTEGER*4 3: INTEGER*4 4: INTEGER*4 5: LOGICAL*1 6: INTEGER*4	MEMOMAP	FP@MINF	FP@MINF
TMODE	2	1: CHARACTER*8 2: CHARACTER*45	ACCOUNTNR	TMODEACC	FP@TMACC
	2	1: INTEGER*1 2: CHARACTER*10	DIALOG	DIALOG	FP@DLOG
	3	1: CHARACTER*4 2: CHARACTER*8 3: CHARACTER*45	TASKANDUSERID	TMODTSN	FP@TMTSN
	5	1: INTEGER*1 2: CHARACTER*4 3: CHARACTER*8 4: CHARACTER*8 5: CHARACTER*45	TMODEALL	TMODALL	FP@TMALL

Tab. 12-1: Generic-Namen, Connect-Namen und spezifische Namen von FPOOL-Unterprogrammen

12.3 Die Einrichtung privater FPOOLs (Dienstprogramm FPOOLITY)

Mit Hilfe des Dienstprogramms FPOOLITY kann der Anwender für seine Unterprogramme eigene FPOOL-Dateien erstellen und bearbeiten (vgl. "FPOOLITY" [22]).
Eine Schnittstellenbeschreibung wird in eine neu zu erstellende oder zu erweiternde FPOOL-Datei mittels der FPOOLITY-Funktion GENERATE und deren Function-Description-Language (FDL) eingetragen.

Die FPOOL-Bearbeitung des FOR1-Compilers erstreckt sich zur Zeit noch nicht auf alle in "FPOOLITY" [22] beschriebenen FDL-Angaben. Die folgende Übersicht zeigt den von FOR1 interpretierten Ausschnitt der FDL-Schnittstellenbeschreibung:

```
CALL NAME              :   name1                  ;
GENERIC                :   sohn1,sohn2,...,sohn20;
CONNECT NAME           :   name2                  ;
CONNECT MODE           :   STANDARD               ;
IMPL-LANGUAGE          :   FORTRAN                ;
ENVIRONMENT            :   FORTRAN                ;
FOR                    :   FORTRAN                ;
RETURNS                :   typname2               ;
P#n KEYWORD            :   parametername          ;
[P#n] DIRECTION        :   IN|OUT|INOUT|SCRATCH   ;
[P#n] MODE             :   REFERENCE              ;
[P#n] TYPE             :   typname1               ;
FDLEND    ;
```

name1	Aufrufname des Unterprogramms, maximal 15 Zeichen lang.
sohnx	CALL NAME eines FPOOL-Eintrags.
name2	FOR1 ersetzt im Objektcode den Aufrufnamen *name1* durch den CONNECT-Namen *name2*. Der CONNECT-Name muß identisch sein mit dem Entrynamen des Objektmoduls. Der CONNECT-Name darf maximal 8 Zeichen lang sein.
P#n	n-ter Parameter der Übergabeliste in der CALL-Anweisung. Bis zu 30 Parameter können angegeben werden.
parametername	Name des n-ten Parameters.
typname1	Datentyp (siehe Tab. 12-2)
typname2	Datentyp des RETURN-Werts (siehe Tab. 12-3)

Den Zusammenhang zwischen FDL-Typnamen und FORTRAN-Datentypen bzw. FORTRAN-RETURN-Werten zeigen die folgende Tabellen:

FDL (TYPE:)	FORTRAN-Datentyp typname1
CHAR[ACTER]	CHARACTER
CHAR-STRING * n	CHARACTER * n (0 ≤ n ≤ 32000)
COMPLEX [*{8|16|32}]	COMPLEX *{8|16|32}
INTEGER [*{1|2|4|8}]	INTEGER *{1|2|4|8}
LOGICAL	LOGICAL * 1
REAL [*{4|8|16}]	REAL *{4|8|16}
SIGNED-INTEGER [*{1|2|4|8}]	INTEGER *{1|2|4|8}

Alle anderen FDL-Angaben werden von FOR1 als Kommentar betrachtet.

Tab. 12-2: FDL-Typnamen und FORTRAN-Datentypen

FDL (RETURNS:)	Datentyp des RETURN-Werts typname2
CHAR[ACTER]	CHARACTER
CHAR-STRING * n	CHARACTER * n (0 ≤ n ≤ 4)
INTEGER [*{1|2|4}]	INTEGER *{1|2|4}
LOGICAL	LOGICAL * 1
REAL [*4]	REAL *4
SIGNED-INTEGER [*{1|2|4}]	INTEGER *{1|2|4}

Tab. 12-3: FDL- und FORTRAN-RETURN-Werte

Einschränkungen

Die Überprüfung von Schnittstellen zu FORTRAN-Unterprogrammen unterliegt einigen Einschränkungen:

- Bei Funktionen werden nur die Parameter, nicht jedoch Typ und Länge der Funktion überprüft.
- Funktionen vom Datentyp REAL*{8|16}, INTEGER*8, CHARACTER*n (n>5) und COMPLEX*{8|16|32} können nicht überprüft werden.
- Namen von SUBROUTINEs, FUNCTIONs und LABELs als Parameter werden nicht überprüft.
- Felder werden nicht überprüft.
- Datenelemente und Funktionen vom Typ LOGICAL*4 können nicht überprüft werden.

- Bei CHARACTER-Datenelementen wird nur ein Fehler (SEVERE ERROR) ausgegeben, wenn die Länge des Aktualparameters kleiner als die des Formalparameters ist.
- Konversionen werden durchgeführt,
  - falls sie möglich sind (z.B. Aktualparameter vom Typ REAL, Formalparameter vom Typ INTEGER) und
  - im FPOOL "DIRECTION IN" eingetragen ist (d.h., wenn es sich um reine Eingabeparameter handelt und wenn nur solche fehlerhaft sind).

  In diesem Fall wird eine WARNING (SA151) ausgegeben. In allen anderen Fällen wird nicht konvertiert und ein SEVERE ERROR ausgegeben.

Beispiel: FDL-Eintrag

Für ein Unterprogramm mit folgenden Anweisungen

```
SUBROUTINE ABWEICH (N,A,WERT)
REAL A (2:101)
REAL*8 WERT
```

sieht der FDL-Eintrag folgendermaßen aus:

```
CALL NAME:           ABWEICH ;
CONNECT NAME:        ABWEICH ;
CONNECT MODE:        STANDARD ;
IMPL-LANGUAGE:       FORTRAN ;
ENVIRONMENT:         FORTRAN ;
FOR:                 FORTRAN ;
P#  1 KEYWORD:     N ;
      DIRECTION:   INOUT ;
      MODE:        REFERENCE ;
      TYPE:        INTEGER *4 ;
P#  2 KEYWORD:     A ;
      DIRECTION:   INOUT ;
      MODE:        REFERENCE ;
      TYPE:        REAL *4
                   ARRAY(100) ;
P#  3 KEYWORD:     WERT ;
      DIRECTION:   INOUT ;
      MODE:        REFERENCE ;
      TYPE:        REAL *8 ;
FDLEND ;
```

12.4 Beispiel: Anwendung von FOR1.FPOOLLIB-Schnittstellen

Im folgenden wird die Anwendung von FOR1.FPOOLLIB-Schnittstellen gezeigt: Abgedruckt sind:

- Quellprogramm
- Kommandos/Steueranweisungen zum Übersetzen, Binden, Ablauf
- Ergebnisprotokoll

1. Quellprogramm

```
      PROGRAM BEISP
      IMPLICIT CHARACTER *2 (C-D)
      IMPLICIT INTEGER   *4 (I,J)
      INTEGER   *1   I1,I2,I3
      INTEGER   *1   UT /2     /
      LOGICAL   *1   L1(2048),L2(256)
      LOGICAL   *1   Q  /.TRUE./
      CHARACTER *260 SCR260
      CHARACTER *49  SCRATCH
      CHARACTER *8   C81,C82,C83,C84,D8(2:4)
      CHARACTER *4   C41,C42,C43,D4(2:3)
      CHARACTER *3   C31,D31
      CHARACTER *1   C1,C2
      CHARACTER *26  T(0:53)
      DIMENSION      I4(4)
      DIMENSION      D2(3)
      DATA T(0)  /'STAPELPROZESS'/,
     .     T(2)  /'DIALOGSTATION    8103'/,
     .     T(4)  /'DATENSICHTGERAET 8150'/,
     .     T(17) /'TRANSDATA        8418,8415'/,
     .     T(21) /'DATENSICHTGERAET 8151'/,
     .     T(22) /'DATENSICHTGERAET 8152'/,
     .     T(23) /'SCHREIBSTATION   8110'/,
     .     T(24) /'DATENSTATION     8161/54'/,
     .     T(25) /'DATENSTATION     8161/64'/,
     .     T(26) /'DATENSTATION     8161/80'/,
     .     T(44) /'DATENSTATION     8162'/,
     .     T(45) /'DATENSTATION     8160/80'/,
     .     T(53) /'DATENSTATION     9750'/
      DATA (T(I),I=5,16),
     .     (T(I),I=18,20),
     .     (T(I),I=27,43),
     .     (T(I),I=46,52),
     .      T(1),T(3) /41* 'GERAET UNDEFINIERT'/
*
      CALL DIALOG(I1,SCRATCH(1:10))
         IF (I1 .EQ. 0) UT = 6
      WRITE (UT,1) I1,T(I1)
  1   FORMAT (' DIALOG:            ',I2,', D.H. ',A)
*
      CALL ACCOUNTNR(C81,SCRATCH(:45))
      WRITE (UT,2) C81
  2   FORMAT (' ABRECHNUNGSNUMMER: ',A)
*
      CALL GDATECHAR(C21,C22,C23,C31,SCRATCH(:12))
```

```
      WRITE (UT,3) C21,C22,C23,C31

  3   FORMAT (' DATUM (CHAR):      ',4(A,:', '))
*
      CALL GDATEINT(I41,I42,I43,I44,SCRATCH(:31))
      WRITE (UT,4) I41,I42,I43,I44
  4   FORMAT (' DATUM (INT):       ',3(I2,', '),I3)
*
      CALL GEPRTCHAR(C24,C25,C26,C41,SCRATCH(:12))
      WRITE (UT,5) C24,C25,C26,C41
  5   FORMAT (' CPUZEIT (CHAR):    ',4(A,:', '))
*
      CALL GEPRTINT(I45,I46,I47,I48,SCRATCH(:45))
      WRITE (UT,6) I45,I46,I47,I48
  6   FORMAT (' CPUZEIT (INT):     ',3(I2,', '),I4)
*
      CALL GETODCHAR(C27,C28,C29,SCRATCH(:6))
      WRITE (UT,7) C27,C28,C29
  7   FORMAT (' TAGESZEIT (CHAR):  ',3(A,:', '))
*
      CALL GETODINT(I49,I4A,I4B,SCRATCH(:23))
      WRITE (UT,8) I49,I4A,I4B
  8   FORMAT (' TAGESZEIT (INT):   ',3(I2,:', '))
*
      CALL GETMEMMAPLONG(J1,J2,L1,C1,SCR260)
      DO 9 I = 2047,1,-1
  9      IF (L1(I).NEQV.L1(2048)) GOTO 10
 10   WRITE (UT,11) J1,J2,:L1(I)
 11   FORMAT (' MEMORYMAPLONG:     ',2(I9,', '),32(:/,10X,64L1))
      WRITE (UT,12) L1(I+1),C1
 12   FORMAT (10X,'ALLE WEITEREN ',L1,', ',Z2)
*
      CALL GETMEMMAPSHORT(J3,L2,C2,SCRATCH(:34))
      WRITE (UT,13) J3,L2,C2
 13   FORMAT (' MEMORYMAPSHORT:    ',I9,', ',4(/,10X,64L1),
     .        ', ',/,10X,Z2)
*
      CALL TASKANDUSERID(C42,C82,SCRATCH(2:46))
      WRITE (UT,14) C42,C82
 14   FORMAT (' TASKANDUSERID:     ',2(A,:', '))
*
      CALL TMODEALL(I2,C43,C83,C84,SCRATCH(3:47))
      WRITE (UT,15) I2,C43,C83,C84
 15   FORMAT (' TMODEALL:          ',I3,', ',3(A,:', '))
      IF (I1.NE.I2.OR.
     .    C42.NE.C43.OR.
     .    C82.NE.C83.OR.
     .    C81.NE.C84) THEN
         Q = .FALSE.
         WRITE (UT,16)
 16      FORMAT (' FEHLER NACH TMODEALL')
         END IF
*

      CALL TMODE(D8(1),SCRATCH(:45))
      IF (C81.NE.D8(1)) THEN
         Q = .FALSE.
         WRITE (UT,17)
 17      FORMAT (' FEHLER BEI ABRECHNUNGSNUMMER = TMODE')
```

```
         END IF
*
      CALL TMODE(I3,SCRATCH(1:10))
      IF (I1.NE.I3) THEN
         Q = .FALSE.
         WRITE (UT,18)
 18      FORMAT (' FEHLER BEI DIALOG = TMODE')
         END IF
*
      CALL GDATE(D2(1),D2(2),D2(3),D31,SCRATCH(:12))
      IF (C21.NE.D2(1).OR.
     .    C22.NE.D2(2).OR.
     .    C23.NE.D2(3).OR.
     .    C31.NE.D31) THEN
         Q = .FALSE.
         WRITE (UT,19)
 19      FORMAT (' FEHLER BEI GDATECHAR = GDATE')
         END IF
*
      CALL GDATE(I4(1),I4(2),I4(3),I4(4),SCRATCH(:31))
      IF (I41.NE.I4(1).OR.
     .    I42.NE.I4(2).OR.
     .    I43.NE.I4(3).OR.
     .    I44.NE.I4(4)) THEN
         Q = .FALSE.
         WRITE (UT,20)
 20      FORMAT (' FEHLER BEI GDATEINT = GDATE')
         END IF
*
      CALL TMODE(D4(2),D8(2),SCRATCH(2:46))
      IF (D4(2) .NE.C42.OR.
     .    D8(2).NE.C82) THEN
         Q = .FALSE.
         WRITE (UT,21)
 21      FORMAT (' FEHLER BEI TASKANDUSERID = TMODE')
         END IF
*
      CALL TMODE(I2,D4(3),D8(3),D8(4),SCRATCH(3:47))
      IF (I1.NE.I2.OR.
     .    C42.NE.D4(3).OR.
     .    C82.NE.D8(3).OR.
     .    C81.NE.D8(4)) THEN
         Q = .FALSE.
         WRITE (UT,22)
 22      FORMAT (' FEHLER BEI TMODEALL = TMODE')
         END IF
*
      IF (Q) THEN
         WRITE (2,31)
         WRITE (6,31)
      ELSE
         WRITE (2,32)
         WRITE (6,32)
         END IF
 31   FORMAT (' FPOOLBEISP SUCCESSFUL')
 32   FORMAT (' FPOOLBEISP FAILURE   ')
      STOP
      END
```

2. Übersetzung

```
/START-PROG $FOR1
 *COMOPT SRC=SRC.FPOOL,FPOOL=$TSOS.FOR1.FPOOL,END
```

Dabei sind:

```
SRC.FPOOL          Quellprogrammdatei
$TSOS.FOR1.FPOOL   FPOOL-Datei
```

3. Binden

```
/START-PROG $TSOSLNK
 PROG FPOOL,FILENAM=L.FPOOL
 INCLUDE *
 RESOLVE ,$TSOS.FOR1MODLIBS
 RESOLVE ,$TSOS.FOR1.FPOOLLIB
 END
```

Dabei sind:

```
L.FPOOL               Dateiname des ablauffähigen Programms
$TSOS.FOR1MODLIBS     FOR1-Modulbibliothek
$TSOS.FOR1.FPOOLLIB   FPOOL-Objektmodulbibliothek
```

4. Ablauf

```
/SET-TASKLIB $TSOS.FOR1MODLIBS
/START-PROG L.FPOOL
```

Auf der Datensichtstation wird folgendes Protokoll ausgegeben:

```
                % BLS0500 PROGRAM 'FPOOL', VERSION ' ' OF '91-08-20' LOADED
                BS2000  F O R 1 : FORTRAN PROGRAM "BEISP"
                STARTED ON 1991-08-20 AT 16:55:49
                DIALOG:            53, D.H. DATENSTATION     9750
                ABRECHNUNGSNUMMER: ABRECH03
                DATUM (CHAR):      08, 20, 91, 232
                DATUM (INT):        8, 20, 91, 232
                CPUZEIT (CHAR):    00, 00, 20, 7920
                CPUZEIT (INT):      0,  0, 20, 7946
                TAGESZEIT (CHAR):  16, 55, 49
                TAGESZEIT (INT):   16, 55, 49
                MEMORYMAPLONG:          2047,       3583,
                         TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
                         TTTTTTTTTTTTTTTTTTTTTT
                         ALLE WEITEREN F, 00
                MEMORYMAPSHORT:          239,
                         TTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTTT
                         TTTTTTTTTTTTTTTTTTTTTTTFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
                         FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
                         FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF,
                         00
                TASKANDUSERID:     9FIK, K3790052
                TMODEALL:           53, 9FIK, K3790052, ABRECH03
                FPOOLBEISP SUCCESSFUL
                STOP AT STMT 112 IN BEISP

FPOOLBEISP SUCCESSFUL
                BS2000  F O R 1 : FORTRAN PROGRAM "BEISP   " ENDED PROPERLY AT 16:55:55
                CPU - TIME USED :       0.0514 SECONDS
                ELAPSED TIME    :       5.9670 SECONDS
```

A Anhang

A.1 Kurzformen für FOR1-Compileroptionen und Optionswerte

Es gibt zwei Möglichkeiten, um Namen von Compileroptionen bzw. Optionswerten abzukürzen:

1. Von rechts beginnend können beliebig viele Buchstaben weggelassen werden, solange der Name noch eindeutig bleibt.

   Beispiel:

   - statt LIST LIS oder LI
   - statt LISTFILE LISTFIL oder LISTFI oder LISTF

2. Es gelten besondere Vereinbarungen:

   - Abkürzungen bei zusammengesetzten Namen, z.B. statt LISTFILE: LF
   - Zuordnung eigentlich mehrdeutiger Abkürzungen, z.B. statt LIST: L

Die folgenden Zusammenstellungen zeigen Kurzformen, die durch Abschneiden von rechts entstanden sind oder als so vereinbart gelten.

Kurzformen der Compileroptionen:

```
CCOM                CC
CODE                CO
COLLECT             CL
COMPATIBLE          COMPAT, COM
DIALOG              D
NODIALOG            ND
DIALOG-SAVE         DIALOG-
EJECT               EJ
ERRKILL             EK
EXPAND              EX
EXPUNDERFLOW        EU
FORTRAN90-CHECK     F90
FPOOL               F
GEN                 G
IMPLICIT            IM
INCLUDE-LIBRARY     INC
LANGUAGE            LA
LINECNT             LC
```

LINEEND	LE
LINKAGE	LNK
LIST	L
LISTFILE	LF
LIST-OUTPUT	LIST-
MAXERR	ME
MODULE-LIBRARY	MOD
MSGLEVEL	MSG
OBJECT	OBJ, O
OPTIMIZE	OPT
OPTIONS	OPTIO
OUTPUT	OU
PAD	P
PROCEDURE-OPTIMIZATION	PR
REAL	R
SAVE-CONSTANT	SAV
SHARE-LIBRARY	SH
SOURCE	SRC
SOURCE-FORMAT	SF
STANDARD-CHECK	STD
SUPPLIEDBOUND	SUP, SB
SYMTEST	SYM
TESTOPT	TO
TEXT-SEPARATOR	TEX
TRUNCONST	TC
UNIT	U
UPDATE	UPD

Kurzformen der TESTOPT-Optionswerte:

ALL	A
ARG	AR
BOUNDS	B
CNTRL	C
DEBUG	D
STNR	STN
STRING	STR
SUBSCR	SU
UNDEF	U

Kurzformen der LIST-Optionswerte:

ALL	A
ATR	AT
CHANGE	CH
DIAG	D
DECOMP	DE
ESD	E
LIST	L
MAP	M
MIN	MI
NONE	N
OBJECT	O
OPTIONS	OP
SOURCE	SRC

```
SUMMARY              S
XREF                 X
```

Kurzformen der OBJECT-Optionswerte:

```
SHARE                S
```

Kurzformen der UNIT-Optionswerte:

```
PUNCH                PU
PRINT                PR
READ                 R
WRITE                W
```

Kurzformen der MSGLEVEL-Optionswerte:

```
DIAG                 D
ERROR                E
NOTE                 N
SOURCE               S, SRC
WARNING              W
```

Kurzformen der COMPATIBLE-Optionswerte:

```
BGFOR                BGF
BS3FOR               BS3
```

Die Abkürzungsregeln für SDF-Operanden werden in Abschnitt 2.2.1 dargestellt.

A.2 Übersetzungsphasen

Der Übersetzungsvorgang ist in mehrere Phasen unterteilt, die je Programmeinheit einmal durchlaufen und einzeln in den virtuellen Speicher geladen werden.
Die Phasen tauschen untereinander über virtuelle Arbeitsdateien Informationen aus; bei einem Überlauf erfolgt der Informationsaustausch über externe temporäre Arbeitsdateien. Das Zusammenspiel der einzelnen Phasen wird vom Compiler Management gesteuert (siehe Bild A.2-1).

Die Phasen des Compilers besitzen folgende Funktionen:

Compilerinitialisierung
Einlesen der Optionen, Öffnen der Dateien, u.a.

Formale Analyse 1
Einlesen des Quellprogramms, Klassifikation der Anweisungen, lexikalische und syntaktische Analyse der Spezifikationsanweisungen.
Aufbau der Konstanten- und Symboltabellen.

Formale Analyse 2
Erkennen der Definition von Formelfunktionen, lexikalische und syntaktische Analyse der ausführbaren Anweisungen, weiterer Aufbau der Konstanten- und Symboltabellen.

Semantische Analyse 1
Semantische Prüfung und Detaillierung von Spezifikationsanweisungen.

Semantische Analyse 2
Semantische Prüfung und Detaillierung der ausführbaren Anweisungen.

Globale Optimierung
Durchführung der Optimierungsmaßnahmen.

Code-Generierung
Zuordnen von Adressen an alle Variablen, Konstanten und temporäre Hilfsgrößen, Zuordnen der Register an Operanden, Erzeugen der Befehlsfolgen, Zuordnen von Adressen an alle Sprungmarken.

Compilerausgabe
Zuordnen der Distanzen und Adressen zu Vorwärtssprüngen, Erzeugen der LSD-, TXT-, RLD- und ESD-Sätze (siehe "Systemkonventionen" [39]), Erzeugen der Standardprotokollisten und der zusätzlichen Listen, Ausgabe der zusammenfassenden Meldungen des Übersetzungsvorgangs.

Für jede Programmeinheit wird der Übersetzungsvorgang mit der Compilerinitialisierung neu gestartet.

Das Bild steht in dieser Online-PDF nicht mehr zur Verfügung.

Bild A.2-1: Phasen des FOR1-Compilers

A.3 Namenskonvention bei Bibliotheksmoduln

Die Namen und ENTRY-Namen der mathematischen Bibliotheksmoduln und der FPOOL-Moduln beginnen grundsätzlich mit IF@ bzw. FP@. Der Name bzw. ENTRY-Name eines mathematischen Bibliotheksmoduls oder einer FPOOL-Routine ist somit nicht identisch mit dem in FORTRAN definierten Namen der Funktion. So hat beispielsweise der Bibliotheksmodul, der durch den INTRINSIC-Namen ABS angesprochen wird, den Namen IF@ABS, der Bibliotheksmodul der durch den INTRINSIC-Namen DSIN angesprochen wird, den Namen IF@DS.

Da das Zeichen "@" nicht zum FORTRAN-Zeichensatz gehört, wird hierdurch eine unbeabsichtigte Namensgleichheit mit selbstdefinierten Unterprogrammen ausgeschlossen. Dies ist insbesondere deshalb wichtig, da manche mathematischen Bibliotheksmoduln ihrerseits wiederum andere Bibliotheksmoduln aufrufen. Durch diesen prinzipiellen Unterschied ist gewährleistet, daß bei solchen "internen" Funktionsaufrufen auf die benötigten Bibliotheksfunktionen zugegriffen wird, und nicht auf eventuell vorhandene namensgleiche Anwender-Routinen.

Vorsicht ist jedoch geboten bei der Verwendung von Namen die mit "ITS" oder "IT0" beginnen, da einige Laufzeitsystemsmoduln mit diesen Zeichenkombinationen beginnen. Eine unbeabsichtigte Namensgleichheit ist hier - im Gegensatz zu mathematischen Bibliotheksfunktionen und FPOOL-Funktionen - *nicht* ausgeschlossen.

A.4 PARAMETER-Operanden und entsprechende Compileroptionen

Das PARAMETER-Kommando ist aus Kompatibilitätsgründen auch für den FOR1-Compiler wirksam. Es kann zur Angabe von Übersetzungsoperanden und von Operanden zur Steuerung des Programmablaufs verwendet werden. Der Anwender sollte Übersetzungsoperanden jedoch vorzugsweise in COMOPT-Anweisungen oder als SDF-Operanden angeben, die wesentlich mehr Auswahlmöglichkeiten bieten.

In zwei Fällen gibt es zu Funktionen von PARAMETER-Operanden keine entsprechenden RUNOPT-Optionen sondern nur entsprechende SDF-Operanden:

– CARD=YES zur Eingabe von Laufzeitoptionen (RUNOPTs, siehe Abschnitt 6.3.1).
  Entsprechender SDF-Operand: RUNTIME-OPTIONS=YES().

– DEBUG=YES zur Programmfortsetzung bei Laufzeitfehlern im Stapelbetrieb (siehe Abschnitt 6.5.2).
  Entsprechender SDF-Operand: OBJECT-CONTINUATION=YES.

Ein PARAMETER-Kommando wird von SYSCMD gelesen und muß bei Bedarf vor Aufruf des Compilers gegeben werden. Das Kommando bleibt bis zum nächsten LOGOFF-Kommando oder SET-JOB-STEP-Kommando (in Prozeduren) oder bis zur nächsten Änderung durch ein anderes PARAMETER-Kommando gültig.

Für FOR1 wirken die Angaben aus dem PARAMETER-Kommmando so, als seien entsprechende Compileroptionen den tatsächlich vorhandenen Compileroptionen vorangestellt. Die Angaben des PARAMETER-Kommandos kommen daher nur zur Wirkung, falls sie nicht durch die vorhandenen Compileroptionen verändert bzw. aufgehoben werden.

Beispiel:

Das Kommando

```
/PARAMETER ERRFIL=YES,MAP=YES,SAVLST=SOURCE,OBJLST=YES,LIST=YES
```

entspricht den Compileroptionen

```
*COMOPT LISTFILE=(SOURCE,DIAG,ESD,XREF,SUMMARY,OPTIONS)
*COMOPT LIST=(MAP,SOURCE,OPTIONS,DIAG,SUMMARY,OBJECT).
```

Die folgende Tabelle stellt den für FOR1 relevanten PARAMETER-Operanden die gleichbedeutenden Compileroptionen gegenüber:

PARAM-Operand	* COMOPT...
CODE = 1	CODE = EBCDIC
CODE = 2	CODE = ISO
CODE = 3	CODE = BCD
DEBUG = YES	TESTOPT = (ALL)
DEBUG = NO	TESTOPT = (STNR)
DIAG = YES	LIST = (DIAG)
DIAG = NO	NOLIST = (DIAG)
DISC = YES	OBJECT = (*)
DISC = NO	NOOBJECT = (*)
ERRFIL = YES	LISTFILE = (DIAG)
ERRFIL = NO	LISTFILE = (NODIAG)
LIST = YES	LIST
LIST = NO	NOLIST
MAP = YES	LIST = (MAP)
MAP = NO	LIST = (NOMAP)
OBJLST = YES	LIST = (OBJECT)
OBJLST = NO	LIST = (NOOBJECT)
SAVLST = SOURCE	LISTFILE = (SOURCE, DIAG, ESD, XREF, SUMMARY, OPTIONS)
SAVLST = LOCMAP	LISTFILE = (MAP)
SAVLST = OBJECT	LISTFILE = (OBJECT)
SAVLST = ALL	LISTFILE = (ALL)
SAVLST = NO	NOLISTFILE
XREF = YES	LIST = (XREF)
XREF = NO	LIST = (NOXREF)

Tab. A.4-1: PARAM-Operanden und entsprechende Compileroptionen

Für eine Listenausgabe nach SYSLST, die über PARAM-Kommandos gesteuert wird, ist stets die Angabe LIST=YES notwendig.

