Deutsch

FUJITSU Software BS2000

AID V3.4B

Testen von FORTRAN-Programmen

Benutzerhandbuch

Ausgabe Juni 2018

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.

Nach DIN EN ISO 9001:2015 zertifizierte Dokumentationserstellung

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 9001:2015 erfüllt.

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

Copyright und Handelsmarken

Copyright © 2018 Fujitsu Technology Solutions GmbH.

Alle Rechte vorbehalten.
Liefermöglichkeiten und technische Änderungen vorbehalten.

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

Inhalt

1 Einleitung

Mit AID (Advanced Interactive Debugger) steht im Betriebssystem BS2000 eine leistungsstarke Dialog-Testhilfe zur Verfügung. Fehlerdiagnose, Test und vorläufige Fehlerbehebung aller im BS2000 erstellten Programme können Sie mit AID wesentlich schneller und mit weniger Aufwand durchführen als mit anderen Mitteln, wie z.B. dem Einfügen von Testhilfe-Anweisungen im Programm. AID ist permanent verfügbar und besitzt eine hohe Anpassungsfähigkeit an die jeweilige Programmiersprache. Ein Programm, das Sie mit AID getestet haben, muss nicht erneut übersetzt werden, sondern kann sofort in den produktiven Einsatz gehen. Der Funktionsumfang von AID und seine Testsprache, die AID-Kommandos, sind primär auf die Dialoganwendung zugeschnitten. AID kann aber ebenso gut im Batch-Betrieb eingesetzt werden. AID bietet Ihnen vielfältige Möglichkeiten zur Ablaufüberwachung, Ablaufsteuerung, Ausgabe und Änderung von Speicherinhalten, Abfrage von Informationen über den Programmablauf und zur Handhabung von AID.

Mit AID können Sie sowohl auf der symbolischen Ebene der jeweiligen Programmiersprache als auch auf Maschinencode-Ebene testen. Wurden beim Übersetzen LSD-Sätze erzeugt, können Sie im Test Daten, Anweisungsmarken und Programmteile mit den Namen ansprechen, die Sie beim Programmieren vergeben haben. Anweisungen können Sie mit den Nummern oder Namen ansprechen, die vom Compiler erzeugt wurden. Wurden keine LSD-Sätze für ein Programm oder Modul erzeugt, können Sie Daten und Anweisungen mit virtuellen Adressen, über CSECT-Namen und Schlüsselwörter ansprechen.

Die BS2000-Kommandos, die in der AID-Dokumentation vorkommen, sind im SDF-Format (System Dialog Facility), EXPERT-Form beschrieben. SDF ist die Dialogschnittstelle zum BS2000. Die SDF-Kommandosprache löst die bisherige Kommandosprache im ISP-Format ab.

1.1 Zielsetzung und Zielgruppen der AID-Dokumentation

AID wendet sich an alle Software-Entwickler, die im BS2000 mit den Programmiersprachen COBOL, FORTRAN, C, PL/I, ASSEMBH arbeiten oder Programme auf Maschinencode-Ebene testen oder korrigieren wollen.

1.2 Konzept der AID-Dokumentation

Die Dokumentation von AID besteht aus einem Basishandbuch und den sprachspezifischen Handbüchern für das symbolische Testen sowie dem Handbuch für das Testen auf Maschinencode-Ebene. Zusammen mit dem Basishandbuch enthält das Handbuch für die von Ihnen gewählte Sprache alle Informationen, die Sie zum Testen brauchen. Das Handbuch für das Testen auf Maschinencode-Ebene kann statt oder zusätzlich zu einem der sprachspezifischen Handbücher eingesetzt werden.

AID - Basishandbuch [1]

Im Basishandbuch finden Sie einen Überblick über AID und die Beschreibung der Sachverhalte und Operanden, die in allen Programmiersprachen gleich sind. Im Überblick wird die BS2000-Umgebung beschrieben, es werden die grundlegenden Begriffe erläutert und der AID-Kommandovorrat vorgestellt. Die anderen Kapitel beschreiben die Testvorbereitung, die Kommandoeingabe, die Operanden Subkommando, komplexe Speicherreferenz und Medium-und-Menge, die AID-Literale und die Schlüsselwörter. Das Handbuch enthält außerdem die in Kommandofolgen unzulässigen BS2000-Kommandos.

AID - Testen auf Maschinencode-Ebene [2]
AID - Testen von COBOL-Programmen [3]
AID - Testen von FORTRAN-Programmen
AID - Testen unter Posix [4]
AID - Testen von ASSEMBH-Programmen [5]
AID - Testen von C-Programmen

In den sprachspezifischen Handbüchern und dem Handbuch für das Testen auf Maschinencode-Ebene finden Sie die Kommandos in alphabetischer Reihenfolge. Alle einfachen Speicherreferenzen sind hierin enthalten.

In den sprachspezifischen Handbüchern ist die Beschreibung der Operanden auf die jeweilige Programmiersprache zugeschnitten. Es wird vorausgesetzt, dass Ihnen der jeweilige Sprachumfang und die Handhabung des entsprechenden Compilers geläufig sind.

Das Handbuch für das Testen auf Maschinencode-Ebene können Sie für Programme einsetzen, zu denen keine LSD-Sätze vorhanden sind oder für die die Informationen aus dem symbolischen Testen zur Diagnose nicht ausreichen. Beim Testen auf Maschinencode-Ebene sind Sie bei der Anwendung der AID-Kommandos unabhängig von der Programmiersprache, in der das Programm erstellt wurde.

Readme-Datei

Funktionelle Änderungen der aktuellen Produktversion und Nachträge zu diesem Handbuch entnehmen Sie bitte ggf. der produktspezifischen Readme-Datei.

Readme-Dateien stehen Ihnen online bei dem jeweiligen Produkt zusätzlich zu den Produkthandbüchern unter *http://manuals.ts.fujitsu.com* zur Verfügung. Alternativ finden Sie Readme-Dateien auch auf der Softbook-DVD.

Informationen unter BS2000

Wenn für eine Produktversion eine Readme-Datei existiert, finden Sie im BS2000-System die folgende Datei:

`SYSRME.<product>.<version>.<lang>`

Diese Datei enthält eine kurze Information zur Readme-Datei in deutscher oder englischer Sprache (<lang>=D/E). Die Information können Sie am Bildschirm mit dem Kommando `/SHOW-FILE` oder mit einem Editor ansehen.
Das Kommando `/SHOW-INSTALLATION-PATH INSTALLATION-UNIT=<product>` zeigt, unter welcher Benutzerkennung die Dateien des Produkts abgelegt sind.

Ergänzende Produkt-Informationen

Aktuelle Informationen, Versions-, Hardware-Abhängigkeiten und Hinweise für Installation und Einsatz einer Produktversion enthält die zugehörige Freigabemitteilung. Solche Freigabemitteilungen finden Sie online unter *http://manuals.ts.fujitsu.com*.

1.3 Änderungen gegenüber dem Vorgänger-Handbuch

AID V3.4B30 bietet gegenüber der Version V3.4B10 folgende neue Funktionalität:

- Erweiterung des %AID-Kommandos: neuer Operand *LEV*. Dieser Operand kann die Ausgabe des AID-Kommandos %SDUMP %NEST um die Ebenen innerhalb der Aufrufhierarchie erweitern.
- Neue Qualifikation *NESTLEV* in den Kommandos %DISPLAY, %MOVE, %SDUMP und %SET zur Qualifikation aller Instanzen rekursiver Daten.
- Erweiterung des %FIND-Kommandos, mit der es möglich wird, den *find-bereich* nach Zeichen aus einem von XHCS unterstützten Coded Character Set (CCS) zu durchsuchen.

1.4 Darstellungsmittel

kursiv Im Fließtext werden Operanden in *kursiven kleinbuchstaben* geschrieben.

Mit diesem Symbol werden Stellen im Text gekennzeichnet, die Sie besonders beachten sollten.

2 Voraussetzungen zum symbolischen Testen

Für das symbolische Testen benötigt AID eine "List for Symbolic Debugging" (LSD), in der die in einem Programm definierten symbolischen Namen verzeichnet sind. Diese LSD-Informationen werden vom Compiler erzeugt und vom Binder, dynamischen Bindelader oder vom statischen Binder und Starter übernommen. Die zum Übersetzen nötigen Steueranweisungen sind nachfolgend kurzgefasst beschrieben.
Im AID-Basishandbuch, Kapitel 3 finden Sie allgemeine Informationen zu LSD-Sätzen, zum Binden, Laden und Starten.

2.1 Übersetzen

Der FOR1-Compiler lässt sich (ab Version 2.1A) auf zwei Arten steuern:

– durch SDF-Optionen oder
– durch COMOPT-Anweisungen.

Entsprechend den beiden Steuerungsmöglichkeiten vereinbaren Sie mit den folgenden Operanden, ob der Compiler LSD-Sätze erzeugen soll oder nicht:

SDF-Steuerung

```
/START-FOR1-COMPILER .....,TEST-SUPPORT = PARAMETER (TOOL-SUPPORT = { NO  })
                                                                    { AID }
```

NO Es werden keine LSD-Sätze erzeugt. Das Programm kann mit AID nur maschinennah getestet werden.

AID Der Compiler erzeugt LSD-Sätze. Das Programm kann mit AID symbolisch getestet werden.

COMOPT-Steuerung

```
/START-PROGRAM $FOR1
*...
                        ⎧  NO   ⎫
*COMOPT SYMTEST = ⎨  MAP  ⎬
                        ⎩  ALL  ⎭
```

NO Es werden keine LSD-Sätze erzeugt.

MAP Es werden keine LSD-Sätze erzeugt, aber Aufrufhierarchien können rückverfolgt werden.

ALL Der Compiler erzeugt LSD-Sätze. Das Programm kann mit AID symbolisch getestet werden.

Ab FOR1 Version 2.1A können auch zu optimierten Programmen LSD-Sätze erzeugt werden. Das optimierte Programm stimmt dann allerdings nicht mehr mit der Übersetzungsliste überein:

– Die Reihenfolge der Anweisungen kann sich ändern.

– Eine Anweisung kann in mehrere Anweisungen aufgeteilt werden.

– Anweisungen können wegfallen.

– Mit %DISPLAY wird im allgemeinen noch der alte Wert einer Variablen ausgegeben, da das Abspeichern eines Wertes in eine Variable nur selten unmittelbar nach Durchlaufen der entsprechenden Zuweisungsanweisung erfolgt.

Wenn Sie ein hoch-optimiertes FORTRAN-Programm (SDF-Option `OPTIMIZATION = HIGH` bzw. COMOPT-Anweisungen `OPTIMIZE={3|4}` und `PROCEDURE-OPTIMIZATION = SPECIAL)` mit AID testen wollen, können Sie sich eine Decompiler-Liste (siehe FOR1-Benutzerhandbuch [8]) erstellen lassen, die das Testen eines optimierten Programms erleichtert.
Das Kommando %JUMP können Sie jedoch nur zum Testen eines nicht-optimierten Programmes verwenden.

In einer Übersetzung kann nicht gleichzeitig mehrfach benutzbarer Code und LSD-Information erzeugt werden. Werden beide Optionen angegeben, setzt FOR1 die SDF-Option `SHAREABLE-CODE = YES` bzw. die COMOPT-Anweisung `OBJECT = (SHARE)` zurück und gibt eine Fehlermeldung aus.

Eine vollständige Darstellung der Operanden, die die Übersetzung steuern, finden Sie im FOR1-Benutzerhandbuch [8].

Beispiel

```
/START-FOR1-COMPILER SOURCE = QUELLE.TEST,
                     TEST-SUPPORT = PARAMETER (TOOL-SUPPORT = AID),
                     MODULE-LIBRARY = PROGRAMLIB
```

Bei der Übersetzung des Quellprogramms QUELLE.TEST soll ein Bindemodul mit LSD-Sätzen erzeugt werden. Der Bindemodul soll direkt in die PLAM-Bibliothek PROGRAMLIB geschrieben werden.
Dasselbe Beispiel würde mit COMOPT-Steuerung folgendermaßen lauten:

```
/DELETE-SYSTEM-FILE FILE-NAME = OMF
/START-PROGRAM $FOR1
*COMOPT SOURCE=QUELLE.TEST
*COMOPT SYMTEST=ALL
*COMOPT MODULE-LIBRARY=PROGRAMLIB
*END
```

2.2 Binden, Laden und Starten

Zu diesem Abschnitt lesen Sie bitte nach im AID Basishandbuch, Abschnitt "Symbolisches Testen".

In der Testphase ist es sinnvoll, das Programm zunächst nur mit LOAD-PROGRAM zu laden, damit Sie die AID-Kommandos eingeben können, die Sie zum Testen benötigen. Binden, laden und starten können Sie das Programm mit START-PROGRAM. Beide SDF-Kommandos sind im AID-Basishandbuch, Kapitel 3 beschrieben, sie sind für alle Programmiersprachen gleich.

Ihr FORTRAN-Programm können Sie auch mit START-FOR1-PROGRAM binden, laden und starten, wobei der SDF-Operand TESTOPT die Behandlung der LSD-Sätze steuert. Ab FOR1 Version 2.2A wird das Programm mit START-FOR1-PROGRAM nur geladen, aber nicht gestartet, wenn Sie die SDF-Optionen TESTOPT = AID und RUNOPT = NO angeben. Sie können dann AID-Kommandos zum Testen eingeben und anschließend mit %RESUME das Programm starten.

Wenn Sie mit einer FOR1-Version < 2.2A arbeiten, können Sie sich dadurch behelfen, dass Sie an geeigneter Stelle im Programm die FORTRAN-Anweisung PAUSE einfügen. PAUSE unterbricht den Programmablauf, sodass Sie Gelegenheit haben, AID-Kommandos einzugeben.

```
/START-FOR1-PROGRAM .....,TESTOPT = { NONE }
                                    { AID  }
```

NONE — Das Programm wird ohne LSD-Sätze geladen.
Wenn die LSD-Sätze mit dem Bindemodul in einer PLAM-Bibliothek stehen, können sie bei Bedarf von AID nachgeladen werden. Hierzu muss die Bibliothek mit %SYMLIB angemeldet werden.

AID — Das Programm wird mit den LSD-Sätzen geladen.
Der Binder überprüft nicht, ob der verarbeitete Bindemodul wirklich LSD-Sätze enthält.

Beispiele

1.
```
/START-FOR1-PROGRAM FROM-FILE = *MODULE (LIBRARY = *OMF),
                    TESTOPT = AID, RUNOPT = NO
```

   Aus der temporären Bindemoduldatei bindet der dynamische Bindelader das Programm und lädt es mit den zugehörigen LSD-Sätzen (ab FOR1 V2.2A).

2.
```
/START-FOR1-PROGRAM FROM-FILE = *PHASE (LIBRARY = PROGRAMLIB,
                    ELEMENT = ROOTMOD), TESTOPT = NONE
```

   Aus der PLAM-Bibliothek PROGRAMLIB wird das gebundene Programm ROOTMOD ohne LSD-Sätze geladen und gestartet.

3 FORTRAN-spezifische Adressierung

In diesem Kapitel werden nur die Speicherreferenzen beschrieben, die für das symbolische Testen von FORTRAN-Programmen verwendet werden. Eine allgemeine Beschreibung der Adressierung finden Sie im AID-Basishandbuch, Kapitel 6.

Qualifikationen

Qualifikationen müssen immer in der Reihenfolge angegeben werden, in der sie hier beschrieben sind. Sie werden durch Punkte getrennt. Zwischen der letzten Qualifikation und dem anschließenden Operanden muss ebenfalls ein Punkt stehen.

E={VM|Dn}
: Die Basis-Qualifikation legt fest, ob der AID-Arbeitsbereich im geladenen Programm (E=VM) oder in einer Dump-Datei (E=Dn) liegen soll. Die Basis-Qualifikation wird beim symbolischen Testen und beim maschinennahen Testen gleich verwendet und ist im AID-Basishandbuch, Kapitel 6 und bei %BASE beschrieben. Auf eine Basis-Qualifikation kann unmittelbar ein Daten- oder Anweisungsname, eine Source-Referenz oder eine komplexe Speicherreferenz folgen.

PROG=program-name
: Als Bereichs-Qualifikation können Sie bei FORTRAN die PROG-Qualifikation verwenden. *program-name* bezeichnet eine Programmeinheit aus einem FORTRAN-Programm.
program-name ist der maximal 7stellige Name, der im Quellprogramm in einer PROGRAM-, SUBROUTINE- oder FUNCTION-Anweisung vergeben wurde.
Operanden, die einen Adressbereich (%CONTROLn, %TRACE) oder einen Namensraum (%SDUMP) angeben, können mit der PROG-Qualifikation enden.
Der Adressbereich bzw. Namensraum umfasst dann die gesamte Programmeinheit.

PROG=program-name**.**program-name
: Wird der Name einer Programmeinheit direkt im Anschluss an eine PROG-Qualifikation wiederholt, dann bezeichnen Sie damit die Adresse der ersten ausführbaren Anweisung dieser Programmeinheit.
Diese Angabe können Sie in %DISASSEMBLE und %INSERT verwenden.

NESTLEV=level-nummer

Die NESTLEV-Qualfikation bezeichnet eine Ebene in der aktuellen Aufrufhierarchie.

Ebenso wie die Qualifikation S=*srcname*.PROC=*funktion* dient die Qualifikation NEST–LEV=*level-nummer* dazu, Datennamen zu manipulieren, die vom Anwender in den Source Units deklariert wurden. Die Qualifikation NESTLEV=*level-nummer* kann nur mit der Basisqualifikation E={VM|Dn} kombiniert werden.

Die Qualifikation NESTLEV akzeptiert als Eingabe die Nummer einer Ebene in der aktuellen Aufrufhierarchie, d.h. eine Referenz auf die aktuelle Aufrufhierarchie. Basierend auf dieser Referenz identifiziert AID eine komplette Liste von verfügbaren Datennamen, die auf der angegebenen Ebene definiert wurden.

Normalerweise müssen Sie sich die Aufrufhierarchie ausgeben lassen und diese analysieren, bevor Sie die Qualifikation NESTLEV verwenden können. Folgende AID-Kommandos geben die um die Aufrufebenen erweiterte aktuelle Aufrufherarchie aus:

```
%AID LEV=ON
%SDUMP %NEST
```

Die Qualifikation NESTLEV können Sie in den Kommandos %DISPLAY, %MOVE, %SDUMP and %SET verwenden. In diesen Kommandos liefert die Qualifikation NEST–LEV=*level-nummer* das gleiche Resultat wie die Qualifikation S=*srcname*.PROC=*funktion*, sofern *level-nummer* korrekt ist.

Ein Beispiel zur Verwendung der NESTLEV-Qualifikation finden Sie im AID-Basishandbuch, Abschnitt „Bereichsqualifikationen“[1].

Speicherreferenzen

Als Speicherreferenzen können alle in den LSD-Sätzen verzeichneten Datennamen und Anweisungsmarken aus dem Programm und die vom Compiler erzeugten Anweisungsnummern verwendet werden, und sie können alle Operationen, wie im AID-Basishandbuch, Kapitel 6 beschrieben, darauf anwenden.

In allen Operanden, in denen *kompl-speicherref* möglich ist, können Sie beliebig wechseln zwischen den in diesem Handbuch beschriebenen Speicherreferenzen und denen für das Testen auf Maschinencode-Ebene [2].

datenname
: steht für alle im Quellprogramm definierten Namen von Konstanten, Variablen und Feldern. Um ein Feldelement anzusprechen, können Sie *feldname* indizieren. Es sind so viele Indizes erforderlich, wie in einer FORTRAN-Anweisung zum Zugriff angegeben werden müssen. Mehrere Indizes müssen durch Komma getrennt werden.
index kann folgendermaßen angegeben werden:

```
{n                    }
{datenname            }
{arithmetischer ausdruck}
```

n
: ist eine Ganzzahl mit einem Wert $-2^{31} \leq n \leq 2^{31}-1$.

datenname
: bezeichnet eine numerische Variable vom Typ Integer, sie muss in derselben Programmeinheit wie *feldname* liegen.

arithmetischer Ausdruck
: Der Wert für *index* wird von AID errechnet. Erlaubt sind die arithmetischen Operatoren (+, –, /, *) und die oben aufgeführten Operanden *n* und *datenname*.

datenname kann in allen Kommandos zur Ausgabe und Änderung von Daten angegeben werden, das sind die Kommandos %DISPLAY, %MOVE, %SDUMP, %SET, und im %FIND-Kommando (Suchen einer Zeichenfolge).