A.5 IOSTAT-Meldungen

IOSTAT-Meldungen bestehen aus

- dem IOSTAT-Code und
- dem Meldungstext.

Auf beide kann der Anwender per Programm zugreifen. Nötig sind dazu:

- der IOSTAT-Parameter in der OPEN, READ- oder WRITE-Anweisung; (siehe "FOR1-Beschreibung" [21]).
- das INCLUDE-Element IFNIOS aus der FOR1-Makrobibliothek (FOR1MACLIB).

Anwendung

Je nach gewünschter Sprache der Meldungstexte (deutsch bzw. englisch) ist folgende Anweisung anzugeben:

a) um **deutsche** Texte zu erhalten:

```
%INCLUDE $FOR1MACLIB(IFNIOS),'*D'='␣␣'
```

b) um **englische** Texte zu erhalten:

```
%INCLUDE $FOR1MACLIB(IFNIOS),'*E'='␣␣'
```

c) um **gleichzeitig deutsche und englische** Texte zu erhalten

```
%INCLUDE $FOR1MACLIB(IFNIOS),'*D'='␣␣','*E'='␣␣'
```

Um die Ausgabe der Meldungstexte des INCLUDE-Elements IFNIOS in der Quellprogrammliste zu unterdrücken, muß der EXPAND-Modus durch COMOPT NOEXPAND ausgeschaltet werden.

Beispiel:

Zugriff auf einen IOSTAT-Meldungstext per Programm: Aufgrund der falschen ACCESS-METHOD-Angabe im SET-FILE-LINK-Kommando gibt das Programm den IOSTAT 19 mit zugehörigem Meldungstext aus.

Gezeigt werden:

- Quellprogramm
- Kommandos für Übersetzung und Ablauf
- Ausgabe der IOSTAT-Meldung

1. Programm (in Datei IO.SRC):

```
      PROGRAM IOSEXAM
      OPEN (UNIT=10,ACCESS='DIRECT',ERR=20,IOSTAT=IOS)
       .
       .
       .
      STOP
   20 WRITE (6,'(A)') IOSTATDEUTXT(IOS)
      STOP
      %INCLUDE LIBNAME(IFNIOS),'*D'='  '
      END
```

2. Übersetzung:

```
/SET-FILE-LINK LINK-NAME=LIBNAME, FILE-NAME=$TSOS.FOR1MACLIB
/START-PROG $FOR1
*COMOPT INCLUDE-LIBRARY=LIBNAME, SOURCE=IO.SRC, NOEXPAND, END
```

3. Ablauf des mit TSOSLNK gebundenen Programms:

```
/SET-TASKLIB LIB=$TSOS.FOR1MODLIBS
/SET-FILE-LINK LINK-NAME=DSET10, FILE-NAME=dateiname, ACCESS-METHOD=SAM
/START-PROG L.IOSBEISP
```

4. Ausgabe über SYSLST:

```
IOSTAT=19: OPEN-PARAMETER WIDERSPRECHEN DATEI/GERAETE-EIGENSCHAFTEN
```

Liste der IOSTAT-Meldungen

Die Datei IFNIOS enthält alle IOSTAT-Meldungen. Alle mit "*D" bezeichneten Zeilen kennzeichnen deutsche Meldungen; alle mit "*E" zeichneten Zeilen kennzeichnen englische Meldungen.

Im folgenden sind die deutschen IOSTAT-Meldungen aus IFNIOS aufgelistet:

```
*D    CHARACTER IOSTATDEUTXT*104 (-1:200)
*D    DATA IOSTATDEUTXT/
*D   F 'IOSTAT= -1: DATEIENDE-BEDINGUNG AUFGETRETEN'
*D   F,'IOSTAT=  0: OPERATION ORDNUNGSGEMAESS BEENDET'
*D   F,'IOSTAT=  1: VORAUSGEGANGENE EIN/AUSGABE-OPERATION NICHT ORDNUNGS
*D   FGEMAESS ABGESCHLOSSEN'
*D   F,'IOSTAT=  2: DATEINUMMER LIEGT NICHT ZWISCHEN 0 UND 99'
*D   F,'IOSTAT=  3: MIT DER ANGEFORDERTEN DATEINUMMER IST KEINE DATEI VE
*D   FRKNUEPFT'
*D   F,'IOSTAT=  4: OPERATION AUF ANGEFORDERTE DATEINUMMER NICHT ZUGELAS
*D   FSEN'
*D   F,'IOSTAT=  5: OPERATION AUF ANGEFORDERTE DATEI NICHT ZUGELASSEN'
*D   F,'IOSTAT=  6: REIHENFOLGE DER OPERATIONEN UNZULAESSIG'
*D   F,'IOSTAT=  7: WIDERSPRUECHLICHE PARAMETER BEIM INITIAL CALL'
*D   F,'IOSTAT=  8: UNGUELTIGER PARAMETER BEIM INITIAL CALL'
*D   F,'IOSTAT= 14: OPEN PAM-DATEI UNZULAESSIG'
```

```
*D   F,'IOSTAT= 15: UNZULAESSIGE AUSGABE-OPERATION, DATEI DARF NUR GELES
*D   FEN WERDEN'
*D   F,'IOSTAT= 16: WIDERSPRUECHLICHER PARAMETER BEIM OPEN'
*D   F,'IOSTAT= 17: UNZULAESSIGER DATEINAME BEIM OPEN'
*D   F,'IOSTAT= 18: UNGUELTIGER PARAMETER BEIM OPEN'
*D   F,'IOSTAT= 19: OPEN-PARAMETER WIDERSPRECHEN DATEI/GERAETE-EIGENSCHA
*D   FFTEN'
*D   F,'IOSTAT= 20: "ALTE" DATEI NICHT KATALOGISIERT ODER LEER'
*D   F,'IOSTAT= 21: DATEI KONNTE NICHT GEOEFFNET WERDEN'
*D   F,'IOSTAT= 22: KEIN GERAET FREI FUER MONTAGE PRIVATER DATEIEN'
*D   F,'IOSTAT= 23: "NEUE" DATEI BEREITS KATALOGISIERT UND NICHT LEER'
*D   F,'IOSTAT= 24: PHYSIKALISCHE SATZGROESSE KLEINER ALS FORTRAN-SATZGR
*D   FOESSE. ABSCHNEIDEN MOEGLICH'
*D   F,'IOSTAT= 25: FALSCHES ODER FEHLENDES PASSWORT BEI GESCHUETZTER DA
*D   FTEI'
*D   F,'IOSTAT= 26: DATEI GESPERRT'
*D   F,'IOSTAT= 27: PRIVATE BANDDATEI HAT KEINE STANDARD-ETTIKETTEN'
*D   F,'IOSTAT= 28: UNGUELTIGER PARAMETER BEIM CLOSE'
*D   F,'IOSTAT= 29: CLOSE-PARAMETER WIDERSPRECHEN DATEI/GERAETE-EIGENSCH
*D   FAFTEN'
*D   F,'IOSTAT= 30: CLOSE: STATUS IST WEDER "KEEP" NOCH "DELETE". "KEEP
*D   FWIRD ANGENOMMEN'
*D   F,'IOSTAT= 31: ZU SCHLIESSENDE DATEI GAR NICHT OFFEN. CLOSE WIRD UE
*D   FBERGANGEN'
*D   F,'IOSTAT= 32: NICHT NAEHER ERKLAERBARER FEHLER BEI AUSFUEHRUNG DES
*D   F CLOSE AUFGETRETEN'
*D   F,'IOSTAT= 33: UNZULAESSIGER STATUS-PARAMETER BEI CLOSE, KEEP WIRD
*D   FANGENOMMEN'
*D   F,'IOSTAT= 34: UNZULAESSIGER DATEINAME BEIM INQUIRE'
*D   F,'IOSTAT= 35: DATEINAME FEHLT BEIM INQUIRE'

*D   F,'IOSTAT= 36: UNGUELTIGER PARAMETER BEIM INQUIRE'
*D   F,'IOSTAT= 37: WRITE-PASSWORT FEHLT, OPEN-MODUS -> INPUT, ZUKUENFTI
*D   FGE AUSGABE-OPERATIONEN WERDEN ABGEWIESEN'
*D   F,'IOSTAT= 38: DATEI GESPERRT, OPEN-MODUS -> INPUT, ZUKUENFTIGE AUS
*D   FGABE-OPERATIONEN WERDEN ABGEWIESEN'
*D   F,'IOSTAT= 39: OPEN AUF STANDARD-FORTRAN-DATEI, UMLENKEN NUR DURCH
*D   FRUNOPTS'
*D   F,'IOSTAT= 40: UNGUELTIGER ZAHLENWERT IN POSITIONIERUNGSANWEISUNG'
*D   F,'IOSTAT= 41: BACKFILE/SKIPFILE NUR AUF BANDDATEIEN ZULAESSIG'
*D   F,'IOSTAT= 42: BACKSPACE/SKIPREC NUR AUF SEQUENTIELLEN DATEIEN ZULA
*D   FESSIG'
*D   F,'IOSTAT= 43: FIND NUR AUF DIREKTZUGRIFFSDATEIEN ZULAESSIG'
*D   F,'IOSTAT= 44: SATZFORMAT UNBEKANNT'
*D   F,'IOSTAT= 48: SATZNUMMER UEBERSTEIGT GROESSTE ZULAESSIGE SATZNUMME
*D   FR (MAXREC)'
*D   F,'IOSTAT= 49: SATZ GROESSER ALS PUFFER. REST WIRD ABGESCHNITTEN'
*D   F,'IOSTAT= 50: UNDEFINIERBARER FEHLER BEIM ZUGRIFF AUF EINE SYSTEMD
*D   FATEI'
*D   F,'IOSTAT= 51: UNDEFINIERBARER FEHLER BEIM ZUGRIFF AUF EINE EAM-DAT
*D   FEI'
*D   F,'IOSTAT= 52: UNDEFINIERBARER FEHLER BEIM ZUGRIFF AUF EINE BENUTZE
*D   FRDATEI'
*D   F,'IOSTAT= 53: UNZULAESSIGE ANWENDUNG EINER DMS-ANWEISUNG'
*D   F,'IOSTAT= 54: GEWUENSCHTER SATZ NICHT AUFFINDBAR'
*D   F,'IOSTAT= 55: ISAM-SCHLUESSEL MEHRFACH ODER NICHT AUFSTEIGEND'
*D   F,'IOSTAT= 56: GERAETEFEHLER AUFGETRETEN'
*D   F,'IOSTAT= 57: GEWUENSCHTER ISAM-SATZ GESPERRT'
*D   F,'IOSTAT= 58: ES WERDEN MEHR DATEN ANGEFORDERT/ANGEBOTEN, ALS IN D
*D   FEN PUFFER PASSEN'
*D   F,'IOSTAT= 59: SATZNUMMER NICHT GROESSER 0'
*D   F,'IOSTAT= 60: NICHT-NUMERISCHE SATZNUMMER ODER SATZNUMMER ZU GROSS'
*D   F,'IOSTAT= 61: REC-PARAMETER NICHT CHARACTER-DATENTYP'
*D   F,'IOSTAT= 62: STRINGLAENGE UEBERSTEIGT SATZLAENGE'
*D   F,'IOSTAT= 64: B-FORMAT NICHT ERLAUBT BEI INTEGER-DATEN'
*D   F,'IOSTAT= 65: B-FORMAT NICHT ERLAUBT BEI REAL-DATEN'
*D   F,'IOSTAT= 66: D,E,F,Q-FORMAT NICHT ERLAUBT BEI CHARACTER-DATEN'
*D   F,'IOSTAT= 67: D,E,F,Q-FORMAT NICHT ERLAUBT BEI BIT-DATEN'
*D   F,'IOSTAT= 68: D,E,F,Q-FORMAT NICHT ERLAUBT BEI LOGICAL-DATEN'
```

```
*D   F,'IOSTAT= 69: I-FORMAT NICHT ERLAUBT BEI CHARACTER-DATEN'
*D   F,'IOSTAT= 70: I-FORMAT NICHT ERLAUBT BEI BIT-DATEN'
*D   F,'IOSTAT= 71: I-FORMAT NICHT ERLAUBT BEI LOGICAL-DATEN'
*D   F,'IOSTAT= 72: L-FORMAT NICHT ERLAUBT BEI INTEGER-DATEN'
*D   F,'IOSTAT= 73: L-FORMAT NICHT ERLAUBT BEI REAL-DATEN'
*D   F,'IOSTAT= 74: L-FORMAT NICHT ERLAUBT BEI CHARACTER-DATEN'
*D   F,'IOSTAT= 75: DATENTYP BIT NOCH NICHT IMPLEMENTIERT'
*D   F,'IOSTAT= 76: KONVERSION VON REAL-DATEN MIT I-FORMAT NICHT IMPLEME
*D   FNTIERT'
*D   F,'IOSTAT= 77: KONVERSION VON INTEGER-DATEN MIT D,E,F,Q-FORMAT NICH
*D   FT IMPLEMENTIERT'
*D   F,'IOSTAT= 78: FORMATANGABE IN SICH WIDERSPRUECHLICH'
*D   F,'IOSTAT= 80: INTEGER-WERT ZU GROSS'
*D   F,'IOSTAT= 81: UNGUELTIGER INTEGER-WERT'
*D   F,'IOSTAT= 82: UNGUELTIGER REAL-WERT'
*D   F,'IOSTAT= 83: UNGUELTIGER BIT-WERT'
*D   F,'IOSTAT= 84: UNGUELTIGER CHARACTER-WERT'
*D   F,'IOSTAT= 85: UNGUELTIGES SEDEZIMALZEICHEN'
*D   F,'IOSTAT= 86: UNGUELTIGER LOGICAL-WERT'
*D   F,'IOSTAT= 96: FORMAT SYNTAX FEHLER: LINKE KLAMMER FEHLT'
*D   F,'IOSTAT= 97: FORMAT SYNTAX FEHLER: LEERES FORMAT'
*D   F,'IOSTAT= 98: FORMAT SYNTAX FEHLER: UNZULAESSIGES FORMAT-ELEMENT'
*D   F,'IOSTAT= 99: FORMAT SYNTAX FEHLER: ZAHLENWERT 0 NICHT ERLAUBT'
*D   F,'IOSTAT=100: FORMAT SYNTAX FEHLER: LEERE FORMAT-GRUPPE'
*D   F,'IOSTAT=101: FORMAT SYNTAX FEHLER: RECHTE KLAMMER FEHLT'
*D   F,'IOSTAT=102: FORMAT SYNTAX FEHLER: WIEDERHOLUNGSFAKTOR OHNE NACHF
*D   FOLGENDES FORMAT'
*D   F,'IOSTAT=103: FORMAT SYNTAX FEHLER: UNZULAESSIGE POSITION EINES +/
*D   F-'
*D   F,'IOSTAT=104: FORMAT SYNTAX FEHLER: WIEDERHOLUNGSFAKTOR UNZULAESSI
*D   FG'
*D   F,'IOSTAT=105: FORMAT SYNTAX FEHLER: LITERAL ENDET NICHT MIT HOCHKO
*D   FMMA'
*D   F,'IOSTAT=106: FORMAT SYNTAX FEHLER: LITERAL MIT LAENGE V NCHT ZUL
*D   FAESSIG'
*D   F,'IOSTAT=107: FORMAT SYNTAX FEHLER: ZU GROSSER WIEDERHOLUNSFAKTOR
*D   F'
*D   F,'IOSTAT=108: FORMAT SYNTAX FEHLER: TRENNZEICHEN FEHLT'
*D   F,'IOSTAT=109: FORMAT SYNTAX FEHLER: FELDWEITENANGABE FEHLT
*D   F,'IOSTAT=110: FORMAT SYNTAX FEHLER: ANGABE FUER DEZIMALSTELLEN FEH
*D   FLT'
*D   F,'IOSTAT=111: FORMAT SYNTAX FEHLER: EXPONENTENSPEZIFIKATION NUR BE
*D   FI E/G-FORMATEN'
*D   F,'IOSTAT=112: FORMAT SYNTAX FEHLER: ANGABE FUER LAENGE DES EXPONEN
*D   FTEN FEHLT'
*D   F,'IOSTAT=113: FORMAT SYNTAX FEHLER: ANGABE FUER DEZIMALSTELLEN UNZ
*D   FULAESSIG'
*D   F,'IOSTAT=114: FORMAT SYNTAX FEHLER: ANGABE EINES ZAHLENWERTES FEHL
*D   FT'
*D   F,'IOSTAT=115: FORMAT SYNTAX FEHLER: FEHLER IN DER FOLGE DES FORMAT
*D   F-ELEMENT-DESKRIPTOREN'
*D   F,'IOSTAT=116: FORMAT SYNTAX FEHLER: ZAHLENWERT ZU GROSS'
*D   F,'IOSTAT=117: FORMAT SYNTAX FEHLER: FELDWEITENANGABE ZU GROSS'
*D   F,'IOSTAT=118: FORMAT SYNTAX FEHLER: FELDWEITE 0 UNZULAESSIG'
*D   F,'IOSTAT=119: FORMAT SYNTAX FEHLER: IM WIEDERHOLBAREN TEIL DES FOR
*D   FMATS FEHLT EIN DATEN-FORMAT'
*D   F,'IOSTAT=120: UNFORMATIERTE EIN/AUSGABE VERARBEITET NUR EINEN SATZ
*D   F'
*D   F,'IOSTAT=121: UNZULAESSIGE KENNZEICHNUNGSANGABE IN DER WAIT-ANWEIS
*D   FUNG'
*D   F,'IOSTAT=122: ZU VIELE TEILSAETZE'
*D   F,'IOSTAT=123: UNFORMATIERTES SCHREIBEN: SAETZE WURDEN GETEILT'
*D   F,'IOSTAT=128: UNZULAESSIGES ZEICHEN IM EINGABESATZ'
*D   F,'IOSTAT=129: UNZULAESSIGE REIHENFOLGE DER ZEICHEN'
*D   F,'IOSTAT=130: UNGUELTIGER KOMPLEXER WERT'
*D   F,'IOSTAT=131: UNZULAESSIGE LINKE KLAMMER'
*D   F,'IOSTAT=132: STRING NICHT IN HOCHKOMMAS EINGESCHLOSSEN'
*D   F,'IOSTAT=144: FALSCHER NAME IM NAMELIST-EINGABESATZ'
*D   F,'IOSTAT=145: VARIABLENNAME NICHT IN NAMELIST-ANWEISUNG DEFINIERT'
```

```
*D   F,'IOSTAT=146: EINFACHE VARIABLE ENTHAELT INDIZES'
*D   F,'IOSTAT=147: UNZULAESSIGE ANZAHL VON INDIZES'
*D   F,'IOSTAT=148: NAMELIST-NAME FEHLT IM ERSTEN SATZ'
*D   F,'IOSTAT=149: NULL-VALUES IN NAMELIST-EINGABESAETZEN NICHT ZULAESS
*D   FIG'
*D   F,'IOSTAT=150: ZU VIELE WERTANGABEN FOLGEN VARIABLENNAME BZW. VARIA
*D   FBLENNAME FEHLT'
*D   F,'IOSTAT=151: &END FEHLT'
*D   F,'IOSTAT=152: UNZULAESSIGER INDEXWERT'
*D   F,'IOSTAT=153: WIEDERHOLUNGSFAKTOR ZU GROSS : REPFAK * DATENLAENGE
*D   F> 32 KBYTE'
*D   F,'IOSTAT=156: UNDEFINIERTER FEHLER BEI AUSFUEHRUNG EINER PAUSE/STO
*D   FP-ANWEISUNG'
*D   F,'IOSTAT=157: UNZULAESSIGER DATENTYP IN PAUSE/STOP-ANWEISUNG'
*D   F,'IOSTAT=158: BEI AUSFUEHRUNG EINER PAUSE/STOP-ANWEISUNG WURDE DIE
*D   F NACHRICHT ABGESCHNITTEN'
*D   F,'IOSTAT=159: BEI AUSFUEHRUNG EINER PAUSE/STOP-ANWEISUNG MIT ANTWO
*D   FRT WURDE DIE ANTWORT ABGESCHNITTEN'
*D   F/
```

A.6 Beispiele für Compilerlisten

A.6.1 Quellprogrammliste mit Diagnoseliste

```
   ****  SOURCE  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 16:35:28   PAGE  1
                                                 PROGRAM UNIT: INV
D DO/IF SEG   STMT  I/H LINE        SOURCE-TEXT                                                  COL73-80 RECORD-ID.
-------------------------------------------------------------------------------------------------------------------
       1/1      1         1        PROGRAM INV                                                          07000000
         1      2         2        COMMON IR                                                            15000000
 *       1      3         3    1   WRITE (2,10)                                                         23000000        *
 ***** WARNING (SA047) *******                                                                        *UNREFERENCED LABEL********
         1      4         4        READ (1,11) IR                                                       30000000
         1      5         5        IF (IR.GT.3) THEN                                                    38000000
 *  1    2      6         6          GO TO 2                                                            46000000        *
 ***** SEVERE  (FA185) *******                                                                        *LABEL 2 NOT DECLARED******
    1    2      7         7        ELSE IF (IR.EQ.0) THEN                                               53000000
    1    3      8         8          CALL DETS (A)                                                      61000000
    1    3      9         9        END IF                                                               69000000
         3     10        10   10   FORMAT('    RANG DER MATRIX EINGEBEN')                             76000000
         3     11        11   11   FORMAT (I1)                                                         84000000
         4     12        12        END                                                                  92000000

   *** DIAGNOSTIC  LISTING ***      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 16:35:28   PAGE  2
                                                 PROGRAM UNIT: INV
D DO/IF SEG   STMT  I/H LINE        SOURCE-TEXT                                                  COL73-80 RECORD-ID.
-------------------------------------------------------------------------------------------------------------------
 *       1      3         3    1   WRITE (2,10)                                                         23000000        *
       WARNING (SA047)  UNREFERENCED LABEL #1
 *  1    2      6         6          GO TO 2                                                            46000000        *
       SEVERE  (FA185)  LABEL 2 NOT DECLARED
```

Anmerkung: In Anweisung 6 müßte es "GOTO 1" heißen.

A.6.2 Quellprogrammlisten (Haupt- und Unterprogramm)

```
   ****  SOURCE  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:41:01   PAGE  1
                                                 PROGRAM UNIT: DIALOG
 DO/IF SEG   STMT  I/H LINE        SOURCE-TEXT                                                  COL73-80 RECORD-ID.
-------------------------------------------------------------------------------------------------------------------
       1/1      1         1        PROGRAM DIALOG                                                       02000000
         1      2         2        REAL A(10),B                                                         03000000
         1      3         3        DIMENSION B(5)                                                       04000000
                          4   ****                                                                      05000000
         2      4         5        DO 10 I=1,5                                                          06000000
    1    3      5         6        B(I)=A(I)+I                                                          07000000
    1    4      6         7   10   A(I)=I                                                              08000000
         4      7         8        WRITE (2,11) A                                                       09000000
         4      8         9   11   FORMAT ('***',5F5.2)                                               10000000
         4      9        10        CALL SUBA(A(1))                                                      11000000
         4     10        11        END                                                                  12000000

   ****  SOURCE  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:43:33   PAGE  2
                                                 PROGRAM UNIT: SUBA
 DO/IF SEG   STMT  I/H LINE        SOURCE-TEXT                                                  COL73-80 RECORD-ID.
-------------------------------------------------------------------------------------------------------------------
       1/1      1         1        SUBROUTINE SUBA(X)                                                   18000000
         1      2         2        Y=X*X                                                                19000000
         1      3         3        END                                                                  20000000
```

A.6.3 Liste der Änderungen

```
 ****  CHANGE  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:41:01    PAGE  1
                                                PROGRAM UNIT: DIALOG
(OLD) @   3.0000:      REEL A(10),B
(NEW) @   3.0000:      REAL A(10),B
(IN)  @P
(IN)  @   5.0000:      DO 10 I=1,5
(OLD) @   5.0000:      DO 10 I=1.5
(NEW) @   5.0000:      DO 10 I=1,5
(IN)  @CON
(OUT)  FOR1: RECOMPILIERUNG DER AKTUELLEN P.U. INITIIERT
(IN)  @I3.5
(IN)  @IN3.5
(NEW) @   3.5000:      DIMENSION B(5)
(IN)  @R
(IN)  @P1-2
(IN)  @D2
(OUT)  FOR1: 1 ZEILE(N) GELOESCHT
(IN)  @CON
(OUT)  FOR1: RECOMPILIERUNG DER AKTUELLEN P.U. INITIIERT
```

A.6.4 Liste der externen Namen

```
****  E S D   LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:41:01   PAGE  1
                                              PROGRAM UNIT: DIALOG
IDENTIFIER ESID  TYPE  DISPL.  LENGTH     IDENTIFIER ESID  TYPE  DISPL.  LENGTH     IDENTIFIER ESID  TYPE  DISPL.  LENGTH
-------------------------------------------------------------------------------------------------------------------------
DIALOG     0001   SD   000000  000250     IF@FCTL    000C   VC   000168             IF@XINI    0009   VC   00001C
DIALOG@@   0002   SD   000250  000278     IF@FEDC    000D   VC   00016C             IF@XPRO    000A   VC   000160
I@@@RTCA   0003   CM   000000  001000     IF@RETN    000F   VC   000000             IF@XTCA    0005   VC   0001E4
IF@@MPI    0002   LD   000000             IF@SINI    000E   VC   000170             IT0PCD     0004   VC   000000
IF@CTER    0010   VC   000000             IF@XFCO    0006   VC   0001E0             SUBA       0008   VC   0001D8
IF@ERROR   000B   VC   000164             IF@XICA    0007   VC   0001DC

****  E S D   LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:43:33   PAGE  2
                                              PROGRAM UNIT: SUBA
IDENTIFIER ESID  TYPE  DISPL.  LENGTH     IDENTIFIER ESID  TYPE  DISPL.  LENGTH     IDENTIFIER ESID  TYPE  DISPL.  LENGTH
-------------------------------------------------------------------------------------------------------------------------
I@@@RTCA   0003   CM   000000  001000     IF@ERROR   0006   VC   000154             IT0PCD     0004   VC   000000
IF@@MPI    0005   ER   000000             IF@RETN    0008   VC   000000             SUBA       0001   SD   000000  0001E8
IF@CTER    0009   VC   000000             IF@SINI    0007   VC   000158             SUBA@@@@   0002   SD   0001E8  000138
```

A.6.5 Datenadreßliste

```
  ****  M A P   LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:41:01   PAGE  1
CODE + CONSTANTS SECTION: DIALOG       ESID=0001     PROGRAM UNIT: DIALOG                                  SORTED BY ADDRESS
SYMBOL           TYPE        ADR           SYMBOL           TYPE        ADR           SYMBOL           TYPE        ADR
-------------------------------------------------------------------------------------------------------------------------
10               STMNT_LABEL 00000080      11               FORMT_LABEL

  ****  M A P   LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:41:01   PAGE  2
CODE + CONSTANTS SECTION: DIALOG       ESID=0001     PROGRAM UNIT: DIALOG                                  SORTED BY SYMBOL
SYMBOL           TYPE        ADR           SYMBOL           TYPE        ADR           SYMBOL           TYPE        ADR
-------------------------------------------------------------------------------------------------------------------------
10               STMNT_LABEL 00000080      11               FORMT_LABEL

  ****  M A P   LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:41:01   PAGE  3
L O C A L  DATA  SECTION: DIALOG@@      ESID=0002     PROGRAM UNIT: DIALOG                                  SORTED BY ADDRESS
SYMBOL           TYPE        ADR           SYMBOL           TYPE        ADR           SYMBOL           TYPE        ADR
-------------------------------------------------------------------------------------------------------------------------
SUBA.A           EXT_SUBR    000001D8      IF@XTCA.A        EXT_SUBR    000001E4      I                I4          00000424
IF@XICA.A        EXT_SUBR    000001DC      A                R4(1)       000003E8      B.D              R4(1)       00000484
IF@XFCO.A        EXT_SUBR    000001E0      B                R4(1)       00000410      A.D              R4(1)       00000494

  ****  M A P   LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:41:01   PAGE  4
L O C A L  DATA  SECTION: DIALOG@@      ESID=0002     PROGRAM UNIT: DIALOG                                  SORTED BY SYMBOL
SYMBOL           TYPE        ADR           SYMBOL           TYPE        ADR           SYMBOL           TYPE        ADR
-------------------------------------------------------------------------------------------------------------------------
A                R4(1)       000003E8      B.D              R4(1)       00000484      IF@XICA.A        EXT_SUBR    000001DC
A.D              R4(1)       00000494      I                I4          00000424      IF@XTCA.A        EXT_SUBR    000001E4
B                R4(1)       00000410      IF@XFCO.A        EXT_SUBR    000001E0      SUBA.A           EXT_SUBR    000001D8

  ****  M A P   LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:43:33   PAGE  5
L O C A L  DATA  SECTION: SUBA@@@@      ESID=0002     PROGRAM UNIT: SUBA                                    SORTED BY ADDRESS
SYMBOL           TYPE        ADR           SYMBOL           TYPE        ADR           SYMBOL           TYPE        ADR
-------------------------------------------------------------------------------------------------------------------------
X                R4          00000304      Y                R4          00000308

  ****  M A P   LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:43:33   PAGE  6
L O C A L  DATA  SECTION: SUBA@@@@      ESID=0002     PROGRAM UNIT: SUBA                                    SORTED BY SYMBOL
SYMBOL           TYPE        ADR           SYMBOL           TYPE        ADR           SYMBOL           TYPE        ADR
-------------------------------------------------------------------------------------------------------------------------
X                R4          00000304      Y                R4          00000308
```

A.6.6 Querverweisliste

```
  **** X R E F   LISTING ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:41:01   PAGE  1
                                                PROGRAM UNIT: DIALOG
IDENTIFIER         DISPL   DESCR   SPEC   ATTRIBUTES AND REFERENCES (/ : SPECIFICATION;  = : ASSIGNMENT)
-------------------------------------------------------------------------------------------------------------------------
A                  0003E8 000494      2   ARRAY, REAL*4(1) ------- /2 5 =6 7 =9
B                  000410 000484      2   ARRAY, REAL*4(1) ------- /2 /3 =5
DIALOG                                *   PROGRAM NAME ------- 1
I                  000424             *   VARIABLE, INTEGER*4 ------- =4 5 5 5 6 6
SUBA                      0001D8      *   EXTERNAL SUBROUTINE ------- 9
10                 000080             6   STATEMENT LABEL ------- 4 /6
11                                    8   FORMAT LABEL ------- 7 8 /8

  **** X R E F   LISTING ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:43:33   PAGE  2
                                                PROGRAM UNIT: SUBA
IDENTIFIER         DISPL   DESCR   SPEC   ATTRIBUTES AND REFERENCES (/ : SPECIFICATION;  = : ASSIGNMENT)
-------------------------------------------------------------------------------------------------------------------------
SUBA                                  *   SUBROUTINE NAME(1) ------- 1
X                  000304             *   VARIABLE, REAL*4, DUMMY ARGUMENT ------- 1 2 2
Y                  000308             *   VARIABLE, REAL*4 ------- =2
```

A.6.7 Objektliste

```
  ****  OBJECT  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:41:01   PAGE  1
                                                   PROGRAM UNIT: DIALOG
FLG DISPL   OPERATION       ADDR1  ADDR2  STMNT    ASSEMBLY CODE                          SYMB.ADDR1, SYMB.ADDR2
                                                1   ************************************
                                                2   **        C O D E    A R E A        **
                                                3   ************************************
                                                4   *
    000000                                      5   DIALOG   EQU   *
    000000                                      6   *
                                                7   ***** STATEMENT 1 (ENTRY) ************
                                                8   *       PROGRAM DIALOG
                                                9   *                                                  **** SEGMENT 1 ****
    000000  17 EE                              10            XR    14,14
    000002  05 B0                              11            BALR  11,0
                                               12   * CODE SLICE BEGIN *                   (SLICE 1)
    000004                                     13            USING *,11
    000004  98 CD B010      000014             14            LM    12,13,16(11)
    000008  47 F0 B024      000028             15            BC    15,36(0,11)
    00000C  05                                 16            DC    AL1(05)
    00000D  C4C9C1D3D6C740                     17            DC    CL7'DIALOG '
    000014  000000D8                           18            DC    A(DIALOG##)
    000018  00000250                           19            DC    A(DIALOG@@)
    00001C  00000000                           20            DC    V(IF@XINI)
    000020  00000024                           21            DC    A(*+4)
    000024                                     22            DC    X'0000000C'
    0000D8                                     23            USING DIALOG##,12
    000250                                     24            USING DIALOG@@,13
    000028  58 F0 B018      00001C             25            L     15,24(0,11)
    00002C  D5 03 B018 C0A0 00001C 000178      26            CLC   24(4,11),160(12)
    000032  47 80 0001      000001             27            BC    8,1(0,0)
    000036  50 E0 D004      000254             28            ST    14,4(0,13)
    00003A  41 10 B01C      000020             29            LA    1,28(0,11)
    00003E  05 EF                              30            BALR  14,15
    000040  58 90 D04C      00029C             31            L     9,76(0,13)
    000044  50 D0 900C      00000C             32            ST    13,12(0,9)
                                               33   ***** STATEMENT 4 (DO) ***************
                                               34   *       DO 10 I=1,5
    000048  41 F0 0001      000001             35            LA    15,1(0,0)              1
    00004C  50 F0 D1D4      000424             36            ST    15,468(0,13)           I
    000050  41 A0 0005      000005             37            LA    10,5(0,0)              5
                                               38   ***** STATEMENT 5 (MOVED STMT) *******
    000054  41 10 0004      000004             39            LA    1,4(0,0)               4
    000058  41 30 0004      000004             40            LA    3,4(0,0)               4
    00005C  58 50 D1D4      000424             41            L     5,468(0,13)            I
    000060  47 F0 B062      000066             42            BC    15,98(0,11)            %L5
                                               43   ***** STATEMENT 5 (INCR STMT) ********          **** SEGMENT 2 ****
    000064  1A 13                              44   %L3      AR    1,3                    %V1, %V2
                                               45   ***** STATEMENT 5 (ASSIGNMENT) *******
                                               46   *       B(I)=A(I)+I
                                               47   *                                                  **** SEGMENT 3 ****
    000066  78 01 D194      0003E4             48   %L5      LE    0,404(1,13)            A
    00006A  68 20 C0E0      0001B8             49            LD    2,224(0,12)
    00006E  50 50 D0A4      0002F4             50            ST    5,164(0,13)            I
    000072  97 80 D0A4      0002F4             51            XI    164(13),128
    000076  6B 20 D0A0      0002F0             52            SD    2,160(0,13)
    00007A  3A 02                              53            AER   0,2                    I
    00007C  70 01 D1BC      00040C             54            STE   0,444(1,13)            B
                                               55   ***** STATEMENT 6 (ASSIGNMENT) *******
                                               56   * 10    A(I)=I
    000080  68 00 C0E0      0001B8             57   #1  0      LD    0,224(0,12)
```

```
   ****  OBJECT  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:41:01   PAGE  2
                                                PROGRAM UNIT: DIALOG
FLG DISPL   OPERATION       ADDR1  ADDR2  STMNT    ASSEMBLY CODE                              SYMB.ADDR1, SYMB.ADDR2
-------------------------------------------------------------------------------------------------------------------------

    000084  50 50 D0A4      0002F4           58            ST    5,164(0,13)            I
    000088  97 80 D0A4      0002F4           59            XI    164(13),128
    00008C  6B 00 D0A0      0002F0           60            SD    0,160(0,13)
    000090  70 01 D194      0003E4           61            STE   0,404(1,13)            A
                                             62    ***** STATEMENT 6 (DOEND) ************
                                             63    * 10    A(I)=I
    000094  1A 5F                            64            AR    5,15                   I, 1
    000096  46 A0 B060      000064           65            BCT   10,96(0,11)            %L3
                                             66    *                                                  **** SEGMENT 4 ****
    00009A  50 50 D1D4      000424           67            ST    5,468(0,13)            I
                                             68    ***** STATEMENT 7 (WRITE) ************
                                             69    *       WRITE (2,11) A
    00009E  41 10 D1F0      000440           70            LA    1,496(0,13)
    0000A2  58 F0 C104      0001DC           71            L     15,260(0,12)
    0000A6  41 00 0003      000003           72            LA    0,3(0,0)
    0000AA  05 EF                            73            BALR  14,15                  IF@XICA
    0000AC  58 F0 C108      0001E0           74            L     15,264(0,12)
    0000B0  41 10 D214      000464           75            LA    1,532(0,13)
    000464                                   76            USING DIALOG@@+532,1
    0000B4  41 00 0001      000001           77            LA    0,1(0,0)
    0000B8  05 EF                            78            BALR  14,15                  IF@XFCO
    0000BA  58 F0 C10C      0001E4           79            L     15,268(0,12)
    0000BE  17 00                            80            XR    0,0
    0000C0  05 EF                            81            BALR  14,15                  IF@XTCA
                                             82    ***** STATEMENT 9 (CALL) *************
                                             83    *       CALL SUBA(A(1))
    0000C2  58 F0 C100      0001D8           84            L     15,256(0,12)
    0000C6  41 10 D1E0      000430           85            LA    1,480(0,13)
    000430                                   86            USING DIALOG@@+480,1
    0000CA  41 00 0001      000001           87            LA    0,1(0,0)
    0000CE  05 EF                            88            BALR  14,15                  SUBA
                                             89    ***** STATEMENT 10 (END) *************
                                             90    *       END
    0000D0  58 F0 C088      000160           91            L     15,136(0,12)           IF@XPRO
    0000D4  05 EF                            92            BALR  14,15
                                             93    ************************************
                                             94    **     C O N S T A N T   A R E A    **
                                             95    ************************************
                                             96    *
    0000D6                                   97            ORG
    0000D8                                   98    DIALOG## DS    0D
    0000D8                                   99            USING *,12
                                            100    *                                      RUNTIME COMMUNICATION AREA
    0000D8  F3F0C1E4C7F9F1                  101            DC    '30AUG91'
    0000DF  5B081E0F29019135                102            DC    X'5B081E0F'
    0000E7  40C6D6D9F140E74040E5F24BF2C1F0  103            DC    ' FOR1 X  V2.2A00'
    0000F7  A0                              104            DC    X'A0'
    0000F8  00000004                        105            DC    A(DIALOG)
    0000FC  FFFFFFFF                        106            DC    F'-1'
    000100  41000411                        107            DC    X'41000411'
    000104  58F0C08C                        108            DC    X'58F0C08C'
    000108  07FF                            109            DC    X'07FF'
    00010C                                  110            ORG   DIALOG##+52
    00010C  00000004                        111            DC    A(DIALOG+4)            CODE SLICE ADDRESS (SLICE 1)
    0001E4                                  112            ORG   DIALOG##+268
    0001E4  00000000                        113            DC    V(IF@XTCA)             EXTERNAL REFERENCE
```

```
   ****  OBJECT  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:41:01   PAGE  3
                                                PROGRAM UNIT: DIALOG
FLG DISPL   OPERATION       ADDR1  ADDR2  STMNT    ASSEMBLY CODE                          SYMB.ADDR1, SYMB.ADDR2
    0001E0                                 114            ORG   DIALOG##+264
    0001E0  00000000                       115            DC    V(IF@XFCO)
    0001DC                                 116            ORG   DIALOG##+260
    0001DC  00000000                       117            DC    V(IF@XICA)
    0001D8                                 118            ORG   DIALOG##+256
    0001D8  00000000                       119            DC    V(SUBA)
    000160                                 120            ORG   DIALOG##+136
    000160  00000000                       121            DC    V(IF@XPRO)
    000164  00000000                       122            DC    V(IF@ERROR)
    000168  00000000                       123            DC    V(IF@FCTL)
    00016C  00000000                       124            DC    V(IF@FEDC)
    000170  00000000                       125            DC    V(IF@SINI)
    000178                                 126            ORG   DIALOG##+160
    000178  00000000                       127            DC    X'00000000'            0.00000000000000000000000000000000+00
    00017C  00000000                       128            DC    X'00000000'
    000180  00000000                       129            DC    X'00000000'
    000184  00000000                       130            DC    X'00000000'
    000198                                 131            ORG   DIALOG##+192
    000198  00000001                       132            DC    X'00000001'            1
    00019C  00000002                       133            DC    X'00000002'            2
    0001A8                                 134            ORG   DIALOG##+208
    0001A8  4E000000                       135            DC    X'4E000000'            0.00000000000000000E+00
    0001AC  00000000                       136            DC    X'00000000'
    0001B0  4E000001                       137            DC    X'4E000001'            0.42949672960000000E+10
    0001B4  00000000                       138            DC    X'00000000'
    0001B8  CE000000                       139            DC    X'CE000000'            -0.21474836480000000E+10
    0001BC  80000000                       140            DC    X'80000000'
    0001C0  40                             141            DC    ' '
    0001CC                                 142            ORG   DIALOG##+244
    0001CC  00000005                       143            DC    X'00000005'            5
    0001D0  00000004                       144            DC    X'00000004'            4
    0001D4  01018600                       145            DC    X'01018600'            16877056
    0001EA                                 146            ORG   DIALOG##+274
    0001EA  4DF3C85C5C5C6BF5C6F54BF25D4040 147            DC    '(3H***,5F5.2)   '
                                           148       *                                   EVENTHANDLER LIST
    0001FC                                 149            ORG   DIALOG##+292
    0001FC  01E70000                       150            DC    X'01E70000'
    000200  00000214                       151            DC    X'00000214'            ADDR OF PU_NAME
    000204  00000000                       152            DC    V(IF@RETN)             TERMINATION ROUTINE
    000208  00000000                       153            DC    V(IF@CTER)             PROCHK ROUTINE
    00020C  00000000                       154            DC    V(IF@CTER)             ERROR ROUTINE
    000210  FFFFFFFF                       155            DC    X'FFFFFFFF'            OTHER EVENTS ROUTINE
    000214  06                             156            DC    X'06'
    000215  C4C9C1D3D6C7                   157            DC    'DIALOG'
    00014C                                 158            ORG   DIALOG##+116
    00014C  0000021C                       159            DC    A(DIALOG##+324)        ADDRESS OF STATEMENT TABLE
    00021C                                 160            ORG   DIALOG##+324
    00021C  0000003000FC0000               161            DC    X'0000003000FC0000'
    000224  000124FC00000004               162            DC    X'000124FC00000004'
    00022C  0608FC00000005FF               163            DC    X'0608FC00000005FF'
    000234  01FC00000005FF0D               164            DC    X'01FC00000005FF0D'
    00023C  0AFC00000006FF05               165            DC    X'0AFC00000006FF05'
    000244  12FC000000090703               166            DC    X'12FC000000090703'
```

```
  ****  OBJECT  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:41:01   PAGE  4
                                   PROGRAM UNIT: DIALOG
FLG DISPL   OPERATION       ADDR1  ADDR2  STMNT    ASSEMBLY CODE                          SYMB.ADDR1, SYMB.ADDR2
-------------------------------------------------------------------------------------------------------------------------

                                          167    ************************************
                                          168    **        D A T A    A R E A        **
                                          169    ************************************
                                          170    *
    00024D                                171             ORG
    000250                                172             DS    0D
    000250                                173    DIALOG@@ CSECT
    000250                                174             USING *,13
    000250                                175             DS    632C
    000250                                176             ORG   DIALOG@@+0
    000250  0001FEFF                      177             DC    X'0001FEFF'              SAVE AREA
    000298                                178             ORG   DIALOG@@+72
    000298  00000000                      179             DC    A(I@@@RTCA)              RUNTIME COMMUNICATION AREA
    00029C  00000000                      180             DC    V(IT0PCD)
    0002A0  000001FC                      181             DC    X'000001FC'              A(IT0EHL)
    0002A0                                182             ORG   DIALOG@@+80
    0002A0  0000021C00000000              183             DC    X'0000021C00000000'      STMT TABLE AND COUNT TABLE ADDR
    0002F0                                184             ORG   DIALOG@@+160
    0002F0  CE00000000000000              185             DC    X'CE00000000000000'
    0002FC                                186             ORG   DIALOG@@+172
    0002FC  05                            187             DC    X'05'
    0002FD  C4C9C1D3D6C740                188             DC    'DIALOG '
    0003DC                                189             ORG   DIALOG@@+396
    0003DC  820004A4                      190             DC    A(DIALOG##+596)          ADDRESS OF FORMAT LABEL TABLE
    0003E0  000004C8                      191             DC    A(DIALOG@@+632)          DATA SLICE ADDRESS (ITA)
    000430                                192             ORG   DIALOG@@+480
    000430  000003E8                      193             DC    X'000003E8'              A()              ARGUMENT LIST
    000434  00000494                      194             DC    X'00000494'
    000438  FF030010                      195             DC    X'FF030010'
    00043C  05060000                      196             DC    X'05060000'
    000440  000001D4                      197             DC    X'000001D4'              16877056         ARGUMENT LIST
    000444  0000019C                      198             DC    X'0000019C'              2
    000448  000004B4                      199             DC    X'000004B4'              '(3H***,5F5.2)
    00044C  00000000                      200             DC    X'00000000'
    000450  00000000                      201             DC    X'00000000'
    000454  00000000                      202             DC    X'00000000'
    000458  FF030010                      203             DC    X'FF030010'
    00045C  01040104                      204             DC    X'01040104'
    000460  010C0000                      205             DC    X'010C0000'
    000464  000003E8                      206             DC    X'000003E8'              A                ARGUMENT LIST
    000468  00000494                      207             DC    X'00000494'
    00046C  FF030010                      208             DC    X'FF030010'
    000470  04060000                      209             DC    X'04060000'
    000484                                210             ORG   DIALOG@@+564
    000484  00000410                      211             DC    X'00000410'              ARRAY DESCRIPTOR
    000488  00000424                      212             DC    X'00000424'
    00048C  0000040C                      213             DC    X'0000040C'
    000490  00010004                      214             DC    X'00010004'
    000494  000003E8                      215             DC    X'000003E8'              ARRAY DESCRIPTOR
    000498  00000410                      216             DC    X'00000410'
    00049C  000003E4                      217             DC    X'000003E4'
    0004A0  00010004                      218             DC    X'00010004'
    0004B4                                219             ORG   DIALOG@@+612
    0004B4  00000000                      220             DC    X'00000000'              OBJECT FORMAT DESCRIPTOR
    0004B8  00000000                      221             DC    X'00000000'
    0004BC  00000010                      222             DC    X'00000010'
    0004C0  000001EA                      223             DC    X'000001EA'


  ****  OBJECT  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:41:01   PAGE  5
                                   PROGRAM UNIT: DIALOG
FLG DISPL   OPERATION       ADDR1  ADDR2  STMNT    ASSEMBLY CODE                          SYMB.ADDR1, SYMB.ADDR2
-------------------------------------------------------------------------------------------------------------------------

    0004C8                                224             END   DIALOG
```

```
****  OBJECT  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:43:33   PAGE  6
                                 PROGRAM UNIT: SUBA
FLG DISPL   OPERATION       ADDR1  ADDR2  STMNT    ASSEMBLY CODE                          SYMB.ADDR1, SYMB.ADDR2

                                          1    ************************************
                                          2    **        C O D E    A R E A        **
                                          3    ************************************
                                          4    *
    000000                                5    SUBA     EQU   *
    000000                                6    *
                                          7    ***** STATEMENT 1 (ENTRY) ************
                                          8    *       SUBROUTINE SUBA(X)
                                          9    *                                                  **** SEGMENT 1 ****
                                         10    * CODE SLICE BEGIN *                  (SLICE 1)
    000000                               11             USING *,15
    000000                               12             USING *,11
    000000  90 EC D00C      00000C       13             STM   14,12,12(13)
    000004  45 E0 F018      000018       14             BAL   14,24(0,15)
    000008  03                           15             DC    AL1(03)
    000009  E2E4C2C1404040               16             DC    CL7'SUBA   '
    000010  000000C8                     17             DC    A(SUBA####)
    000014  000001E8                     18             DC    A(SUBA@@@@)
    0000C8                               19             USING SUBA####,12
    0001E8                               20             USING SUBA@@@@,13
    000018  18 BF                        21             LR    11,15
    00001A  18 9D                        22             LR    9,13
    00001C  58 D0 B014      000014       23             L     13,20(0,11)
    000020  58 C0 B010      000010       24             L     12,16(0,11)
    000024  50 D0 9008      000008       25             ST    13,8(0,9)
    000028  17 77                        26             XR    7,7
    00002A  58 70 D04C      000234       27             L     7,76(0,13)
    00002E  D2 03 D110 700C 0002F8 00000C 28             MVC   272(4,13),12(7)
    000034  50 D0 700C      00000C       29             ST    13,12(0,7)
    000038  92 EC 9000      000000       30             MVI   0(9),236
    00003C  95 00 D000      0001E8       31             CLI   0(13),0
    000040  47 80 B068      000068       32             BC    8,104(0,11)
    000044  90 C0 9068      000068       33             STM   12,0,104(9)
    000048  41 00 0403      000403       34             LA    0,1027(0,0)
    00004C  18 D9                        35             LR    13,9
    00004E  58 E0 900C      00000C       36             L     14,12(0,9)
    000052  58 C0 9044      000044       37             L     12,68(0,9)
    000056  D2 02 D001 B061 0001E9 000061 38             MVC   1(3,13),97(11)
    00005C  47 F0 C02C      0000F4       39             BC    15,44(0,12)
    000060  00000064                     40             DC    AL4(*+4)
    000064  98 C0 9068      000068       41             LM    12,0,104(9)
    000068  50 90 D004      0001EC       42             ST    9,4(0,13)
    00006C  58 A0 1000      000000       43             L     10,0(0,1)
    000070  D2 03 D11C A000 000304 000000 44             MVC   284(4,13),0(10)        X
    000076  45 90 B0AE      0000AE       45             BAL   9,174(0,11)            %L3
                                         46    *** END OF PROLOG
    00007A                               47             USING *,15
    00007A  58 20 D004      0001EC       48             L     2,4(0,13)
    00007E  58 20 2018      000018       49             L     2,24(0,2)
    000082  58 40 2000      000000       50             L     4,0(0,2)
    000086  D2 03 4000 D11C 000000 000304 51             MVC   0(4,4),284(13)         X
    00008C  58 E0 D04C      000234       52             L     14,76(0,13)
    000090  58 F0 D110      0002F8       53             L     15,272(0,13)
    000094  58 D0 D004      0001EC       54             L     13,4(0,13)
    000098  50 F0 E00C      00000C       55             ST    15,12(0,14)
    00009C  92 00 D000      0001E8       56             MVI   0(13),0
    0000A0  58 E0 D00C      0001F4       57             L     14,12(0,13)
```

```
****  OBJECT  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:43:33   PAGE  7
                                  PROGRAM UNIT: SUBA
FLG DISPL   OPERATION       ADDR1  ADDR2  STMNT    ASSEMBLY CODE                          SYMB.ADDR1, SYMB.ADDR2
    0000A4  58 F0 D010      0001F8           58            L     15,16(0,13)
    0000A8  98 2C D01C      000204           59            LM    2,12,28(13)
    0000AC  07 FE                            60            BCR   15,14
                                             61            DROP  15
                                             62    *** END OF EPILOG
                                             63    *** BEGIN OF TRAILER
    0000AE  50 90 D0B4      00029C           64    %L3     ST    9,180(0,13)
                                             65    ***** STATEMENT 2 (ASSIGNMENT) *******
                                             66    *       Y=X*X
    0000B2  78 00 D11C      000304           67    %L4     LE    0,284(0,13)              X
    0000B6  7C 00 D11C      000304           68            ME    0,284(0,13)              X
                                             69    ***** STATEMENT 3 (END) **************
                                             70    *       END
    0000BA  70 00 D120      000308           71            STE   0,288(0,13)              Y
    0000BE  58 F0 D0B4      00029C           72            L     15,180(0,13)
    0000C2  17 11                            73            XR    1,1
    0000C4  05 EF                            74            BALR  14,15
                                             75    ************************************
                                             76    **     C O N S T A N T   A R E A    **
                                             77    ************************************
                                             78    *
    0000C6                                   79            ORG
    0000C8                                   80    SUBA#### DS    0D
    0000C8                                   81            USING *,12
                                             82    *                                      RUNTIME COMMUNICATION AREA
    0000C8  F3F0C1E4C7F9F1                   83            DC    '30AUG91'
    0000CF  5B081E0F2B219135                 84            DC    X'5B081E0F'
    0000D7  40C6D6D9F140E74040E5F24BF2C1F0   85            DC    ' FOR1 X  V2.2A00'
    0000E7  A0                               86            DC    X'A0'
    0000E8  00000000                         87            DC    A(SUBA)
    0000EC  FFFFFFFF                         88            DC    F'-1'
    0000F0  41000411                         89            DC    X'41000411'
    0000F4  58F0C08C                         90            DC    X'58F0C08C'
    0000F8  07FF                             91            DC    X'07FF'
    0000FC                                   92            ORG   SUBA####+52
    0000FC  00000000                         93            DC    A(SUBA+0)              CODE SLICE ADDRESS (SLICE 1)
    000154                                   94            ORG   SUBA####+140
    000154  00000000                         95            DC    V(IF@ERROR)
    000158  00000000                         96            DC    V(IF@SINI)
    000160                                   97            ORG   SUBA####+152
    000160  00000000                         98            DC    X'00000000'            0.00000000000000000000000000000000+00
    000164  00000000                         99            DC    X'00000000'
    000168  00000000                        100            DC    X'00000000'
    00016C  00000000                        101            DC    X'00000000'
    000180                                  102            ORG   SUBA####+184
    000180  00000001                        103            DC    X'00000001'            1
    000190                                  104            ORG   SUBA####+200
    000190  4E000000                        105            DC    X'4E000000'            0.00000000000000000E+00
    000194  00000000                        106            DC    X'00000000'
    000198  4E000001                        107            DC    X'4E000001'            0.42949672960000000E+10
    00019C  00000000                        108            DC    X'00000000'
    0001A0  CE000000                        109            DC    X'CE000000'            -0.21474836480000000E+10
    0001A4  80000000                        110            DC    X'80000000'
    0001A8  40                              111            DC    ' '
                                            112    *                                      EVENTHANDLER LIST
```

```
   ****  OBJECT  LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:43:33   PAGE  8
                                                PROGRAM UNIT: SUBA
FLG DISPL   OPERATION       ADDR1  ADDR2  STMNT    ASSEMBLY CODE                              SYMB.ADDR1, SYMB.ADDR2

    0001B8                                  113             ORG   SUBA####+240
    0001B8  01E70000                        114             DC    X'01E70000'
    0001BC  000001D0                        115             DC    X'000001D0'              ADDR OF PU_NAME
    0001C0  00000000                        116             DC    V(IF@RETN)               TERMINATION ROUTINE
    0001C4  00000000                        117             DC    V(IF@CTER)               PROCHK ROUTINE
    0001C8  00000000                        118             DC    V(IF@CTER)               ERROR ROUTINE
    0001CC  FFFFFFFF                        119             DC    X'FFFFFFFF'              OTHER EVENTS ROUTINE
    0001D0  04                              120             DC    X'04'
    0001D1  E2E4C2C1                        121             DC    'SUBA'
    00013C                                  122             ORG   SUBA####+116
    00013C  000001D8                        123             DC    A(SUBA####+272)          ADDRESS OF STATEMENT TABLE
    0001D8                                  124             ORG   SUBA####+272
    0001D8  0000000D00FC0000                125             DC    X'0000000D00FC0000'
    0001E0  0001590406                      126             DC    X'0001590406'
                                            127     ************************************
                                            128     **        D A T A    A R E A        **
                                            129     ************************************
                                            130     *
    0001E6                                  131             ORG
    0001E8                                  132             DS    0D
    0001E8                                  133     SUBA@@@@ CSECT
    0001E8                                  134             USING *,13
    0001E8                                  135             DS    312C
    0001E8                                  136             ORG   SUBA@@@@+0
    0001E8  0001FEFF                        137             DC    X'0001FEFF'              SAVE AREA
    000230                                  138             ORG   SUBA@@@@+72
    000230  00000000                        139             DC    A(I@@@RTCA)              RUNTIME COMMUNICATION AREA
    000234  00000000                        140             DC    V(IT0PCD)
    000238  000001B8                        141             DC    X'000001B8'              A(IT0EHL)
    000238                                  142             ORG   SUBA@@@@+80
    000238  000001D800000000                143             DC    X'000001D800000000'      STMT TABLE AND COUNT TABLE ADDR
    000288                                  144             ORG   SUBA@@@@+160
    000288  CE00000000000000                145             DC    X'CE00000000000000'
    000294                                  146             ORG   SUBA@@@@+172
    000294  03                              147             DC    X'03'
    000295  E2E4C2C1404040                  148             DC    'SUBA   '
    0002FC                                  149             ORG   SUBA@@@@+276
    0002FC  82000310                        150             DC    A(SUBA####+296)          ADDRESS OF FORMAT LABEL TABLE
    000300  00000320                        151             DC    A(SUBA@@@@+312)          DATA SLICE ADDRESS (ITA)
    000320                                  152             END
```