L'n'
: ist ein Anweisungsname und bezeichnet die Adresse der ersten ausführbaren FORTRAN-Anweisung nach einer Anweisungsmarke.
n ist eine maximal 5stellige Anweisungsmarke des Quellprogramms, die vom Programmierer vergeben wurde. Führende Nullen dürfen nicht angegeben werden.
L'n' kann in allen Operanden angegeben werden, die entweder eine Adresse im ausführbaren Teil des Programms bezeichnen (%DISASSEMBLE, %FIND, %INSERT, %JUMP) oder die zur Ausgabe und Änderung von Speicherstellen angegeben werden (%DISPLAY, %MOVE, %SET).

S'n'

ist eine Source-Referenz und bezeichnet die Adresse einer ausführbaren FORTRAN-Anweisung.
n ist die Nummer einer Anweisung des Quellprogramms, die vom Compiler vergeben wird und in der Übersetzungsliste der Spalte STMT zu entnehmen ist.
S'n' kann in allen Operanden angegeben werden, die einen Bereich (%CONTROLn, %TRACE) oder eine Adresse (%DISASSEMBLE, %FIND, %INSERT) im ausführbaren Teil des Programms bezeichnen oder die zur Ausgabe und Änderung von Speicherstellen angegeben werden (%DISPLAY, %MOVE, %SET).

index

Bereich von Indizes

```
array (index{,...})
```

Es kann ein Bereich von Indizes angegeben werden. Dies erfolgt in der Form:

index1 : *index2*
Damit wird der Bereich zwischen *index1* und *index2* bezeichnet. Beide müssen innerhalb der Indexgrenzen liegen und *index1* muss kleiner oder gleich *index2* sein.

*

Damit bezeichnen Sie den gesamten Indexbereich der Dimension. Bei eindimensionalen Arrays ist diese Angabe gleichbedeutend mit der Verwendung des Array-Namens ohne Indizierung.

Die Bereichsangabe können Sie nur im %DISPLAY Kommando verwenden. Der Array-Name mit Bereichsangabe darf nicht in einer Adressrechnung eingesetzt werden. Es darf keine Typ- oder Längenmodifikation folgen.

Beispiele

```
%D array (*,3)
```

Von einem zweidimensionalen Array werden alle die Elemente ausgegeben, die zur ersten Dimension gehören und deren Index der zweiten Dimension gleich 3 ist.

```
%D array (1 : 3,*,5 : 15)
```

Von einem dreidimensionalen Array werden folgende Elemente ausgegeben:
- der Index der ersten Dimension ist 1, 2 oder 3,
- der Index der zweiten Dimension läuft von der Indexuntergrenze bis zur Indexobergrenze,
- der Index der dritten Dimension läuft von 5 bis 15.

4 Metasyntax

Für die Darstellung der Kommandos wird folgende Metasyntax verwendet. Die Aufstellung zeigt die verwendeten Symbole und beschreibt ihre Bedeutung.

`GROSSBUCHSTABEN`
Zeichenfolge, die Sie unverändert übernehmen müssen, wenn Sie eine Funktion auswählen.

`kleinbuchstaben`
Zeichenfolge, die eine Variable bezeichnet. An ihre Stelle müssen Sie einen der zugelassenen Operandenwerte setzen.

```
{ alternativ }
{    ...     }
{ alternativ }
```

`{alternativ | ... | alternativ}`
Alternativen, unter denen Sie eine auswählen müssen. Die beiden Formate sind gleichbedeutend.

`[wahlweise]`
Die in eckige Klammern eingeschlossenen Angaben können entfallen.
Bei AID-Kommandona
men kann der in eckigen Klammern stehende Teil nur komplett entfallen, andere Abkürzungen ergeben einen Syntaxfehler.

`[...]`
Wiederholbarkeit einer wahlfreien syntaktischen Einheit. Muss vor jede Wiederholung ein Trennzeichen, z.B. Komma gesetzt werden, steht es vor den Wiederholungspunkten.

`{...}`
: Wiederholbarkeit einer syntaktischen Einheit, die einmal angegeben werden muss. Muss vor jede Wiederholung ein Trennzeichen, z.B. Komma gesetzt werden, steht es vor den Wiederholungspunkten.

Unterstreichung
: Die Unterstreichung kennzeichnet den Standardwert, den AID einsetzt, wenn Sie für einen Operanden keinen Wert angeben.

.
: Der dickere Punkt trennt Qualifikationen, steht für eine *vorqualifikation* (siehe %QUALIFY), ist der Operator für einen Adressversatz oder Teil des Durchlaufzählers bzw. Subkommandonamens. Eingegeben wird der dickere Punkt mit dem Punkt, der auf der Tastatur ist. Er wurde nur der besseren Lesbarkeit wegen dicker dargestellt.

Im Fließtext werden Operanden in *kursiven kleinbuchstaben* dargestellt.

5 AID-Kommandos

In den Abschnitten dieses Kapitels werden die AID-Kommandos in alphabetischer Reihenfolge ausführlich beschrieben. Die folgende Tabelle gibt eine Übersicht über die zur Verfügung stehenden Kommandos:

Kommando	**Funktion**
%AID	Verändern globaler Einstellungen
%BASE	Globale Festlegung der Basis-Qualifikation
%CONTINUE	Starten oder Fortsetzen des Programms, eventuell Weiterführen eines noch aktiven %TRACE
%CONTROLn	Überwachen ausgewählter Anweisungen
%DISASSEMBLE	Rückübersetzen von Speicherinhalten in symbolische Assembler-Notation
%DISPLAY	Ausgeben des Inhalts von Datenelementen, von deren Adressen und Längen und von Systeminformationen und Literalen
%DUMPFILE	Öffnen oder Schließen von Dump-Dateien und Zuweisen von Linknamen
%FIND	Suchen einer Zeichenfolge
%HELP	Help-Funktion für AID-Kommandos und -Meldungen
%INSERT	Testpunkte setzen zur Überwachung des Programmablaufs
%JUMP	Vereinbaren einer Fortsetzungsadresse
%MOVE	Ändern der Inhalte von Datenelementen ohne Typüberprüfung und ohne Konvertierung numerischer Werte
%ON	Überwachen ausgewählter Ereignisse
%OUT	Vereinbaren von Ausgabe-Medien und Zusatzinformationen für Ausgabe-Kommandos
%OUTFILE	Öffnen oder Schließen von AID-Ausgabedateien und Zuweisen von Linknamen
%QUALIFY	Definieren einer Vorqualifikation
%REMOVE	Löschen von Überwachungsvereinbarungen

Kommando	Funktion
%RESUME	Starten oder Fortsetzen des Programms und Beenden eines aktiven %TRACE
%SDUMP	Symbolischer Dump; Ausgeben von Datenelementen oder der Programmnamen der aktuellen Aufrufhierarchie
%SET	Ändern der Inhalte von Datenelementen mit Typüberprüfung und mit Konvertierung numerischer Werte
%STOP	Anhalten des Programms und Übergang in den Kommandomodus
%SYMLIB	Anmelden von Bibliotheken zum Nachladen von LSD-Sätzen
%TITLE	Definieren von Seitenüberschriften und Einschalten der Seitenzählung für Ausgaben nach SYSLST
%TRACE	Starten oder Fortsetzen des Programms mit Ablaufverfolgung

%AID

Mit %AID können Sie globale Einstellungen vereinbaren oder bis zu diesem Kommando geltende Einstellungen wieder zurücknehmen.

– Mit *CHECK* legen Sie fest, ob vor der Ausführung von %MOVE oder %SET ein Änderungsdialog durchgeführt werden soll.

– Mit *REP* legen Sie fest, ob die Speicheränderungen eines %MOVE-Kommandos als REP abgelegt werden sollen.

– Mit *SYMCHARS* legen Sie fest, ob AID den Bindestrich "-" in Programm-, Daten- und Anweisungsnamen als Bindestrich oder als Minuszeichen interpretieren soll.
Damit der Bindestrich entsprechend den FORTRAN-Konventionen stets als Minuszeichen interpretiert wird, muss SYMCHARS = NOSTD gesetzt werden.

– Mit *OV* legen Sie fest, ob AID die Überlagerungsstruktur eines Programms (Overlay) berücksichtigen soll.

– Mit *LOW* legen Sie fest, ob AID Kleinbuchstaben aus Character-Literalen und Namen in Großbuchstaben konvertieren oder als Kleinbuchstaben interpretieren soll.

  Voreinstellung ist OFF.

– Mit *DELIM* legen Sie die Begrenzer (Delimiter) für die AID-Ausgabe alphanumerischer Daten fest. Der senkrechte Strich ist der Standard-Begrenzer.

– Mit *LANG* legen Sie fest, ob AID die Informationen des %HELP-Kommandos auf Deutsch oder Englisch ausgeben soll.

– Mit dem Operanden *LEV* legen Sie fest, ob beim AID-Kommando %SDUMP %NEST die Aufruftiefe ausgegeben werden soll.

Kommando	Operand
%AID	CHECK [= {ALL|NO}] REP [= {YES|NO}] SYMCHARS [= {STD|NOSTD}] OV [= {YES|NO}] LOW [= {ON|OFF}] DELIM [= {C'x'|'x'C|'x' '|'}] LANG [= {D | E}] LEV [= {ON|OFF}]

Mit %AID getroffene Vereinbarungen gelten, bis sie durch ein neues %AID-Kommando geändert werden oder bis /LOGOFF.

%AID darf nur als Einzelkommando eingegeben werden, es darf nicht in einer Kommandofolge oder einem Subkommando stehen.

%AID verändert den Programmzustand nicht.

CHECK

ALL
: Vor der Ausführung eines %MOVE oder %SET führt AID folgenden Änderungsdialog:

```
OLD CONTENT:
AAAAAAAA
NEW CONTENT:
BBBBBBBB
%  IDA0129 CHANGE? (Y = YES; N = NO) ?

N

I342 NOTHING CHANGED
```

  Nach der Eingabe **Y** wird der alte Datenfeld-Inhalt ohne weitere Meldung überschrieben.
  In Prozeduren im Stapelbetrieb kann AID keinen Dialog führen und nimmt immer **Y** an.

NO
: %MOVE und %SET werden ohne Änderungsdialog ausgeführt.

Wird der *CHECK*-Operand ohne Wertangabe eingegeben, setzt AID den Standardwert (NO) ein.

REP

YES
Zu Änderungen im Speicher mit %MOVE werden LMS-UPDR-Korrektursätze (REPs) erstellt. Wenn die Objekt-Strukturliste nicht zur Verfügung steht, erstellt AID keine Korrektursätze und gibt eine Fehlermeldung aus.

AID hinterlegt die Korrekturen mit den nötigen LMS-UPDR-Anweisungen in einer Datei mit dem Linknamen F6, aus der sie als fertiges Paket übernommen werden können. Achten Sie deshalb darauf, dass Sie in die Datei mit dem Linknamen F6 keine anderen Ausgaben schreiben lassen. Ist keine Datei mit dem Linknamen F6 angemeldet (siehe %OUTFILE), so wird der REP in der von AID angelegten Datei AID.OUTFILE.F6 abgelegt.

Benutzerspezifische REP-Dateien müssen mit FCBTYPE = SAM angelegt sein. Von AID angelegte REP-Dateien werden ebenfalls mit FCBTYPE=SAM, RECFORM=V und OPEN=EXTEND angelegt.
Die Datei bleibt geöffnet, bis sie mit %OUTFILE geschlossen wird oder bis /LOGOFF.

NO
Es werden keine REPs erstellt.

Wird der *REP*-Operand ohne Wertangabe eingegeben, setzt AID den Standardwert (NO) ein. Der *REP*-Operand des %MOVE-Kommandos kann die mit %AID getroffene Vereinbarung für dieses eine %MOVE-Kommando ersetzen. Für nachfolgende %MOVE-Kommandos ohne *REP*-Operand gilt dann wieder die Vereinbarung mit %AID.

SYMCHARS

STD
Der Bindestrich (-) wird als alphanumerisches Zeichen interpretiert und kann somit in Programm-, Daten- und Anweisungsnamen verwendet werden. Er wird nur dann als Minus-Zeichen interpretiert, wenn vor dem Bindestrich ein Leerzeichen steht.

NOSTD
Der Bindestrich (-) wird immer als Minus-Zeichen interpretiert und kann in Namen nicht verwendet werden.

Wird der *SYMCHARS*-Operand ohne Wertangabe eingegeben, setzt AID den Standardwert (STD) ein. SYMCHARS = NOSTD muss gesetzt werden, wenn das Zeichen "-" entsprechend den FORTRAN-Konventionen stets als Minuszeichen interpretiert werden soll.

OV

YES
müssen Sie angeben, wenn Sie ein Programm mit Überlagerungsstruktur (Overlay) testen. AID überprüft dann jedesmal, ob der angesprochene Programmteil eventuell aus einem nachgeladenen Segment stammt.

NO
AID geht davon aus, dass das zu testende Programm ohne Überlagerungsstruktur gebunden ist. AID benutzt die einmal geladenen LSD-Sätze, ohne zu prüfen, ob der angesprochene Programmteil in einem nachgeladenen Segment liegt.

Wird der *OV*-Operand ohne Wertangabe eingegeben, setzt AID den Standardwert (NO) ein.

LOW

ON
: Kleinbuchstaben in Character-Literalen und in Programm-, Daten- und Anweisungsnamen werden nicht in Großbuchstaben konvertiert.

OFF
: Alle Kleinbuchstaben aus Benutzereingaben werden in Großbuchstaben umgesetzt.

ALL
: Wirkt wie %AID LOW=ON, wobei zusätzlich die Unterscheidung von Klein-/Großbuchstaben bei der Eingabe von allen BLS-Namen berücksichtigt wird.

  Außerdem wird, wie bei der Angabe von %AID LOW=ON, die Klein-/Großschreibung in Character-Literalen und in Programm-, Daten- und Anweisungsnamen beibehalten.

  BLS-Namen, die von AID verwendet werden, sind:
  – Kontextnamen der CTX-Qualifikation
  – Namen von Ladeeinheiten der L-Qualifikation
  – Bindemodulnamen der O-Qualifikation
  – CSECT-Namen der C-Qualifikation
  – COMMON-Namen der COM-Qualifikation
  – Namen von Programmeinheiten der S-Qualifikation

Wenn in einer Testsitzung noch kein *LOW*-Operand eingegeben wurde, gilt die Voreinstellung OFF.
Wird der *LOW*-Operand ohne Wertangabe eingegeben, setzt AID den Standardwert (ON) ein. Um wieder die Umsetzung in Großbuchstaben einzuschalten, müssen Sie LOW=OFF eingeben.

DELIM

C’x’|’x’C|’x’
: Mit diesem Operanden legen Sie ein Zeichen als linke und rechte Begrenzung (Delimiter) für die AID-Ausgabe von symbolischen Daten vom Typ Character (Kommandos %DISPLAY und %SDUMP) fest.

|
: Der Standard-Begrenzer ist der senkrechte Strich.

Wird der *DELIM*-Operand ohne Wertangabe eingegeben, setzt AID den Standardwert (|) ein.

LANG

D
AID gibt die Informationen, die mit %HELP angefordert wurden, in Deutsch aus.

E
AID gibt die Informationen, die mit %HELP angefordert wurden, in Englisch aus.

Wird der *LANG*-Operand ohne Wertangabe eingegeben, setzt AID den Standardwert (D) ein.

LEV

ON Ausgabe der Aufruftiefe einschalten.

Wird die Ausgabe der Aufruftiefe eingeschaltet, dann gibt %SDUMP %NEST für jede Prozedur in der Aufrufhierarchie (Funktion oder Block in C/C++) zusätzlich zwei Werte aus:

– Die einfache Aufruftiefe (Counter) mit einer rückläufigen Nummerierung, d.h. von der aktuellen Prozedur zur Haupt-Prozedur. Diese Aufruftiefe kann in der *NESTLEV*-Qualifikation verwendet werden.

– Die Rekursionstiefe (RLEV) oder einen individuellen Zähler für jede Prozedur mit einer rückläufigen Nummerierung, beginnend bei 0. Die Rekursionstiefe dient nur zur Information.

OFF Ausgabe der Aufruftiefe ausschalten.

Beispiel

Im Programm SYMCHAR soll der Inhalt des Feldelements IFELD(L+M) mit einem %SET-Kommando durch den Inhalt des Feldelements IFELD(L-M) ersetzt werden.

Quellprogrammliste des Programms SYMCHAR:

```
DO/IF SEG STMT  I/H LINE          SOURCE-TEXT
──────────────────────────────────┐──────────────────────────────────────────────
      1/1    1          1  │ PROGRAM SYMCHAR
        1    2          2  │ PARAMETER (B=3, C=5)
        1    3          3  │ DIMENSION IFELD(B+C)
        1    4          4  │ INTEGER IFELD /1,2,3,4,5,6,7,8/
        1    5          5  │ L=5
        1    6          6  │ M=3
        1    7          7  │ WRITE *,IFELD
        1    8          8  │ WRITE *,IFELD(L+M)
        1    9          9  │ WRITE *, ' SYMCHAR BEENDET!'
```

```
     1   10        10 | END
```

1. Da %AID SYMCHARS = STD eingestellt ist, wird L-M als Name eines Datenelements interpretiert; AID gibt die Meldung "L-M NOT FOUND" aus.

```
/LOAD-PROG FROM-FILE=*MOD(LIB=*OMF),TEST-OPT=AID
%  BLS0001 DLL VER 823
%  BLS0517 MODULE 'SYMCHAR' LOADED
/%IN S'9' <%D IFELD;%SET IFELD(L-M) INTO IFELD(L+M);%D IFELD>
/%R
BS2000  F O R 1 : FORTRAN PROGRAM "SYMCHAR"
STARTED ON 91-02-18 AT 12:04:11
1 , 2 , 3 , 4 , 5 , 6 , 7 , 8
8
** ITN: #'00000047' *** TSN* 8438 ********************************************
SCR_REF:     9 SOURCE: SYMCHAR     PROC: SYMCHAR ********************************
IFELD( 1: 8)
(  1)          1  (  2)            2  (  3)          3  (  4)      4
(  5)          5  (  6)            6  (  7)          7  (  8)      8
I375  SYMBOL    L-M NOT FOUND

STOPPED AT SCR REF: 9, SOURCE: SYMCHAR , PROC: SYMCHAR
```

2. Nachdem %AID SYMCHARS = NOSTD eingegeben wurde, wird der Bindestrich in L-M als Minuszeichen interpretiert. AID führt nun das %SET-Kommando richtig aus.

```
/LOAD-PROG FROM-FILE=*MOD(LIB=*OMF),TEST-OPT=AID
%  BLS0001 DLL VER 823
%  BLS0517 MODULE 'SYMCHAR' LOADED
/%AID SYMCHARS=NOSTD
/%IN S'9' <%D IFELD;%SET IFELD(L-M) INTO IFELD(L+M);%D IFELD>
/%R
BS2000  F O R 1 : FORTRAN PROGRAM "SYMCHAR"
STARTED ON 91-02-18 AT 12:05:37
1 , 2 , 3 , 4 , 5 , 6 , 7 , 8
8
** ITN: #'00000047' *** TSN* 8438 ********************************************
SCR_REF:     9 SOURCE: SYMCHAR     PROC: SYMCHAR ********************************
IFELD( 1: 8)
(  1)          1  (  2)            2  (  3)          3  (  4)      4
(  5)          5  (  6)            6  (  7)          7  (  8)      8
IFELD( 1: 8)
(  1)          1  (  2)            2  (  3)          3  (  4)      4
(  5)          5  (  6)            6  (  7)          7  (  8)      2
 SYMCHAR BEENDET !
BS2000  F O R 1 : FORTRAN PROGRAM "SYMCHAR " ENDED PROPERLY AT 12:05:41
CPU - TIME USED :       0.2124 SECONDS
ELAPSED TIME    :       4.6430 SECONDS
```

%BASE

Mit %BASE legen Sie die Basis-Qualifikation fest. Alle nachfolgend eingegebenen Speicherreferenzen ohne eigene Basis-Qualifikation übernehmen die mit %BASE vereinbarte. Mit %BASE wird zugleich festgelegt, wo sich der AID-Arbeitsbereich befinden soll.

– Mit *basis* bezeichnen Sie den virtuellen Speicherbereich des geladenen Programms oder einen Speicherabzug in einer Dump-Datei.

Kommando	Operand
%BASE	[basis]

Beim Testen von FORTRAN-Programmen entspricht der AID-Arbeitsbereich dem Bereich, den die aktuelle Programmeinheit im virtuellen Speicher oder in einer Dump-Datei belegt. Geben Sie in einer Testsitzung kein %BASE oder geben Sie ein %BASE ohne Operanden ein, gilt die Basis-Qualifikation E=VM (Standardwert) und der AID-Arbeitsbereich entspricht der Programmeinheit im virtuellen Speicher, in der die aktuelle Unterbrechungsstelle liegt (AID-Standard-Arbeitsbereich).