A.6.8 Überblicklisten für Haupt- und Unterprogramm

```
****  SUMMARY LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:41:01   PAGE  1
                                          PROGRAM UNIT: DIALOG
OBJECT MODULES GENERATED

( NAME | LENGTH | TYPE )
                      DIALOG     |  0004C8 (    1224)  |  CODE & DATA MODULE OF MAIN PROGRAM
COMPILATION STATISTICS

    OPTIMIZATION :          2 COMMON SUBEXPRESSIONS
                            2 CONSTANT EVALUATIONS
                            1 STRENGTH REDUCTIONS
     DIAGNOSTICS :          0 NOTES                       MEMORY USAGE :       582 VIRTUAL MEMORY PAGES
                            1 WARNINGS                                           0 WORK FILE PAGES
                            0 ERRORS
                            0 SEVERE ERRORS
                            0 FAILURES
                                                          COMPILE TIME :       152 SECONDS ELAPSED TIME
  SOURCE PROGRAM :         11 SOURCE LINES                                    1167 LINES / MINUTE CPU TIME
                            1 COMMENT LINES                                   .565 SECONDS CPU TIME
                                                                                   (COMPILER NOT PRELOADED)
****  SUMMARY LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:43:33   PAGE  2
                                          PROGRAM UNIT: SUBA
OBJECT MODULES GENERATED

( NAME | LENGTH | TYPE )
                      SUBA       |  000320 (     800)  |  CODE & DATA MODULE OF SUBROUTINE
COMPILATION STATISTICS

    OPTIMIZATION :            ACTIVE
     DIAGNOSTICS :          0 NOTES                       MEMORY USAGE :       582 VIRTUAL MEMORY PAGES
                            0 WARNINGS                                           0 WORK FILE PAGES
                            0 ERRORS
                            0 SEVERE ERRORS
                            0 FAILURES
                                                          COMPILE TIME :         6 SECONDS ELAPSED TIME
  SOURCE PROGRAM :          3 SOURCE LINES                                     420 LINES / MINUTE CPU TIME
                            0 COMMENT LINES                                   .429 SECONDS CPU TIME
                                                                                   (COMPILER NOT PRELOADED)
```

A.6.9 Gesamt-Überblickliste

```
****  SUMMARY LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:43:39   PAGE  3
OBJECT MODULES GENERATED

( NAME | LENGTH | TYPE )
                      DIALOG     |  0004C8 (    1224)  |  CODE & DATA MODULE OF MAIN PROGRAM
                      SUBA       |  000320 (     800)  |  CODE & DATA MODULE OF SUBROUTINE
COMPILATION STATISTICS

    OPTIMIZATION :          2 COMMON SUBEXPRESSIONS
                            2 CONSTANT EVALUATIONS
                            1 STRENGTH REDUCTIONS
     DIAGNOSTICS :          0 NOTES                       MEMORY USAGE :       582 VIRTUAL MEMORY PAGES
                            1 WARNINGS                                           0 WORK FILE PAGES
                            0 ERRORS
                            0 SEVERE ERRORS
                            0 FAILURES
                                                          COMPILE TIME :       158 SECONDS ELAPSED TIME
  SOURCE PROGRAM :         14 SOURCE LINES                                    836 LINES / MINUTE CPU TIME
                            1 COMMENT LINES                                 1.005 SECONDS CPU TIME
                                                                                   (COMPILER NOT PRELOADED)
```

A.6.10 Optionenliste

```
****  OPTIONS LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:43:39   PAGE  1
ENVIRONMENT

                          PROCESSOR          :     UNDEFINED
                          OPERATING SYSTEM   :     BS2000 V10.0
                          COMPILER           :     F O R  1    V2.2A00
                          USER-ID.           :     F3390052
                          TSN                :     91TE
                          CALLER             :     UNDEFINED
                          PREPROCESSOR       :     UNDEFINED
OPTIONS FILE

                          |COMOPT SOURCE=S.DIALOG                                                  |
                          |COMOPT LISTFILE=L.DIALOG(SOURCE,OPTIONS,MAP,ESD,CHANGE,XREF,OBJECT,SUMMA|
                          |       RY)                                                              |
                          |COMOPT COLLECT=(LF)                                                     |
                          |END                                                                     |
OPTIONS IN EFFECT

( D = DEFAULT )
( P = PARAM COMMAND )
                               SOURCE            = S.DIALOG
                          D    CODE              = EBCDIC
                               UPD               = SRCFCB
                          D    UPD CODE          = EBCDIC
                          D    INCLUDE-LIBRARY   = *NO
                          D    CCOM              = ''
                          D    LINEEND           = ''
                          D    IMPLICIT
                          D    STANDARD-CHECK    = NO
                          D    EXTENDED-SYSTEM   = YES
                          D    SYMTEST           = MAP
                          D    LANGUAGE          = ENGLISH
                          D  NODIALOG
                               LIST              = ( SOURCE, ESD, MAP, XREF, OBJECT, SUMMARY, OPTIONS )
                               LIST-OUTPUT       = L.DIALOG
                          D    LINECNT           = 64
                               COLLECT           = ( LISTFILE )
                          D    EJECT
                          D    EXPAND
                          D    TEXT-SEPARATOR    = |
                          D    OBJECT            = ( * )
                          D    MODULE-LIBRARY    = *OMF
                          D  NOFPOOL
                               PROCEDURE         = NO
                          D    ALIGN
                          D  NOSUPPLIEDBOUND
                          D    TRUNCONST
                          D  NOPAD
                          D  NOCOMPATIBLE
                          D    GEN
                          D    OPTIMIZE          = 1


****  OPTIONS LISTING  ****      SIEMENS-NIXDORF FORTRAN COMPILER   FOR1 V2.2A00  DATE = 1991-08-30 TIME = 15:43:39   PAGE  2
                          D    SAVE-CONSTANT     = YES
                          D    REAL              = 4
                          D    UNIT
                          D    TESTOPT           = ( STNR )
                          D  NOEXPUNDERFLOW
                          D    ERRKILL           = FAILURE
                          D    MAXERR            = 100
                          D    SOURCE-FORMAT     = FIXED
                          D    MSGLEVEL          = WARNING
                          D    FORTRAN90-CHECK   = YES
                          D    LINKAGE           = STD
```

A.7 Koexistenz von ALT-, NXS- und XS-Programmen

Der FOR1-Compiler erzeugt ab Version 2.2A grundsätzlich XS-Bindemoduln. Bei den FOR1-Versionen 2.0A und 2.1A konnte der Anwender mit der Option EXTENDED-SYSTEM={YES|NO} steuern, ob die übersetzten Programmeinheiten als XS- oder als NXS-Moduln (Voreinstellung war NO) erzeugt werden sollten.

Bei NXS-Programmen hat die Parameteradreßliste ein Format, das sowohl mit dem Format der Parameteradreßliste von ALT-Programmen (= Programme, die mit einer FOR1-Version ≤1.6A übersetzt wurden), als auch mit dem Format der Parameteradreßliste von XS-Programmen kompatibel ist. NXS-Unterprogramme verarbeiten deshalb neben NXS- auch ALT- und XS-Parameteradreßlisten, sofern bei letzteren nur 24 Bit der 31-Bit-Adressen relevant sind.

A.7.1 31-Bit-Adreßmodus und 24-Bit-Adreßmodus

Ein XS-Bindemodul hat die Eigenschaften AMODE=ANY und RMODE=ANY, ein NXS-Bindemodul die Eigenschaften AMODE=24 und RMODE=24. Diese Eigenschaften werden vom Binder-/Ladersystem ausgewertet bzw. können durch das Binder-/Ladersystem noch geändert werden. Schließlich kann der durch den Lader ermittelte Adreßmodus noch durch die Laufzeitoption START=XS aktualisiert werden (siehe Abschnitt 6.3.5).

Somit liegen die Ladeadresse und der Maschinenadreßmodus, in dem ein Programm abläuft, erst nach dem Starten des Programms fest. Ein Programm kann entweder im 24-Bit-Raum oder im 31-Bit-Raum ablaufen.

Im 24-Bit-Raum läuft ein geladenes Programm (d.h. Code und statische Daten und dynamische Daten) ab, wenn es im 24-Bit-Raum liegt und im 24-Bit-Adressierungsmodus der Hardware abgearbeitet wird.

Im 31-Bit-Raum läuft ein geladenes Programm ab, wenn es im 31-Bit-Raum liegt und im 31-Bit-Adressierungsmodus der Hardware abgearbeitet wird.

Die Bilder A.7-1 und A.7-2 zeigen die verschiedenen Möglichkeiten, nach denen entweder ein NXS-Programm, ein XS-Programm im Adreßraum unterhalb 16 Megabyte oder ein XS-Programm im Adreßraum oberhalb 16 Megabyte ablaufen.

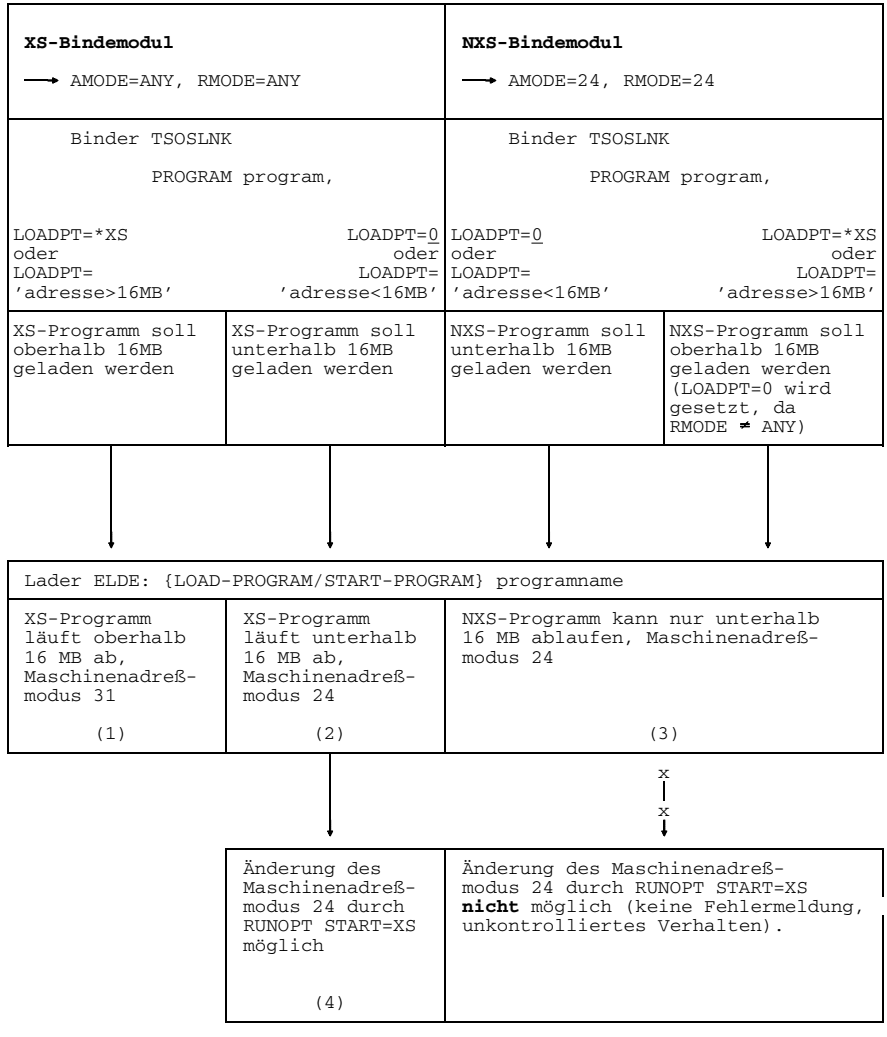


Bild A.7-1: Ladeadresse und Maschinenadreßmodus beim Binden mit TSOSLNK uns Laden mit ELDE

Zu (1),(4): Dynamischer Speicherplatz wird bei CALL ALLOC(...,'ANY') oberhalb 16MB angelegt. Falls dies nicht möglich ist, wird eine Fehlermeldung ausgegeben.
Bei CALL ALLOC(...,'NXS') wird dynamischer Speicherplatz unterhalb 16MB angelegt.

Zu (2),(3): Dynamischer Speicherplatz wird bei CALL ALLOC(...) unterhalb 16MB angelegt.

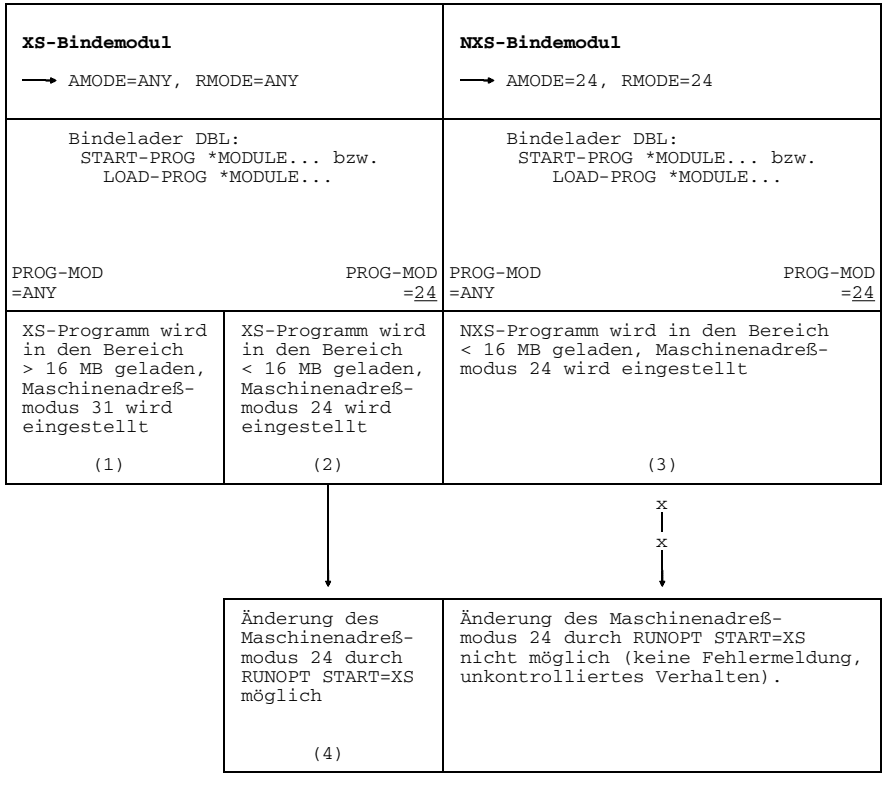


Bild A.7-2: Ladeadresse und Maschinenadreßmodus beim Binden und Laden mit dem DBL

Zu (1),(4): Dynamischer Speicherplatz wird bei CALL ALLOC(...,'ANY') oberhalb 16MB angelegt. Falls dies nicht möglich ist, wird eine Fehlermeldung ausgegeben.
Bei CALL ALLOC(...,'NXS') wird dynamischer Speicherplatz unterhalb 16MB angelegt.

Zu (2),(3): Dynamischer Speicherplatz wird bei CALL ALLOC(...) unterhalb 16MB angelegt.

A.7.2 XS-, ALT- und Glue-Programme

ALT-Programme, d.h. Programme, die mit einer Vorgängerversion des FOR1 V2.0A übersetzt wurden, können durch Neuübersetzen mit einer FOR1-Version ab 2.0A als XS-Programme erzeugt werden. Wird mit den FOR1-Versionen 2.0A oder 2.1A neu übersetzt, so muß hierfür bei der Übersetzung EXTENDED-SYSTEM=YES angegeben werden. Ab Version 2.2A erzeugt FOR1 grundsätzlich XS-Moduln, ohne daß dies durch eine Option angefordert werden müßte.

In manchen Fällen wird jedoch eine Programmverknüpfung zwischen XS-Programmen und ALT-Programmen notwendig bleiben, wenn z.B. Quellprogramme nicht mehr verfügbar sind. Bei der Verknüpfung zwischen ALT- und XS-Programmen muß der Anwender dann für die jeweils richtige Einstellung des Adressierungsmodus und für die Übergabe der Parameter in der jeweils erwarteten Form sorgen. Dies erfolgt durch Glue-Programme (glue engl. für Leim), die der Anwender selbst erstellen muß. Unter einem Glue-Programm versteht man ein Programm, das zwischen zwei Programmen eingefügt werden muß, zwischen denen kein direkter Unterprogrammaufruf möglich ist. Bei der Verknüpfung von XS-Programmen und ALT-Programmen übernimmt das Glue-Programm das Umschalten des Adressierungsmodus und das Umsetzen der Parameterformate. Die genaue Form der notwendigen Glue-Programme bei Ablauf im selben Adreßraum wird in Abschnitt A.7.3 beschrieben, bzw. bei Ablauf in verschiedenen Adreßräumen in Abschnitt A.7.4.

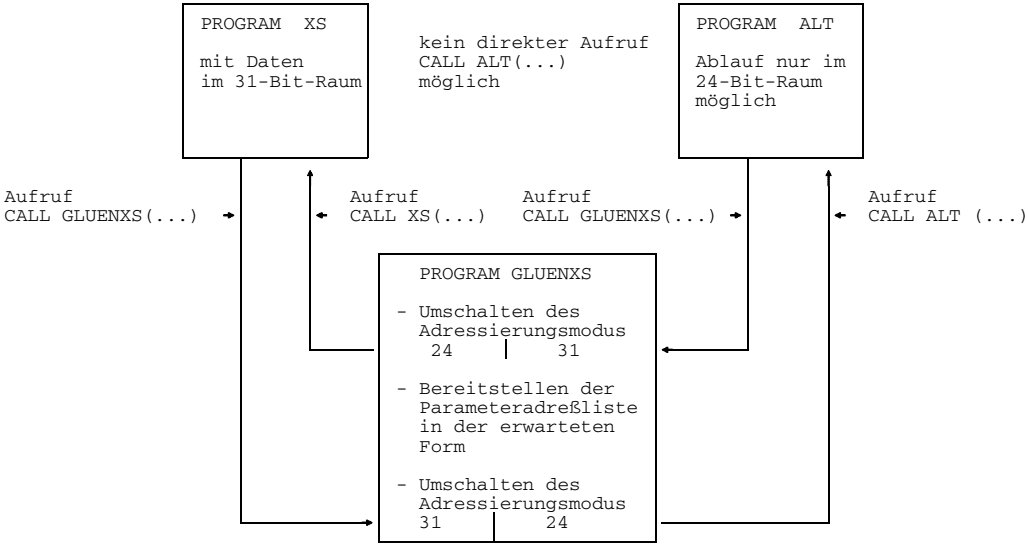


Bild A.7-3: Zwischenschalten eines NXS-Glueprogramms bei der Verknüpfung von XS- und ALT-Programmen

A.7.3 Programmverknüpfung bei Ablauf im gleichen Adreßraum

Programmverknüpfung im 31-Bit-Raum

Die einzig mögliche Programmverknüpfung ist:
XS-Programm ruft XS-Programm auf.

Programmverknüpfungen im 24-Bit-Raum ohne Zwischenschalten eines Glue-Programms

Das folgende Schema zeigt die zulässigen direkten Programmverknüpfungen im 24-Bit-Raum:

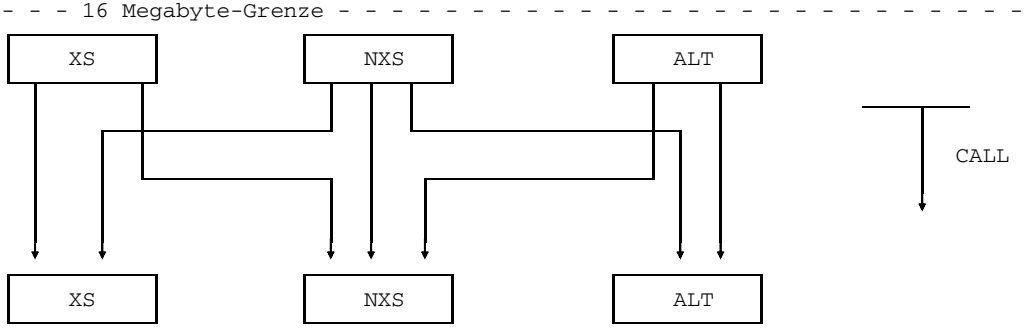


Bild A.7-4: Zulässige direkte Programmverknüpfungen im 24-Bit-Raum

Da alle Programme im 24-Bit-Raum ablaufen, d.h. im Bereich unter 16 Megabyte liegen und im 24-Bit-Adressierungsmodus der Hardware abgearbeitet werden, sind alle Programmverknüpfungen zulässig, bei denen die Parameteradreßlisten miteinander verträglich sind.

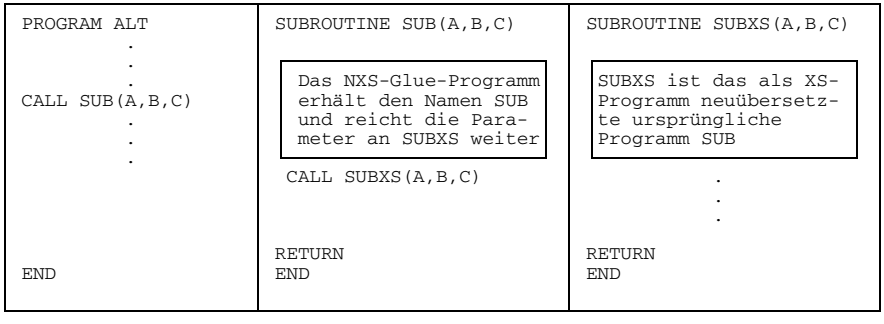
NXS-Unterprogramme verarbeiten neben XS- und NXS- auch ALT-Parameteradreßlisten. XS-Unterprogramme verarbeiten nur XS- und NXS-Parameteradreßlisten. ALT-Unterprogramme verarbeiten nur ALT- und NXS-Parameteradreßlisten.

Programmverknüpfungen im 24-Bit-Raum mit Zwischenschalten eines Glue-Programms

Bei der Programmverknüpfung zwischen XS- und ALT-Programmen muß stets ein NXS-Programm zwischengeschaltet werden.

Aufruf eines XS- durch ein ALT-Programm

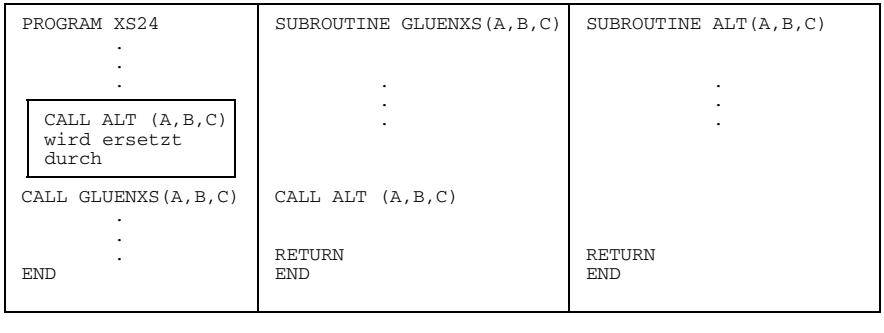
Ein ALT-Programm ruft ein XS-Programm auf, das im 24-Bit-Raum ablaufen soll. Der Anwender muß ein NXS-Programm mit dem Namen des im ALT-Programm aufgerufenen Unterprogramms schreiben. Dieses NXS-Glue-Programm hat nur die Aufgabe, die vom ALT-Programm übergebenen Parameter in einem weiteren Aufruf an das XS-Programm weiterzureichen.



Das XS-Programm liegt bei dieser Konstellation immer unterhalb 16 Megabyte und läuft immer im 24-Bit-Adreßmodus (vgl. Bild A.7-1 und A.7-2), sofern der Anwender den Adreßmodus nicht fehlerhafterweise mit RUNOPT START=XS im Adreßmodus 31 gestartet hat.

Aufruf eines ALT- durch ein XS-Programm

Ein XS-Programm, das im 24-Bit-Raum abläuft, ruft ein ALT-Programm auf. Der Anwender muß ein NXS-Unterprogramm schreiben, das nur die Aufgabe hat, die vom XS-Programm übergebenen Parameter an das ALT-Programm weiterzureichen:



A.7.4 Programmverknüpfung bei Ablauf in verschiedenen Adreßräumen

Sollen Programme im 31-Bit-Raum und Programme im 24-Bit-Raum miteinander verknüpft werden, dann muß der Anwender stets ein NXS-Glue-Programm erstellen.

Unterprogramme zur Bedienung des Maschinenadreßmodus

Zur Bedienung des Maschinenadreßmodus stehen die vorgefertigten Unterprogramme GETMODE, NXSTOXS und XSTONXS zur Verfügung. Sie können in NXS-Glueprogrammen verwendet werden.

```
CALL GETMODE (mode)
```

Das Unterprogramm GETMODE (mode) liefert den beim Aufruf eingestellten aktuellen Maschinenadreßmodus in der INTEGER*4-Variablen oder dem INTEGER*4-Feldelement mode zurück.

Mögliche Ausgabewerte für mode: 24 oder 31

Auf NXS-Anlagen wird von GETMODE der Wert 24 zurückgegeben.

```
CALL XSTONXS (progname,par1,...,parn)
```

Das Unterprogramm XSTONXS setzt die Parameteradreßliste der Parameter $par_1,...,par_n$ vom XS- in das NXS-Format um und ruft das NXS-Programm progname auf.

progname	Name eines NXS-Unterprogramms, der im rufenden XS-Programm als EXTERNAL deklariert ist.
par_i	*i*-ter an das NXS-Unterprogramm zu übergebender Parameter; $0 \le i \le n$, $0 \le n \le 408$

Das Unterprogramm XSTONXS kopiert die ihm übergebene Parameteradreßliste, entfernt den EXTERNAL-Namen und überführt die XS-Parameteradreßliste in eine entsprechende NXS-Parameteradreßliste. Der Maschinenadreßmodus wird von 31 auf 24 umgestellt. Mit der Adresse der Parameteradreßliste in Register 1 und der Parameteranzahl in Register 0 wird dann das NXS-Programm angesprungen.

Nach der Rückkehr aus dem NXS-Programm wird der ursprüngliche Maschinenadreßmodus 31 wiederhergestellt und die Kontrolle wird wieder an das Programm zurückgegeben, das das XSTONXS-Unterprogramm aufgerufen hat.

Die Parameter par_1...par_n müssen im 24-Bit-Adreßraum liegen. Liegen die Parameter im 31-Bit-Adreßraum, dann erfolgt eine Fehlermeldung. Der Anwender muß die Parameter im 31-Bit-Adreßraum vor dem Aufruf von XSTONXS durch Kopieren in entsprechende Parameter im 24-Bit-Adreßraum überführen. Falls notwendig, müssen die Parameter nach Rückkehr aus XSTONXS wieder in Parameter im 31-Bit-Adreßraum kopiert werden (siehe Bild A.7-5 und zugehöriges Beispiel).

CALL NXSTOXS (progname,par_1,...,par_n)

Das Unterprogramm NXSTOXS setzt die Parameteradreßliste der Parameter par_1,...,par_n vom NXS- in das XS-Format um und ruft das XS-Unterprogramm progname auf.

progname	Name eines XS-Unterprogramms, der im rufenden NXS-Programm als EXTERNAL deklariert ist
par_i	*i*-ter an das XS-Unterprogramm zu übergebender Parameter; $0 \leq i \leq n$, $0 \leq n \leq 408$

Das Unterprogramm NXSTOXS kopiert die ihm übergebene Parameteradreßliste, entfernt den EXTERNAL-Namen und überführt die NXS-Parameteradreßliste in eine entsprechende XS-Parameteradreßliste. Der Maschinenadreßmodus wird von 24 auf 31 umgestellt. Mit der Adresse der Parameteradreßliste in Register 1 und der Parameteranzahl in Register 0 wird dann das XS-Programm angesprungen. Nach der Rückkehr aus dem XS-Programm wird der ursprüngliche Maschinenadreßmodus 24 wiederhergestellt und die Kontrolle wird wieder an das Programm zurückgegeben, das das NXSTOXS-Unterprogramm aufgerufen hat (siehe Bild A.7-6 und zugehöriges Beispiel).