Ein %BASE gilt bis zum nächsten %BASE, bis /LOGOFF oder bis zum Schließen der Dump-Datei (siehe %DUMPFILE), die als Basis-Qualifikation vereinbart war.

In einem Subkommando werden Speicherreferenzen bei der Eingabe mit den aktuellen Qualifikationen ergänzt, d.h. dass ein %BASE keine Auswirkung hat auf Subkommandos, die vorher vereinbart wurden.

%BASE darf nur als Einzelkommando eingegeben werden, es darf nicht in einer Kommandofolge oder einem Subkommando stehen.

%BASE verändert den Programmzustand nicht.

basis

legt die Basis-Qualifikation fest. Alle nachfolgend eingegebenen Speicherreferenzen ohne eigene Basis-Qualifikation übernehmen die mit %BASE vereinbarte.

basis-OPERAND

E = { VM | Dn }

E=VM

Der virtuelle Speicherbereich des geladenen Programms ist als Basis-Qualifikation vereinbart. VM ist der Standardwert.

E=Dn

Ein Speicherabzug in einer Dump-Datei mit dem Linknamen Dn ist als Basis-Qualifikation vereinbart.
n ist eine Zahl mit einem Wert $0 \leq n \leq 7$.

Bevor Sie eine Dump-Datei als Basis-Qualifikation vereinbaren, müssen Sie mit %DUMPFILE die entsprechende Dump-Datei einem Linknamen zuweisen und öffnen.

%CONTINUE

Mit %CONTINUE starten Sie das geladene Programm oder setzen es an der unterbrochenen oder mit %JUMP vereinbarten Stelle fort.
Im Gegensatz zu %RESUME wird ein unterbrochener, noch aktiver %TRACE durch %CONTINUE nicht beendet, sondern entsprechend den Vereinbarungen fortgesetzt.

Kommando	Operand
%CONT[INUE]	

Ein %TRACE gilt in den folgenden Fällen als unterbrochen und wird von %CONTINUE fortgesetzt:

1. Ein Subkommando wurde ausgeführt, weil eine Überwachungsbedingung aus einem %CONTROLn, %INSERT oder %ON zutraf, und das Subkommando enthielt ein %STOP.
2. Ein %INSERT endet mit einer Programmunterbrechung, weil der *steuerung*-Operand K oder S lautet.
3. Die K2-Taste wurde gedrückt.
4. Das Programm wurde durch die FORTRAN-Anweisung PAUSE angehalten.

Steht in einem Subkommando nur das Kommando %CONTINUE, wird nur der Durchlaufzähler erhöht.

Steht %CONTINUE in einer Kommandofolge oder in einem Subkommando, werden nachfolgende Kommandos nicht mehr ausgeführt.

%CONTINUE verändert den Programmzustand.

%CONTROLn

Mit %CONTROLn können Sie nacheinander bis zu sieben Ablaufüberwachungs-Funktionen vereinbaren, die dann gleichzeitig wirken. Es gibt %CONTROL1 bis %CONTROL7.

– Mit *kriterium* wählen Sie verschiedene Typen von FORTRAN-Anweisungen aus. Steht eine Anweisung des gewählten Typs zur Ausführung an, unterbricht AID das Programm und bearbeitet *subkdo*.

– Mit *control-bereich* legen Sie den Programmbereich fest, in dem *kriterium* überwacht werden soll.

– Mit *subkdo* definieren Sie ein Kommando oder eine Kommandofolge und eventuell eine Bedingung. Bei zutreffendem *kriterium* und erfüllter Bedingung wird *subkdo* ausgeführt. *subkdo* muss angegeben werden.

Kommando	Operand
%C[ONTROL]n	[kriterium][,...] [IN control-bereich] [<subkdo>]

Mehrere %CONTROLn mit unterschiedlichen Nummern beeinflussen einander nicht, so dass Sie mehrere Kommandos mit demselben *kriterium* für verschiedene Bereiche oder mit unterschiedlichen *kriterien* für denselben Bereich aktivieren können. Treffen an einer Anweisung mehrere %CONTROLn zusammen, so werden die zugehörigen Subkommandos in der Reihenfolge %C1 bis %C7 bearbeitet.

Der einzelne Operandenwert eines %CONTROLn gilt solange, bis Sie ihn durch neue Angaben in einem späteren %CONTROLn mit derselben Nummer überschreiben, bis Sie den %CONTROLn löschen oder bis zum Programmende.

Mit %REMOVE löschen Sie einen einzelnen %CONTROLn oder alle aktiven %CONTROLn-Vereinbarungen.

%CONTROLn kann nur im laufenden Programm eingesetzt werden, deshalb muss die Basis-Qualifikation E=VM eingestellt sein (siehe %BASE) oder explizit angegeben werden.

%CONTROLn verändert den Programmzustand nicht.

kriterium

Schlüsselwort, das den Typ der FORTRAN-Anweisungen festlegt, vor deren Ausführung AID *subkdo* bearbeiten soll.
Sie können mehrere Schlüsselwörter gleichzeitig angeben, die dann gemeinsam wirken. Zwischen zwei Schlüsselwörtern muss ein Komma stehen.

Wird kein *kriterium* vereinbart, arbeitet AID mit dem Standardwert %STMT, falls nicht noch aus einem vorhergehenden %CONTROLn eine *kriterium*-Vereinbarung gültig ist.

kriterium	*subkdo* **wird bearbeitet** vor**:**
%STMT	jeder FORTRAN-Anweisung, die durchlaufen wird
%ASSGN	Zuweisungs-Anweisungen
%CALL	SUBROUTINE-Aufrufen (CALL-Anweisungen)
%COND	IF(...) THEN-, ELSE IF(...) THEN-, ELSE- und IF(...)-Anweisungen
%GOTO	GOTO-Anweisungen
%IO	Ein-/Ausgabe-Anweisungen
%LAB	jeder Anweisung mit einer Anweisungsmarke
%PROC	STOP-, END-, RETURN-Anweisungen sowie der ersten ausführbaren Anweisung nach SUBROUTINE oder FUNCTION

control-bereich

legt den Programmbereich fest, in dem die Überwachungsfunktion wirksam wird. Beim Verlassen des festgelegten Programmbereichs wird die Überwachungsfunktion inaktiv, bis wieder eine Anweisung ausgeführt wird, die in dem zu überwachenden Programmbereich liegt.

Eine *control-bereich*-Definition gilt bis zum nächsten %CONTROLn derselben Nummer mit neuer Definition, bis zum entsprechenden %REMOVE %CONTROLn oder bis Programmende. Ein %CONTROLn ohne eigenen *control-bereich*-Operanden, übernimmt eine wirksame Bereichsdefinition. Ein wirksamer *control-bereich* muss in einem %CONTROLn mit derselben Nummer definiert sein und die aktuelle Unterbrechungsstelle muss innerhalb dieses Bereichs liegen. Gibt es keine wirksame Bereichsdefinition, so umfasst der *control-bereich* standardmäßig die aktuelle Programmeinheit.

```
control-bereich-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                          ⎧PROG=program-name                        ⎫
IN  [•][E=VM•] ⎨                                                    ⎬
                          ⎩[PROG=program-name•]( S'n' : S'n' )      ⎭

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

- Steht der Punkt an führender Stelle, so ist er das Kennzeichen für eine *vorqualifikation*. Sie muss mit einem vorhergehenden %QUALIFY-Kommando definiert worden sein.

Aufeinanderfolgende Qualifikationen werden durch einen Punkt getrennt. Außerdem muss zwischen der letzten Qualifikation und dem anschließenden Operandenteil ein Punkt stehen.

E=VM

Da *control-bereich* nur im virtuellen Speicher des geladenen Programms liegen kann, geben Sie *E=VM* nur an, wenn als aktuelle Basis-Qualifikation eine Dump-Datei vereinbart ist (siehe %BASE).

PROG=program-name

program-name ist der maximal 7stellige Name einer Programmeinheit.
Diese Programmeinheit muss zum Zeitpunkt der Eingabe des %CONTROLn bzw. bei der Abarbeitung des Subkommandos, in dem der %CONTROLn enthalten ist, geladen sein.

Eine PROG-Qualifikation ist nur erforderlich, wenn ein Lademodul aus mehreren Programmeinheiten entstanden ist und sich der %CONTROLn nicht auf die aktuelle bezieht oder um eine bisher geltende *control-bereich*-Vereinbarung zu überschreiben.

Endet *control-bereich* mit einer PROG-Qualifikation, so umfasst er die gesamte angegebene Programmeinheit.

(S'n' : S'n')

Der *control-bereich* wird durch die Angabe einer Anfangs- und einer Endadresse festgelegt. Beide müssen innerhalb derselben Programmeinheit liegen, und es gilt: Anfangsadresse ≤ Endadresse.
n ist die Nummer einer Anweisung; siehe Spalte STMT der Übersetzungsliste.

Soll *control-bereich* nur eine Anweisung umfassen, müssen Anfangs- und Endadresse gleich sein.

subkdo

wird immer dann bearbeitet, wenn im *control-bereich* eine Anweisung zur Ausführung ansteht, die *kriterium* entspricht. *subkdo* wird vor der Ausführung der *kriterium*-Anweisung bearbeitet.

Wenn *subkdo* nicht angegeben wird, setzt AID bei %CONTROL*n* <%STOP> ein.

Vollständig beschrieben finden Sie *subkdo* im Basishandbuch, Kapitel 5.

```
subkdo-OPERAND  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                                          ⎧AID-kommando    ⎫
<[subkdoname:] [(bedingung):] [           ⎨                ⎬ {;...}]>
                                          ⎩BS2000-kommando ⎭

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Das Subkommando kann einen Namen, eine Bedingung und einen Kommandoteil enthalten. Zu jedem Subkommando gehört ein Durchlaufzähler. Der Kommandoteil kann aus einem einzelnen Kommando oder einer Kommandofolge bestehen, er kann AID- und BS2000-Kommandos und Kommentare enthalten.

Wenn das Subkommando aus einem Namen oder einer Bedingung besteht, aber der Kommandoteil fehlt, erhöht AID beim Erreichen einer Anweisung vom Typ *kriterium* nur den Durchlaufzähler.

Im *subkdo* eines %CONTROLn sind zusätzlich zu den Kommandos, die in allen Subkommandos nicht zugelassen sind, die AID-Kommandos %CONTROLn, %INSERT, %JUMP und %ON nicht erlaubt.

Die Kommandos in einem *subkdo* werden nacheinander ausgeführt. Danach wird das Programm fortgesetzt. Die Kommandos zur Ablaufsteuerung verändern auch in einem Subkommando sofort den Programmzustand. Sie brechen *subkdo* ab und starten das Programm (%CONTINUE, %RESUME, %TRACE) oder halten es an (%STOP). Sie sind nur als letztes Kommando in einem *subkdo* sinnvoll, da nachfolgende *subkdo*-Kommandos nicht mehr ausgeführt werden. Auch ein Löschen des gerade aktiven Subkommandos mit %REMOVE ist nur als letztes Kommando in *subkdo* sinnvoll.

Beispiele

1. `%CONTROL1 %CALL, %PROCIN(S'123':S'250') <%DISPLAYZAEHLER;%STOP>`
   `%C1 %CALL,%PROC IN(S'123':S'250') <%D ZAEHLER;%STOP>`

   Die beiden AID-Kommandos unterscheiden sich nur in der Schreibweise.

   Das erste Beispiel ist voll ausgeschrieben und enthält unterschiedlich viele Leerzeichen an den zulässigen Stellen, das zweite ist abgekürzt.

   Das %CONTROL1-Kommando gilt für die Kriterien %CALL und %PROC und soll zwischen den Anweisungen 123 bis einschließlich 250 wirken.

   Tritt im Programmablauf im genannten Bereich eine der mit den Kriterien %CALL und %PROC bezeichneten FORTRAN-Anweisungen auf, wird aus *subkdo* der %DISPLAY für die Variable ZAEHLER ausgeführt. Anschließend wird durch %STOP der Programmablauf unterbrochen, und es können AID- oder BS2000-Kommandos eingegeben werden.

2. `%CONTROL1 %CALL <%DISPLAY 'CALL' T=MAX; %STOP>`
   Vor Ausführung jeder CALL-Anweisung führt AID den %DISPLAY aus *subkdo* aus und unterbricht dann das Programm aufgrund des %STOP-Kommandos.

3. `%CONTROL2 %IO <%SDUMP %NEST P=MAX; %REMOVE %C1>`
   Bevor eine IO-Anweisung ausgeführt wird, gibt AID die aktuelle Aufrufhierarchie in die Systemdatei SYSLST aus und führt dann das %REMOVE-Kommando aus, mit dem die Vereinbarungen des %CONTROL1 gelöscht werden. Das Programm läuft weiter.

4. `%C3 %PROC IN PROG=TAUSCH <%STOP>`
   Mit dem %C3 wird vereinbart, dass AID das Programm anhalten soll, unmittelbar nachdem das Unterprogramm TAUSCH aufgerufen und bevor es mit RETURN wieder verlassen wird.

5. `%C4 %PROC IN PROG=AUSGABE <(SLF LE 10): %D IFELD(1)>`
   Mit dem %C4 wird vereinbart, dass AID das erste Feldelement von IFELD ausgeben soll, bevor die erste bzw. die letzte (RETURN) Anweisung der Programmeinheit AUSGABE ausgeführt wird und wenn der Wert in SLF kleiner oder gleich 10 ist.

%DISASSEMBLE

Mit %DISASSEMBLE können Sie Speicherinhalte in symbolische Assembler-Notation rückübersetzen und ausgeben lassen.
Die Ausgabe erfolgt über SYSOUT, SYSLST oder in eine katalogisierte Datei (siehe Kommando %OUT).

– Mit *anzahl* legen Sie fest, wieviele Befehle rückübersetzt und ausgegeben werden sollen.

– Mit *start* bestimmen Sie die Adresse, bei der AID mit der Rückübersetzung beginnen soll.

Kommando	Operand
{ %DISASSEMBLE | %DA }	[anzahl] [FROM start]

Die Rückübersetzung des Speicherinhalts beginnt mit dem ersten Byte. Für Speicherinhalt, der nicht als Befehl interpretiert werden kann, wird eine Ausgabezeile erzeugt, die die sedezimale Darstellung des Speicherinhalts und den Hinweis INVALID OPCODE enthält. Die Suche nach gültigem Befehlscode geht dann in 2-Byte-Schritten vorwärts.

Mit einem %DISASSEMBLE ohne *start*-Operanden können Sie ein vorher gegebenes %DISASSEMBLE-Kommando solange fortsetzen, bis Sie mit einem BS2000- oder AID-Kommando (/LOAD-PROGRAM, /EXEC-PROGRAM, %BASE) das Testobjekt wechseln oder einen neuen Operandenwert vereinbaren. AID setzt die Rückübersetzung an der Speicheradresse fort, die an die Adresse anschließt, die mit dem vorhergehenden %DISASSEMBLE-Kommando zuletzt bearbeitet wurde. Ist auch *anzahl* nicht angegeben, so erzeugt AID dieselbe Anzahl von Ausgabezeilen wie bisher vereinbart.

Haben Sie in einer Testsitzung noch kein %DISASSEMBLE-Kommando gegeben, oder haben Sie das Testobjekt gewechselt und geben nun im %DISASSEMBLE-Kommando keine aktuellen Werte für einen oder beide Operanden, dann arbeitet AID mit Standardwerten 10 für *anzahl* und V'0' für *start*.

Mit %OUT können Sie steuern, wie die aufbereitete Speicherinformation dargestellt wird und auf welches Ausgabemedium sie übertragen werden soll. Der Aufbau der möglichen Ausgabezeilen ist im Anschluss an die Beschreibung des *start*-Operanden nachzulesen.

%DISASSEMBLE verändert den Programmzustand nicht.

anzahl

gibt an, wieviele Assembler-Befehle ausgegeben werden sollen.

Wird für *anzahl* kein Wert angegeben und ist auch aus einem vorherigen %DISASSEMBLE kein Wert gültig, so setzt AID den Standardwert 10 ein.

anzahl
ist eine Ganzzahl mit einem Wert:
$1 \leq anzahl \leq 2^{31}-1$

start

legt die Adresse fest, an der die Rückübersetzung von Speicherinhalt in Assembler-Befehle beginnen soll. Wird *start* nicht angegeben, setzt AID beim ersten %DISASSEMBLE den Standardwert V'0' ein; bei jedem weiteren %DISASSEMBLE wird hinter dem zuletzt rückübersetzten Assembler-Befehl fortgefahren.

```
start-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                 ⎧program-name      ⎫
                 ⎪L'n'              ⎪
FROM  [•][qua•] ⎨S'n'               ⎬
                 ⎪                  ⎪
                 ⎩kompl-speicherref ⎭

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

•
Steht der Punkt an führender Stelle, ist er das Kennzeichen für eine *vorqualifikation*. Sie muss mit einem vorhergehenden %QUALIFY-Kommando definiert worden sein. Aufeinanderfolgende Qualifikationen werden durch einen Punkt getrennt. Außerdem muss zwischen der letzten Qualifikation und dem anschließenden Operandenteil ein Punkt stehen.

qua
Eine Qualifikation geben Sie nur an, wenn *start* nicht im aktuellen AID-Arbeitsbereich liegt.

E={VM | Dn}
geben Sie nur an, wenn für *start* die aktuelle Basis-Qualifikation nicht gelten soll (siehe %BASE).

PROG=program-name
geben Sie nur an, wenn *start* nicht in der aktuellen Programmeinheit liegt (siehe Kapitel 3).

program-name
: Diese Angabe ist nur nach einer expliziten PROG-Qualifikation möglich:
`PROG=program-name.program-name`
Mit der Wiederholung von *program-name* legen Sie *start* auf die Anfangsadresse der bezeichneten Programmeinheit.

L'n'
: ist ein Anweisungsname und bezeichnet die Adresse der ersten ausführbaren FORTRAN-Anweisung nach einer Anweisungsmarke.
n ist eine maximal 5stellige Anweisungsmarke. Führende Nullen dürfen nicht angegeben werden.

S'n'
: ist eine Source-Referenz und bezeichnet die Adresse einer ausführbaren FORTRAN-Anweisung.
n ist die Nummer einer Anweisung; siehe Spalte STMT in der Übersetzungsliste.

kompl-speicherref
: sollte die Anfangsadresse eines Maschinenbefehls sein, andernfalls erhalten Sie eine unsinnige Disassemblierung. Folgende Operationen können darin vorkommen (siehe AID-Basishandbuch Kapitel 6):

  – Adressversatz (**.**)
  – indirekte Adressierung (->)
  – Typmodifikation (%A)
  – Längenmodifikation (%Ln)
  – Adressselektion (%@(...))

  Einen Anweisungsnamen *L'n'* oder eine Source-Referenz *S'n'* können Sie innerhalb von *kompl-speicherref* nur in Verbindung mit dem Pointer-Operator benutzen, z.B. L'n' ->.4
  Eine Typmodifikation ist nur sinnvoll, wenn der Inhalt eines Datenelements als Adresse eingesetzt werden kann oder wenn Sie die Adresse aus einem Register entnehmen, z.B. %3G.2 %AL2 ->

Ausgabe des %DISASSEMBLE-Protokolls

Das %DISASSEMBLE-Protokoll wird standardmäßig mit Zusatzinformationen über SYSOUT ausgegeben (T=MAX). Mit %OUT können Sie die Ausgabe-Medien wählen und festlegen, ob AID Zusatzinformationen ausgeben soll oder nicht.

AID berücksichtigt die Modi XMAX und XFLAT für die Ausgabe des %DISASSEMBLE-Protokolls nicht. Statt dessen generiert es die Standardausgabe (T=MAX).

Eine %DA-Ausgabezeile enthält folgende Elemente, wenn der Standardwert T=MAX gilt:

– CSECT-relative Speicheradresse,

– in symbolische Assembler-Notation rückübersetzter Speicherinhalt, wobei Distanzen im Gegensatz zum Assembler-Format als Sedezimalzahlen dargestellt werden,

– für Speicherinhalt, der nicht mit einem gültigen Operationscode beginnt, wird eine Assembler-Anweisung DC im Sedezimal-Format mit der Länge von 2 Bytes aufgebaut, der der Hinweis INVALID OPCODE folgt,

– sedezimale Darstellung des Speicherinhalts (Maschinencode).

Beispiel zum Zeilenaufbau mit T=MAX

Die Anweisungsnummer im %DISASSEMBLE-Kommando bezieht sich auf das Beispiel in Abschnitt 6.1.