NXSTOXS muß im Maschinenadreßmodus 24 aufgerufen werden. Wird NXSTOXS im Maschinenadreßmodus 31 aufgerufen, dann tritt ein Bibliotheksprogrammfehler auf.

Fall 1: XS-Programm mit dynamisch angelegten Daten oberhalb 16 Megabyte ruft ALT-Unterprogramm

Ein XS-Programm mit dynamischen Daten oberhalb 16 Megabyte ruft ein ALT-Unterprogramm auf, das Lese- oder Schreibzugriffe auf diese Daten enthält.

Mit Ausnahme der dynamisch angelegten Daten liegt der gesamte Lademodul unterhalb 16 Megabyte, da mit dem XS-Programm ein ALT-Programm gebunden wird.

Voraussetzung:

RUNOPT START=XS

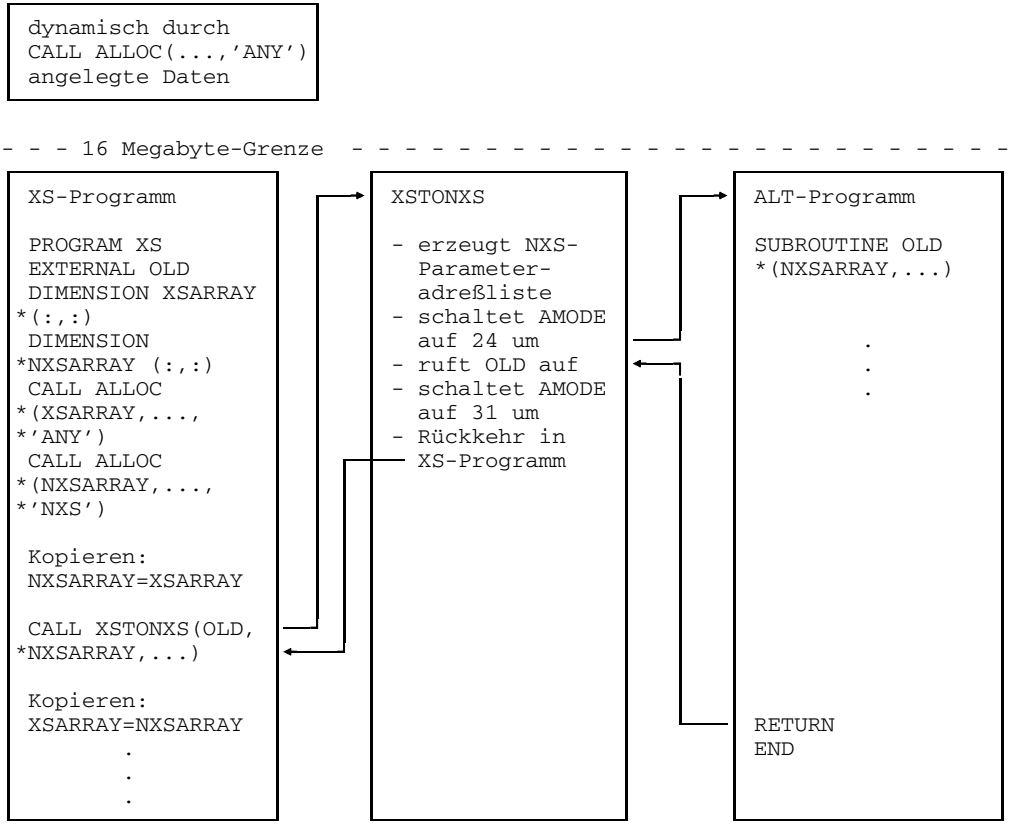
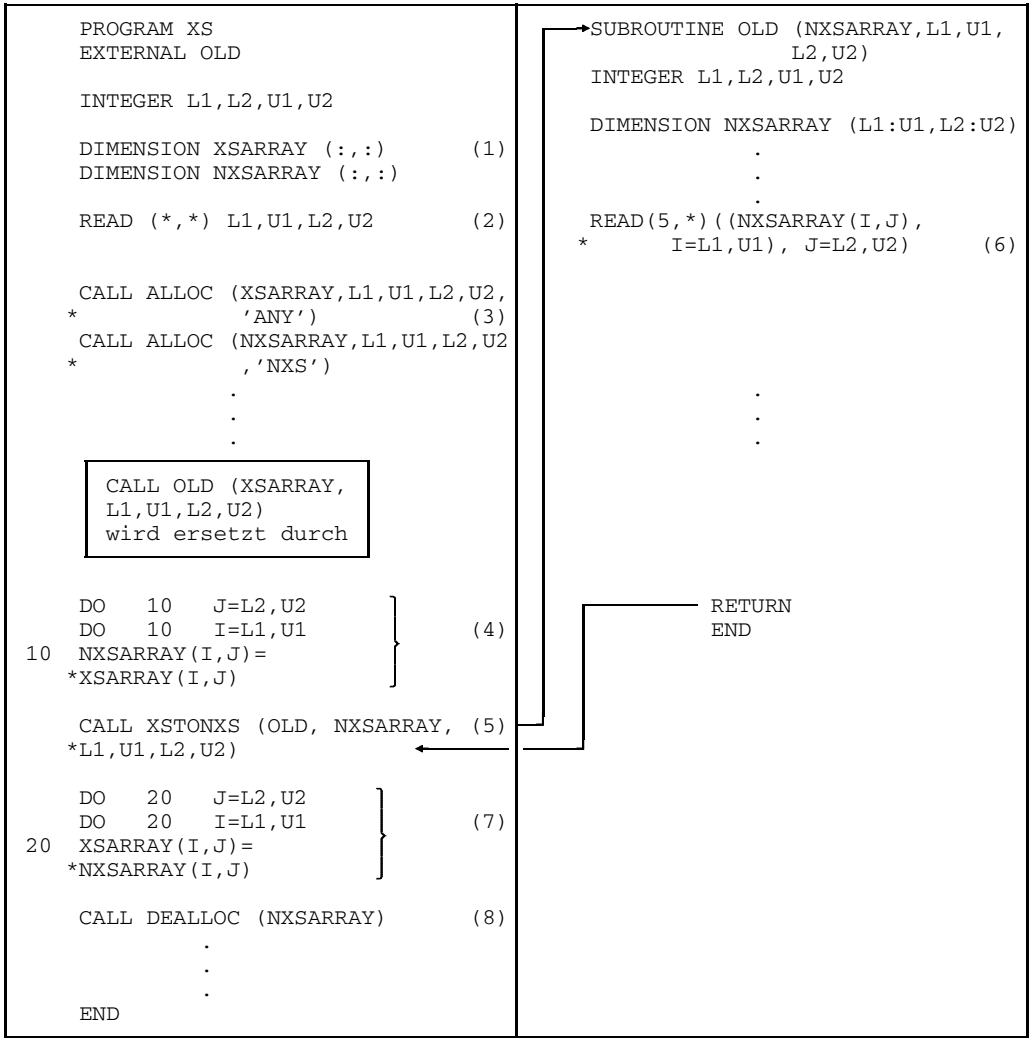


Bild A.7-5: XS-Programm mit Daten oberhalb 16 Megabyte ruft ALT-Unterprogramm

Beispiel zu Fall(1): *XS-Programm mit dynamisch angelegten Daten oberhalb 16 Megabyte ruft ALT-Unterprogramm*



(1) Die Felder XSARRAY und NXSARRAY werden als zweidimensionale Felder mit variablen Indexgrenzen angelegt.

(2) Die Indexgrenzen werden erst zur Laufzeit eingegeben.

(3) Durch CALL ALLOC wird XSARRAY als dynamisches Feld mit den Indexgrenzen (L1:U1,L2:U2) angelegt. Durch die Angabe 'ANY' wird das Feld oberhalb 16 Megabyte angelegt, falls der aktuelle Maschinenadreßmodus 31 ist, und unterhalb

16 Megabyte, falls der aktuelle Maschinenadreßmodus 24 ist. Der aktuelle Maschinenadreßmodus ist durch die Angaben beim Übersetzen, Binden, Laden und bei der Laufzeitoption RUNOPT START festgelegt (siehe Bild A.7-1 und A.7-2). Um zu erreichen, daß das dynamische Feld XSARRAY oberhalb 16 Megabyte angelegt wird, muß die Laufzeitoption RUNOPT START=XS angegeben werden.

(4) Das Feld XSARRAY oberhalb 16 Megabyte wird in das Feld NXSARRAY kopiert, das unterhalb 16 Megabyte liegt. Durch das Kopieren werden die 31-Bit-Adressen in 24-Bit-Adressen umgewandelt.

(5) Die Laufzeitroutine XSTONXS wird aufgerufen. XSTONXS hat als ersten Parameter den Namen des aufzurufenden ALT-Programms, der im XS-Programm in der EXTERNAL-Anweisung deklariert ist.
XSTONXS stellt den Maschinenadreßmodus von 31 auf 24 um und ruft das ALT-Programm auf. Nach der Rückkehr aus dem ALT-Programm wird der Maschinenadreßmodus wieder auf 31 gestellt und in das XS-Programm zurückgesprungen.

(6) Im ALT-Unterprogramm erfolgt ein Lesezugriff auf das dynamisch angelegte Feld NXSARRAY.

(7) Nach der Rückkehr aus XSTONXS werden durch Kopieren wieder 31-Bit-Adressen erzeugt.

(8) Durch CALL DEALLOC wird der dynamisch angelegte Speicherplatz wieder freigegeben.

Fall 2: ALT-Programm ruft XS-Unterprogramm mit dynamisch angelegten Daten oberhalb 16 Megabyte

Ein ALT-Programm ruft ein XS-Unterprogramm auf, das dynamisch angelegte Daten oberhalb von 16 Megabyte hat. Mit Ausnahme der dynamisch angelegten Daten liegt der gesamte Lademodul unterhalb 16 Megabyte, da mit dem XS-Programm ein ALT-Programm gebunden wird. Diese vermutlich seltene Konstellation tritt z.B. auf, wenn die Quelle eines Hauptprogramms nicht mehr vorhanden ist, während die Programme einer Unterprogrammbibliothek als XS-Programme neu übersetzt werden können.

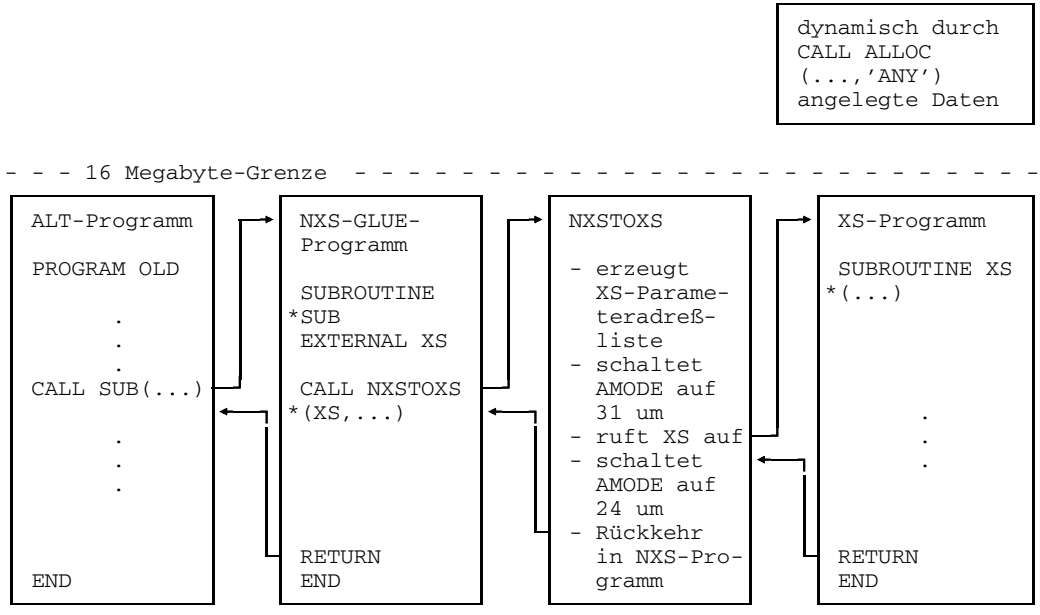
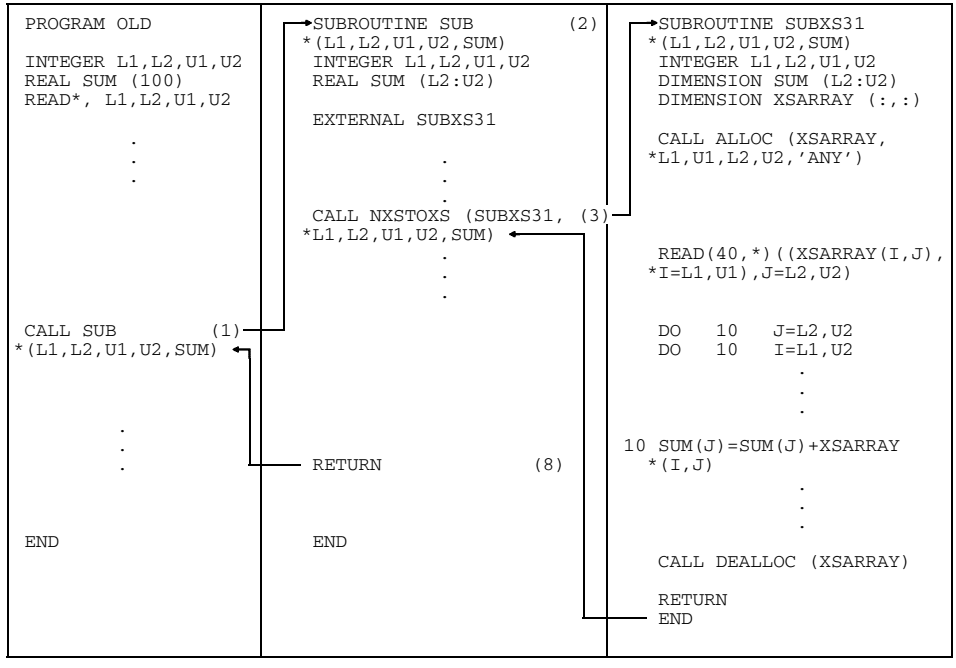


Bild A.7-6: ALT-Programm ruft XS-Programm mit dynamisch angelegten Daten oberhalb 16 Megabyte

Beispiel zu Fall(2): *ALT-Programm ruft XS-Programm mit dynamisch angelegten Daten oberhalb 16 Megabyte*



(1) Das ALT-Programm übergibt seine Parameter an ein NXS-Glue-Programm, das der Anwender erstellen muß und das den Namen des ursprünglichen Unterprogramms erhält.

(2) In diesem GLUE-Programm ist der Name des aufzurufenden XS-Programms als EXTERNAL deklariert.

(3) Die Laufzeitroutine NXSTOXS wird im Glue-Programm aufgerufen. Im Aufruf der NXSTOXS-Routine werden als erster Parameter der Name des aufzurufenden XS-Unterprogramms und als weitere Parameter die Parameter des XS-Programms angegeben.

(4) NXSTOXS ruft das XS-Programm mit der entsprechenden XS-Parameteradreßliste auf.

(5) Das ursprüngliche Unterprogramm wurde geändert und als XS-Programm übersetzt. Im XS-Unterprogramm ist das Feld XSARRAY als Feld mit variablen Indexgrenzen deklariert.

(6) Durch CALL ALLOC wird XSARRAY als dynamisches Feld angelegt mit den Indexgrenzen (L1:U2,L2:U2), die dem XS-Unterprogramm als Parameter übergeben wurden. Durch die Angabe 'ANY' wird das Feld oberhalb 16 Megabyte angelegt, falls der aktuelle Maschinenadreßmodus 31 ist, und unterhalb 16 Megabyte, falls der aktuelle Maschinenadreßmodus 24 ist. Der aktuelle Maschinenadreßmodus ist durch die Angaben beim Übersetzen, Binden, Laden und bei der Laufzeitoption RUNOPT START festgelegt (siehe Bild A.7-1 und A.7-2).

(7) Durch CALL DEALLOC wird der dynamische angelegte Speicherplatz wieder freigegeben.

(8) Nach der Rückkehr aus der NXSTOXS-Routine in das Glue-Programm wird von dort in das ursprünglich aufrufende ALT-Programm zurückgekehrt.

A.8 Sprachverknüpfungen in Nicht-ILCS-Umgebungen

A.8.1 Routinen für die Sprachverknüpfung in Nicht-ILCS-Umgebungen

Die Aufrufe von Laufzeitsystem-Initialisierungsroutinen (wie z.B. INITFOR1), Laufzeitsystem-Beendigungsroutinen (wie z.B. IF@PROT) und STXIT-Aktivierungsroutinen (wie z.B. IF@STXT) sind nur bei Sprachverknüpfungen in Nicht-ILCS-Umgebungen notwendig.

In ILCS-Umgebungen erfolgen alle beim Sprachübergang erforderlichen Maßnahmen automatisch, ein Aufruf dieser Routinen ist somit überflüssig. Trotzdem können Programme, die Aufrufe dieser Routinen enthalten, auch in ILCS-Umgebungen ablaufen, da in ILCS-Umgebungen die Aufrufe (zumindest soweit sie das FOR1-Laufzeitsystem betreffen) ignoriert werden.

Programmbeendigungsroutine IF@PROT

Die Programmbeendigungsroutine wird in Nicht-ILCS-Umgebungen benötigt bei Sprachverknüpfungen (FOR1 mit Assembler, PLI1) und bei Datenbanksystemen.

Die Programmbeendigungsroutine wird (in Nicht-ILCS-Umgebungen) aufgerufen:
- implizit vom Programmende des FOR1-Hauptprogramms;
- explizit durch die Anweisungen STOP oder CALL EXIT im Quellprogramm.

In ILCS-Umgebungen wird der Aufruf von IF@PROT ignoriert.

Reihenfolge der Funktionen

1) Bei Angabe der Testanweisung %COUNT (siehe Abschnitt 7.4.7) Ausgabe des dynamischen Programmprofils.
2) Aufruf der angemeldeten Abschlußprozeduren in der Reihenfolge absteigender Gewichtung (15, 14, ..., 0), falls diese keinen Term machen.
3) Schließen aller geöffneten FOR1-Dateien in der Reihenfolge aufsteigender Dateinummern (0, 1, ..., 99).
4) Rücksetzen (REMOVE-FILE-LINK) aller vom Quellprogramm implizit veranlaßten SET-FILE-LINK-Kommandos (falls eine neue Datei eingerichtet wurde). Reihenfolge nach aufsteigenden Dateinummern (0, 1, ..., 99).
5) Freigabe (RELM) des durch REQM angeforderten Speichers.
6) Endemeldung mit Ausgabe der verbrauchten CPU- und Anschaltzeit (elapsed).

7) Bei fehlerfreier Beendigung:

```
TERM UNIT=PRGR,MODE=NORMAL
```

Bei fehlerhafter Beendigung:

```
TERMJ UNIT=STEP,MODE=ABNORMAL,DUMP=N
```

Wird der Modul über den Entry IF@PTERM oder I$PTERM (nur aus Nicht-FORTRAN-Programmen) angesprungen, werden die Funktionen 2 bis 5 ausgeführt. Anschließend erfolgt der Rücksprung zum Aufrufer.

Soll das aufrufende Programm mit AMODE=31 laufen, dann muß statt IF@PTERM oder I$PTERM die Vorschaltroutine IF@XPTR aufgerufen werden.

IF@VAP-Routine zum Anmelden von Abschlußprozeduren

Zur Anmeldung von PLI1-Abschlußprozeduren in Nicht-ILCS-Umgebungen steht eine Laufzeitroutine mit dem Namen IF@VAP oder mit dem bedeutungsgleichen Namen I$VAP zur Verfügung. In ILCS-Umgebungen wird der Aufruf ignoriert.

Das Anmelden von PLI1-Abschlußprozeduren durch den Aufruf von IF@VAP kann nicht direkt aus einem FORTRAN-Programm erfolgen. IF@VAP kann nur von Assembler- oder PLI1-Programmen aus aufgerufen werden, die der Anwender selbst schreiben muß. Die angemeldeten PLI1-Abschlußprozeduren werden von der FOR1-Programmbeendigungsroutine IF@PROT aufgerufen. PLI1-Abschlußprozeduren dürfen keine Verknüpfungen mit FOR1-SUBROUTINEs oder -FUNCTIONs haben.

Parameter für den Aufruf des Assembler- oder PLI1-Programms

An die IF@VAP-Routine müssen drei Parameter übergeben werden, die Informationen über die aufzurufende Abschlußprozedur enthalten:

param1 INTEGER*4-Variable oder INTEGER-Konstante. Gewichtung der Abschlußprozedur. Zulässig sind die Werte 0 bis 15. Die Abschlußprozeduren werden in der Reihenfolge absteigender Gewichtung (15,14,...,0) von der Programmbeendigungsroutine IF@PROT aufgerufen.

param2 kann sein:

- Name der Abschlußprozedur.
- der Nullpointer X'FFFEFFFF' (entspricht F'-65537'). Wird der Nullpointer angegeben, dann wird die Abschlußprozedur mit der in *param1* angegebenen Gewichtung gelöscht.

param3 Schutzschlüssel mit folgenden möglichen Werten:

0 die Adresse der Abschlußprozedur mit der in *param1* angegebenen Gewichtung kann geändert oder gelöscht werden;

1 die Adresse der Abschlußprozedur mit der in *param1* angegebenen Gewichtung kann weder geändert noch gelöscht werden.

Die FOR1-Laufzeitroutine IF@VAP (I$VAP) benötigt die übergebenen Parameter in bestimmten Registern:

- der Wert von *param1* (Gewichtung) muß in Register 1 geladen werden;
- der Wert von *param2* (Adresse der Abschlußprozedur) muß in Register 2 geladen werden;
- der Wert von *param3* (Schutzschlüssel) muß in Register 3 geladen werden.

Diese Register müssen im Assembler- oder PLI1-Programm versorgt werden.

Rückmeldungen der Routine IF@VAP(I$VAP)

Mögliche Rückmeldungen in Register 1:

0 Die Anmeldung der Abschlußprozedur wurde akzeptiert.

1 Für *param1* wurde eine unzulässige Gewichtung angegeben.

2 Für *param3* wurde ein ungültiger Schutzschlüssel übergeben.

3 Eine geschützte Adresse sollte verändert werden.

Mögliche Rückmeldungen in Register 2:

undefiniert falls in Register 1 ein Wert ungleich 0 rückgemeldet wurde.

Nullpointer X'FFFEFFFF' falls für die Gewichtung *param1* bisher keine Abschlußprozedur angemeldet war (X'FFFEFFFF' entspricht F'-65537').

alte Adresse alte Adresse, die bisher unter der Gewichtung *param1* gemeldet war.

In der folgenden Übersicht ist die Anmeldung von Abschlußprozeduren in Nicht-ILCS-Umgebung durch IF@VAP schematisch dargestellt.

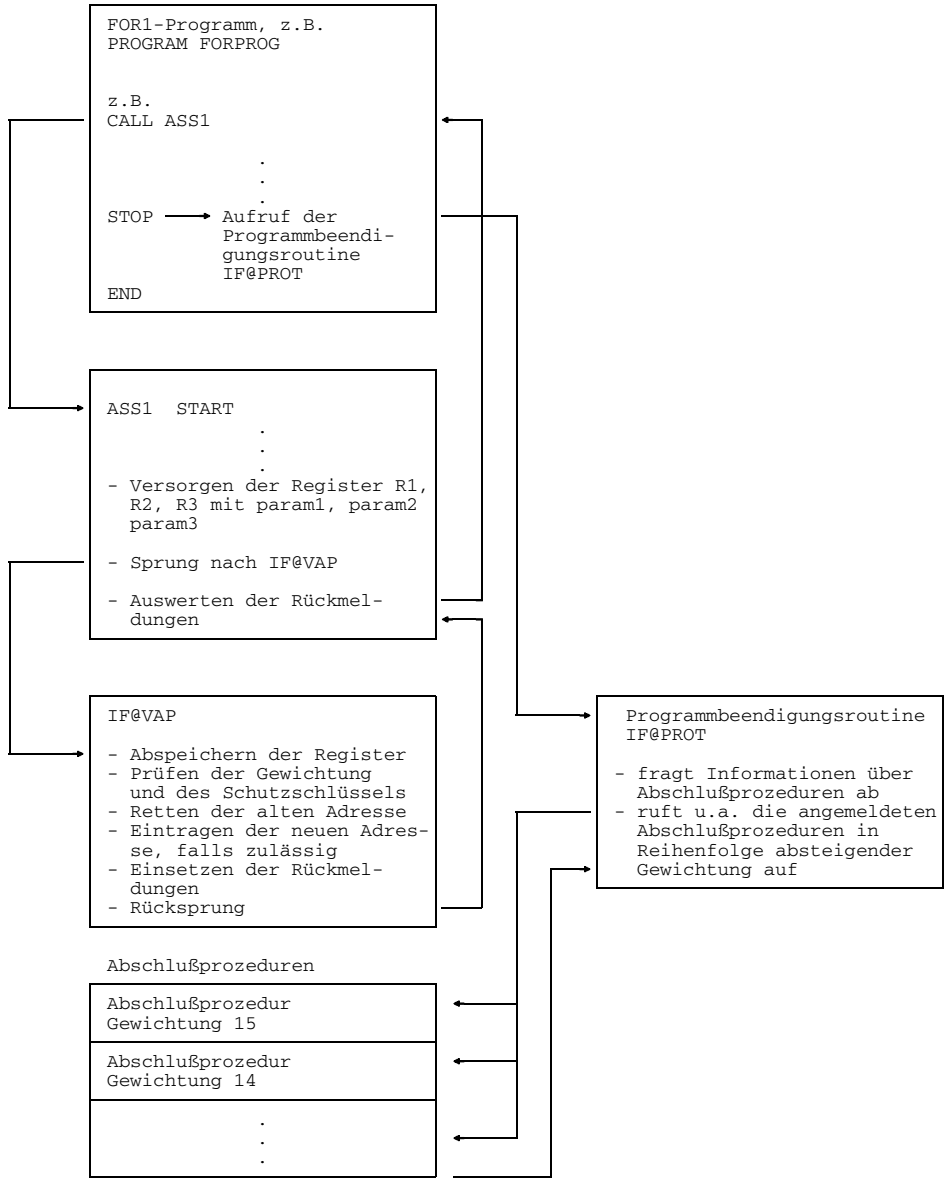


Bild A.8-1: Anmelden von Abschlußprozeduren durch IF@VAP

FOR1 STXIT-Routine

In Nicht-ILCS-Umgebungen erfolgt die Einrichtung der FOR1-Programmaske und das Anmelden der FOR1-STXIT-Routine normalerweise beim Starten des FOR1-Objektmoduls. Ab FOR1 V1.5 wird STXIT (CONTINGENCY) benutzt. Das Anmelden der FOR1-STXIT-Routine kann durch die Laufzeitoption RUNOPT STXIT=NO unterdrückt werden.

Falls in einem aufgerufenen Nicht-FORTRAN-Unterprogramm eigene STXIT-Routinen angemeldet werden oder eine andere Programmaske gesetzt wird kann dadurch die FOR1-STXIT-Routine deaktiviert werden.

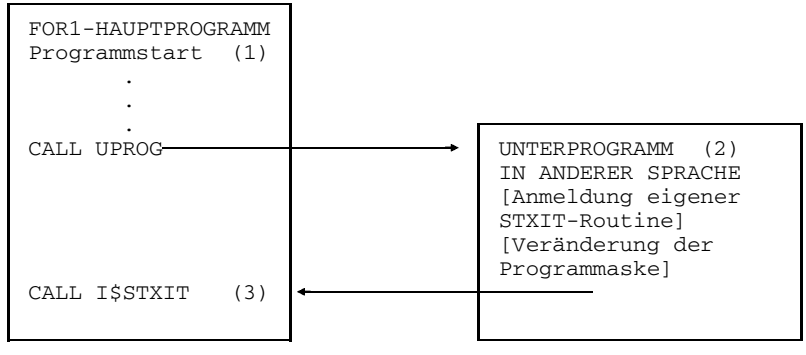
Wird im umgekehrten Fall ein FORTRAN-Unterprogramm von einem anderssprachigen Programm aufgerufen, dann kann es notwendig sein, die FOR1-STXIT-Routine im FORTRAN-Unterprogramm zu aktivieren.