```
/LOAD-PROG FROM-FILE=*MOD(LIB=*OMF),TEST-OPT=AID
 %  BLS0001 DLL VER 823
 %  BLS0517 MODULE 'B1' LOADED
/%DISASSEMBLE 10 FROM PROG=SORT.S'22'
 SORT+90          L     R15,1B0(R0,R13)            58 F0 D1B0
 SORT+94          A     R15,B0(R0,R12)             5A F0 C0B0
 SORT+98          ST    R15,1B0(R0,R13)            50 F0 D1B0
 SORT+9C          BC    B'1111',76(R0,R11)         47 F0 B076
 SORT+A0          DC   X'0000'  INVALID OPCODE     00 00
 SORT+A2          BCR   B'1100',R8                 07 C8
 SORT+A4          DC   X'0000'  INVALID OPCODE     00 00
 SORT+A6          ISK   R3,R8                      09 38
 SORT+A8          L     R15,1B4(R0,R13)            58 F0 D1B4
 SORT+AC          MH    R15,EE(R0,R12)             4C F0 C0EE
```

Mit dem %OUT-Operandenwert T=MIN baut AID verkürzte Ausgabezeilen auf, in denen die CSECT-relative Adresse durch die virtuelle Adresse ersetzt wird und die sedezimale Darstellung des Speicherinhalts entfällt.

Beispiel zum Zeilenaufbau mit T=MIN

```
/%OUT %DA T=MIN
/%DISASSEMBLE 10 FROM PROG=SORT.S'22'
 000005F8  L     R15,1B0(R0,R13)
 000005FC  A     R15,B0(R0,R12)
 00000600  ST    R15,1B0(R0,R13)
 00000604  BC    B'1111',76(R0,R11)
 00000608  DC   X'0000'  INVALID OPCODE
 0000060A  BCR   B'1100',R8
 0000060C  DC   X'0000'  INVALID OPCODE
 0000060E  ISK   R3,R8
 00000610  L     R15,1B4(R0,R13)
 00000614  MH    R15,EE(R0,R12)
```

Beispiele

1. `%DISASSEMBLE FROM PROG=BEISPIEL.L'22'`
   Das Kommando veranlasst die Rückübersetzung von 10 Befehlen (Standardwert) ab der Adresse der ersten ausführbaren Anweisung, die auf die Anweisungsmarke 22 in der Programmeinheit BEISPIEL folgt.

2. `%DA 2 FROM E=D1.PROG=BEISPIEL.BEISPIEL`
   Ab der Anfangsadresse der Programmeinheit BEISPIEL in der Dump-Datei mit dem Linknamen D1 sollen zwei Befehle disassembliert werden.

3. `%DA FROM S'67'`
   Da für *anzahl* kein Wert angegeben wurde, setzt AID entweder den Standardwert 10 ein, wenn es der erste %DISASSEMBLE für dieses Programm ist, oder übernimmt den Wert aus dem vorherigen %DISASSEMBLE.
   Die Rückübersetzung beginnt mit dem ersten Befehl, der zur Anweisung mit der Nummer 67 erzeugt wurde.

%DISPLAY

Mit %DISPLAY veranlassen Sie die Ausgabe von Speicherinhalten, Adressen, Längen, Systeminformationen und AID-Literalen, und Sie können damit den Vorschub nach SYSLST steuern. Daten bereitet AID entsprechend der Definition im Quellprogramm auf, wenn Sie nicht mit Typmodifikation einen anderen Ausgabetyp wählen.
Die Ausgabe erfolgt über SYSOUT, SYSLST oder in eine katalogisierte Datei.

– Mit *daten* bezeichnen Sie Datenelemente, deren Adressen oder Längen, Anweisungen, Register, Durchlaufzähler von Subkommandos und Systeminformationen, Sie definieren AID-Literale, oder Sie steuern den Vorschub nach SYSLST.

– Mit *medium-u-menge* geben Sie an, welche Ausgabe-Medien AID verwenden soll und ob Zusatzinformationen ausgegeben werden sollen. Mit diesem Operanden setzen Sie eine mit %OUT getroffene Vereinbarung für das aktuelle %DISPLAY-Kommando außer Kraft.

Kommando	Operand
%D[ISPLAY]	daten {,...} [medium-u-menge][,...]

Ohne Qualifikation zu *daten* sprechen Sie *daten* der aktuellen Programmeinheit an.
Mit einer Qualifikation können Sie *daten* in einer Dump-Datei oder in einer anderen geladenen Programmeinheit ansprechen, jedoch nur dann, wenn sich die entsprechende Programmeinheit in der aktuellen Aufrufhierarchie befindet.
Ohne den *medium-u-menge*-Operanden gibt AID die Daten entweder gemäß den Vereinbarungen im %OUT-Kommando oder standardmäßig mit Zusatzinformationen über SYSOUT aus (siehe AID-Basishandbuch, Kapitel 7).

Es empfiehlt sich, das Kommando nicht unmittelbar nach dem Laden einzugeben, da Sie Daten und Anweisungen erst dann ohne explizite Qualifikation ansprechen können, wenn das Programm vor der ersten ausführbaren Anweisung steht. Dies erreichen Sie mit der folgenden Kommandofolge:
%INSERT PROG=program-name.program-name
%RESUME

Mit %DISPLAY %SORTEDMAP erhalten Sie eine nach Namen und Adressen sortierte Liste aller CSECTs des Programms.

Neben den hier beschriebenen Operandenwerten können Sie auch die im Handbuch für das Testen auf Maschinencode-Ebene beschriebenen Operandenwerte einsetzen. Sie können mit diesem Kommando im geladenen Programm und in einer Dump-Datei arbeiten.

%DISPLAY verändert den Programmzustand nicht.

daten

beschreibt, welche Informationen AID ausgeben soll. Sie können sich Inhalt, Adresse und Länge von Variablen, Feldern oder Feldelementen, Inhalt und Länge von Konstanten und die Adresse von Anweisungen ausgeben lassen. Den Inhalt von Registern und Durchlaufzählern sowie für Ihr Programm relevante Systeminformationen können Sie über Schlüsselwörter adressieren. Um die Protokolle Ihrer Tests übersichtlicher zu gestalten, können Sie AID-Literale definieren oder den Vorschub nach SYSLST steuern.

Datenelemente bereitet AID entsprechend der Definition im Quellprogramm auf, wenn Sie nicht mit Typmodifikation einen anderen Ausgabetyp festlegen (siehe AID-Basishandbuch, Abschnitt 6.8). Passt der Inhalt nicht zum definierten Speichertyp, wird die Ausgabe mit einer Fehlermeldung abgelehnt. Sie können sich den Inhalt des Datenelements trotzdem ansehen, z.B. indem Sie sich durch die Typmodifikation %X den Inhalt sedezimal aufbereiten lassen.

Geben Sie in einem %DISPLAY mehrere *daten*-Operanden an, so können Sie von Operand zu Operand wechseln zwischen den hier beschriebenen symbolischen Angaben und den nicht-symbolischen, wie sie im Handbuch für das Testen auf Maschinencode-Ebene [2] beschrieben sind. Auch innerhalb einer komplexen Speicherreferenz können Sie symbolische und maschinennahe Angaben mischen.

Für Namen, die in den LSD-Sätzen nicht verzeichnet sind, gibt AID eine Fehlermeldung aus. Die anderen *daten* desselben Kommandos werden ordnungsgemäß bearbeitet.

```
daten-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

┌                ┌datenname          ┐          ┐
│                │L'n'               │          │
│ [•][qua•] {    │S'n'               }          │
│                │schlüsselwort      │          │
│                └kompl-speicherref  ┘          │
│                                               │
│ ┌%@┐                ┌datenname          ┐     │
{ └  ┘ ([•][qua•] {    │                   } }   │
│ └%L┘               └kompl-speicherref  ┘     │
│                                               │
│ %L=(ausdruck)                                 │
│                                               │
│ AID-literal                                   │
│                                               │
└ vorschubsteuerung                             ┘

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

- Steht der Punkt an führender Stelle, ist er das Kennzeichen für eine *vorqualifikation*. Sie muss mit einem vorhergehenden %QUALIFY definiert worden sein.

Aufeinanderfolgende Qualifikationen werden durch einen Punkt getrennt. Außerdem muss zwischen der letzten Qualifikation und dem anschließenden Operandenteil ein Punkt stehen.

qua

Eine Qualifikation geben Sie nur an für Speicherobjekte, die nicht im aktuellen AID-Arbeitsbereich liegen.

E={VM | Dn}

geben Sie nur an, wenn für einen Daten- oder Anweisungsnamen oder für eine Source-Referenz oder ein Schlüsselwort die aktuelle Basis-Qualifikation nicht gelten soll (siehe %BASE).

PROG=program-name

geben Sie nur an, wenn Sie einen Daten- oder Anweisungsnamen oder eine Source-Referenz ansprechen, die nicht in der aktuellen Programmeinheit liegen (siehe Kapitel 3).

NESTLEV= level-nummer

level-nummer Nummer einer Ebene in der aktuellen Aufrufhierarchie

Auf *level-nummer* muss *datenname* folgen.
Das %DISPLAY-Kommando gibt das Datenelement *datenname* aus, das auf der Ebene *level-nummer* der aktuellen Aufrufhierarchie definiert wurde.

datenname

ist der im Quellprogramm definierte Name einer Konstanten, einer Variablen, eines Feldes oder Feldelementes.

datenname ist eine maximal 15stellige alphanumerische Zeichenfolge.

Ist *datenname* der Name eines Feldes, dann können Sie ihn wie in einer FORTRAN-Anweisung indizieren, um ein Feldelement anzusprechen. Wenn Sie den Namen eines Felds ohne Indexliste angeben, werden alle Feldelemente ausgegeben.

feldname (index1[, index2][, ...])

index gibt die Position innerhalb eines Feldes an. Es sind so viele Indizes erforderlich, wie in einer FORTRAN-Anweisung zum Zugriff angegeben werden müssen. Mehrere Indizes müssen durch Komma getrennt werden.

index kann folgendermaßen angegeben werden:

```
{n                    }
{datenname            }
{arithmetischer ausdruck}
```

Die folgenden FORTRAN-Datendefinitionen werden anders ausgegeben:

– INTEGER*8 wie REAL*8

– REAL*16 wie REAL*8
– COMPLEX*32 wie COMPLEX*16

Ist *datenname* ein Datenelement vom Typ COMPLEX, dann wird der Real- und der Imaginärteil der komplexen Zahl ausgegeben. Sie können sich auch gezielt den Real- bzw. den Imaginärteil ausgeben lassen:

%D *datenname* ._REAL	gibt den Real-Teil aus.
%D *datenname* ._IMAG	gibt den Imaginär-Teil aus.

L'n'

ist ein Anweisungsname und bezeichnet die Adresse der ersten ausführbaren FORTRAN-Anweisung nach einer Anweisungsmarke.
n ist eine maximal 5stellige Anweisungsmarke. Führende Nullen dürfen nicht angegeben werden.
Geben Sie *L'n'* ohne Pointer-Operator an, so wird die entsprechende Adresse in sedezimaler Darstellung ausgegeben. Mit Pointer-Operator, also mit %DISPLAY L'n'->, gibt AID 4 Bytes des an der entsprechenden Adresse stehenden Maschinencodes aus.

S'n'

ist eine Source-Referenz und bezeichnet die Adresse einer ausführbaren FORTRAN-Anweisung.
n ist die Nummer einer Anweisung; siehe Spalte STMT der Übersetzungsliste.
Geben Sie *S'n'* ohne Pointer-Operator an, so wird die entsprechende Adresse in sedezimaler Darstellung ausgegeben. Mit Pointer-Operator, also mit %DISPLAY S'n'->, gibt AID 4 Bytes des an der entsprechenden Adresse stehenden Maschinencodes aus.

schlüsselwort

Sie können alle Schlüsselwörter für Programmregister, AID-Register und Systemtabellen sowie das Schlüsselwort für den Durchlaufzähler oder die symbolische Lokalisierungsinformation angeben (siehe AID-Basishandbuch, Kapitel 9).
Vor *schlüsselwort* können Sie nur eine Basis-Qualifikation angeben.

%n	Mehrzweckregister, $0 \le n \le 15$
%nD|E	Gleitpunktregister, n = 0,2,4,6
%nQ	Gleitpunktregister, n = 0,4
%nG	AID-Mehrzweckregister, $0 \le n \le 15$
%nDG	AID-Gleitpunktregister, n = 0,2,4,6
%nAR	Zugriffsregister, $0 \le n \le 15$
%MR	alle 16 Mehrzweckregister in Tabellenform
%FR	alle 4 Gleitpunkt-Register mit doppelter Genauigkeit in Tabellenform aufbereitet
%AR	alle 16 Zugriffsregister in Tabellenform
%PC	Befehlszähler (Program Counter)
%CC	Condition Code

%PCB	Process Control Block
%PCBLST	Liste aller Process Control Blocks
%SORTEDMAP	Liste aller CSECTs des Benutzerprogramms (namen- und adresssortiert)
%IFR	Interrupt Flag Register
%IMR	Interrupt Mask Register
%ISR	Interrupt Status Register
%PM	Program Mask
%AMODE	Adressierungsmodus (24- oder 31-Bit-Adressen)
%ASC	ASC-Modus (bzgl. AR-Modus bedeutet: X'00' = aus; X'01' = ein)
%AUD1	P1-Audit-Tabelle, wenn vorhanden auch SAVE-Tabelle
%LINK	Name des zuletzt nachgeladenen Segments (siehe %ON, Ereignis %LPOV)
%HLLOC(speicherref)	Lokalisierungsinformation auf symbolischer Ebene für eine Speicherreferenz im ausführbaren Teil des Programms (High-Level-Location)
%LOC(speicherref)	Lokalisierungsinformation auf Maschinencode-Ebene für eine Speicherreferenz im ausführbaren Teil des Programms (Low-Level-Location)
%DS[(ALET/SPID-qua)]	Information über SPIDs und/oder ALETs der aktiven Datenräume
%**.**subkdoname	Durchlaufzähler
%**.**	Durchlaufzähler des gerade aktiven Subkommandos

kompl-speicherref

Folgende Operationen können darin vorkommen (siehe AID-Basishandbuch, Kapitel 6):

- Adressversatz (**.**)
- indirekte Adressierung (->)
- Typmodifikation (%T(datenname), %X, %C, %E, %P, %D, %F, %A)
- Längenmodifikation (%L(...), %L=(ausdruck), %Ln)
- Adressselektion (%@(...))

Nach Adressversatz oder indirekter Adressierung gibt AID den Speicherinhalt an der errechneten Adresse standardmäßig im Dump-Format und der Länge 4 aus (%XL4). Mit der Typmodifikation können Sie sich *daten* in jeder gewünschten Aufbereitung ausgeben lassen, vorausgesetzt der Inhalt von *daten* passt zum angegebenen Speichertyp.

Mit %X können Sie sich jedes Datenelement stets in sedezimaler Darstellung ausgeben lassen, unabhängig davon, wie es im Quellprogramm definiert ist und welchen Inhalt es hat.

Mit der Längenmodifikation können Sie die Ausgabelänge selbst bestimmen, z.B. wenn Sie nur Teile eines Datenelements oder ein Datenelement in der Länge eines anderen Datenelements ausgeben lassen wollen.

%@(...)

Mit dem Adressselektor können Sie sich die Adresse eines Datenelements oder von *kompl-speicherref* ausgeben lassen.
Der Adressselektor lässt sich nicht auf symbolische Konstanten anwenden, dazu gehören auch die Anweisungsnamen *L'n'* und die Source-Referenzen *S'n'*.

%L(...)

Mit dem Längenselektor können Sie sich die Länge eines Datenelements ausgeben lassen.

Beispiel

```
%DISPLAY %L(AFELD)
```

Die Länge von AFELD wird ausgegeben.

%L=(ausdruck)

Mit der Längenfunktion können Sie sich einen Wert errechnen lassen (siehe AID-Basishandbuch, Abschnitte 6.9 und 6.10). In *ausdruck* können Sie den Inhalt von Speicherreferenzen und Konstanten vom Typ Integer und ganze Zahlen mit den arithmetischen Operatoren (+,-,*,/) verknüpfen.

Beispiel

```
%DISPLAY %L=(AFELD)
```

Wenn AFELD vom Typ Integer ist, wird der Inhalt von AFELD ausgegeben. Andernfalls gibt AID eine Fehlermeldung aus.

AID-literal

Alle im AID-Basishandbuch, Kapitel 8 beschriebenen AID-Literale können Sie angeben:

```
{C'x...x' | 'x...x'C | 'x...x'}Character-Literal
{X'f...f' | 'f...f'X}   Sedezimal-Literal
{B'b...b' | 'b...b'B}   Binär-Literal
[{±}]n                  Ganzzahl
#'f...f'                    Sedezimalzahl
[{±}]n.m                    Dezimalpunktzahl
[{±}]mantisseE[{±}]exponent Gleitpunktzahl
```

vorschubsteuerung

Für das Ausgabemedium SYSLST kann die Druckaufbereitung durch die folgenden beiden Schlüsselwörter gesteuert werden:

%NP	bewirkt einen Seitenvorschub
%NL[(n)]	bewirkt einen Zeilenvorschub um n Leerzeilen. $1 \leq n \leq 255$. Standardwert für n ist 1.

medium-u-menge

legt fest, über welches oder über welche Medien die Ausgabe erfolgen soll und ob AID Zusatzinformationen ausgeben soll. Ohne diesen Operanden und ohne eine Vereinbarung mit dem %OUT-Kommando arbeitet AID mit der Voreinstellung T = MAX.

medium-u-menge ist ausführlich im AID-Basishandbuch, Kapitel 7 beschrieben.

T Terminal-Ausgabe
H Hardcopy-Ausgabe
Fn Datei-Ausgabe
P Ausgabe nach SYSLST

MAX Ausgabe mit Zusatzinformationen.

MIN Ausgabe ohne Zusatzinformationen.

XMAX Für AID V3.4B gilt: Im Kommando %DISPLAY wird der Operandenwert XMAX nicht berücksichtigt, so dass das Verhalten identisch zum Standardwert MAX ist.

XFLAT Für AID V3.4B gilt: Im Kommando %DISPLAY wird der Operandenwert XFLAT nicht berücksichtigt, so dass das Verhalten identisch zum Standardwert MAX ist.

Beispiele

1. `%DISPLAY E=D1.PROG=BEISP.INTVAR,'DUMP-INHALT'`

   Es handelt sich hier um die Auswertung eines Dumps.

```
** D1: DUMP.BEISPIEL ************************************************************
INTVAR         =                      -89
DUMP-INHALT
```

2. `%DISPLAY %L=(S'13'-S'12')`

   AID gibt die Länge der Maschinencode-Sequenz aus, die für die Anweisung mit der Nummer 12 erzeugt wurde.

```
        +52
```

3. `%BASE`
   `%DISPLAY L'200'`

   Mit %BASE wird zurückgeschaltet auf den AID-Standard-Arbeitsbereich. Danach gibt AID die Adresse der ersten ausführbaren Anweisung nach der Anweisungsmarke 200 als Sedezimalzahl aus.

```
** ITN: #'00010053' *** TSN: 6567 ***********************************************
SRC_REF:    26 SOURCE: B1        PROC: B1  ************************************
200             = 0000051C
```

4. `%DISPLAY L'200'->`

   AID gibt 4 Bytes des an der Adresse der Anweisungsmarke 200 erzeugten Maschinencodes aus. Der Pointer-Operator bewirkt den Übergang zur Maschinencode-Ebene, sodass AID zusätzlich eine entsprechende Kopfzeile ausgibt.

```
CURRENT PC: 0000CEFA    CSECT: IF@STOP   ****************************************
V'0000051C' = B1       + #'0000051C'
0000051C (0000051C) 9500D17C                                n.J@
```

5. `%DISPLAY %HLLOC(L'200'->)`

   AID gibt die symbolische Lokalisierungsinformation zur Anweisungsmarke 200 aus.

```
V'0000051C' = SMOD     :  B1
              PROC     :  B1
              SRC-REF  :  82
              LABEL    :  200
```

6. `%DISPLAY %LOC(L'200'->)`

   AID gibt die Lokalisierungsinformation auf Maschinencode-Ebene zur Anweisungsmarke 200 aus.

```
V'0000051C' = PROG :  QSORT
              LMOD :  %ROOT
              SMOD :  B1
              OMOD :  B1
              CSECT :  B1       (00000000) + 0000051C
```

7. `%DISPLAY CHARFELD`

   Das Feld CHARFELD besteht aus 26 Feldelementen und ist im Programm folgendermaßen definiert:
   `CHARACTER CHARFELD (26)/'A','B','C','D',...,'X','Y','Z'/`
   Da im %DISPLAY keine Indexliste angegeben wurde, gibt AID alle Elemente des Feldes aus:

```
** ITN: #'00010053' *** TSN: 6567 ********************************************
SRC_REF:    66 SOURCE: BEISPIEL  PROC: BEISPIEL  ********************************
CHARFELD(1:26)
(  1) |A|  (  2) |B|  (  3) |C|  (  4) |D|  (  5) |E|  (  6) |F|  (  7) |G|
(  8) |H|  (  9) |I|  ( 10) |J|  ( 11) |K|  ( 12) |L|  ( 13) |M|  ( 14) |N|
( 15) |O|  ( 16) |P|  ( 17) |Q|  ( 18) |R|  ( 19) |S|  ( 20) |T|  ( 21) |U|
( 22) |V|  ( 23) |W|  ( 24) |X|  ( 25) |Y|  ( 26) |Z|
```

8. Das FORTRAN-Programm AUSGABE gibt alle Datentypen aus, die in FOR1 definiert werden können.

```
 DO/IF SEG  STMT  I/H LINE      SOURCE-TEXT
──────────────────────────────────────────────────────────────────────────────
       1/1     1         1 |    PROGRAMM AUSGABE
                         2 |  *
         1     2         3 |    INTEGER * 1       INT1    /-12/
         1     3         4 |    INTEGER * 2       INT2    /234/
         1     4         5 |    INTEGER * 4       INT4    /997/
         1     5         6 |    INTEGER * 8       INT8    /757/
                         7 |  *
         1     6         8 |    REAL    *   4      REAL4  /123.456/
         1     7         9 |    REAL    *   8      REAL8  /128.996/
         1     8        10 |    REAL    *  16      REAL16 /-987.772/
                        11 |  *
         1     9        12 |    COMPLEX *   8     CPLX8   /(2.5,4.7)/
         1    10        13 |    COMPLEX *  16     CPLX16  /(1.6,9.6)/
         1    11        14 |    COMPLEX *  32     CPLX32  /(3.7,8.9)/
                        15 |  *
         1    12        16 |    CHARACTER *   5       CHARC(3)   /'AAAAA','BBBBB','CCCCC'/
         1    13        17 |    CHARACTER * (45,V)    CHARV      /'44778'/
                        18 |  *
         1    14        19 |    LOGICAL * 1   LOG1     /.TRUE./
         1    15        20 |    LOGICAL * 4   LOG4     /.FALSE./
                        21 |  *
         1    16        22 |    CHARACTER CTEXT*30 /'ENTSPRECHENDE FORTRAN-AUSGABE:'/
                        23 |  *
5                       24 |  *
                        25 |  *
         1    17        26 |    WRITE(2,*) CTEXT
         1    18        27 |    WRITE(2,*) INT1
         1    19        28 |    WRITE(2,*) INT2
         1    20        29 |    WRITE(2,*) INT4
         1    21        30 |    WRITE(2,*) INT8
         1    22        31 |    WRITE(2,*)
                        32 |  *
         1    23        33 |    WRITE(2,*) CTEXT
         1    24        34 |    WRITE(2,*) REAL4
         1    25        35 |    WRITE(2,*) REAL8
         1    26        36 |    WRITE(2,*) REAL16
         1    27        37 |    WRITE(2,*)
                        38 |  *
         1    28        39 |    WRITE(2,*) CTEXT
         1    29        40 |    WRITE(2,*) CPLX8
         1    30        41 |    WRITE(2,*) CPLX16
         1    31        42 |    WRITE(2,*) CPLX32
         1    32        43 |    WRITE(2,*) REAL(CPLX8)
         1    33        44 |    WRITE(2,*) IMAG(CPLX8)
         1    34        45 |    WRITE(2,*)
                        46 |  *
         1    35        47 |    WRITE(2,*) CTEXT
         1    36        48 |    WRITE(2,*) CHARC
         1    37        49 |    WRITE(2,*) CHARV
         1    38        50 |    WRITE(2,*)
                        51 |  *
         1    39        52 |    WRITE(2,*) CTEXT
         1    40        53 |    WRITE(2,*) LOG1
         1    41        54 |    WRITE(2,*) LOG4
         1    42        55 |    WRITE(2,*)
                        56 |  *
         1    43        57 |    END
```

```
/START-FOR1-COMPILER SOURCE=Q.AUSGABE,OPTIMIZATION=NO,-
/TEST-SUPPORT=PARAMETER(TOOL-SUPPORT=AID),-
/LISTING=PARAMETER(OUTPUT=LF.AUSGABE)
%  BLS0500 PROGRAM 'FOR1', VERSION '2.1A00' OF '91-03-06' LOADED.
 FOR1:   V2.1A00 READY, GIVE COMPILER OPTION
 FOR1: LIST FILE REPLACED = LF.AUSGABE
 FOR1: NO ERRORS DURING COMPILATION OF P.U. AUSGABE
 END OF  F O R 1  COMPILATION;  CPU TIME USED: 1.675 SEC.
/SET-TASKLIB LIBRARY=$FOR1MODLIBS
/LOAD-PROG FROM-FILE=*MOD(LIB=*OMF),TEST-OPT=AID
%  BLS0001 DLL VER 823
%  BLS0517 MODULE 'AUSGABE' LOADED
/%INSERT S'17' <%DISPLAY C'AID-AUSGABE I*1, I*2, I*4, I*8',INT1, -
/INT2, INT4, INT8>
/%INSERT S'23' <%DISPLAY C'AID-AUSGABE R*4, R*8, R*16',REAL4,REAL8,REAL16>
/%INSERT S'28' <%DISPLAY C'AID-AUSGABE C*8, C*16, C*32, REAL(C*8), IMAG(C*8)',-
/CPLX8, CPLX16, CPLX32, CPLX8._REAL, CPLX8._IMAG>
/%INSERT S'35' <%DISPLAY C'AID-AUSGABE CHAR*L, CHAR*(MAXL,V)',CHARC, CHARV>
/%INSERT S'39' <%DISPLAY C'AID-AUSGABE LOG*1, LOG*4', LOG1, LOG4>
/%RESUME
 BS2000  F O R 1 : FORTRAN PROGRAM "AUSGABE"
 STARTED ON 91-06-28 AT 11:33:32
```

Programm AUSGABE wurde fehlerfrei übersetzt und mit LSD-Sätzen gebunden und geladen. Mit den %INSERTs wurden Testpunkte gesetzt, so dass jeweils auf ein %DISPLAY-Kommando die entsprechende FOR1-Ausgabe folgt. Zur besseren Lesbarkeit sind die Textzeilen "AID-AUSGABE" und "ENTSPRECHENDE FORTRAN-AUSGABE" fettgedruckt.

```
AID-AUSGABE I*1, I*2, I*4, I*8
** ITN: #'000000DF' *** TSN: 1627 ***********************************************
SRC_REF:    17 SOURCE: AUSGABE   PROC: AUSGABE  *********************************
INT1            =    -12
INT2            =         234
INT4            =         997
INT8            = +.7570000000000000 E+003
ENTSPRECHENDE FORTRAN-AUSGABE:
-12
234
997
757
```

Zunächst wurden alle Integer-Variablen ausgegeben. Im Unterschied zu FOR1 gibt AID Datenelemente vom Typ INTEGER*8 wie REAL*8-Datenelemente aus.

```
AID-AUSGABE R*4, R*8, R*16
SRC_REF:    23 SOURCE: AUSGABE    PROC: AUSGABE   ******************************
REAL4           = +.1234559 E+003
REAL8           = +.1289960021972656 E+003
REAL16          = -.9877719726562500 E+003
ENTSPRECHENDE FORTRAN-AUSGABE:
0.12345599E+03
0.12899600219726562 5E+03
-0.98777197265625000000000000000000E+03
```

Die Datenelemente vom Typ REAL*4, REAL*8 UND REAL*16 wurden ausgegeben. Unterschiedlich ist die Ausgabe von REAL*16-Variablen: AID gibt sie wie REAL*8-Variable aus.

```
AID-AUSGABE C*8, C*16, C*32, REAL(C*8), IMAG(C*8)
** ITN: #'000000DF' *** TSN: 1627 ***********************************************
SRC_REF:    28 SOURCE: AUSGABE   PROC: AUSGABE  *********************************
CPLX8
_REAL           = +.2500000 E+001
_IMAG           = +.4699999 E+001
CPLX16
_REAL           = +.1600000381469726 E+001
_IMAG           = +.9600000381469726 E+001
CPLX32
_REAL           = +.3699999809265136 E+001
_IMAG           = +.8899999618530273 E+001
CPLX8._REAL     = +.2500000 E+001
CPLX8._IMAG     = +.4699999 E+001
ENTSPRECHENDE FORTRAN-AUSGABE:
(0.2500000E+01,0.46999998E+01)
(0.16000003814697265E+01,0.96000003814697265E+01)
(0.36999998092651367187500000000000E+01,0.88999996185302734375000000000000E+01)
0.25000000E+01
0.46999998E+01
```

Komplexe Zahlen gibt AID stets nach Real- und Imaginärteil getrennt aus.
Der geklammerten Darstellung von komplexen Zahlen in FOR1 entspricht bei AID die nach Real- und Imaginärteil getrennte Ausgabe.
Wie in FOR1 können auch in AID Realteil oder Imaginärteil gesondert angesprochen werden. Mit *datenname*._REAL bezeichnen Sie den Realteil und mit *datenname*._IMAG den Imaginärteil der komplexen Variablen.

```
AID-AUSGABE CHAR*L, CHAR*(MAXL,V)
SRC_REF:    35 SOURCE: AUSGABE   PROC: AUSABE  **********************************
CHARC( 1: 3)
( 1) |AAAAA|   ( 2) |BBBBB|   ( 3) |CCCCC|
CHARV           = |44778|
ENTSPRECHENDE FORTRAN-AUSGABE:
AAAAABBBBBCCCCC
44778

AID-AUSGABE LOG*1, LOG*4
SRC_REF:    39 SOURCE: AUSGABE   PROC: AUSGABE  **********************************
LOG1            = %TRUE
LOG4            = %FALSE
ENTSPRECHENDE FORTRAN-AUSGABE:
T
F
```

Auch Felder vom Typ Character werden von AID nach Feldelementen gegliedert ausgegeben.
Die Aufbereitung logischer Variablen bei FOR1 und bei AID schließt die Gegenüberstellung der verschiedenen Datentypen ab.

%DUMPFILE

Mit %DUMPFILE weisen Sie einem der Linknamen eine Dump-Datei zu und veranlassen AID, diese Datei zu öffnen oder zu schließen.

– Mit *link* wählen Sie den Linknamen für die Dump-Datei aus, die geöffnet oder geschlossen werden soll.

– Mit *datei* bezeichnen Sie die Dump-Datei, die geöffnet werden soll.

Kommando	Operand
{%DUMPFILE | %DF}	[link [=datei]]

Ohne den *datei*-Operanden veranlassen Sie AID, die Datei zu schließen, die dem angegebenen Linknamen zugewiesen ist.

Mit einem %DUMPFILE ohne Operanden veranlassen Sie AID, alle offenen Dump-Dateien zu schließen. Lag bis dahin der AID-Arbeitsbereich in der nun geschlossenen Dump-Datei, gilt anschließend wieder der AID-Standard-Arbeitsbereich (siehe %BASE).

%DUMPFILE darf nur als Einzelkommando eingegeben werden, es darf nicht in einer Kommandofolge oder in einem Subkommando stehen.

%DUMPFILE verändert den Programmzustand nicht.

link

bezeichnet einen der AID-Linknamen für Dump-Dateien und hat das Format Dn, wobei n eine Zahl ist mit einem Wert $0 \leq n \leq 7$.

datei

gibt den vollqualifizierten Dateinamen an, unter dem die Dump-Datei katalogisiert ist, die AID öffnen soll.
Ohne diesen Operanden wird die Dump-Datei mit dem Linknamen *link* geschlossen.
Eine offene Dump-Datei muss erst mit einem eigenen %DUMPFILE geschlossen worden sein, bevor eine andere demselben Linknamen zugewiesen werden kann.

Beispiele

1. `%DUMPFILED3=DUMP.1234.00001`
   Die Datei DUMP.1234.00001 mit dem Linknamen D3 wird geöffnet.

2. `%DF D3`
   Die Datei, die dem Linknamen D3 zugewiesen ist, wird geschlossen.

3. `%DF`
   Alle offenen Dump-Dateien werden geschlossen.

%FIND

Mit %FIND können Sie ein Literal in einem Datenelement oder im ausführbaren Teil eines Programms suchen und Treffer auf Terminal (SYSOUT) ausgeben lassen. Außerdem werden in den AID-Registern %0G und %1G Trefferadresse und Fortsetzungsadresse abgelegt. Mit %FIND können Sie sowohl im virtuellen Speicher als auch in einer Dump-Datei suchen.

– *suchbegriff* ist ein Character- oder Sedezimal-Literal, das gesucht werden soll.

– Mit *find-bereich* geben Sie an, in welchem Datenelement oder in welchem Abschnitt des ausführbaren Teils des Programms AID *suchbegriff* suchen soll. Ohne Angabe von *find-bereich* durchsucht AID den gesamten Speicherbereich zur aktuell eingestellten Basis-Qualifikation (siehe %BASE).

– Mit *alignment* geben Sie an, ob *suchbegriff* an Doppelwort-, Wort-, Halbwort- oder Byte-Grenze gesucht werden soll. Ohne Angabe von *alignment* wird an Byte-Grenze gesucht.

– Mit *ALL* geben Sie an, dass die Suche nicht nach der Ausgabe des ersten Treffers abgebrochen werden soll, sondern dass der gesamte *find-bereich* durchsucht und alle Treffer ausgegeben werden sollen. Die Suche kann dann nur mit der K2-Taste abgebrochen werden.

Kommando	Operanden
%F[IND]	[[ALL] suchbegriff [IN find-bereich] [alignment]]

Ein %FIND ohne *ALL*-Operanden können Sie hinter der Adresse des letzten Treffers fortsetzen, bis das Ende von *find-bereich* erreicht ist, indem Sie ein neues %FIND-Kommando ohne eigene Operandenwerte eingeben.

Ein %FIND mit eigenem *suchbegriff* und ohne weitere Operanden übernimmt Vereinbarungen für *find-bereich* und *alignment* aus einem vorhergehenden %FIND. Gibt es keinen vorhergehenden %FIND, setzt AID die Standardwerte ein.

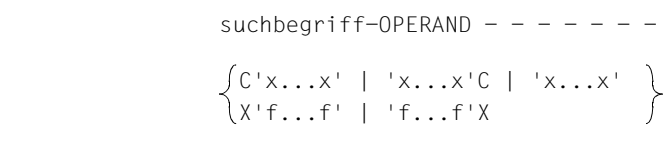
Die Ausgabe von Treffern erfolgt immer im Ausgabetyp DUMP (Sedezimal- und Character-Darstellung) in der Länge von 12 Bytes auf das Medium Terminal (SYSOUT). Zusätzlich zum Treffer wird seine Adresse und (soweit möglich) der Name der Programmeinheit, in der der Treffer gefunden wurde, und die relative Adresse des Treffers zum Anfang der Programmeinheit ausgegeben.

Im Trefferfall wird die Trefferadresse im AID-Register %0G und die Fortsetzungsadresse (Trefferadresse + Suchstringlänge) im AID-Register %1G abgespeichert. Bei der Angabe von *ALL* wird die Adresse des letzten Treffers in %0G und die Fortsetzungadresse des letzten Treffers in %1G abgespeichert. Falls *suchbegriff* nicht gefunden wurde, setzt AID %0G auf –1; %1G bleibt unverändert.
Die beiden Registerinhalte ermöglichen es Ihnen, das %FIND-Kommando auch in Prozeduren oder Subkommandos einzusetzen und mit den Ergebnissen weiterzuarbeiten.

%FIND verändert den Programmzustand nicht.

suchbegriff

ist ein Character- oder Sedezimal-Literal. *suchbegriff* kann Wildcard-Symbole enthalten. Diese Symbole sind immer Treffer. Sie werden durch '%' dargestellt.



`{C'x...x' | 'x...x'C | 'x...x'}`

Character-Literal
mit einer maximalen Länge von 80 Zeichen. Kleinbuchstaben können nur nach Eingabe von %AID LOW [=ON] als Character-Literal gesucht werden.

x kann jedes darstellbare Zeichen annehmen, insbesondere das Wildcard-Symbol '%', welches immer einen Treffer darstellt. Das Zeichen '%' selbst kann in dieser Form nicht gesucht werden, da es als C'%' in einem Character-Literal stets zu einem Treffer führt. Es muss deshalb als Sedezimal-Literal X'6C' gesucht werden.

Bitte beachten Sie, dass der CCS von *find-bereich* mit dem CCS des Eingabemediums (SYSCMD) übereinstimmen muss, damit die Character-Literale gefunden werden können. Legen Sie daher den CCS von *find-bereich* fest, bevor Sie in *find-bereich* nach einem Character-Literal suchen:

`%AID CCS=` *CCS-name*

Eine komplette Liste der von XHCS unterstützten CCS-Namen und den aktuellen CCS von SYSCMD können Sie mit dem folgenden AID-Kommando ausgeben:

```
%SHOW %CCSN
```

Den CCS von SYSCMD können Sie mit dem folgenden SDF-Kommando ändern:

`MODIFY-TERMINAL-OPTION CODED-CHARACTER-SET= {`*EBCDIC-CCS-name* | *UTFE*`}`

Den aktuellen CCS von *find-bereich* können Sie mit dem folgenden AID-Kommando ausgeben:

```
%SHOW %AID
```

Beachten Sie bitte, dass das %DISPLAY-Kommando seit der AID-Version V3.4B11 als Voreinstellung den CCS-Wert von `%AID` verwendet, wenn kein CCS-Wert angegeben wurde.

```
%D char-data ['CCS-name']
```

Siehe Basishandbuch, Abschnitt „Character-Literal“ [1]) für ein Beispiel zur Suche nach Character-Literalen in unterschiedlichen Coded Character Sets.

`{X'f...f' | 'f...f'X}`

Sedezimal-Literal
mit einer maximalen Länge von 80 Sedezimal-Stellen bzw. 40 Zeichen. Ein Literal mit ungerader Stellenzahl wird rechts mit X’0’ ergänzt.

f kann jeden Wert zwischen 0 und F sowie das Wildcard-Symbol X’%’ annehmen. Das Wildcard-Symbol stellt für jede Sedezimal-Stelle zwischen 0 und F einen Treffer dar.

find-bereich

legt einen Speicherbereich fest, in dem *suchbegriff* gesucht werden soll. *find-bereich* kann ein Datenelement oder ein Abschnitt des ausführbaren Teils des geladenen Programms oder einer Dump-Datei sein. *find-bereich* darf nicht länger als 65 535 Bytes sein.

Ist kein *find-bereich* angegeben, so setzt AID den Standardwert %CLASS6 ein (siehe AID-Basishandbuch), d.h. %FIND beginnt die Suche bei Adresse V'0'.

```
find-bereich-OPERAND  - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                     ⎧datenname          ⎫
                     ⎪L'n'->             ⎪
IN [•][qua•] [(]     ⎨S'n'->             ⎬[)]
                     ⎪                   ⎪
                     ⎩kompl-speicherref  ⎭

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

•
: Steht der Punkt an führender Stelle, ist er das Kennzeichen für eine *vorqualifikation*. Sie muss mit einem vorhergehenden %QUALIFY definiert worden sein.
Aufeinanderfolgende Qualifikationen werden durch einen Punkt getrennt. Außerdem muss zwischen der letzten Qualifikation und dem anschließenden Operandenteil ein Punkt stehen.

qua
: Eine Qualifikation geben Sie nur an, wenn *find-bereich* nicht im aktuellen AID-Arbeitsbereich liegt.

  E={VM | Dn}
  : geben Sie nur an, wenn für *find-bereich* die aktuelle Basis-Qualifikation nicht gelten soll (siehe %BASE).

  PROG=program-name
  : geben Sie nur an, wenn *find-bereich* nicht in der aktuellen Programmeinheit liegt (siehe Kapitel 3).

datenname
: ist der im Quellprogramm definierte Name einer Variablen, eines Felds oder Feldelements.
datenname ist eine maximal 15stellige alphanumerische Zeichenfolge.

  Ist *datenname* der Name eines Felds, dann können Sie ihn wie in einer FORTRAN-Anweisung indizieren, um ein Feldelement anzusprechen. Wenn Sie den Namen eines Felds ohne Indexliste angeben, wird *such-begriff* im gesamten Feld gesucht.

feldname (index1[, index2][, ...])

index gibt die Position innerhalb eines Felds an. Es sind so viele Indizes erforderlich, wie in einer FORTRAN-Anweisung zum Zugriff angegeben werden müssen. Mehrere Indizes müssen durch Komma getrennt werden.

index kann folgendermaßen angegeben werden:

```
{n                      }
{datenname              }
{arithmetischer ausdruck}
```

L'n'->

bezeichnet die Speicherstelle an der Adresse der ersten ausführbaren FORTRAN-Anweisung nach einer Anweisungsmarke.
n ist eine maximal 5stellige Anweisungsmarke. Führende Nullen dürfen nicht angegeben werden.

Ohne Längenmodifikation werden 4 Bytes ab der Adresse durchsucht, die in der Adresskonstanten *L'n'* hinterlegt ist.

S'n'->

bezeichnet die Speicherstelle an der Adresse der FORTRAN-Anweisung mit der angegebenen Nummer.
n ist die Nummer einer Anweisung; siehe Spalte STMT der Übersetzungsliste.

Ohne Längenmodifikation werden 4 Bytes ab der Adresse durchsucht, die in der Adresskonstanten *S'n'* hinterlegt ist.

kompl-speicherref

bezeichnet einen Bereich von 4 Bytes ab der errechneten Adresse. Soll eine andere Anzahl von Bytes durchsucht werden, muss *kompl-speicherref* mit der entsprechenden Längenmodifikation enden. Bei der Längenmodifikation von Datenelementen müssen Sie die Bereichsgrenzen beachten oder mit %@(datenname)-> auf Maschinencode-Ebene wechseln. Folgende Operationen können in *kompl-speicherref* vorkommen (siehe AID-Basishandbuch, Kapitel 6):

- Adressversatz (**.**)
- indirekte Adressierung (->)
- Typmodifikation (%A)
- Längenmodifikation (%L(...), %L=(ausdruck), %Ln)
- Adressselektion (%@(...))

alignment

legt fest, dass *suchbegriff* nur an bestimmten ausgerichteten Adressen gesucht werden soll.