In ILCS-Umgebungen sind Aufrufe der FOR1-STXIT-Routine überflüssig und werden ignoriert. Wird ein Programm, das Aufrufe von FOR1-STXIT-Routinen enthält mit LINKAGE=STD und TESTOPT=(ARG) übersetzt, gibt der Compiler folgende Warnung aus:

```
SA 249  ILCS-DEVIATION:  USELESS FUNCTION CALL
```

Auch die Angabe der Laufzeitoption RUNOPT=NO bleibt in ILCS-Umgebungen wirkungslos.

Beispiel: Aufruf der FOR1-STXIT-Routine (Nicht-ILCS-Umgebung)



Erläuterung des Beispiels:

(1) Die Entries IF@STXIT bzw. I$STXIT der STXIT-Routine befinden sich im Modul IF@INIT, der standardmäßig zu jedem FOR1-Bindemodul gebunden wird. Bei Programmstart wird durch den Modul IF@INIT die STXIT-Routine aktiviert. Dadurch werden die FOR1-Fehlerbehandlungsroutinen aktiviert.

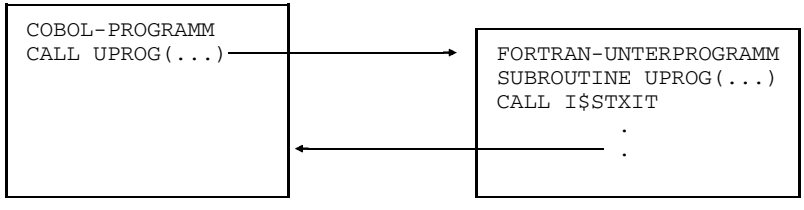
Aktionen des IF@STXIT- bzw. des I$STXIT-Entry:

- Abspeichern der Register
- Setzen der Programmaske auf X'0C', wenn mit COMOPT NOEXPUNDERFLOW übersetzt wurde (Voreinstellung), bzw.
- Setzen der Programmaske auf X'0E', wenn mit COMOPT EXPUNDERFLOW übersetzt wurde.
- Ausführen des STXIT-Makros mit Streichung aller Ausgänge, die durch FOR1 nicht ausgewertet werden.
- Zurückladen der Register
- Rücksprung

(2) Im Nicht-FORTRAN-Unterprogramm kann eine eigene STXIT-Routine angemeldet und die Programmaske verändert werden.

(3) In diesem Fall muß die FOR1-STXIT-Routine durch CALL I$STXIT im FORTRAN-Hauptprogramm reaktiviert werden. Bei diesem Aufruf sind keine Parameter erforderlich.

COBOL setzt z.B. die Programmaske auf sedezimal '00'; eine STXIT-Routine wird nicht angemeldet:



Der Aufruf der FOR1-STXIT-Routine kann entfallen, wenn die FOR1-STXIT-Routine im aufrufenden Programm durch `CALL "INITFOR1"` implizit angemeldet wird.

A.8.2 Verknüpfung FOR1-COBOL in Nicht-ILCS-Umgebungen

FOR1-Programm ruft COBOL-Programm

Für den Aufruf von COBOL-Unterprogrammen aus FOR1-Programmen sind keine besonderen Vorkehrungen notwendig.

Aufruf: `CALL unterprog (par`$_1$`,... ,par`$_n$`)`

COBOL-Programm ruft FOR1-Unterprogramm

In Nicht-ILCS-Umgebungen muß vor Aufruf des FOR1-Unterprogramms die Initialisierungsroutine INITFOR1 aufgerufen werden, um eine einheitliche FOR1-Umgebung zu schaffen:

```
CALL "INITFOR1"
CALL "unterprog" USING par1,... ,parn
```

INITFOR1 darf in einem Programmsystem nur einmal aufgerufen werden. In ILCS-Umgebungen wird der Aufruf von INITFOR1 ignoriert.

A.8.3 Verknüpfung FOR1-PLI1 in Nicht-ILCS-Umgebungen

FOR1-Programm ruft PLI1-Unterprogramm

In Nicht-ILCS-Umgebungen muß in der PROCEDURE- bzw. ENTRY-Anweisung des PLI1-Programms das Attribut OPTIONS (FORTRAN) angegeben werden.

– Anweisung in FOR1:

```
CALL name (par1,...,parn)
```

– Anweisung in PLI1:

```
name: {PROCEDURE} (par1,...,parn) OPTIONS (FORTRAN);
      {         }
      {ENTRY    }
```

PLI1-Programm ruft FOR1-Unterprogramm

In Nicht-ILCS-Umgebungen muß im PLI1-Programm in der DEL-Anweisung ein Sprachattribut deklariert werden:

– Anweisungen in PLI1:

```
DCL forupro ENTRY OPTIONS (FORTRAN[INTER]);
CALL forupro (par1,...parn);
```

`forupro` Name des FOR1-Unterprogramms

`FORTRAN` Es wird ein FOR1-Unterprogramm aufgerufen; STXIT wird in FOR1 nicht aktiviert.

`FORTRAN INTER`
Es wird ein FOR1-Unterprogramm aufgerufen; STXIT wird aktiviert. Programmunterbrechungen, die während des Ablaufs des FOR1-Unterprogramms auftreten, werden von der FOR1-Fehlerbehandlungsroutine bearbeitet.
Die PLI1-Fehlerbehandlung ist abgeschaltet. Bei Fehlerabbruch im FOR1-Unterprogramm wird aber auf jeden Fall die PLI1-Endebehandlung durchgeführt. Bei Rückkehr ins PLI1-Programm wird die PLI1-Unterbrechungsbehandlung (STXIT) wieder aktiviert.

– Anweisungen in FOR1:

```
{SUBROUTINE forupro (par1,...,parn)}
{FUNCTION forupro (par1,...,parn)  }
```

A.8.4 Verknüpfung FOR1-C in Nicht-ILCS-Umgebungen

C-Programm ruft FOR1-Programm

In Nicht-ILCS-Umgebungen muß das FOR1-Unterprogramm im C-Quellprogramm deklariert werden. Falls das FOR1-Unterprogramm die FOR1-Laufzeitumgebung benötigt, muß diese vor Aufruf des Unterprogramms initialisiert werden. In den folgenden beiden Abschnitten werden diese Schritte beschrieben.

Das gerufene FOR1-Unterprogramm darf nicht mit TESTOPT=(ARG) übersetzt worden sein.

Deklaration des FOR1-Unterprogramms im C-Quellprogramm

Bevor in einer Nicht-ILCS-Umgebung ein FOR1-Unterprogramm aufgerufen werden kann, muß es mit dem Sprachattribut for1 versehen werden. Dies geschieht mit der Präprozessor-Anweisung #pragma, z.B.

```
#pragma for1
void forsub();   /* SUBROUTINE-Unterprogramm */

#pragma for1
<typ> forsub();  /* FUNCTION-Unterprogramm mit Datentyp <typ> */
```

forsub ist der mit einer SUBROUTINE-, FUNCTION- oder ENTRY-Anweisung vereinbarte Einsprungname.

Das Sprachattribut läßt sich auch blockweise für mehrere Unterprogramme angeben, z.B.

```
#pragma for1 {
        forsub1();
        char forsub2();
        forsub3();
#pragma      }  [for1]
```

Die #pragma-Anweisung und das Sprachattribut for1 müssen in einer separaten Zeile (ab Spalte 1) stehen.
Bei blockweiser Angabe kann wahlweise bei der zweiten #pragma-Anweisung nach der schließenden Klammer } das Sprachattribut for1 angegeben werden, z.B. aus Dokumentationsgründen.

In ILCS-Umgebungen darf das C-Programm keine #pragma-Anweisungen für FOR1-Unterprogramme enthalten.

Initialisieren der FOR1-Laufzeitumgebung

Wird in Nicht-ILCS-Umgebungen die FOR1-Laufzeitumgebung vom FOR1-Unterprogramm benötigt, muß diese im C-Quellprogramm mit der externen FOR1-Routine initfor1 initialisiert werden. Diese Routine wird in der FOR1-Laufzeitbibliothek zur Verfügung gestellt.

Ein FOR1-Unterprogramm benötigt das FOR1-Laufzeitsystem, wenn es

- Ein-/Ausgaben vorsieht, inkl. PAUSE, STOP,
- INTRINSIC-Funktionen aufruft,
- Potenzierungen, komplexe Arithmetik/Vergleichsoperationen, vierfach genaue Gleitpunkt-Divisionen/Vergleiche ausführt,
- Zeichen-Kettung oder die Behandlung von Zeichenketten variierender Länge vorsieht,
- Testhilfen benutzt (Testoptionen, Testanweisungen, Testhilfe-Unterprogramme) oder
- das Programm beendet.

initfor1 muß mit einer #pragma-Anweisung mit dem Sprachattribut for1 deklariert werden.

initfor1 muß vor dem ersten Aufruf eines FOR1-Unterprogramms aufgerufen werden. Pro Programmsystem darf initfor1 nur einmal aufgerufen werden. Auch bei mehrstufigem Sprachwechsel (C → FOR1 → C → FOR1 → usw.) genügt dieser eine Aufruf.

In ILCS-Umgebungen wird der Aufruf von initfor1 ignoriert.

Beispiel:

```
#pragma for1
initfor1();
#pragma for1
forsub1();
#pragma for1
float forsub2();
  .
  .
main()
{
  float x;

  initfor1();
  forsub1();
  x = forsub2();
}
```

FOR1-Programm ruft C-Programm

In Nicht-ILCS-Umgebungen sind für den Aufruf eines C-Programms aus einem FOR1-Programm besondere Maßnahmen notwendig, die in den folgenden beiden Abschnitten beschrieben werden.

Definition der externen C-Funktion im C-Quellprogramm

In Nicht-ILCS-Umgebungen müssen C-Funktionen, die von FOR1-Programmen aufgerufen werden, bei ihrer Definition mit dem Umgebungsattribut for1 versehen werden. Dies geschieht mit der Präprozessor-Anweisung #pragma:

```
#pragma for1
<typ> functname (formalparameterliste)
<typ> param-1;
<typ> param-2;
...
<typ> param-n;
{
 .
 .
}
```

Die #pragma-Anweisung und das Umgebungsattribut for1 müssen in einer separaten Zeile (ab Spalte 1) und unmittelbar vor der Funktionsdefinition stehen.

In ILCS-Umgebungen darf die aufgerufene C-Funktion keine #pragma-Anweisung zur Definition des Umgebungsattributs enthalten.

Initialisieren der C-Laufzeitumgebung

In Nicht-ILCS-Umgebungen muß das FOR1-Programm vor Aufruf der ersten C-Funktion die C-Laufzeitumgebung initialisieren. Dies geschieht durch Aufruf der C-Bibliotheksfunktionen cinit1 bzw. cinit2.
cinit1 liefert als Ausgangsparameter die Adresse des C-Laufzeitstacks. Bei cinit2 kann zusätzlich die Größe des ersten Segments des C-Laufzeitstacks bestimmt werden.
cinit1 bzw. cinit2 dürfen nur einmal pro Programmsystem aufgerufen werden:

```
INTEGER*4 ADR
  .
  .
CALL CINIT1(ADR)
```

Auch bei mehrstufigem Sprachwechsel (FOR1 → C → FOR1 → C → usw.) genügt dieser eine Aufruf.

In ILCS-Umgebungen darf das FOR1-Programm keine Aufrufe von cinit1 bzw. cinit2 enthalten.

A.9 Verknüpfung von FOR1- mit Assembler-Programmen

Von FOR1 werden in der FOR1MACLIB Makros zur Verfügung gestellt, die eine Verknüpfung von FOR1- und Assembler-Programmen ermöglichen.

Mit Hilfe dieser Makros lassen sich Assembler-Programme erzeugen, die sich bei der Sprachverknüpfung wie mit FOR1 erzeugte Nicht-ILCS-Objekte verhalten. Die Makros ermöglichen jedoch nicht die Erzeugung von ILCS-Assemblerprogrammen. Deshalb können zwar Assembler-Unterprogramme, die diese Makros verwenden, in ILCS-Umgebungen von FOR1-Programmen aufgerufen werden oder ihrerseits FOR1-Programme aufrufen - *nicht* möglich ist es jedoch, mit diesen Makros Assembler-Hauptprogramme zu erstellen, die in ILCS-Umgebungen ablauffähig sind.

Für die ILCS-Sprachverknüpfung FOR1-Assembler stellt jedoch ASSEMBH (ab Version 1.1A) entsprechende Makros zur Verfügung (siehe "ASSEMBH - Beschreibung" [10]).

Von FOR1 zur Verfügung gestellte Verknüpfungsmakros

Assembler-Programm ruft FOR1-Unterprogramm		FOR1-Programm ruft Assembler-Unterprogramm	
IFEPL	generiert die Parameteradreßliste	IFAEN	generiert die Einsprungsroutine für das Assembler-Programm
IFECL	Aufruf des FOR1-Programms	IFART	generiert die Rücksprungsroutine in das FOR1-Programm
IFESDS	generiert Deskriptor für CHARACTER-Datenelement		
IFEADS	generiert Deskriptor für ein Feld		
IFEEDS	generiert Deskriptor für ein Feldelement		
IFESAV	generiert einen Sicherstellungsbereich		

Tab. A.9-1: Makros für die Verknüpfung FOR1-Assembler

Die Verknüpfung von Assembler-Unterprogrammen mit FOR1-ILCS-Programmen ist problemlos möglich, wenn beim Makro IFART (bzw. IFARTO) die Parameterwerte FIRST=1 und LAST=12 gewählt werden.

A.9.1 Assembler-Programm ruft FOR1-Unterprogramm

Makro IFEPL (Parameteradreßlisten-Makro)

Dieser Makro generiert die an das Unterprogramm zu übergebende Parameteradreßliste bzw. die bei Verwendung des FORTRAN-Sprachmittels "RETURN i" notwendige Adreßtabelle.

Aufruf:

```
label IFEPL operandenliste

        [,PARMOD={ OLD                    }]
                 { NEW[(PL=XS|NXS)]       }
```

label — symbolische Adresse der Parameteradreßliste.

operandenliste

```
{ symbadr   [,symbadr]... }
{                         }
{ (param)   [,(param)]... }
```

symbadr — symbolische Rückkehradresse

param

```
{ E[,upadr]...                          }
{ { CH  }                               }
{ {     }   [,chadr,sdsadr]...          }
{ { CHV }                               }
{ typ [,adr]...                         }
{ (typ1) [,arradr,adsadr]...            }
{ <typ1> [,eladr,edsadr]...             }
```

E — Anzeige, daß die folgenden Parameter Unterprogrammadressen sind.

CH — Anzeige, daß die folgenden Parameter CHARACTER-Datenelemente mit fester Länge sind.

CHV — Anzeige, daß die folgenden Parameter CHARACTER-Datenelemente mit variabler Länge sind.

typ — {L1|L4|I1|I2|I4|I8|R4|R8|R16|C8|C16|C32|H}

Anzeigen für den Typ der folgenden Parameter.

L	LOGICAL*
I	INTEGER*
R	REAL*
C	COMPLEX*
H	Hollerith-Datenelement

typ 1 — {typ | CH}

upadr symbolische Adresse eines Unterprogramms.

chadr symbolische Adresse eines CHARACTER-Datenelements.

sdsadr symbolische Adresse des Deskriptors dieses Datenelements.

adr symbolische Adresse einer einfachen Variablen vom Typ ungleich CHARACTER.

arradr symbolische Adresse eines Feldes.

adsadr symbolische Adresse des Deskriptors dieses Feldes.

eladr symbolische Adresse eines Feldelements.

edsadr symbolische Adresse des Deskriptors dieses Feldelements.

PARMOD

=OLD Die gleiche Parameteradreßliste wie in FOR1-Versionen < V2.0A wird erzeugt.

=NEW [(PL={XS|NXS})]
Die gleiche Parameteradreßliste wie in FOR1-Versionen ≥ V2.0A wird erzeugt. Bei Angabe von (PL=XS) wird eine Parameteradreßliste im XS-Format, bei Angabe von (PL=NXS) eine Parameteradreßliste im NXS-Format erzeugt.

Bei CHARACTER-Datenelementen, bei Feldern und Feldelementen wird neben der Adresse des 1. Bytes des Datenelements auch ein Deskriptor dieses Datenelements übergeben. Die Deskriptoren für Datenelemente vom Typ CHARACTER, für Felder und für Feldelemente können durch die Makros IFESDS (Zeichenketten-Deskriptor), IFEADS (Feld-Deskriptor) und IFEEDS (Feldelement-Deskriptor) erzeugt werden (siehe unten).

Bei CHARACTER-Feldern und CHARACTER-Feldelementen wird kein Zeichenketten-Deskriptor übergeben, sondern der Feld- bzw. Feldelement-Deskriptor.

Die Anzahl der Parameter ist auf 255 beschränkt.

Bei INTEGER-Datenelementen mit einer Länge kleiner als 4 byte wird eine modifizierte Adresse generiert: bei Länge 1 (Adresse-3), bei Länge 2 (Adresse-2). Diese Modifikation wird vom Makro durchgeführt und braucht nicht vom Anwender angegeben zu werden.

Beispiel für einen gültigen Makroaufruf:

```
BETA IFEPL     (R8,ALPHA,SUM),
               ((I2),FELD,FDESCR),
               (<L1>,BOOL3,BOOL7,DBOOL7),
               (CH,CHAR,CHDESCR)
```

Makro IFECL (Aufruf-Makro)

Dieser Makro bildet die Schnittstelle zum FOR1-Unterprogramm. Der generierte Code übernimmt die Rettung und Wiederherstellung der Registerinhalte, die Initialisierung der FOR1-"Run Time Communication Area", das Setzen und Wiederherstellen der Programmmaske, den Ansprung zum FOR1-Programm und die Rückkehr von dort. IFECL erzeugt XS-fähigen Code, der ab Laufzeitsystem V2.0A ablauffähig ist.

Aufruf:

```
[label] IFECL PROG=name [,INIT =⎧YES⎫⎫]      [,PARLIST =⎧NO  ⎫⎫]
                           ⎩NO  ⎭⎭                    ⎪name⎪⎪
                                                      ⎩(reg)⎭⎭

                        [,RESTORE =⎧YES⎫⎫]   [,RETURNI =⎧NO  ⎫⎫]
                                   ⎩NO  ⎭⎭               ⎪name⎪⎪
                                                         ⎩(reg)⎭⎭

                        [,MAINSAV =⎧YES ⎫⎫] [,FTERM =⎧YES⎫⎫]
                                   ⎪name⎪⎪           ⎩NO  ⎭⎭
                                   ⎩(reg)⎭⎭

                        [,USREQM = ⎧NO  ⎫⎫]  [,USRELM =⎧NO  ⎫⎫]
                                   ⎩name⎭⎭               ⎩name⎭⎭
```

label — symbolische Adresse des Makroaufrufs.

PROG=name — Name des aufzurufenden FOR1-Unterprogramms. Bei fehlender Angabe wird kein Absprung durchgeführt (BALR 14,0).
Dies kann unter Umständen sinnvoll sein, wenn z.B. nur die Initialisierung gewünscht ist.

INIT

- =YES — FOR1-Initialisierung wird durchgeführt.
- =NO — Es wird keine Initialisierung durchgeführt.

PARLIST

- =NO — Dem FOR1-Unterprogramm werden keine Parameter übergeben.
- =name — Symbolische Adresse der Parameteradreßliste, welche durch den Makro IFEPL generiert werden kann.
- =(reg) — Angabe des Registers, welches die Adresse der Parameteradreßliste enthält.

RESTORE

=YES — Der durch den Makro IFECL generierte Code zerstört den Inhalt der Register und der Programmaske. Bei Angabe von RESTORE=YES wird der Inhalt gerettet und nach der Rückkehr wieder hergestellt.

=NO — Keine Rettung und Wiederherstellung.

RETURNI

=NO — Das aufgerufene FOR1-Unterprogramm verwendet nicht das Sprachmittel "RETURN i".

=name — Symbolische Adresse einer Adreßliste zur Realisierung des Sprachmittels "RETURN i". Diese Adreßliste kann mit dem Makro IFEPL generiert werden.

=(reg) — Angabe eines Registers, das die Adresse einer Adreßliste enthält.

MAINSAV

=YES — Es wird implizit eine Save Area durch den Makro IFECL generiert.

=name — Symbolische Adresse einer Save Area, in welche das FOR1-Unterprogramm die Register rettet. Diese Save Area kann durch den Makro IFESAV generiert werden.

=(reg) — Angabe eines Registers, welches die Adresse der Save Area enthält.

FTERM

=YES — Letzter Aufruf eines FOR1 Unterprogramms. Vor der Rückkehr werden alle FOR1-Dateien geschlossen, Speicher an das System zurückgegeben und der Verständigungsbereich RTCA gelöscht.

=NO — nicht letzter Aufruf eines FOR1-Unterprogramms

USREQM

=NO — Es wird keine Speicherbeschaffung durch benutzereigene Routinen gewünscht, d.h. der ganze nicht belegte Adreßraum steht für das Laufzeitsystem zur Verfügung.

=name — Symbolische Adresse einer benutzereigenen Routine zur Anforderung von Speicherplatz.

USRELM

=NO — Es wird keine Speicherfreigabe durch benutzereigene Routinen gewünscht, d.h. der ganze nicht belegte Adreßraum steht für das Laufzeitsystem zur Verfügung.

=name — Symbolische Adresse einer benutzereigenen Routine zur Freigabe des angeforderten Speicherplatzes. Eine FOR1-Initialisierung muß durch INIT=YES durchgeführt worden sein.

Die Initialisierung der Run Time Communication Area (RTCA) des FOR1 ist notwendig, wenn im FOR1-Unterprogramm Potenzierungen, Ein-/Ausgabe-Operationen oder Divisionen mit R*16 - Datenelementen durchgeführt oder mathematische Funktionen benutzt werden, ferner wenn komplexe Arithmetik eingesetzt wird. Bei mehrmaligem Durchlaufen des generierten Code wird die Initialisierung nur beim ersten Mal durchgeführt. Die Initialisierung muß jedoch ausgeschaltet sein, wenn dieser Makro in einem Assemblerprogramm benutzt wird, welches seinerseits von einem FOR1-Programm aufgerufen wurde.

Makro IFESDS (String-Deskriptor-Makro)

Dieser Makro generiert einen Deskriptor für ein Datenelement vom Typ CHARACTER. Die Adresse des Deskriptors muß in der Parameterliste an das aufgerufene FOR1-Unterprogramm übergeben werden.

Aufruf:

```
[label] IFESDS adr, ALLOCL = a [,CURRL = c]
                    [,PARMOD = {OLD                    }]
                               {NEW[(PL={XS|NXS})]     }
```

label — Symbolische Adresse des Makroaufrufs. Dieser Name erscheint als Operand im Makro IFEPL.

adr — Symbolische Adresse des zu übergebenden Datenelements.

ALLOCL=a — Länge des belegten Speicherplatzes des Datenelements.

CURRL=c — Aktuell genutzte Länge des Datenelements.
Falls nicht angegeben, wird die zugewiesene Länge angegeben.

PARMOD

=OLD — Der gleiche Deskriptor wie in FOR1-Versionen < V2.0A wird erzeugt.

=NEW [(PL={XS|NXS})]
Der gleiche Deskriptor wie in FOR1-Versionen ≥ V2.0A wird erzeugt. Bei Angabe von (PL=XS) wird ein Deskriptor im XS-Format, bei Angabe von (PL=NXS) ein Deskriptor im NXS-Format erzeugt.

Makro IFEADS (Feld-Deskriptor-Makro)

Dieser Makro generiert einen Deskriptor für ein Feld. Die Adresse des Deskriptors muß in der Parameterliste an das aufgerufene FOR1-Unterprogramm übergeben werden.

Aufruf:

```
[label] IFEADS name,ELEN = n, BOUNDS = (u:o[,u:o][,...])
```

label — Symbolische Adresse des Makroaufrufs. Dieser Name erscheint als Operand im Makro IFEPL.

name — Symbolische Adresse des zu übergebenden Feldes.

ELEN=n — Länge eines Feldelements in byte.

BOUNDS =(u:o[,u:o] [,...]) — Indexgrenzen der einzelnen Dimensionen,
u untere Grenze,
o obere Grenze

Makro IFEEDS (Feldelement-Deskriptor-Makro)

Dieser Makro generiert einen Deskriptor für ein Feldelement. Die Adresse des Deskriptors muß in der Parameterliste an das aufgerufene FOR1-Unterprogramm übergeben werden.

Aufruf:

```
[label] IFEEDS ename, aend, ELEN = n
                  [,PARMOD = {OLD|NEW}]
```

label — Symbolische Adresse des Makroaufrufs. Dieser Name erscheint als Operand im Makro IFEPL.

ename — Symbolische Adresse des zu übergebenden Datenelements.

aend — Symbolische Adresse des Feldendes (letztes Byte+1).

ELEN=n — Länge eines Feldelements in byte.

PARMOD =OLD — Der gleiche Deskriptor wie in FOR1-Versionen < V2.0A wird erzeugt.

=NEW — Der gleiche Deskriptor wie in FOR1-Versionen ≥ V2.0A wird erzeugt.

Makro IFESAV (Save-Area-Makro)

Dieser Makro generiert einen Sicherstellungsbereich (Save Area), in welchen das gerufene FOR1-Unterprogramm die Register rettet.

Aufruf:

```
[label] IFESAV
```

label Symbolische Adresse des Makroaufrufs.

Wird der Makro zum ersten Mal innerhalb eines Assemblerprogramms aufgerufen, so wird zusätzlich eine DSECT generiert, welche die Struktur des Sicherstellungsbereichs beschreibt.

Beispiel: Assemblerprogramm ruft FOR1-Unterprogramm

Das Assemblerprogramm ASSFOR1 ruft das FOR1-Unterprogramm AFOR auf.

Assemblerprogramm ASSFOR1:

```
ASSFOR1  START
         PRINT NOGEN
ANF      BALR  3,0
         USING *,3
         B     BEG
ITAB     DC    20F'0'
IERG     DC    F'0'
K        DC    F'0'
IERG1    DS    D
PARAL    IFEPL ((I4),ITAB,IND),(I4,IERG),(I4,K)                    (1)
IND      IFEADS ITAB,ELEN=4,BOUNDS=(1:20)                          (2)
DRU      DS    0CL133
SATZL    DC    X'00854040C1'
         DS    CL128
MASKE    DC    X'4020202020202120'
         DS    0F
BEG      MVI   DRU+5,X'40'
LOESCH   MVC   DRU+6(127),DRU+5
         MVC   DRU+6(27),='SPRUNG INS FORTRAN-PROGRAMM'
         WRLST DRU,FEHL
         MVI   DRU+4,X'40'
         MVI   DRU+5,X'40'
         EX    0,LOESCH
         IFECL PROG=AFOR,PARLIST=PARAL                             (3)
RUECK    MVI   DRU+4,X'40'                                         (5)
         MVC   DRU+6(35),='IM ASSEMBLER-HAUPT-PROGRAMM ZURUECK'
         WRLST DRU,FEHL
         MVI   DRU+5,X'40'
         EX    0,LOESCH
         MVC   DRU+6(41),='ERGEBNIS VOM FORTRAN-PROGRAMM UEBERGEBEN:'
         L     5,IERG
         CVD   5,IERG1
         MVC   DRU+60(8),MASKE
         ED    DRU+60(8),IERG1+4
         WRLST DRU,FEHL
         MVI   DRU+5,X'40'
         EX    0,LOESCH
         MVC   IERG1,=XL8'0'
         L     7,K
         XR    4,4
         LA    5,ITAB
SCHLEIFE A     4,0(5)
         LA    5,4(5)
         BCT   7,SCHLEIFE
         CVD   4,IERG1
         MVC   DRU+60(8),MASKE
         ED    DRU+60(8),IERG1+4
         MVC   DRU+6(37),='ERGEBNIS IM ASSEMBLER-HAUPT-PROGRAMM:'
         WRLST DRU,FEHL
         TERM
```

```
FEHL     TERMD
         END   ANF
```

FOR1-Unterprogramm AFOR

```
         SUBROUTINE AFOR(ITAB1,IERG1,K1)
         DIMENSION ITAB1(20)
         WRITE(99,1)
    1    FORMAT(' ',35X,'IM FORTRAN-UNTERPROGRAMM')
         IERG1=0
         DO 10 I=1,20
         READ(1,2,END=100)ITAB1(I)                                    (4)
    2    FORMAT(I8)
   10    IERG1=IERG1+ITAB1(I)
  100    WRITE(99,3)
    3    FORMAT(' ',35X,'FESTPUNKTZAHLEN IN TABELLE EINGELESEN')
         K1=I-1
         WRITE(99,7)K1
    7    FORMAT(' ',35X,'ANZAHL DER SAETZE VOM LESEVORGANG:',I8)
         WRITE(99,4)
    4    FORMAT(' ',35X,'BERECHNUNG IM FORTRAN-UNTERPROGRAMM')
         WRITE(99,5)IERG1
    5    FORMAT(' ',35X,'ERGEBNIS IM FORTRAN-UNTERPROGRAMM:',I8)
         WRITE(99,6)
    6    FORMAT(' ',35X,'RUECKSPRUNG INS ASSEMBLER-PROGRAMM'/' ')
         RETURN
         END
```

(1) Aufruf des Parameteradreßlisten-Makros IFEPL:

```
PARAL      IFEPL ((I4),ITAB,IND),(I4,IERG),(I4,K)
```

Der Makro IFEPL erzeugt die an das FOR1-Unterprogramm zu übergebede Parameteradreßliste. Der erste zu übergebende Parameter ist der Datenbereich ITAB, dem im FOR1-Unterprogramm das Feld ITAB1 entsprcht. Im ersten Operanden ((I4),ITAB,IND) des Makros IFEPL wird der Typ des Feldes (I4), die symbolische Adresse des Feldes (ITAB) und die symbolische Adresse des Deskriptors dieses Feldes (IND) angegeben.Die symbolische Adresse IND muß als symbolische Adresse des Feld-Deskriptor-Makros IFEADS genannt werden.