```
alignment-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
                  ⎧1⎫
                  ⎪2⎪
ALIGN [=]         ⎨ ⎬
                  ⎪4⎪
                  ⎩8⎭
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

suchbegriff wird gesucht an:

1 Byte-Grenze (Standardwert)
2 Halbwort-Grenze
4 Wort-Grenze
8 Doppelwort-Grenze

Beispiele

1.
```
%AID LOW
%FIND 'r' IN CFELD; %F
```

   Mit %AID LOW wird die Unterscheidung der Klein-/Großschreibung eingeschaltet. Das erste %FIND-Kommando sucht ein kleines ’r’ im Feld CFELD, das im Common CB definiert ist. Mit dem anschließenden %FIND ohne Operanden wird die Suche fortgesetzt und das zweite ’r’ in CFELD gefunden. Die Treffer werden am Bildschirm ausgegeben.

```
CB      +00000002=0000B002 : 99A381C8 8195A289 C195A396  rtaHansiAnto
CB      +00000010=0000B010 : 99898388 C9939695 81C48981  richIlonaDia
```

2.
```
%FIND ALL 'AB' IN PROG=BSP.CVAR
```

   Das Character-Literal ’AB’ wird in der Variablen CVAR gesucht, die in der Programmeinheit BSP definiert ist. Alle Treffer werden am Bildschirm ausgegeben.

```
BSP@@@@@+00000B66=00001516 : C1C2C3C4 C1C2C3C4 C1C2C3C4  ABCDABCDABCD
BSP@@@@@+00000B6A=0000151A : C1C2C3C4 C1C2C3C4 C1C2C3C4  ABCDABCDABCD
BSP@@@@@+00000B6E=0000151E : C1C2C3C4 C1C2C3C4 00F000F0  ABCDABCD.0.0
BSP@@@@@+00000B72=00001522 : C1C2C3C4 00F000F0 000F00F0  ABCD.0.0.0.0
```

3. `%F X'D2' IN PROG=BSP.S'56'->%L=(S'57'-S'56') ALIGN=2;%F`

   Das Sedezimal-Literal X’D2’ wird im Maschinencode, der für die Anweisung S’56’ erzeugt wurde, gesucht. Der folgende %FIND ohne Operandenwerte sucht das nächste X’D2’. Wegen der Angabe ALIGN=2 sucht AID nur an Halbwortgrenzen.

```
BSP      +00000208=00000208 : D2245AF0 C0E050F0 D22446A0  K.!0..&0K...
BSP      +00000210=00000210 : D22446A0 B1B4  K.....
```

%HELP

Mit %HELP können Sie sich über die Bedienung von AID informieren. Auf das gewählte Medium werden ausgegeben: entweder alle AID-Kommandos oder das gewählte Komando und seine Operanden oder die gewählte Fehlermeldung mit Bedeutung und möglichen Maßnahmen.

– Mit *info-ziel* geben Sie das Kommando an, über das Sie weitere Angaben brauchen oder die AID-Meldung, zu der Sie Bedeutung und Maßnahmen nachlesen wollen.

– Mit *medium-u-menge* geben Sie an, über welche Ausgabe-Medien AID die angeforderten Informationen ausgeben soll. Mit diesem Operanden setzen Sie vorübergehend eine mit %OUT getroffene Vereinbarung außer Kraft.

Kommando	Operand	
%H[ELP]	[info-ziel]	[medium-u-menge][,...]

%HELP informiert Sie über alle Operanden des gewählten Kommandos, d.h. sowohl über alle sprachspezifischen Operanden für das symbolische Testen, als auch über alle Operanden für das maschinennahe Testen. Was für die Sprache, in der Ihr Programm geschrieben wurde, erlaubt ist, können Sie dem jeweiligen sprachspezifischen Handbuch entnehmen.

Die Meldungen von AIDSYS haben im Meldungsschlüssel den Aufbau IDA0n und werden mit /HELP abgefragt.

%HELP darf nur als Einzelkommando eingegeben werden, es darf nicht in einer Kommandofolge oder in einem Subkommando stehen.

%HELP verändert den Programmzustand nicht.

info-ziel

bezeichnet ein Kommando oder eine Meldungsnummer, über die Informationen ausgegeben werden sollen.
Ohne den *info-ziel*-Operanden bewirkt das Kommando die Ausgabe einer Übersicht über die AID-Kommandos mit einer Kommando-Kurzbeschreibung und über den AID-Meldungsnummernkreis.

Ein %HELP-Kommando mit falschem *info-ziel*-Operanden beantwortet AID mit einer Fehlermeldung. Daran schließt sich die vorher beschriebene Übersicht an. Diese Übersicht erhalten Sie auch, wenn Sie %?, %H? oder %H %? angeben.

```
info-ziel-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

( %AID | %AINT | %BASE | %CONT[INUE] | %C[ONTROL]          )
( %DISASSEMBLE | %DA | %D[ISPLAY] | %DUMPFILE | %DF        )
( %F[IND] | %H[ELP] | %IN[SERT] | %JUMP | %M[OVE]          )
( %ON | %OUT | %OUTFILE | %Q[UALIFY]                       )
{ %REM[OVE] | %R[ESUME] | %SD[UMP]                         }
( %S[ET] | %STOP | %SYMLIB | %TITLE | %T[RACE]             )
(                                                          )
( In                                                       )

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Die AID-Kommandonamen können auch in der zulässigen Abkürzung angegeben werden.

In bezeichnet die Meldungsnummer, zu der Bedeutung und mögliche Maßnahmen ausgegeben werden sollen.
n ist die dreistellige Meldungsnummer.

medium-u-menge

legt fest, über welche Medien die Informationen zu *info-ziel* ausgegeben werden sollen. Die Angabe {MAX | MIN | XMAX | XFLAT} hat bei %HELP keine Auswirkungen, eine der Angaben ist aber syntaktisch erforderlich.

Ohne diesen Operanden und ohne eine Vereinbarung mit dem %OUT-Kommando arbeitet AID mit dem Standardwert T=MAX.

```
medium-u-menge-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - -

( I  )   ( MIN  )
( H  )   ( MAX  )
{ Fn } = { XMAX }
( P  )   ( XFLAT )

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

medium-u-menge ist ausführlich im AID-Basishandbuch, Kapitel 7 beschrieben.

T Terminal-Ausgabe
H Hardcopy-Ausgabe
Fn Datei-Ausgabe
P Ausgabe nach SYSLST

%INSERT

Mit %INSERT legen Sie einen Testpunkt fest und definieren ein Subkommando. Wenn der Programmablauf den Testpunkt erreicht, bearbeitet AID das zugehörige Subkommando. Zusätzlich können Sie angeben, ob AID nach einer angegebenen Anzahl von Durchläufen den Testpunkt löschen und das Programm danach anhalten soll.

– Mit *testpunkt* bezeichnen Sie die Adresse eines Befehls im Programm, vor deren Ausführung AID den Programmablauf unterbrechen soll, um *subkdo* zu bearbeiten.

– Mit *subkdo* vereinbaren Sie ein Kommando oder eine Kommandofolge und eventuell eine Bedingung. Wird *testpunkt* erreicht und ist die Bedingung erfüllt, wird *subkdo* ausgeführt.

– Mit *steuerung* vereinbaren Sie, ob *testpunkt* nach einer vorgegebenen Anzahl von Durchläufen gelöscht und ob danach das Programm angehalten werden soll.

Kommando	Operand
%IN[SERT]	testpunkt [<subkdo>] [steuerung]

Ein *testpunkt* wird in folgenden Fällen gelöscht:

1. Das Programmende wird erreicht.
2. Die mit *steuerung* vorgegebene Durchlauf-Anzahl wird erreicht, und das Löschen von *testpunkt* ist vereinbart.
3. Der *testpunkt* wird mit %REMOVE gelöscht.

Ohne *subkdo*-Operanden setzt AID das *subkdo* <%STOP> ein.

Das *subkdo* eines %INSERT für einen bereits gesetzten *testpunkt* überschreibt nicht das bestehende *subkdo*, sondern das neue *subkdo* wird vor das bestehende gekettet.
Die geketteten Subkommandos werden somit nach dem LIFO-Prinzip abgearbeitet.

Mit %REMOVE löschen Sie ein Subkommando, einen Testpunkt oder alle eingetragenen Testpunkte.

testpunkt kann nur eine Adresse im geladenen Programm sein, deshalb muss die Basis-Qualifikation E=VM eingestellt sein (siehe %BASE) oder explizit angegeben werden.

%INSERT verändert den Programmzustand nicht.

testpunkt

muss die Adresse eines ausführbaren Maschinenbefehls sein, der für eine FORTRAN-Anweisung erzeugt wurde. *testpunkt* wird sofort durch das gezielte Überschreiben der adressierten Speicherstelle eingetragen und muss deshalb zum Zeitpunkt der %INSERT-Eingabe bzw. bei der Abarbeitung des Subkommandos, in dem der %INSERT enthalten ist, im virtuellen Speicher geladen sein. Da durch das Eintragen von *testpunkt* der Code des Programms verändert wird, führt ein falsch gesetzter Testpunkt zu Fehlern im Programmablauf (z.B. Daten- oder Adressierungsfehler).

Kommt der Programmablauf an den *testpunkt*, unterbricht AID das Programm und startet *subkdo*.

```
testpunkt-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                  ⎧program-name      ⎫
                  ⎪L'n'              ⎪
 [.][qua.] [.]    ⎨S'n'              ⎬
                  ⎪                  ⎪
                  ⎩kompl-speicherref ⎭

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

.
: Steht der Punkt an führender Stelle, ist er das Kennzeichen für eine *vorqualifikation*. Sie muss mit einem vorhergehenden QUALIFY-Kommando definiert worden sein. Aufeinanderfolgende Qualifikationen werden durch einen Punkt getrennt. Außerdem muss zwischen der letzten Qualifikation und dem anschließenden Operandenteil ein Punkt stehen.

qua
: Eine Qualifikation gegen Sie nur an, wenn *testpunkt* nicht im aktuellen AID-Arbeitsbereich liegt.

  E=VM
  : Da *testpunkt* nur im virtuellen Speicher des geladenen Programms eingetragen werden kann, geben Sie *E=VM* nur an, wenn als aktuelle Basis-Qualifikation eine Dump-Datei vereinbart ist (siehe %BASE).

  PROG=program-name
  : geben Sie nur an, wenn *testpunkt* nicht in der aktuellen Programmeinheit liegt (siehe Kapitel 3).

program-name

Diese Angabe ist nur nach einer expliziten PROG-Qualifikation möglich:

`PROG=program–name•program–name`

Mit der Wiederholung von *program-name* setzen Sie *testpunkt* auf die erste Anweisung der bezeichneten Programmeinheit.

L'n'

ist ein Anweisungsnname und bezeichnet die Adresse der ersten ausführbaren FORTRAN-Anweisung nach einer Anweisungsmarke, d.h. die Adresse des ersten Maschinenbefehls der Code-Sequenz, die für diese Anweisung erzeugt wurde.
n ist eine maximal 5stellige Anweisungsmarke. Führende Nullen dürfen nicht angegeben werden.

S'n'

ist eine Source-Referenz und bezeichnet die Adresse einer ausführbaren FORTRAN-Anweisung, d.h. die Adresse des ersten Maschinenbefehls der Code-Sequenz, die für diese Anweisung erzeugt wurde.
n ist die Nummer einer Anweisung; siehe Spalte STMT der Übersetzungsliste.

kompl-speicherref

Das Ergebnis von *kompl-speicherref* muss die Anfangsadresse eines ausführbaren Maschinenbefehls sein. *kompl-speicherref* kann folgende Operationen enthalten (siehe AID-Basishandbuch, Kapitel 6):

- Adressversatz (**.**)
- indirekte Adressierung (->)
- Typmodifikation (%A)
- Längenmodifikation (%Ln)
- Adressselektion (%@(...))

Einen Anweisungsnamen *L'n'* oder eine Source-Referenz *S'n'* können Sie innerhalb von *kompl-speicherref* nur in Verbindung mit dem Pointer-Operator benutzen, z.B. L'200' ->.4.
Eine Typmodifikation ist nur sinnvoll, wenn der Inhalt eines Datenelements als Adresse eingesetzt werden kann oder wenn Sie die Adresse aus einem Register entnehmen, z.B. %3G.2 %AL2 ->.

subkdo

wird immer dann bearbeitet, wenn der Programmablauf an die mit *testpunkt* bezeichnete Adresse kommt.
Wird der *subkdo*-Operand nicht angegeben, so setzt AID ein <%STOP> ein.

Vollständig beschrieben finden Sie *subkdo* im Basishandbuch, Kapitel 5.

```
subkdo-OPERAND  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                                        ⎧AID-kommando    ⎫
<[subkdoname:] [(bedingung):] [         ⎨                ⎬ {;...}]>
                                        ⎩BS2000-kommando ⎭

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Das Subkommando kann einen Namen, eine Bedingung und einen Kommandoteil enthalten. Zu jedem Subkommando gehört ein Durchlaufzähler. Der Kommandoteil kann aus einem einzelnen Kommando oder einer Kommandofolge bestehen, er kann AID- und BS2000-Kommandos und Kommentare enthalten.

Wenn das Subkommando aus einem Namen oder einer Bedingung besteht, aber der Kommandoteil fehlt, erhöht AID beim Erreichen des Testpunkts nur den Durchlaufzähler.

subkdo überschreibt nicht ein bestehendes Subkommando zu demselben *testpunkt*, sondern das neue Subkommando wird vor das bestehende gekettet. In *subkdo* sind die Kommandos %CONTROLn, %INSERT, %JUMP und %ON zugelassen. Sie können damit eine Schachtelung über maximal 5 Stufen vornehmen.

Die Kommandos in einem *subkdo* werden nacheinander ausgeführt. Danach wird das Programm fortgesetzt. Die Kommandos zur Ablaufsteuerung verändern auch in einem Subkommando sofort den Programmzustand. Sie brechen *subkdo* ab und starten das Programm (%CONTINUE, %RESUME, %TRACE) oder halten es an (%STOP). Sie sind nur als letztes Kommando in einem *subkdo* sinnvoll, da nachfolgende *subkdo*-Kommandos nicht mehr ausgeführt werden. Auch ein Löschen des gerade aktiven Subkommandos mit %REMOVE ist nur als letztes Kommando in *subkdo* sinnvoll.

steuerung

gibt an, ob *testpunkt* nach dem n-ten Durchlauf gelöscht werden soll, und ob das Programm angehalten werden soll, damit neue Kommandos eingegeben werden können.
Wird der *steuerung*-Operand nicht angegeben, setzt AID die Standardwerte 2^{31}-1 (für *n*) und K ein.

```
steuerung-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

ONLY  n [{ K }]
        { S }
        { C }

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

n ist eine Zahl mit dem Wert $1 \leq n \leq 2^{31}$-1.
Sie gibt an, beim wievielten Durchlaufen von *testpunkt* die weiteren Vereinbarungen dieses *steuerung*-Operanden ausgeführt werden sollen.

K *testpunkt* wird nicht gelöscht (KEEP).
Der Programmablauf wird unterbrochen, und AID erwartet die Eingabe von Kommandos.

S *testpunkt* wird gelöscht (STOP).
Der Programmablauf wird unterbrochen, und AID erwartet die Eingabe von Kommandos.

C *testpunkt* wird gelöscht (CONTINUE).
Keine Unterbrechung des Programms.

Beispiele

1. `%IN S'48'`
   Als *testpunkt* wird die Anweisung mit der Nummer 48 angegeben.

2. `%IN L'0' <%DISPLAY X>`
   Als *testpunkt* wird die Anweisung mit der Anweisungsmarke 0 angegeben.

3. `%IN S'3' <%DISPLAY PERSNR> ONLY 10 S`
   Als *testpunkt* wird die Anweisung mit der Nummer 3 angegeben. D.h. immer wenn der Programmablauf zur dritten Anweisung kommt, wird der %DISPLAY aus *subkdo* ausgeführt. Wird *testpunkt* zum 10. Mal erreicht, versetzt AID das Programm in den Zustand STOP, löscht den Testpunkt, und Sie können neue Kommandos eingeben.

4. ```
   %IN L'2' <%DISPLAY TEXTDAT, 'L2'>
   %IN S'3' <%DISPLAY 'INSERT1', TEXTDAT; %IN L'20' <%D 'INSERT2', -
   I,J,K, ANZAHL; %IN S'172' <%D 'INSERT3' ,I,J; %REMOVE L'20'>>>
   ```

   Mit dem ersten %INSERT wird als *testpunkt* die Anweisung mit der Anweisungsmarke 2 festgelegt. Kommt nach der Beendigung der Kommandoeingabe der Programmablauf zu L'2', wird das Subkommando ausgeführt. Es besteht aus einem %DISPLAY für den Datennamen TEXTDAT und dem Literal 'L2'. Anschließend läuft das Programm weiter.

   Mit dem zweiten %INSERT wird der *testpunkt* S'3' vereinbart. Dieser %INSERT enthält noch zwei geschachtelte %INSERT-Kommandos. Ihre *testpunkt*-Werte sind für AID noch nicht wirksam. Sie können erst aktiv werden, wenn der *testpunkt* des %INSERT erreicht wird, in dessen *subkdo* sie definiert sind.

   Kommt der Programmablauf zur Anweisung S'3', wird das zugehörige *subkdo* ausgeführt, d.h. das %DISPLAY-Kommando für das Literal 'INSERT1' und die Variable TEXTDAT wird ausgeführt und der *testpunkt* L'20' gesetzt.
   Das *subkdo* zum *testpunkt* L'20' ist noch nicht wirksam. Im zu testenden Programm sind also bis zu dieser Stelle des Programmablaufs drei *testpunkte* gesetzt: L'2', S'3' und L'20'.

   Da auch das *subkdo* zum *testpunkt* S'3' kein %STOP-Kommando enthält, wird das Programm nach Ausführung von *subkdo* fortgesetzt. Wird der Programmablauf nicht aus einem anderen Grund, z.B. einem Fehler oder dem Eintreten eines mit %ON vereinbarten Ereignisses, unterbrochen und erreicht schließlich die symbolische Adresse L'20', wird nun der %D 'INSERT2', I, J, K, ANZAHL ausgeführt. Außerdem enthält *subkdo* wieder ein %INSERT-Kommando, dessen *testpunkt* diesmal mit S'172', das ist die Anweisung mit der Nummer 172, bezeichnet ist.

   Wird im weiteren Programmablauf die mit S'172' bezeichnete Stelle erreicht, führt AID den %DISPLAY für das Literal 'INSERT3' und die Inhalte der Variablen I und J aus. Mit dem zweiten Kommando in diesem *subkdo*, dem %REMOVE L'20', wird der *testpunkt* L'20' gelöscht. Das ist z.B. dann erforderlich, wenn ein *testpunkt* in einer Schleife liegt und es dadurch zur unerwünschten Kettung geschachtelter *subkdo's* kommen würde. Ohne das %REMOVE-Kommando würde beim zweiten Durchlaufen von L'20' zum *testpunkt* S'172' folgendes *subkdo* entstehen:

   ```
   <%D 'INSERT3',I,J; %D 'INSERT3',I,J>
   ```
   ```

5.
```
%OUT %DISPLAY P=MAX
%IN L'100' <%D 'I GE 10',I,X(I),K,Y(I,K)>
%IN L'100' <(I LT 10): %D 'I LT 10',I,X(I); %CONT>
```

   Zunächst wird vereinbart, dass alle Ausgaben des Kommandos %DISPLAY nach SYSLST erfolgen.
   Durch die danach eingegebenen beiden %INSERTs entsteht am *testpunkt* L'100' das folgende Subkommando:

```
<(I LT 10): %D 'I LT 10',I,X(I); %CONT; %D 'I GE 10',I,X(I),K,Y(I,K)>
```

   Jedes Mal, wenn der Programmablauf an die Anweisung mit der Anweisungsmarke 100 kommt, wird geprüft, ob der Index I einen Wert < 10 enthält. Falls die Bedingung zutrifft, schreibt AID den Kommentar 'I LT 10' sowie den Inhalt von I und X(I) nach SYSLST und setzt wegen des %CONTINUE den Programmablauf fort, evtl. mit Ablaufverfolgung, falls durch das Subkommando ein %TRACE unterbrochen wurde.
   Ist der Wert von I ≥ 10, dann schreibt AID den Kommentar 'I GE 10' und zusätzlich zu I und X(I) auch die Werte von Index K und dem Feldelement Y(I,K) nach SYSLST und setzt ebenfalls das Programm fort. Auch in diesem Fall läuft das Programm mit Ablaufverfolgung weiter, wenn noch ein %TRACE aktiv ist.

%JUMP

Mit %JUMP legen Sie eine Fortsetzungsadresse fest, an der das Programm mit %CONTINUE, %RESUME oder %TRACE fortgesetzt werden soll. Mit dieser Adresse weichen Sie vom codierten Programmablauf ab. Das Kommando wird mit der Meldung beantwortet, dass der Sprung ausgeführt wurde.

– Mit *fortsetzung* bezeichnen Sie die Stelle im Programm, an der AID nach Beendigung der Kommandoeingabe das Programm fortsetzen soll. *fortsetzung* kann nur die Adresse einer FORTRAN-Anweisung sein.

Kommando	Operand
%JUMP	fortsetzung

%JUMP kann nur für Programmeinheiten verwendet werden, die mit FOR1 ab Version V2.1A übersetzt wurden und die nicht optimiert sind (SDF-Option OPTIMIZATION = NO bzw. COMOPT-Anweisung OPTIMIZE = NO).

Die Fortsetzungsadresse muss in derselben Programmeinheit liegen, in der das Programm unterbrochen wurde, sonst gibt AID eine Fehlermeldung aus. Andere Prüfungen nimmt AID nicht vor. Sie müssen selbst dafür sorgen, dass die Voraussetzungen (z.B. Index- oder Zählerstände, Datei-Status) für den fehlerfreien Ablauf des Programms von *fortsetzung* an erfüllt sind. Das gilt vor allem, wenn Sie mit %JUMP auf einer Adresse aufsetzen, die im Programmablauf logisch vor der Unterbrechungsstelle liegt.

Nicht eingeben können Sie %JUMP:

– unmittelbar nach dem LOAD-PROGRAM-Kommando,

– wenn das Programm vom System unterbrochen wurde, z.B. weil eine Datei, die geöffnet werden soll, nicht zugewiesen ist,

– wenn das Programm mit der K2-Taste unterbrochen wurde,

– wenn das Programm durch die FORTRAN-Anweisung PAUSE angehalten wurde.

%JUMP verändert den Programmzustand nicht.

fortsetzung

definiert die Stelle, an der das Programm fortgesetzt werden soll. *fortsetzung* muss die Adresse einer ausführbaren Anweisung in der aktuellen Programmeinheit sein. Steht der %JUMP in einem Subkommando, so muss *fortsetzung* eine Anweisung in der Programmeinheit bezeichnen, in der auch der Testpunkt liegt bzw. das mit %ON definierte Ereignis eingetreten ist.

```
fortsetzung-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

{L'n' }
{S'n' }

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

L’n’
: ist ein Anweisungsname und bezeichnet die Adresse der ersten ausführbaren FORTRAN-Anweisung nach einer Anweisungsmarke.
n ist eine maximal 5stellige Anweisungsmarke. Führende Nullen dürfen nicht angegeben werden.

S’n’
: ist eine Source-Referenz und bezeichnet die Adresse einer ausführbaren FORTRAN-Anweisung.
n ist die Nummer einer Anweisung; siehe Spalte STMT der Übersetzungsliste.

Beispiele

1. `%JUMP S'24'`
   Als Fortsetzungsadresse wird die Anweisung mit der Nummer 24 vereinbart.
2. `%JUMP L'100'`
   Als Fortsetzungsadresse wird die erste ausführbare Anweisung nach der Anweisungsmarke 100 vereinbart.

%MOVE

Mit %MOVE übertragen Sie Speicherinhalte oder AID-Literale auf Speicherstellen im geladenen Programm. Die Übertragung wird ohne Überprüfung und ohne Anpassung der Speichertypen von Sender und Empfänger durchgeführt.

– Mit *sender* bezeichnen Sie eine Variable, ein Feld oder ein Feldelement, eine Länge, eine Adresse, einen Durchlaufzähler, ein AID-Register oder ein AID-Literal. *sender* kann im virtuellen Speicher des geladenen Programms oder in einer Dump-Datei liegen.

– Mit *empfänger* bezeichnen Sie eine Variable, ein Feld oder ein Feldelement, einen Durchlaufzähler oder ein AID-Register, das überschrieben werden soll. *empfänger* kann nur im virtuellen Speicher des geladenen Programms liegen.

– Mit *REP* geben Sie an, ob AID zu einer durchgeführten Änderung einen REP-Satz erzeugen soll. Mit diesem Operanden setzen Sie eine mit dem Kommando %AID vereinbarte Voreinstellung für das aktuelle %MOVE-Kommando außer Kraft.

Kommando	Operand
%M[OVE]	sender INTO empfänger [REP]

Im Gegensatz zu %SET überprüft AID beim %MOVE nicht die Verträglichkeit der Speichertypen von *sender* und *empfänger* und passt *sender* nicht an den Speichertyp von *empfänger* an.

AID überträgt linksbündig in der Länge von *sender*. Ist *sender* länger als *empfänger*, weist AID die Übertragung mit einer Fehlermeldung ab.

Im Gegensatz zum %SET können Sie mit %MOVE Datenelemente vom Typ COMPLEX als Gesamtes übertragen bzw. überschreiben. Sie haben aber wie beim %SET auch die Möglichkeit, mit *datenname*._REAL bzw. *datenname*._IMAG gezielt Real- bzw. Imaginärteil der komplexen Zahl zu verändern.

Es empfiehlt sich, das Kommando nicht unmittelbar nach dem Laden einzugeben, da Sie Daten und Anweisungen erst dann ohne explizite Qualifikation ansprechen können, wenn das Programm vor der ersten ausführbaren Anweisung steht. Dies erreichen Sie mit der folgenden Kommandofolge:
%INSERT PROG=program-name.program-name
%RESUME

Neben den hier beschriebenen Operandenwerten können Sie auch die im Handbuch für das Testen auf Maschinencode-Ebene beschriebenen Operandenwerte einsetzen.

Mit `%AID CHECK = ALL` können Sie zur Kontrolle einen Änderungsdialog einschalten, der Ihnen vor Durchführung der Übertragung den alten und neuen Inhalt des *empfängers* zeigt und Ihnen die Möglichkeit zum Abbruch des %MOVE-Kommandos bietet.

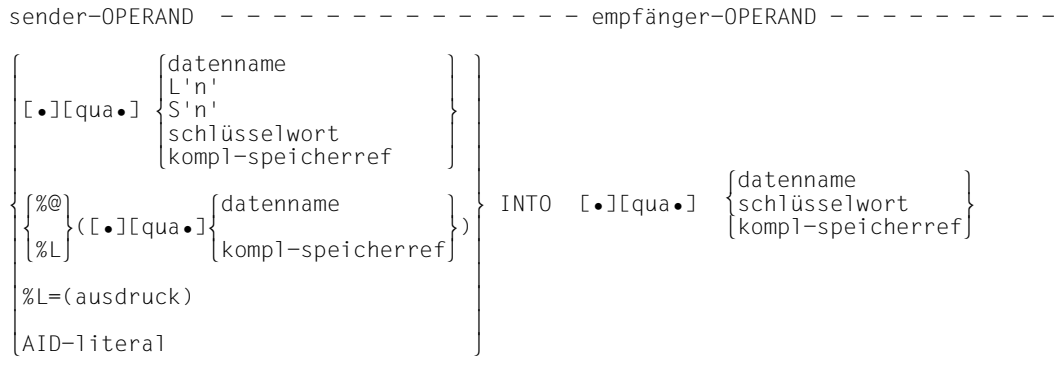
%MOVE verändert den Programmzustand nicht.

sender	INTO	empfänger

Für *sender* oder *empfänger* können Sie eine Variable, ein Feld, ein Feldelement oder eine komplexe Speicherreferenz, einen Durchlaufzähler oder ein Register angeben. Symbolische Konstanten, Adressen und Längen von Datenelementen sowie AID-Literale können Sie nur als *sender* einsetzen.

sender kann sowohl im virtuellen Speicherbereich des geladenen Programms als auch in einer Dump-Datei liegen; *empfänger* kann dagegen nur im virtuellen Speicherbereich des geladenen Programms liegen.

Mehr als 3900 Bytes können mit einem %MOVE nicht übertragen werden. Wenn Sie einen größeren Bereich übertragen wollen, müssen Sie daher mehrere %MOVE-Kommandos verwenden.



- Steht der Punkt an führender Stelle, ist er das Kennzeichen für eine *vorqualifikation*. Sie muss mit einem vorhergehenden %QUALIFY definiert worden sein.
  Aufeinanderfolgende Qualifikationen werden durch einen Punkt getrennt. Außerdem muss zwischen der letzten Qualifikation und dem anschließenden Operandenteil ein Punkt stehen.

qua

Eine Qualifikation geben Sie nur an für Speicherobjekte, die nicht im aktuellen AID-Arbeitsbereich liegen.

E={VM | Dn} für *sender*
E=VM für *empfänger*

geben Sie nur an, wenn für einen Daten- oder Anweisungsnamen oder für eine Source-Referenz oder ein Schlüsselwort die aktuelle Basis-Qualifikation nicht gelten soll (siehe %BASE).
sender kann sowohl im virtuellen Speicher als auch in einer Dump-Datei liegen.
empfänger kann dagegen nur im virtuellen Speicher liegen.

PROG=program-name

geben Sie nur an, wenn Sie einen Daten- oder Anweisungsnamen oder eine Source-Referenz ansprechen, die nicht in der aktuellen Programmeinheit liegt (siehe Kapitel 3).

NESTLEV= level-nummer

level-nummer Nummer einer Ebene in der aktuellen Aufrufhierarchie

Auf *level-nummer* muss *datenname* folgen.
NESTLEV= *level-nummer* geben sie an, wenn Sie einen Datennamen in einer bestimmten Ebene der aktuellen Aufrufhierachie ansprechen wollen. Diese Qualifikation kann nur mit E= kombiniert werden, nicht mit anderen Qualifikationen.

datenname

ist der im Quellprogramm definierte Name einer Konstanten, einer Variablen, eines Felds oder Feldelements. Konstanten können nur als *sender* eingesetzt werden. *datenname* ist eine maximal 15stellige alphanumerische Zeichenfolge.

Ist *datenname* der Name eines Feldes, dann können Sie ihn wie in einer FORTRAN-Anweisung indizieren, um ein Feldelement anzusprechen.
Wenn Sie den Namen eines Feldes ohne Indexliste angeben, so bedeutet dies bei *sender*, dass alle Feldelemente übertragen werden. Geben Sie bei *empfänger* ein Feld ohne Indexliste an, dann wird das Feld beginnend bei der Anfangsadresse in der Länge von *sender* überschrieben, ohne dass die Unterteilung in Feldelemente beachtet wird.

feldname (index1[, index2][, ...])

index gibt die Position innerhalb eines Feldes an. Es sind so viele Indizes erforderlich, wie in einer FORTRAN-Anweisung zum Zugriff angegeben werden müssen. Mehrere Indizes müssen durch Komma getrennt werden.
index kann folgendermaßen angegeben werden:

```
{n                    }
{datenname            }
{arithmetischer ausdruck}
```

L'n'

ist ein Anweisungsname und bezeichnet die Adresse der ersten ausführbaren FORTRAN-Anweisung nach einer Anweisungsmarke.
n ist eine maximal 5stellige Anweisungsmarke. Führende Nullen dürfen nicht angegeben werden.

S'n'

ist eine Source-Referenz und bezeichnet die Adresse einer ausführbaren FORTRAN-Anweisung.
n ist die Nummer einer Anweisung; siehe Spalte STMT der Übersetzungsliste.

Anweisungsnamen und Source-Referenzen sind Adress-Konstanten und können daher nur als *sender* angegeben werden. Es wird dann die mit *L'n'* bzw. *S'n'* bezeichnete Adresse übertragen.

Beispiel

```
%MOVE S'5' INTO %2G
```

Die Adresse der Anweisung mit der Nummer 5 wird in das AID-Register %2G geschrieben.

Mit *L'n'*-> bzw. *S'n'*-> bezeichnen Sie 4 Bytes des an der entsprechenden Adresse stehenden Maschinencodes (siehe AID-Basishandbuch, Abschnitt 6.4).
Die Maschinenbefehle können Sie sich mit %DISASSEMBLE ausgeben lassen, um eventuell eine Längenmodifikation vorzunehmen.
Bei *empfänger* können Sie Anweisungsnamen und Source-Referenzen nur in Verbindung mit dem Pointer-Operator (->) verwenden.

Beispiel

```
%MOVE S'12'->%L=(S'13'-S'12') INTO S'24'->
```

Durch diesen %MOVE ändern Sie den Code Ihres Programms. Der Maschinencode zu Anweisung 24 wird durch den der Anweisung 12 überschrieben. Aus der Angabe %L=(S'13'-S'12') ergibt sich die Länge des zu Anweisung 12 erzeugten Maschinencodes.

schlüsselwort

ist ein Durchlaufzähler, der Befehlszähler oder ein Register.
Vor *schlüsselwort* können Sie nur eine Basis-Qualifikation angeben.

%**.**subkdoname	Durchlaufzähler
%**.**	Durchlaufzähler des gerade aktiven Subkommandos
%PC	Befehlszähler (Program Counter)
%n	Mehrzweckregister, $0 \leq n \leq 15$
%nD|E	Gleitpunktregister, n = 0,2,4,6
%nQ	Gleitpunktregister, n = 0,4
%nG	AID-Mehrzweckregister, $0 \leq n \leq 15$
%nDG	AID-Gleitpunktregister, n = 0,2,4,6

kompl-speicherref

Folgende Operationen können darin vorkommen (siehe AID-Basishandbuch, Kapitel 6):

- Adressversatz (**.**)
- indirekte Adressierung (->)
- Typmodifikation %E
- Längenmodifikation (%L(...), %L=(ausdruck), %Ln)
- Adressselektion (%@(...))

Eine abschließende Typmodifikation ist bei *kompl-speicherref* sinnlos, da unnabhängig vom Speichertyp von *sender* und *empfänger* stets binär übertragen wird. Allerdings kann eine Typmodifikation vor einer Pointer-Operation (->) notwendig sein.

Beispiel

```
%2G.2%AL2->
```

Die letzten beiden Bytes des AID-Registers %2G sollen als Adresse benutzt werden.

Nach Adressversatz (**.**) oder Pointer-Operation (->) gehen impliziter Speichertyp und implizite Länge der Ausgangsadresse verloren. An der errechneten Adresse gilt der Speichertyp %X in der Länge 4, falls Sie nicht Typ und Länge explizit angeben.
Für jeden Operanden in einer komplexen Speicherreferenz darf der zugeordnete Spicherbereich durch einen Adressversatz oder eine Längenmodifikation nicht überschritten werden, sonst führt AID das Kommando nicht aus und schreibt eine Fehlermeldung. Durch die Verbindung von Adressselektion (%@) mit Pointer-Operator (->) verlassen Sie die symbolische Ebene. Nun können Sie die Adresse eines Datenelements verwenden, ohne auf dessen Bereichsgrenzen achten zu müssen.

Beispiel

Die Variablen CFELD und CFELD1 belegen je 5 Bytes. Die letzten 2 Bytes von CFELD sowie die 3 anschließenden Bytes sollen nach CFELD1 übertragen werden. Das folgende Kommando würde AID wegen Bereichsverletzung von CFELD ablehnen:

```
%MOVE CFELD.3%L5 INTO CFELD1
```

Richtig müsste es dagegen heißen:

```
%MOVE %@(CFELD)->.3%L5 INTO CFELD1
```

%@(...)

Mit dem Adressselektor können Sie die Adresse eines Datenelements oder einer komplexen Speicherreferenz als *sender* verwenden (siehe AID-Basishandbuch, Abschnitt 6.11). Der Adressselektor liefert als Ergebnis eine Adresskonstante.

%L(...)

Mit dem Längenselektor können Sie die Länge eines Datenelements oder einer komplexen Speicherreferenz als *sender* verwenden (siehe AID-Basishandbuch, Abschnitt 6.11). Der Längenselektor liefert als Ergebnis eine Ganzzahl.

Beispiel

```
%MOVE %L(FELD1) INTO %2G
```

Die Länge von FELD1 wird übertragen.

%L=(ausdruck)

Mit der Längenfunktion können Sie sich den Wert von *ausdruck* berechnen und in *empfänger* abspeichern lassen (siehe AID-Basishandbuch, Abschnitte 6.9 und 6.10). In *ausdruck* können Sie den Inhalt von Speicherreferenzen und Konstanten vom Typ Integer und ganze Zahlen mit den arithmetischen Operatoren (+,-,*,/) verknüpfen. Die Längenfunktion liefert als Ergebnis eine Ganzzahl.

Beispiel

```
%MOVE %L=(FELD1) INTO %2G
```

Der Inhalt von FELD1 wird übertragen. FELD1 muss vom Typ Integer sein, sonst gibt AID eine Fehlermeldung aus.