(2) Aufruf des Feld-Deskriptor-Makros IFEADS:

```
IND        IFEADS ITAB,ELEN=4,BOUNDS=(1:20)
```

Der Makro IFEADS erzeugt einen Deskriptor für ein Feld. Die Adresse dieses Deskriptors muß in der Parameteradreßliste an das aufgerufene FOR1-Unterprogramm übergeben werden. Als erster Operand erscheint der Name des Feldes ITAB. ELEN=4 bezeichnet die Länge 4 eines Feldelements in byte,
BOUNDS=(1:20) die untere und obere Indexgrenze der Dimension.

(3) Aufruf des Aufruf-Makros IFECL:

```
IFECL PROG=AFOR,PARLIST=PARAL                    (3)
```

Im Aufruf des Makros IFECL wird mit PROG=AFOR der Name des aufgerufenen FOR1-Programms AFOR genannt. Mit PARLIST=PARAL wird die symbolische Adresse PARAL der durch den Makro IFEPL erzeugten Parameteradreßliste angegeben. Für die übrigen nicht genannten Operanden von IFECL sind die Voreinstellungen wirksam:

INIT=YES	FOR1-Laufzeitsystem-Initialisierung
RESTORE=YES	Retten und Wiederherstellen aller Register und der Programmaske
MAINSAV=YES	Ein Sicherstellungsbereich wird durch den Makro IFECL generiert.
RETURNI=NO	FOR1-Unterprogramm enthält keine RETURN i-Anweisung
FTERM=NO	nicht letzter Aufruf eines FOR1-Unterprogramms
USREQM=NO, USRELM=NO	keine Speicherverwaltung durch benutzereigene Routinen

(4) Im FORTRAN-Programm werden 20 INTEGER*4-Zahlen eingelesen und die Summe IERG1 berechnet.

(5) Das im FOR1-Programm berechnete Ergebnis IERG1 wird an das Assembler-Programm übergeben und von dort aus ausgegeben. Im Assembler-Programm wird ebenfalls die Summe der Feldelemente gebildet und ausgegeben.

Im Beispiel werden die Zahlen von 1 bis 20 eingelesen. Die Ausgabe des Programms nach SYSLST zeigt linksbündig die Meldungen des Assembler-Programms und rechts die Meldungen des FOR1-Programms:

```
SPRUNG INS FORTRAN-PROGRAMM

                                  IM FORTRAN-UNTERPROGRAMM
                                  FESTPUNKTZAHLEN IN TABELLE EINGELESEN
                                  ANZAHL DER SAETZE VOM LESEVORGANG:   20
                                  BERECHNUNG IM FORTRAN-UNTERPROGRAMM
                                  ERGEBNIS IM FORTRAN-UNTERPROGRAMM:  210
                                  RUECKSPRUNG INS ASSEMBLER-PROGRAMM

IM ASSEMBLER-HAUPT-PROGRAMM ZURUECK
ERGEBNIS VOM FORTRAN-PROGRAMM UEBERGEBEN:                             210
ERGEBNIS IM ASSEMBLER-HAUPT-PROGRAMM:                                 210
```

A.9.2 FOR1-Programm ruft Assemblerprogramm

Dazu stehen zwei Makros zur Verfügung: IFAEN, IFART

Eingangsmakro IFAEN (bzw. IFAENO)

Dieser Makro generiert die Einsprungsroutine für das Assemblerprogramm.

Der Makro IFAENO hat dieselben Operanden wie der Makro IFAEN und zusätzlich den Operand SB=adresse. IFAENO berechnet die Adresse des letzten Parameters, setzt gegebenenfalls das Endebit und speichert die Adresse des Endebits im Feld mit der Adresse SB=adresse.

Aufruf:

```
[label] IFAEN  [LABEL =⎧ENTRY⎫]        [,FIRST = reg][,LAST = reg]
                      ⎩CSECT⎭

                [,MAINSAV = ⎧NO  ⎫]    [,PMASK = ⎧STANDARD        ⎫]
                            ⎨YES ⎬                ⎩YES(PMSAV=name)⎭
                            ⎩name⎭
```

```
[label] IFAENO [LABEL =⎧ENTRY⎫] , ..., [SB = adresse]
                       ⎩CSECT⎭
```

label — Symbolische Adresse des Makroaufrufs.
Dieser Name wird im FOR1-Programm für den Aufruf verwendet. Wird kein Name angegeben, so wird ein Standardname angegeben, in der Form IFdddd generiert, wobei dddd eine vierstellige Zahl ist.

LABEL

- =ENTRY — Der Eingangspunkt ist ein ENTRY-Name.
- =CSECT — Der Eingangspunkt ist ein CSECT-Name.

FIRST=reg — Angabe des ersten zu rettenden Registers (Standard = 14).

LAST =reg — Angabe des letzten zu rettenden Registers (Standard = 12).

MAINSAV

- =NO — Das aufgerufene Assemblerprogramm benötigt keinen Sicherstellungsbereich (Save Area), da von ihm aus keine weiteren Unterprogrammaufrufe vorgesehen sind.
- =YES — Sicherstellungsbereich für das Assemblerprogramm ist bereits durch den Makro IFECL generiert worden.
- =name — Symbolische Adresse eines Sicherstellungsbereichs, der z.B. durch den Makro IFESAV generiert wurde.

PMASK

=YES Falls AMODE=31 gilt, kann die Programmaske nicht im Register 14 gesichert werden. Bei der Angabe von PMASK=YES (PMSAV= name) wird die Programmaske in einem Feld der Länge 1 byte mit dem symbolischen Namen name abgelegt.

=STANDARD Es erfolgt keine Sicherung der Programmaske.

SB=adresse Adresse eines 4 byte langen Feldes, in dem die Adresse des Endebits der Parameteradreßliste gespeichert wird (höchstwertiges Bit im n-ten Parameteradreßliste; n: Anzahl der Parameter).

Hinweise:

- Bei Angabe von MAINSAV=NO (Voreinstellung) wird bei Auftreten eines Laufzeitfehlers im Assemblerprogramm die Aufrufhierarchie nur bis zur rufenden FOR1-Programmeinheit ausgegeben. Die Namen der Assemblerprogramme in der Aufrufhierarchie werden ebenfalls ausgegeben, wenn in ihnen im IFAEN-Makroaufruf MAINSAV=YES bzw. MAINSAV=name angegeben wurde.
- Falls das aufgerufene Assembler-Programm die Programmaske ändert, so werden dadurch u.U. die ILCS-Konventionen verletzt. Deshalb sollte in einem solchen Fall die Programmaske des aufrufenden Programms durch die Angabe von PMASK= YES(PMSAV=name) gesichert werden, und beim Rücksprung durch den PMASK-Parameter des Makros IFART (bzw. IFARTO) zurückgesetzt werden.

Rücksprungmakro IFART (bzw. IFARTO)

Dieser Makro generiert die Rücksprungsroutine aus dem Assemblerprogramm zurück zum FOR1-Programm.

Der Makro IFARTO hat dieselben Operanden wie der Makro IFART und zusätzlich den Operand SB=adresse. IFARTO entfernt gegebenenfalls das Endebit, dessen Adresse im Feld mit der Adresse SB=adresse gespeichert ist.

Aufruf:

```
[label] IFART  [FIRST = reg] [,LAST = reg]  [,MAINSAV={NO  }]
                                            {YES }
                                            {name}

               [,PMASK = {STANDARD          }] [,RETURN = i]
                         {YES(PMSAV=name)   }
                         {NO                }

               [,STXIT = {YES}]                [,CHARFUN = {YES }]
                         {   }                             {NO  }
                         {NO }                             {name}
```

```
[label] IFARTO [FIRST = reg] ,..., [SB = adresse]
```

label — Symbolische Adresse des Makroaufrufs.

FIRST=reg — Erstes wiederherzustellendes Register (Standard = 14).

LAST=reg — Letztes wiederherzustellendes Register (Standard = 12).

MAINSAV

=NO — Es war kein Sicherstellungsbereich (Save Area) notwendig.

=YES — Ein Sicherstellungsbereich wurde durch den Makro IFECL generiert.

=name — Symbolische Adresse eines Sicherstellungsbereichs. Dieser Operand muß mit dem gleichnamigen Operanden im Makro IFAEN übereinstimmen.

PMASK

=STANDARD — Falls AMODE=24 gilt, wird die Programmaske des rufenden Programms zurückgesetzt. Falls AMODE=31 gilt, wird die Programmaske des rufenden Programms nicht zurückgesetzt.

=YES — Die durch die Angabe PMASK=YES(PMSAV=name) des Makros IFAEN gesicherte Programmaske des aufrufenden Programms wird zurückgesetzt. *name* im PMASK-Parameter von IFART muß das gleiche Feld wie im PMASK-Parameter des Makros IFAEN bezeichnen.

=NO — Die Programmaske wird nicht zurückgesetzt. Es erfolgt keine Sicherung der Programmaske.

RETURN=i	Rückkehrcode. Es wird zur *i*-ten Anweisungsmarke in der Parameterliste des rufenden FOR1-Programms zurückgekehrt (Standard = 0).
STXIT	
=YES	Die FOR1-STXIT-Routinen werden reaktiviert. Angaben nur zulässig, wenn MAINSAV=YES oder MAINSAV=NO angegeben wird.
=NO	Keine Reaktivierung der FOR1-STXIT-Routine.
CHARFUN	
=YES	R1 enthält die Adresse des String Descriptors (Simulation einer CHARACTER-FUNCTION).
=NO	R1 enthält den Return Code.
=name	Symbolische Adresse eines String Descriptors, wie z.B. durch den Makro IFESDS generiert.
SB=adresse	Adresse eines 4 byte langen Feldes in dem die Adresse des Endebits der Parameteradreßliste gespeichert ist (höchstwertiges Bit im *n*-ten Parameteradreßliste; n: Anzahl der Parameter).

Hinweis:

Bei FUNCTION-Unterprogrammen vom Typ INTEGER wird der Funktionswert in Register 0 zurückgeliefert. In diesem Fall muß für die wiederherzustellenden Register FIRST=1 und LAST=12 beim Aufruf des Makro IFART angegeben werden. Verwendet man die Standardwerte, dann wird der Funktionswert in Register 0 überschrieben. Außerdem müssen Register 14 und 15 im Assemblerprogramm zusätzlich gesichert werden.

Beispiel: FOR1-Programm ruft Assemblerprogramm auf

Das FOR1-Programm FORASS ruft das Assemblerprogramm FAAS auf.

FOR1-Programm FORASS:

```
            PROGRAM FORASS
            INTEGER ITAB(20)
            WRITE(99,1)
         1  FORMAT('1'/' ','SPRUNG INS ASSEMBLER-PROGRAMM')
            WRITE(99,2)
         2  FORMAT(' ')
            CALL EINSPR(ITAB,IERG,K)                                    (1)
     C
            WRITE(99,3)
         3  FORMAT(' ','IM FORTRAN-PROGRAMM ZURUECK')
            WRITE(99,4) IERG
         4  FORMAT(' ','ERGEBNIS VOM ASSEMBLER-PROGRAMM UEBERGEBEN:',I8)
            WRITE(99,5) K
         5  FORMAT(' ',I4,' UEBERGEBENE WERTE WERDEN SUMMIERT')
            IERG1=0
            DO 10 I=1,K
        10  IERG1=IERG1+ITAB(I)
            WRITE(99,6) IERG1
         6  FORMAT('0'/' ','ERGEBNIS IM FORTRAN-PROGRAMM:',I8/'1')
            STOP
            END
```

Assemblerprogramm FAAS:

```
FAAS        START
            PRINT NOGEN
EINSPR      IFAEN LABEL=ENTRY                                           (2)
            LM    6,8,0(1)                                              (3)
            BALR  3,0
            USING *,3
            XR    5,5
            MVI   DRU+5,X'40'
LOESCH      MVC   DRU+6(127),DRU+5
            MVC   DRU+35(20),='IM ASSEMBLERPROGRAMM'
            MVI   DRU+5,X'40'
            WRLST DRU,FEHL
            MVI   DRU+4,X'02'
            XR    9,9
LESEN       RDATA  EINB,LEKA,84                                         (4)
VER         PACK  DOWO,EINB+4(8)
            CVB   4,DOWO
            ST    4,0(6)
            AR    5,4
            LA    6,4(6)
            A     9,=F'1'
            B     LESEN
LEKA        STC   15,RETCO
            CLI   RETCO,X'10'
            BE    LEKA1
            CLI   RETCO,X'0C'
```

```
         B     VER
LEKA1    MVC   DRU+6(127),DRU+5                                    (5)
         MVC   DRU+35(L'TEXT),TEXT
         WRLST DRU,FEHL
         MVI   DRU+5,X'40'
         EX    0,LOESCH
         MVC   DRU+35(L'TEXTA),TEXTA
         CVD   9,ERG
         MVC   DRU+76(8),MASKE
         ED    DRU+76(8),ERG+4
         WRLST DRU,FEHL
         MVI   DRU+5,X'40'
         EX    0,LOESCH
         MVC   DRU+35(L'TEXT1),TEXT1
         WRLST DRU,FEHL
         ST    5,0(7)
         ST    9,0(8)
         CVD   5,ERG
         MVI   DRU+5,X'40'
         EX    0,LOESCH
         MVC   DRU+66(8),MASKE
         ED    DRU+66(8),ERG+4
         MVC   DRU+35(L'TEXT2),TEXT2
         WRLST DRU,FEHL
         MVI   DRU+5,X'40'
         EX    0,LOESCH
         MVC   DRU+35(L'TEXT3),TEXT3
         WRLST DRU,FEHL
         MVI   DRU+5,X'40'
         EX    0,LOESCH
         IFART                                                    (6)
         TERM
FEHL     TERMD
EINB     DS    CL84
DOWO     DS    D
DRU      DS    0CL133
SATZL    DC    X'0085404001'
         DS    CL128
RETCO    DS    C
TEXT     DC    'FESTPUNKTZAHLEN IN TABELLE EINGELESEN'
TEXT1    DC    'BERECHNUNG IM ASSEMBLERPROGRAMM'
TEXT2    DC    'ERGEBNIS IM ASSEMBLER-PROGRAMM'
TEXT3    DC    'RUECKSPRUNG INS FORTRAN-PROGRAMM'
TEXTA    DC    'ANZAHL DER SAETZE VOM LESEVORGANG'
ERG      DS    D
MASKE    DC    X'4020202020202120'
         END
```

(1) Aufruf des Assembler-Programms mit dem ENTRY-Namen EINSPR

(2) Aufruf des Eingangs-Makros IFAEN:

```
EINSPR   IFAEN LABEL=ENTRY
```

EINSPR ist die symbolische Adresse des Makroaufrufs, der im FOR1-Programm in der CALL-Anweisung verwendet wurde. Mit LABEL=ENTRY wird festgelegt, daß der Eingangspunkt ein ENTRY-Name ist.

(3) Mit dem LOAD MULTIPLE-Befehl wird die Adresse des Parameters ITAB in Register 6, die Adresse des Parameters IERG in Register 7 und die Adresse des Parameters K in Register 8 geladen.

(4) Das Assembler-Programm liest Festpunktzahlen ein, legt sie im Bereich ITAB ab und berechnet anschließend die Summe aus den eingelesenen Zahlen.

(5) Die Anzahl der eingelesenen Sätze und das Ergebnis werden ausgegeben.

(6) Aufruf des Rücksprung-Makros IFART:

Es gelten die Voreinstellungen des Makros IFART:

FIRST=14 Register 14 ist das erste wiederherzustellende Register.

LAST=12 Register 12 ist das letzte wiederherzustellende Register

MAINSAV=NO Es war kein Sicherstellungsbereich notwendig.

PMASK=STANDARD
Die Programmaske des rufenden Programms wird zurückgesetzt, falls AMODE=24 gilt bzw. nicht zurückgesetzt, falls AMODE=31 gilt.

RETURN=0 Die Parameterliste enthält keine Anweisungsmarke.

STXIT=NO Keine Reaktivierung der FOR1-STXIT-Routine.

CHARFUN=NO Register 1 enthält Return Code.

(7) SYSDTA wird einer Datei zugewiesen, die die Zahlen von 1 bis 20 enthält. Die Ausgabe des Programms nach SYSLST zeigt linksbündig die Meldungen des FORTRAN-Programms und rechts die Meldungen des Assembler-Programms:

```
                                                                        (7)
SPRUNG INS ASSEMBLER-PROGRAMM
                              IM ASSEMBLERPROGRAMM
                              FESTPUNKTZAHLEN IN TABELLE EINGELESEN
                              ANZAHL DER SAETZE VOM LESEVORGANG    20
                              BERECHNUNG IM ASSEMBLERPROGRAMM
                              ERGEBNIS IM ASSEMBLER-PROGRAMM      210
                              RUECKSPRUNG INS FORTRAN-PROGRAMM

IM FORTRAN-PROGRAMM ZURUECK
ERGEBNIS VOM ASSEMBLER-PROGRAMM UEBERGEBEN:                         210
  20 UEBERGEBENE WERTE WERDEN SUMMIERT

ERGEBNIS IM FORTRAN-PROGRAMM:                                       210
```

FOR1-XS-Programm ruft Assembler-Unterprogramm

Ruft ein FOR1-XS-Programm ein Assembler-Unterprogramm auf, das die (variable) Parameteranzahl durch Lokalisieren des sogenannten Ende-Bits feststellt, dann muß der Anwender die XS-Parameteradreßliste anpassen. Während bei ALT- und NXS-Parameteradreßlisten mit n Parametern das höchstwertige Bit im n-ten Wort der Adreßliste gleich 1 gesetzt wird, wird bei XS-Parameteradreßlisten dieses Ende-Bit nicht mehr gesetzt.

Bei der Anpassung der XS-Parameteradreßliste kann man 3 Fälle unterscheiden:

1. Das Assembler-Unterprogramm soll nicht verändert werden.

   Lösung:

   Im FORTRAN-Quellprogramm ersetzt der Anwender den Aufruf des Assembler-Unterprogramms progname

   CALL progname (par_1,...,par_n)

   durch den Aufruf des Unterprogramms

   ```
   CALL OLDASS (progname,par1,...,parn)
   ```

progname	Name des Assembler-Unterprogramms, der im aufrufenden FORTRAN-Programm als EXTERNAL deklariert werden muß.
par_i	i-ter an das Assembler-Unterprogramm zu übergebender Parameter; $0 \le i \le n$, $0 \le n \le 408$

   Das Unterprogramm OLDASS kopiert die ihm übergebene XS-Parameteradreßliste, entfernt den EXTERNAL-Namen progname und setzt das höchstwertige Bit im n-ten Wort der Parameteradreßliste. Mit der Adresse dieser angepaßten Parameteradreßliste in Register 1 und der Anzahl der Parameter in Register 0 wird dann das Assembler-Unterprogramm angesprungen. Nach der Rückkehr aus dem Assembler-Unterprogramm wird das Ende-Bit wieder gelöscht und die Kontrolle an das aufrufende FORTRAN-Programm zurückgegeben.

2. Das FOR1-XS-Programm soll nicht verändert werden.

   Lösung:

   In das Assembler-Unterprogramm fügt der Anwender den Aufruf des Makros IFAENO mit dem Operanden SB=adresse ein, der die Adresse eines 4 byte langen Feldes angibt. IFAENO berechnet die Adresse des letzten Parameters, setzt gegebenenfalls das Endebit und speichert die Adresse des Endebits im Feld mit der Adresse SB=adresse. Vor der Rückkehr in das FOR1-XS-Programm wird der Makro IFARTO mit dem Operanden SB=adresse aufgerufen, der das Ende-Bit wieder entfernt. Die Makros IFAENO und IFARTO sind in der Makrobibliothek FOR1MACLIB enthalten.

3. Weder das FOR1-XS-Programm noch das Assembler-Unterprogramm sollen verändert werden.

   Lösung:

   Der Anwender benennt das nicht XS-fähige Assembler-Programm um und erstellt unter dem alten Namen des Assembler-Programms ein Glue-Programm. Dieses Assembler-Glue-Programm

   – setzt mit Hilfe des Makros IFAENO das Ende-Bit an die Adresse "SB=adresse";
   – ruft das nicht-XS-fähige Assembler-Programm auf;
   – entfernt vor der Rückkehr in das FOR1-XS-Programm mit Hilfe des Makros IFARTO das Ende-Bit.

Assembler-Programme, die sowohl von FOR1-XS- als auch von COBOL-Programmen aufrufbar sein sollen

COBOL teilt die Anzahl der Parameter stets durch Setzen eines Ende-Bits in der Parameteradreßliste mit. Ein von COBOL aufgerufenes Assembler-Unterprogramm stellt die Anzahl der Parameter durch Lokalisieren dieses Ende-Bits fest.

Ruft ein FOR1-XS-Programm ein Assembler-Programm auf, dann stellt das FOR1-Programm die Anzahl der Parameter in Register 0 zur Verfügung. Das Ende-Bit wird nicht gesetzt.

Ein Assembler-Programm, das die Anzahl der Parameter durch Lokalisieren des Ende-Bits feststellt, kann durch folgende Änderung sowohl von COBOL- als auch von FOR1-XS-Programmen aufgerufen werden:

Die Makroaufrufe von IFAEN und IFART werden in die Aufrufe der entsprechenden Makros IFAENO und IFARTO geändert, die im zusätzlichen Operanden SB=adresse die Adresse des Ende-Bits enthalten.

A.10 Software-Produkte für den FOR1-Anwender

A.10.1 Dienstprogramm FPOOLITY

Charakterisierung des Produktes

Mit Hilfe des Dienstprogramms FPOOLITY kann der FOR1-Anwender Schnittstellen zu eigenen Unterprogrammen einer Fehleranalyse unterziehen. Formale Fehler in Unterprogrammaufrufen können damit während der Übersetzung erkannt werden. Die Einrichtung privater FPOOL-Dateien mit Hilfe des Dienstprogramms FPOOLITY wird in Abschnitt 12.3 beschrieben.

Dokumentation:

"FPOOLITY" Beschreibung [22]

A.10.2 Unterprogrammbibliothek für Hochpräzisionsarithmetik ARITHMOS

Charakterisierung des Produktes

ARITHMOS ist eine Unterprogrammbibliothek zur Lösung des Rundungsfehlerproblems bei technisch-wissenschaftlichen Berechnungen mit Gleitkommazahlen.
ARITHMOS basiert auf der mathematischen Theorie der Rechnerarithmetik von Professor Dr. Kulisch, Universität Karlsruhe.

Die Funktionen von ARITHMOS können in FORTRAN-Programmen über die CALL-Schnittstelle und in PLI1-Programmen über die Aufrufschnittstelle zu fremdsprachigen (d.h. hier FORTRAN) Unterprogrammen aufgerufen werden.

ARITHMOS stellt Operationen zur Vektor- und Matrizenrechnung von maximaler Genauigkeit zur Verfügung. Maximale Genauigkeit bedeutet hier, daß zwischen dem exakten Ergebnis und dem von ARITHMOS gelieferten Ergebnis keine weitere in FORTRAN REAL*4 oder REAL*8 darstellbare Gleitkommazahl liegt. Dazu wird das Skalarprodukt zweier Vektoren von beliebiger Länge exakt berechnet.

Weitere Funktionen dienen zur Lösung von Standardproblemen der Linearen Algebra (z.B. Lineare Gleichungssysteme, Matrizeninvertierung, Eigenwerte) mit höchster Genauigkeit, wobei mathematisch garantierte Fehlerschranken geliefert werden. Viele der Funktionen in ARITHMOS erlauben die Eingabe von Intervalldaten. Damit kann z.B. der Einfluß unsicherer Eingabedaten auf das Rechenergebnis bestimmt werden.

Für den Siemens PC-2000 in Verbindung mit dem Betriebssystem BS2000-PC gibt es die Varianten ARITHMOS-PC und ARITHMOS-DL mit demselben Funktionsumfang wie ARITHMOS.

Dokumentation:

"ARITHMOS" Kurzbeschreibung [6]
"ARITHMOS" Benutzerhandbuch [5]
"ARITHMOS" Tabellenheft [7]

A.10.3 Methodenbank-Bibliothek normierter Unterprogramme für Wirtschaft und Wissenschaft MEB

Charakterisierung des Produktes

MEB enthält Programmbausteine zur Lösung betriebswirtschaftlicher, organisatorischer, statistischer und technisch-wissenschaftlicher Probleme.
Die gesamte Methodenbank-Bibliothek MEB umfaßt 420 Methoden, die aufgrund der ihnen zugrunde liegenden Theorien in 13 Klassen gegliedert sind:

Klasse 0 Ein-/Ausgabeprogramm und Hilfsroutinen.
Klasse 1 Matrizenrechnung.
Klasse 2 Differentialrechnung.
Klasse 3 Integralrechnung.
Klasse 4 Gleichungen und Polynome.
Klasse 5 Approximation und Interpolation.
Klasse 6 Statistik.
Klasse 7 Optimierung.
Klasse 8 Simulation.
Klasse 9 Berichts- und Planungswesen.
Klasse A Spezielle Funktionen.
Klasse B Zeitreihenanalyse und Prognose.
Klasse C Finanz- und Versicherungswesen.

Im Hinblick auf Einsatz- und Nutzungsschwerpunkte werden aus diesen Klassen sechs Methodenpakete gebildet, die beliebig miteinander kombiniert werden können:

MEB-MATH Angewandte Mathematik; Klassen 2, 3, 4, 5, A
MEB-OPT Optimierung; Klasse 7
MEB-PLAN Berichts- und Planungswesen; Klasse 9
MEB-PROG Prognose; Klassen B, 6 (Teil E)
MEB-STAT Statistik; Klassen 6, 8
MEB-FINANZ Finanz- und Versicherungswesen; Klasse C

Zusätzlich wurde ein Basispaket gebildet, das von den übrigen Methodenpaketen benötigt wird:

MEB-BASIS Basisroutinen; Klassen 0, 1

Die einzelnen Methodenpakete sind praxisgerecht zusammengestellt und tragen den Namen eines bestimmten Themenkreises, ohne darauf beschränkt zu sein.

Einsatzmöglichkeiten

Die Methodenbank-Bibliothek MEB ist eine offene Bibliothek, d.h. für spezielle Anwendungen kann der Anwender den Methodenvorrat, den ihm die MEB standardmäßig bietet, jederzeit um eigene Programm-Bausteine erweitern.

Neben den Methoden enthält die Methodenbank-Bibliothek MEB einen ausführlichen Informationsteil. Er ist streng hierarchisch gegliedert und bietet in drei Stufen
- eine Übersicht über die gesamte Bibliothek bzw. Übersichten über die MEB-Pakete
- Inhaltsverzeichnisse der MEB-Klassen
- einheitlich aufgebaute Beschreibungen der MEB-Methoden.

Damit kann der Anwender allein mit seinem problemspezifischen Wissen die zur dv-technischen Lösung erforderlichen Programmnamen ermitteln.

Die Beschreibungen der Methoden, die im MEB-Informationsteil enthalten sind, sind stets nach dem gleichen Schema aufgebaut, und umfassen - in dieser Reihenfolge -
- eine Beschreibung der zugrunde liegenden Theorie
- eine dv-technische Beschreibung
- ein Beispiel.

Für die Nutzung der MEB-Programme gibt es zwei Möglichkeiten:

- das Softwareprodukt Methodenmonitor MEMO als eine Abfragesprache für die MEB-Pakete oder
- benutzereigene Hauptprogramme in den Programmiersprachen FORTRAN, COBOL, ALGOL, PASCAL, PLI1 und Assembler, in die MEB-Methoden eingebunden werden.

Um allgemeine Anwendbarkeit der Methoden zu gewährleisten, sind alle Felder variabel dimensioniert, wobei die Feldgrenzen, die den FORTRAN-Konventionen genügen, in der Parameterliste übergeben werden.

Dokumentation (deutsch):

"MEB-BASIS" [28]
"MEB-MATH" [30]
"MEB-STAT" [34]
"MEB-PLAN" [32]
"MEB-OPT" [31]
"MEB-PROG" [33]
"MEB-FINANZ [29]
"MEB-Anwendungsbeschreibung und Bedienungsanleitung" (BS2000) [27]

A.10.4 Bibliotheksprogramm LMS

Charakterisierung des Produktes

LMS ist ein einheitliches Bibliothekssystem für Bibliotheken im BS2000.

Programme und Programmteile (Source, Makro, Copy, Include), Bindemoduln, Bindelademoduln und Prozeduren (JCL) und die dazugehörende Dokumentation können mit LMS in PLAM-Bibliotheken gespeichert, bearbeitet und verwaltet werden.

Auf PLAM-Bibliotheken können die Compiler, die Dienstprogramme (Binder/ Lader) und das Management der Systemdateien zugreifen. Prozeduren können direkt aus der PLAM-Bibliothek gestartet werden.

Mit LMS steht dem Anwender ein in die Systemumgebung des BS2000 eingebettetes Bibliothekssystem zur Verfügung. Bei der Ablösung der veralteten Dienstprogramme, die eine Untermenge von LMS darstellen (MLU/LMR), ist keine Umstellung der Bibliotheken notwendig. COBLUR- und FMS-Bibliotheken können von LMS direkt gelesen werden.

LMS unterstützt Programmbibliotheken, d.h. PAM-Dateien, die mit der Bibliotheks-Zugriffsmethode PLAM (Program Library Access Method) bearbeitet werden.

Für den Siemens PC-2000 in Verbindung mit dem BS2000-PC gibt es die Varianten LMS-PC/-DL. In diesen Varianten werden PLAM-Bibliotheken noch nicht unterstützt.

Beschreibung der Funktionen

Mit LMS können alle Formen von Programmelementen gespeichert, verwaltet und verändert werden.

LMS verwaltet, speichert und verändert in Bibliotheken Source-, Makro-, Bindemodul-, Bindelademodul-, Prozedur- und Textelemente.

Es können Source/Makro-, Modul- und die neuen Programmbibliotheken erstellt und bearbeitet werden.

Daneben können auch die bereits vorhandenen MLU-, LMR-, COBLUR- und FMS-Bibliotheken von LMS verarbeitet werden. Die Umstellung der bisherigen Verfahren auf LMS wird somit wesentlich erleichtert und vereinfacht.

LMS und die Zugriffsmethode PLAM ermöglichen im einzelnen:

- das Einrichten und Kopieren von LMS-Bibliotheken und das Erstellen von Bibliotheksbändern (Archiv- und Transportfunktion)
- das Lesen und Verarbeiten von COBLUR- und FMS-Bibliotheken und deren Elementen
- das gemeinsame Speichern in einer Bibliothek

  - von Programmelementen aller Arten, d.h. vor allem von Quellprogrammen, Binde-, Lade- und Bindelademoduln, aber auch von Listen und Prozeduren
  - von Elementen mit gleichen Namen, die sich durch Typ- oder Versionsbezeichnungen unterscheiden

- das Verwalten und Warten aller Arten von Bibliothekselementen, d.h. unter anderem
  - die Aufnahme, die Korrektur und das Löschen von Bibliothekselementen
  - die Umbenennung, Übertragung, Numerierung und Kennzeichnung von Elementen;
  - die Erstellung und Führung von Versionsnummern und des jeweiligen Archivierungsdatums für Elemente;
  - den voll- oder teilqualifizierten Zugriff auf Bibliothekselemente, auch unter Einbeziehung der Versionsnummern und des Archivierungsdatums mit und ohne die Verwendung von ein-/ausschließenden Zeichenfolgen;
  - einen RUN-/TEST-Modus zur ereignisfreien Kontrolle von Funktionsfolgen oder ein automatisches Umschalten nach Kommandofehlern zur Vermeidung von Folgefehlern;
  - den Aufruf des Editors EDT innerhalb des LMS-Systems zum direkten Bearbeiten von Elementen am Terminal;

- den Zugriff
  - von Compilern, Binder/Lader und des Managements der Systemdateien;
  - mehrerer simultaner Anwender auf eine gemeinsame Bibliothek; (auch schreibend)

- eine platzsparende Speichertechnik durch Komprimierung der Elemente.

Programmbeschreibung

Der Einsatz des BS2000 Bibliothekssystems LMS ermöglicht eine wesentliche Entlastung des BS2000 Katalogs. Durch die komprimierte Speicherung aller Elemente erfolgt gleichzeitig eine umfangreiche Reduzierung des für diese Elemente ursprünglich benötigten Speicherplatzbedarfs.

Durch die Zugriffe der Compiler, Dienstprogramme und des Managements der Systemdateien werden die Systemleistungen verbessert. Die Verarbeitungstechnik von LMS beinhaltet eine Erhöhung der Sicherheit, da durch interne Mechanismen die Bibliotheken immer in einem konsistenten Zustand gehalten werden.

Dokumentation

LMS Beschreibung [25]

A.10.5 Jobvariablen

Charakterisierung des Produktes

Jobvariablen sind Datenobjekte zum Austausch von Informationen zwischen Anwendern einerseits und Betriebssystem und Benutzern andererseits.

Der Anwender kann Jobvariablen einrichten und verändern. Er kann das Betriebssystem anweisen, beim Eintreten gewisser Ereignisse bestimmte Jobvariablen auf vereinbarte Werte zu setzen.

Jobvariablen sind ein flexibles Werkzeug zur Auftragssteuerung unter Benutzerkontrolle. Sie bieten die Möglichkeit, Abhängigkeiten von komplexen Produktionsabläufen einfach zu definieren und bilden die Basis für eine ereignisgesteuerte Auftragsverarbeitung.

Beschreibung der Funktionen

Jobvariablen sind vom Betriebssystem verwaltete Objekte, die über Namen adressiert werden und in die Daten bis zu einer Länge von 256 byte abgespeichert werden können. Sie dienen zum Austausch von Informationen zwischen Anwendern einerseits sowie Betriebssystem und Anwendern andererseits. Auf sie kann über die Kommando- und Makroschnittstelle zugeriffen werden.
In Bedingungsanweisungen kann man Jobvariablen über boolesche Operationen verknüpfen und somit die Ausführung einzelner Aktionen vom Wahrheitswert der Bedingung abhängig machen. Benutzer-Jobvariablen und überwachende Jobvariablen (s.u.) bieten zudem die Möglichkeit der synchronen und asynchronen Ereignissteuerung auf Kommando- und Programmebene.

Für die verschiedenen Aufgabengebiete gibt es unterschiedliche Jobvariablen:

Benutzer-Jobvariablen

Die allgemeinste Form, in der Jobvariablen angeboten werden, ist die Form der Benutzer-Jobvariablen. Ihr Name, ihre Lebensdauer und die abzuspeichernden Daten werden ausschließlich vom Anwender bestimmt. Sie kann mit Schutzattributen wie Paßwörtern, Schreibschutz und Verfallsdatum versehen werden. Der Zugriff auf sie kann auf eine Benutzerkennung beschränkt oder generell gestattet sein.

Benutzer-Jobvariablen sind besonders geeignet zum Austausch von Informationen. Sie können aber auch zur Auftragssteuerung verwendet werden.

Überwachende Jobvariablen

Die überwachende Jobvariable ist eine Spezialform der Benutzer-Jobvariablen. Sie wird einem Auftrag oder Programm zugeordnet. Name, Lebensdauer und Schutzattribute bestimmt der Anwender. Im Gegensatz zur Benutzer-Jobvariablen wird sie aber vom Betriebssystem mit fest vorgegebenen Werten versorgt, die den Status des zugeordneten Auftrags oder Programms widerspiegeln.

Überwachende Jobvariablen sind besonders geeignet zur Auftragssteuerung, wie sie u.a. bei Abhängigkeiten in Produktionsabläufen notwendig ist.

Dokumentation

Jobvariablen Benutzerhandbuch [24]

A.10.6 Grafisches Kernsystem GKS-GA

Charakterisierung des Produktes

GKS-GA (BS2000) ist die Implementierung des genormten grafischen Kernsystems GKS (ISO 7942, DIN 66252) im BS2000. GKS bietet die Grundfunktion für die Erzeugung und Behandlung computergenerierter, zweidimensionaler Grafik. Es erlaubt die Speicherung und dynamische Änderung von grafischen Objekten und deren bildliche Darstellung auf geeigneten Ausgabegeräten.

Die Funktionen des GKS-GA sind unabhängig vom grafischen Gerätetyp, vom Anwendungsgebiet und von der Programmiersprache. Dadurch sind auch die Anwenderprogramme, die GKS verwenden, unabhängig vom jeweiligen Gerätetyp.

GKS-GA bietet über mehrere Programmiersprachen eine genormte Schnittstelle zum Anwendungsprogramm sowie eine einheitliche interne Schnittstelle zu den geräteabhängigen Treibern.

Beschreibung der Funktionen

GKS-GA legt über eine genormte Anwenderprogrammschnittstelle einen Satz von Funktionen fest, die die Erstellung beliebiger zweidimensionaler Grafiken durch ein Anwenderprogramm ermöglichen.

Alle GKS-Funktionen sind anwendungsunabhängig. Sie sind in die folgenden Funktionsbereiche unterteilt:

- Darstellungselemente (Polygon, Polymarke, Füllgebiet, Zellmatrix, Verallgemeinertes Darstellungselement, Text).
- Darstellungsattribute (Farbe, Strichstärke, Linienart, Textausrichtung u.a.).
- Grafischer Arbeitsplatz (Verallgemeinerung realer grafischer Geräte für die geräteunabhängige Programmierung).
- Transformation (Koordinierung von Anwender- (oder Welt-) koordinatensystem / normalisiertem / Gerätekoordinatensystem, Vergrößerung, Verkleinerung).
- Bildsegmentierung (Definition und Manipulation von Teilbildern).
- Bilddatei (o.a. Metafile: Langfristspeichern von Bildern, Übertragen, Wiedereinlesen).
- Fehlerbehandlung (Steuerung durch GKS oder Anwenderprogramm).

GKS ist für Anwendungen unter dem Transaktionsmonitor UTM, für VTX-Anwendungen und für Anwendungen im Teilnehmerbetrieb (TIAM) verfügbar.

Programmiersprachen für den Anwender: COBOL, FORTRAN, Assembler.

Dokumentation

Grafisches Kernsystem [23]

Literatur

[1] **AID** (BS2000)
Advanced Interactive Debugger
Basishandbuch
Benutzerhandbuch

Zielgruppe
Programmierer im BS2000
Inhalt
Überblick über AID, Beschreibung der Sachverhalte und Operanden, die für alle Programmiersprachen gleich sind.
Meldungen, Gegenüberstellung von AID-IDA
Einsatz
Testen von Programmen im Dialog- und Stapelbetrieb

[2] **AID** (BS2000)
Advanced Interactive Debugger
Testen auf Maschinencode-Ebene
Benutzerhandbuch

Zielgruppe
Programmierer im BS2000
Inhalt
Beschreibung der AID-Kommandos für das Testen auf Maschinencode-Ebene; Anwendungsbeispiel
Einsatz
Testen von Programmen im Dialog- und Stapelbetrieb

[3] **AID** (BS2000)
Advanced Interactive Debugger
Testen von FORTRAN-Programmen
Benutzerhandbuch

Zielgruppe
FORTRAN-Programmierer
Inhalt
Beschreibung der AID-Kommandos für das symbolische Testen von FORTRAN-Programmen;
Anwendungsbeispiel
Einsatz
Testen von FORTRAN-Programmen im Dialog- und Stapelbetrieb

[4] **ARCHIVE** (BS2000)
Benutzerhandbuch

Zielgruppe
BS2000-Anwender, -Systemverwalter, -Operator
Inhalt
Beschreibung der ARCHIVE-Funktionen und der Anweisungen zum Sichern und Rekonstruieren von Dateien mit ARCHIVE.

[5] **ARITHMOS**
Benutzerhandbuch

Zielgruppe
FORTRAN- und Assembler-Programmierer
Inhalt
Probleme der herkömmlichen Gleitpunktarithmetik und ARITHMOS-Konzept, Problemlösungsroutinen, Basisoperationen, Elementaranweisungen, Fehlerbehandlung und Grundbefehle von ARITHMOS.

[6] **ARITHMOS**
Kurzbeschreibung

Zielgruppe
FORTRAN- und Assembler-Programmierer
Inhalt
Probleme der herkömmlichen Gleitpunktarithmetik und ARITHMOS-Konzept, Übersicht der ARITHMOS-Funktionen, praktische Anwendungen, dv-technische Voraussetzungen.

[7] **ARITHMOS**
Tabellenheft

Zielgruppe
FORTRAN- und Assembler-Programmierer
Inhalt
Kurzübersicht der ARITHMOS-Funktionen, Auflistung der ARITHMOS-Funktionsaufrufe, Grundbefehle von ARITHMOS.

[8] **Assemblerbefehle** (BS2000)
Beschreibung

Zielgruppe
BS2000-Assembler-Programmierer
Inhalt
Beschrieben sind alle Assemblerbefehle (nicht privilegiert) der vom BS2000 unterstützten Zentraleinheiten in alphabetischer Reihenfolge
Bei jedem Befehl sind dargestellt:
- seine Funktion
- sein Assemblerformat, d.h. seine Schreibweise in Assemblersprache
- sein Maschinenformat, d.h. seine Darstellung in der Zentraleinheit
- sein Ablauf im Detail
- etwaige von ihm gesetzte Werte der Anzeige
- die bei seinem Ablauf möglichen Programmunterbrechungen
- Programmierhinweise
- ein oder mehrere Beispiele

Einsatz
BS2000-Assembler-Anwendungsprogrammierer

[9] **ASSEMBH**
Benutzerhandbuch

Zielgruppe
Assembler-Anwender im BS2000
Inhalt
Aufruf und Steuerung des ASSEMBH;
Übersetzen, Binden, Laden und Starten;
Eingabequellen und Ausgaben des ASSEMBH;
Laufzeitsystem, Listenausgabe der strukturierten Programmierung;
Sprachverknüpfungen;
Assembler-Diagnoseprogramm ASSDIAG;
Dialogtesthilfe AID;
Meldungen des ASSEMBH;
Format der Assemblerbefehle

[10] **ASSEMBH**
Beschreibung

Zielgruppe
Assembler-Anwender im BS2000
Inhalt
Beschreibung des Sprachumfangs des Assemblers ASSEMBH; Struktur der Assemblersprache, Assembleranweisungen; Struktur und Elemente sowie Instruktionen der Makrosprache; Strukturierte Programmierung mit ASSEMBH, vordefinierte Makros für die strukturierte Programmierung

[11] BS2000
Benutzerkommandos (ISP-Format)
Benutzerhandbuch

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

[12] 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

[13] BS2000
Binder-Lader-Starter (BLS)
Benutzerhandbuch

Zielgruppe
Software-Entwickler
Inhalt
Das System Binder-Lader-Starter (BLS) besteht aus den Funktionseinheiten
- Binder BINDER
- dynamischer Bindelader DBL
- statischer Lader ELDE

Die einzelnen Abschnitte enthalten jeweils eine Beschreibung aus funktioneller Sicht mit Beispielen, sowie einen Nachschlageteil mit den Anweisungen, Kommandos und ggf. Makroaufrufen.

[14] **C** (BS2000)
C-Compiler
Benutzerhandbuch

Zielgruppe
C-Anwender im BS2000
Inhalt
Beschreibung aller Tätigkeiten zum Erzeugen eines ablauffähigen C-Programms: Übersetzen, Binden, Laden, Testen;
Programmierhinweise und weitergehende Informationen zu: Programmablaufsteuerung, Dateiverarbeitung, Sprachverknüpfung, Sprachumfang des C-Compilers, Lokalität, Ereignisbehandlung, Meldungen

[15] **COBOL85** (BS2000)
COBOL-Compiler
Benutzerhandbuch

Zielgruppe
COBOL-Anwender im BS2000
Inhalt
Bedienung des COBOL85-Compilers und der Software für das Binden;
Laden und Testen von COBOL-Programmen;
Dateiverarbeitung mit COBOL-Programmen;
Programmverknüpfung;
Aufbau des COBOL85-Systems;
Meldungen des Compilers und des Laufzeitsystems;

[16] BS2000
Einführung in die Dialogschnittstelle (SDF)
Benutzerhandbuch

Zielgruppe
BS2000-Anwender
Inhalt
Die verschiedenen Eingabemöglichkeiten im Systembetrieb mit SDF. Bedienungshinweise und Beispiele mit der optionalen Benutzerführung über Menüs.
Einsatz
Allgemein

[17] BS2000
Dienstprogramme
Benutzerhandbuch

Zielgruppe
BS2000-Anwender (nicht privilegiert)
Inhalt
Dienstprogramme für den nichtprivilegierten Benutzer des BS2000
Einsatz
BS2000-Teilnehmerbetrieb

[18] BS2000
DVS Einführung und Kommandoschnittstelle
Benutzerhandbuch

Zielgruppe
nicht privilegierte BS2000-Anwender
Inhalt
- Funktionen des Datenverwaltungssystems im BS2000
- Verarbeitung von Platten- und Banddateien
- Zugriffsmethoden UPAM, SAM, BTAM, EAM, ISAM
- DVS-Kommandos

[19] BS2000
DVS Assembler-Schnittstelle
Benutzerhandbuch

Zielgruppe
nicht privilegierte BS2000-Anwender/Assembler-Programmierer
Inhalt
- Funktionen des Datenverwaltungssystems im BS2000 (auf Makro-Ebene)
- Verarbeitung von Platten- und Banddateien (auf Makro-Ebene)
- Zugriffsmethoden UPAM, SAM, BTAM, EAM, ISAM (jew. inkl. Aktionsmakroaufrufe)
- Makroaufrufe der Dateiverarbeitung

[20] **EDT** (BS2000)
Anweisungen
Benutzerhandbuch

Zielgruppe
EDT-Einsteiger und EDT-Anwender
Inhalt
Bearbeiten von SAM- und ISAM-Dateien und Elementen aus Programm-Bibliotheken.
Einstieg in die Arbeitsweise des EDT und Beschreibung der Arbeitsmodi.
Erstellen von EDT-Prozeduren. Beschreibung aller EDT-Anweisungen. Häufige Anwendungsfälle werden anhand zahlreicher Beispiele erklärt.
Einsatz
Aufbereiten von Dateien

[21] **FOR1** (BS2000)
FORTRAN-Compiler
Beschreibung

Zielgruppe
FORTRAN-Anwender im BS2000
Inhalt
Beschreibung des Sprachumfangs des FOR1-Compilers: Grundelemente von FORTRAN, Steueranweisungen, Ein-/Ausgabeanweisungen, Spezifikationsanweisungen und Anfangswertfestlegungen, Zuweisungsanweisungen und Formate; Struktur und Aufbau eines FORTRAN-Programms;

[22] **FPOOLITY** (BS2000)
Beschreibung

Zielgruppe
Programmierer im BS2000
Inhalt
Das FPOOL-Konzept ermöglicht einem Compiler die Verträglichkeit von Schnittstellen zu gerufenen Funktionen zu prüfen.
Beschreibung des FPOOL-Konzepts, Bedienen von FPOOLITY zum Erzeugen von Schnittstellenbeschreibungen und Handhaben des FPOOL's.
Diese Beschreibung ist nur zusammen mit dem entsprechenden Compiler-Benutzerhandbuch zu verwenden.

[23] **Grafisches Kernsystem** (BS2000)
Benutzerhandbuch

Zielgruppe
FORTRAN-, COBOL-, Assemblerprogrammierer
Inhalt
Beschreibung der Anwendung des Grafischen Kernsystems GKS

[24] BS2000
Jobvariablen
Benutzerhandbuch

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

[25] **LMS** (BS2000)
Beschreibung

Zielgruppe
BS2000-Anwender
Inhalt
- Beschreibung der LMS-Anweisungen im ISP-Format zum Erstellen und Verwalten von PLAM-Bibliotheken
- Speicherung mit Delta-Technik
- Aufruf als Unterprogramm aus COBOL- und Assembler-Programmen

Einsatz
Dialog- und Stapelbetrieb

[26] BS2000
Makroaufrufe an den Ablaufteil
Benutzerhandbuch

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

[27] **MEB**
Anwendungsbeschreibung und
Bedienungsanleitung

Zielgruppe
FORTRAN-, COBOL-, ALGOL-, PASCAL-, PLI1- und Assembler-Programmierer.
Inhalt
Anwendungsbeschreibung der Methodenbank-Bibliothek normierter Unterprogramme für Wirtschaft und Wissenschaft MEB.

[28] **MEB-BASIS**
Basisroutinen
Programmbeschreibung

Zielgruppe
FORTRAN-, COBOL-, ALGOL-, PASCAL-, PLI1- und Assembler-Programmierer.
Inhalt
Beschreibung von Programmbausteinen, die von den übrigen Methodenpaketen benötigt werden (Ein-/Ausgabeprogramm und Hilfsroutinen, Matrizenrechnung).

[29] **MEB-FINANZ**
Finanz- und Versicherungsmathematik
Programmbeschreibung

Zielgruppe
FORTRAN-, COBOL-, ALGOL-, PASCAL-, PLI1- und Assembler-Programmierer.
Inhalt
Beschreibung von Programmbausteinen für das Finanz- und Versicherungswesen.

[30] **MEB-MATH**
Angewandte Mathematik
Programmbeschreibung

Zielgruppe
FORTRAN-, COBOL-, ALGOL-, PASCAL-, PLI1- und Assembler-Programmierer.
Inhalt
Beschreibung von Programmbausteinen der angewandten Mathematik (Differential-, Integralrechnung, Gleichungen und Polynome, Approximation und Interpolation, spezielle Funktionen).

[31] **MEB-OPT**
Optimierung
Programmbeschreibung

Zielgruppe
FORTRAN-, COBOL-, ALGOL-, PASCAL-, PLI1- und Assembler-Programmierer.
Inhalt
Beschreibung von Programmbausteinen für Optimierung.

[32] **MEB-PLAN**
Planungs- und Berichtwesen
Programmbeschreibung

Zielgruppe
FORTRAN-, COBOL-, ALGOL-, PASCAL-, PLI1- und Assembler-Programmierer.
Inhalt
Beschreibung von Programmbausteinen für das Berichts- und Planungswesen.

[33] **MEB-PROG**
Zeitreihenanalyse und Prognose
Programmbeschreibung

Zielgruppe
FORTRAN-, COBOL-, ALGOL-, PASCAL-, PLI1- und Assembler-Programmierer.
Inhalt
Beschreibung von Programmbausteinen für Prognosen (Zeitreihenanalyse und Prognose, Statistik).

[34] **MEB-STAT**
Statistik
Beschreibung

Zielgruppe
FORTRAN-, COBOL-, ALGOL-, PASCAL-, PLI1- und Assembler-Programmierer
Inhalt
Beschreibung von Programmbausteinen der Statistik (Statistik und Optimierung)

[35] **Pascal-XT** (BS2000)
Benutzerhandbuch

Zielgruppe
Pascal-XT-Anwender im BS2000
Inhalt
Bedienung des Programmierersystems und des Compilers;
Beschreibung der BS2000-spezifischen Eigenschaften des Compilers;
Binden und Ausführen von Programmen;
Sprachanschlüsse;
Die Testhilfe PATH;
Laufzeitfehlermeldungen;
Beschreibung vordefinierter Pakete;
Vergleich mit Pascal Version 3;

[36] **PERCON** (BS2000)
Beschreibung

Zielgruppe
BS2000-Anwender
Inhalt
Beschreibung der PERCON-Anweisungen im ISP-Format zum Übertragen und Umsetzen von Dateien mit PERCON.
Einsatz
BS2000-Dialogbetrieb und -Stapelbetrieb

[37] **PLI1** (BS2000)
PL/I-Compiler
Benutzerhandbuch

Zielgruppe
PL/I-Anwender im BS2000
Inhalt
Aufruf und Steuerung des PLI1-Compilers, Eingeben und Übersetzen von Quellprogrammen, Erstellen und Verwalten von Binde- und Lademodulen, Erzeugen gemeinsam benutzbarer Programme, Steuern des Programmablaufs, Dateizugriffe, Testhilfen, Optimierungen, Interne Darstellung von Daten, Prozedur-Schnittstellen, Dienstleistungsprozeduren, PLI1/Assembler-Makroschnittstelle.

[38] **Systeminstallation** (BS2000)
Benutzerhandbuch

Zielgruppe
BS2000-Systemverwalter
Inhalt
Neuinstallation, Versionsumstellung, Generierung eines neuen Public-Volume-Sets, Generierung eines Subsystemkatalogs, Anweisungen für SIR und UGEN.
Einsatz
Systemverwaltung, Rechenzentrum

[39] **Systemkonventionen** (BS1000, BS2000, TRANSDATA, PDN)
Beschreibung

Zielgruppe
Benutzer von SIEMENS-Großrechenanlagen
Inhalt
Betriebssystemkonventionen für BS1000, BS2000 und TRANSDATA PDN;
Konventionen für Datenträger;
Codes für die Zeichendarstellung;

[40] BS2000
Systemverwaltung
Benutzerhandbuch

Zielgruppe
BS2000-Systemverwaltung
Inhalt
Beschreibung der Möglichkeiten und Aufgaben der Systemverwaltung zur Steuerung und Verwaltung des Betriebssystems.
Das Handbuch enthält folgende Kapitel:
- Systemadministration (Benutzer- und Dateiverwaltung, Abrechnung, Systemdiagnose, Systemkorrekturen, Parameterservice)
- Systemsteuerung und -optimierung (Auftrags-, Task- und Speicherverwaltung, DSSM, MPVS)
- Datensicherheit (SRPM, FACS, SAT)
- Datensicherung (Sicherungskonzepte, SW-Produkte zur Datensicherung, Rekonstruktion von Dateien)
- Automatisierung der Systembedienung
- Kommandos im SDF-Format

Einsatz
Systemverwaltung, Rechenzentrum

[41] BS2000
TSOSLNK
Benutzerhandbuch

Zielgruppe
Software-Entwickler
Inhalt
Anweisungen und Makroaufrufe des Binders TSOSLNK zum Binden von Lade- und Großmodulen.
Kommandos des statischen Laders ELDE.

Bestellen von Handbüchern

Die aufgeführten Handbücher finden Sie mit ihren Bestellnummern im *Druckschriftenverzeichnis* der Siemens Nixdorf Informationssysteme AG. Dort ist auch der Bestellvorgang erklärt. Neu erschienene Titel finden Sie in den *Druckschriften-Neuerscheinungen*.

Beide Veröffentlichungen erhalten Sie regelmäßig, wenn Sie in den entsprechenden Verteiler aufgenommen sind. Wenden Sie sich bitte hierfür an eine Geschäftsstelle unseres Hauses.

Stichwörter

Q

R

T

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*.

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.