AID-literal

Die folgenden AID-Literale (siehe AID-Basishandbuch, Kapitel 8) können mit %MOVE übertragen werden:

{C'x...x' | 'x...x'C | 'x...x'}	Character-Literal
{X'f...f' | 'f...f'X}	Sedezimal-Literal
{B'b...b' | 'b...b'B}	Binär-Literal
[{±}]n	Ganzzahl
#'f...f'	Sedezimalzahl

REP

gibt an, ob AID zu einer durchgeführten Änderung einen REP-Satz erzeugen soll. Mit *REP* setzen Sie eine mit dem Kommando %AID getroffene Vereinbarung vorübergehend außer Kraft. Wird *REP* nicht angegeben, und gibt es keine gültige Vereinbarung im %AID-Kommando, so wird kein REP-Satz erstellt.

```
rep-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

REP = {Y[ES] | NO}

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

REP=Y[ES]
: Zu der durch das aktuelle %MOVE-Kommando durchgeführten Änderung werden LMS-UPDR-Korrektursätze (REPs) erstellt. Wenn die Objekt-Strukturliste nicht zur Verfügung steht, erstellt AID keine Korrektursätze und gibt eine Fehlermeldung aus.

  Auch wenn *empfänger* nicht vollständig innerhalb einer CSECT liegt, gibt AID eine Fehlermeldung aus und schreibt keinen REP. Um dennoch REP-Sätze zu erhalten, müssen Sie die Übertragung auf mehrere %MOVE-Kommandos verteilen, bei denen Sie die CSECT-Grenzen beachten (siehe [2]).

  AID hinterlegt die Korrekturen mit den nötigen LMS-UPDR-Anweisungen in einer Datei mit dem Linknamen F6, aus der sie als fertiges Paket übernommen werden können. Achten Sie deshalb darauf, dass Sie in die Datei mit dem Linknamen F6 keine anderen Ausgaben schreiben lassen.

  Ist keine Datei mit dem Linknamen F6 angemeldet (siehe %OUTFILE), so wird der REP in der von AID angelegten Datei AID.OUTFILE.F6 abgelegt.

REP=NO
: Zum aktuellen %MOVE-Kommando werden keine REPs erstellt.

Beispiele

In einem FORTRAN-Programm sind die folgenden Variablen und Felder definiert:

```
INTEGER*2 IFELD(10)
INTEGER*4 JFELD(10)
REAL*4 RZAHL
CHARACTER*4 CVAR
```

1. `%MOVE IFELD INTO JFELD`
   Bei beiden Feldern ist kein Index angegeben; AID überträgt daher den Inhalt von IFELD ohne Beachtung der Unterteilung in Feldelemente sedezimal linksbündig an die symbolische Adresse JFELD.

2. `%MOVE 20 INTO JFELD(2)`
   In das Feldelement JFELD(2) vom Typ INTEGER*4 schreibt AID ein Wort, das die Ganzzahl mit dem Wert 20 enthält.

3. `%MOVE 20 INTO RZAHL`
   Wie im Beispiel 2 wird auch hier ein Wort mit dem Inhalt X’00000014’ nach RZAHL geschrieben, was natürlich bei einer REAL-Zahl keinen Sinn ergibt. Um den Wert 20 nach RZAHL zu übertragen, müssten Sie ein %SET-Kommando eingeben (siehe %SET), das vor der Übertragung eine Konvertierung durchführt.

4. `%MOVE X'58F0C160' INTO CVAR REP=YES`
   Der Inhalt der Variablen CVAR wird mit dem Sedezimal-Literal X’58F0C160’ überschrieben. Zu der Korrektur wird ein REP erstellt und in der Datei AID.OUTFILE.F6 bzw. der dem Linknamen F6 zugewiesenen Datei abgelegt.

%ON

Mit %ON legen Sie Ereignisse fest und definieren Subkommandos. Wenn ein ausgewähltes *ereignis* eintritt, bearbeitet AID das zugehörige *subkdo*.

- Mit *write-ereignis* beschreiben Sie das Ereignis des schreibenden Zugriffs auf einen Speicherbereich. Immer wenn das Programm den angegebenen Speicherbereich verändert, soll AID den Programmablauf unterbrechen, um *subkdo* zu bearbeiten.
- Mit *ereignis* beschreiben Sie eines der anderen Ereignisse (normale oder abnormale Programmbeendigung, Supervisor-Call (SVC), Programmfehler, etc.), bei dem AID den Programmablauf unterbrechen soll, um *subkdo* zu bearbeiten.
- Mit *subkdo* definieren Sie ein Kommando oder eine Kommandofolge und eventuell eine Bedingung. Bei zutreffendem *ereignis* und erfüllter Bedingung wird *subkdo* ausgeführt.

Kommando	Operand	
%ON	{write-ereignis / ereignis}	[<subkdo>]

Ohne *subkdo*-Operanden setzt AID das *subkdo* <%STOP> ein.

Das *subkdo* eines %ON für ein bereits angemeldetes *ereignis* überschreibt nicht das bestehende *subkdo*, sondern das neue *subkdo* wird vor das bestehende gekettet. Das bedeutet, dass gekettete Subkommandos nach dem LIFO-Prinzip abgearbeitet werden. Dies gilt nicht für *write-ereignis*. Die Eingabe eines neuen *write-ereignis* überschreibt ein bereits bestehendes.

Ein eingetragenes Ereignis gilt, bis es mit %REMOVE gelöscht wird oder bis zum Programmende.

Für %ON muss die Basis-Qualifikation E=VM eingestellt sein (siehe %BASE).

%ON verändert den Programmzustand nicht.

write-ereignis

Mit dem Schlüsselwort %WRITE schalten Sie die Schreibüberwachung ein. In Klammern geben Sie den zu überwachenden Speicherbereich an. Immer dann, wenn auf ein Byte innerhalb des angegebenen Speicherbereichs schreibend zugegriffen wird, bearbeitet AID das im %ON definierte Subkommando. Die Unterbrechungsstelle liegt hinter dem Hardware-Befehl, der die Unterbrechung verursacht hat. Dieser Hardware-Befehl kann sich im Programm oder im Laufzeitsystem befinden. Der zu überwachende Bereich kann höchstens 65535 Bytes lang sein.

Subkommandos zum *write-ereignis* können nicht gekettet werden. Mehrere Bereiche können Sie nicht gleichzeitig überwachen lassen. Das *write-ereignis* kann jedoch mit anderen Ereignissen gleichzeitig angemeldet sein.

Die Lage des zu überwachenden Speicherbereiches bleibt beim Programmablauf unverändert. Durch Überlagerung kann sich die logische Bedeutung dieser Speicherstelle ändern. Beim Eintritt des *write-ereignis* ist deshalb die logische Bedeutung der Speicherstelle zu prüfen. (%WRITE (VAR) definiert den Speicherbereich von VAR zum Zeitpunkt der AID-Kommandoausführung. Dieser Speicher wird überwacht, aber nicht die Variable VAR selbst).

Es kann immer nur ein *write-ereignis* definiert sein. Die Eingabe von einem neuen *write-ereignis* überschreibt ein bestehendes.

Trifft ein *ereignis* gleichzeitig mit *write-ereignis* ein, so bearbeitet AID das Subkommando zu *write-ereignis* als erstes.
Das *write-ereignis* löschen Sie mit %REMOVE %WRITE ohne Angabe der Speicherreferenz.

Folgende Wechselwirkungen bestehen zwischen %ON *write-ereignis* und anderen AID-Kommandos:

– Wenn ein %CONTROLn oder ein %TRACE mit maschinennahem *kriterium* angemeldet ist, wird die Eingabe von %ON *write-ereignis* mit einer Fehlermeldung abgewiesen.

– Wenn ein Maschinenbefehl durch einen %CONTROLn oder %TRACE mit symbolischem *kriterium* mit der AID-internen Markierung (X'0A81') überschrieben wurde, bemerkt AID den schreibenden Zugriff dieses Befehls nicht.

– Wenn ein Maschinenbefehl durch den mit %INSERT vereinbarten Testpunkt mit der AID-internen Markierung überschrieben wurde, erkennt AID auch hier den schreibenden Zugriff dieses Befehls nicht.

Für eine lückenlose Schreibüberwachung empfiehlt es sich, alle %CONTROLn- und %INSERT-Kommandos mit %REMOVE zu löschen und einen eventuell noch eingetragenen %TRACE zu löschen, indem Sie nach dem %ON mit %RESUME fortfahren.

Der zu überwachende Speicherbereich kann jedes, wie auch immer angesprochene Speicherobjekt sein. Er wird durch Anfangsadresse und implizite oder explizite Länge festgelegt. Die Länge darf 1024 Bytes nicht überschreiten, sonst wird eine Fehlermeldung ausgegeben.

Wenn bei einem Programm mit Überlagerungsstruktur die Adresse des angegebenen Speicherobjekts überladen wird, wird der entsprechende Bereich des neu geladenen Programmteils überwacht.

Weitere Informationen zur Schreibüberwachung finden Sie im AID-Basishandbuch, Abschnitte 10.13 und 12.1.

```
write-ereignis-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - -

                                         ⎧datenname          ⎫
                ⎧•                  ⎫    ⎪L'n'->             ⎪
%WRITE ( [⎨                    ⎬ ] ⎨S'n'->             ⎬)
                ⎩PROC=program-name. ⎭    ⎪                   ⎪
                                         ⎩kompl-speicherref  ⎭
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

•

Steht der Punkt an führender Stelle, ist er das Kennzeichen für eine *vorqualifikation*. Sie muss mit einem vorhergehenden %QUALIFY definiert worden sein.
Außerdem muss zwischen der PROG-Qualifikation und dem anschließenden Operandenteil ein Punkt stehen.

PROG=program-name

geben Sie nur an, wenn Sie einen Daten- oder Anweisungsnamen oder eine Source-Referenz ansprechen, die nicht in der aktuellen Programmeinheit liegt (siehe Kapitel 3).

datenname

ist der im Quellprogramm definierte Name einer Variablen, eines Felds oder Feldelements.
datenname ist eine maximal 15stellige alphanumerische Zeichenfolge.

Ist *datenname* der Name eines Felds, dann können Sie ihn wie in einer FORTRAN-Anweisung indizieren, um ein Feldelement anzusprechen. Wenn Sie den Namen eines Felds ohne Indexliste angeben, wird das gesamte Feld überwacht.

feldname (index1[, index2][, ...])

index gibt die Position innerhalb eines Felds an. Es sind so viele Indizes erforderlich, wie in einer FORTRAN-Anweisung zum Zugriff angegeben werden müssen.
Mehrere Indizes müssen durch Komma getrennt werden.

index kann folgendermaßen angegeben werden:

```
{n                      }
{datenname              }
{arithmetischer ausdruck}
```

L'n'->
: bezeichnet die Speicherstelle an der Adresse der ersten ausführbaren FORTRAN-Anweisung nach einer Anweisungsmarke.
n ist eine maximal 5stellige Anweisungsmarke. Führende Nullen dürfen nicht angegeben werden.

  Ohne Längenmodifikation werden 4 Bytes ab der Adresse überwacht, die in der Adresskonstanten *L'n'* hinterlegt ist.

S'n'->
: bezeichnet die Speicherstelle an der Adresse der FORTRAN-Anweisung mit der angegebenen Nummer.
n ist die Nummer einer Anweisung; siehe Spalte STMT der Übersetzungsliste.

  Ohne Längenmodifikation werden 4 Bytes ab der Adresse überwacht, die in der Adresskonstanten *S'n'* hinterlegt ist.

kompl-speicherref
: bezeichnet einen Bereich von 4 Bytes ab der errechneten Adresse. Soll eine andere Anzahl von Bytes überwacht werden, muss *kompl-speicherref* mit der entsprechenden Längenmodifikation enden. Bei der Längenmodifikation von Datenelementen müssen Sie die Bereichsgrenzen beachten oder mit %@(datenname)-> auf Maschinencode-Ebene wechseln. Folgende Operationen können in *kompl-speicherref* vorkommen (siehe AID-Basishandbuch, Kapitel 6):

  – Adressversatz (**.**)
  – indirekte Adressierung (->)
  – Typmodifikation (%A)
  – Längenmodifikation (%L(...), %L=(ausdruck), %Ln)
  – Adressselektion (%@(...))

ereignis

Ein Schlüsselwort legt fest, bei welchem Ereignis (Programmfehler, abnormale Programmbeendigung, Supervisor-Call etc.) AID das angegebene *subkdo* bearbeiten soll.

Wenn mehrere %ON-Kommandos mit unterschiedlichen *ereignis*-Vereinbarungen gleichzeitig aktiv sind und auch zutreffen, bearbeitet AID die zugehörigen Subkommandos in der Reihenfolge, in der die Schlüsselwörter in der folgenden Tabelle aufgeführt sind. Treffen verschiedene %TERM-Ereignisse zu, werden die zugehörigen Subkommandos entgegen der Reihenfolge abgearbeitet, in der die %TERM-Ereignisse erklärt wurden (LIFO-Prinzip wie bei der Verkettung d andos). Wenn ein *write-ereignis* gleichzeitig mit einem anderen *er-*

eignis eintritt, wird erst das Subkommando zum *write-ereignis* abgearbeitet.
Zur Auswahl der SVC-Nummern und Ereigniscodes siehe Makroaufrufe an den Ablaufteil [6].

Es ist nicht sinnvoll, in einem %ON Ereignisse anzumelden, die schon durch die FOR1-Fehlerbehandlungsroutinen abgefangen werden. Dazu gehören die Unterbrechungsbedingungen:
`%ERRFLG(zzz), %INSTCHK, %ARTHCHK, %ABNORM und %ERRFLG`

Diese Ereignisse können deshalb in einem %ON-Kommando nur dann angesprochen werden, wenn das Anmelden der FOR1-Fehlerbehandlungsroutinen unterdrückt wurde (nur möglich bei FOR1-Programmen ohne Standard Linkage).
Dazu geben Sie folgende Kommandos ein:

```
/PARAMETER CARD = YES
.
.
⎧/LOAD-PROGRAM  ⎫
⎨               ⎬ ..., TEST-OPT = AID
⎩/START-PROGRAM ⎭
.
.
GIVE 'RUNOPT' OR 'END' OR '?'
*RUNOPT STXIT = NO
.
.
```

Für die Ereignisse `%ERRFLG[(zzz)], %INSTCK, %ARTHCHK` und `%ABNORM` gilt die folgende Einschränkung: diese Ereignisse können mit AID nicht bearbeitet werden, wenn dazu STXIT-Routinen z.B. des Laufzeitsystems oder von ILCS angemeldet sind. Näheres zu %ON und STXIT finden Sie im AID-Basishandbuch, Abschnitt 11.1.

ereignis	*subkdo* **wird bearbeitet:**
%ERRFLG (zzz)	nach Auftreten eines Fehlers mit dem Ereigniscode zzz und vor Abbruch des Programms.
%INSTCHK	nach Auftreten eines Adressierungsfehlers, eines unzulässigen Systemaufrufs (SVC), nicht decodierbaren Operations-Codes, Seitenwechsel-Fehlers oder einer priviligierten Operation und vor Abbruch des Programms.
%ARTHCHK	nach Auftreten eines Datenfehlers, Divisionsfehlers, Exponenten-Überlaufs oder einer Mantisse Null und vor Abbruch des Programms.
%ABNORM	nach Auftreten eines der Fehler, die mit den vorher beschriebenen Ereignissen erfasst werden
%ERRFLG	nach Auftreten eines Fehlers mit beliebigem Ereigniscode.
%SVC(zzz)	vor Ausführung des Systemaufrufs (SVC) mit der angegebenen Nummer.
%LPOV(x...x)	nach dem Laden des Segments mit dem angegebenen Namen
%LPOV	nach dem Laden eines beliebigen Segments (der Name wird mit %D %LINK ausgegeben)
%TERM(N[ORMAL])	vor normaler Beendigung des Programms
%TERM(A[BNORMAL])	vor abnormaler Beendigung des Programms, jedoch nach der Ausgabe eines Speicherabzugs
%TERM	vor Beendigung des Programms durch alle vorher beschriebenen %TERM-Ereignisse
%ANY	vor der Beendigung des Programms aufgrund eines Programmfehlers bzw. durch die oben beschriebenen %TERM-Ereignisse
%SVC	vor Ausführung eines beliebigen Systemaufrufs (SVC).

zzz kann in zwei verschiedenen Formaten angegeben werden:
n maximal dreistellige vorzeichenlose Dezimalzahl
#'ff' zweistellige Sedezimalzahl
Für den Wert von *zzz* gilt: $1 \leq zzz \leq 255$

Es wird nicht überprüft, ob der angegebene Ereigniscode oder die SVC-Nummer sinnvoll oder zulässig ist.

x...x ist der Name eines Segments mit maximal acht alphanumerischen Zeichen.

subkdo

wird immer dann bearbeitet, wenn im Programmablauf das vereinbarte *ereignis* eintritt. Wird der Operand *subkdo* nicht angegeben, so setzt AID ein <%STOP> ein.

Vollständig beschrieben finden Sie *subkdo* im Basishandbuch, Kapitel 5.

```
subkdo-OPERAND  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                                      ⎧AID-kommando    ⎫
<[subkdoname:] [(bedingung):] [⎨                       ⎬ {;...}]>
                                      ⎩BS2000-kommando ⎭

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

Das Subkommando kann einen Namen, eine Bedingung und einen Kommandoteil enthalten. Zu jedem Subkommando gehört ein Durchlaufzähler. Der Kommandoteil kann aus einem einzelnen Kommando oder einer Kommandofolge bestehen, er kann AID- und BS2000-Kommandos und Kommentare enthalten.
Wenn das Subkommando aus einem Namen oder einer Bedingung besteht, aber der Kommandoteil fehlt, erhöht AID beim Eintreten des vereinbarten Ereignisses nur den Durchlaufzähler.

subkdo überschreibt nicht ein bestehendes Subkommando zu demselben *ereignis*, sondern das neue Subkommando wird vor das bestehende gekettet. In *subkdo* sind die Kommandos %CONTROLn, %INSERT, %JUMP und %ON zugelassen. Sie können damit eine Schachtelung über maximal 5 Stufen vornehmen. Ein Beispiel dazu finden Sie in der %INSERT-Beschreibung.

Die Kommandos in einem *subkdo* werden nacheinander ausgeführt. Danach wird das Programm fortgesetzt. Die Kommandos zur Ablaufsteuerung verändern auch in einem Subkommando sofort den Programmzustand. Sie brechen *subkdo* ab und setzen das Programm fort (%CONTINUE, %RESUME, %TRACE) oder halten es an (%STOP). Sie sind nur als letztes Kommando in einem *subkdo* sinnvoll, da nachfolgende *subkdo*-Kommandos nicht mehr ausgeführt werden. Auch ein Löschen des gerade aktiven Subkommandos mit %REMOVE ist nur als letztes Kommando in *subkdo* sinnvoll.

Beispiele

1. `%ON %LPOV (MONA12) <%D '%LPOV (MONA12)'; %STOP>`
   Nach dem Laden von MONA12 gibt AID das Literal '%LPOV (MONA12)' aus und unterbricht den Programmablauf.

2. `%ON %ERRFLG (108)`
   `%ON %ERRFLG (#'6C')`
   Beide Angaben bezeichnen den gleichen Programmfehler (Mantisse gleich Null).

3. `%ON %ERRFLG (107) <%D 'ERROR'>`
   Dieses Fehlergewicht gibt es nicht; deshalb wird das für dieses *ereignis* definierte *subkdo* nie gestartet.

4.

```
%ON %WRITE(PAGE) <Schreib: %D 'Schreibender Zugriff auf PAGE'>
%R

SCHREIBENDER ZUGRIFF AUF PAGE
SCHREIBENDER ZUGRIFF AUF PAGE
SCHREIBENDER ZUGRIFF AUF PAGE
SCHREIBENDER ZUGRIFF AUF PAGE
SCHREIBENDER ZUGRIFF AUF PAGE
```

   Fünfmal wurde vom Programm auf PAGE schreibend zugegriffen.

5.

```
%ON %WRITE(X) <SCHREIB: >
%ON %TERM <%DISPLAY %.SCHREIB;%STOP>
%R

BS2000  F O R 1 : FORTRAN PROGRAM "ONTST    " ENDED PROPERLY AT 13:41:13
CPU - TIME USED :       0.1885 SECONDS
ELAPSED TIME    :     507.8210 SECONDS
CURRENT PC: 0006952E    CSECT: ITOTRM@   ****************************************
%.SCHREIB                        =          5

STOPPED AT V'6952E' = ITOTRM@  + #'2E'   ,  EVENT: TERM (NORMAL,PROGRAM,NODUMP)
```

   Diesmal wird nur der Durchlaufzähler %.SCHREIB als Subkommando zum %WRITE angelegt. Er wird bei jedem Durchlauf um eins erhöht. Bei Programmende, Ereignis %TERM, wird das Programm angehalten, und der Durchlaufzähler und eine STOP-Meldung werden ausgegeben.

%OUT

Mit %OUT können Sie für die Ausgabe-Kommandos %DISASSEMBLE, %DISPLAY, %HELP, %SDUMP und %TRACE festlegen, über welche Medien die Daten ausgegeben werden und ob in der Ausgabe Zusatzinformationen enthalten sein sollen.

– Mit *ziel-kommando* bezeichnen Sie das Ausgabe-Kommando, für das Sie *medium-u-menge* festlegen wollen.

– Mit *medium-u-menge* geben Sie an, welche Ausgabe-Medien verwendet und ob Zusatzinformationen ausgegeben werden sollen.

```
Kommando          Operand
-------------------------------------------------------------------------------
%OUT              [ziel-kommando          [medium-u-menge][,...]  ]
-------------------------------------------------------------------------------
```

Bei %DISPLAY, %HELP und %SDUMP können Sie einen *medium-u-menge*-Operanden angeben, der für diese Kommandos die Vereinbarungen des %OUT-Kommandos vorübergehend außer Kraft setzt. %DISASSEMBLE und %TRACE haben keinen eigenen *medium-u-menge*-Operanden, ihre Ausgaben können Sie nur über %OUT steuern.

Bevor Sie mit %OUT das Ausgabemedium Datei wählen, müssen Sie die Datei mit %OUTFILE einem Linknamen zuweisen und öffnen; ansonsten legt AID eine Standard-Ausgabedatei mit dem Namen AID.OUTFILE.Fn an.

Die Vereinbarungen mit %OUT gelten, bis sie durch ein neues %OUT-Kommando überschrieben werden oder bis /LOGOFF.

Ein %OUT-Kommando ohne Operanden setzt für alle *ziel-kommandos* den Standardwert T=MAX ein.

%OUT darf nur als Einzelkommando eingegeben werden, es darf nicht in einer Kommandofolge oder einem Subkommando stehen.

%OUT verändert den Programmzustand nicht.

ziel-kommando

bezeichnet das Kommando, für das die Vereinbarungen gelten sollen. Jeweils eines der folgenden Kommandos kann hier angegeben werden:

```
{%D[IS]A[SSEMBLE]}
{%D[ISPLAY]      }
{%H[ELP]         }
{%SD[UMP]        }
{%T[RACE]        }
```

medium-u-menge

legt für *ziel-kommando* fest, über welches oder über welche Medien die Ausgabe erfolgen soll und ob AID Zusatzinformationen ausgeben soll über den AID-Arbeitsbereich, die aktuelle Unterbrechungsstelle und die auszugebenden Daten.

Wird der *medium-u-menge*-Operand nicht angegeben, so gilt für *ziel-kommando* der Standardwert T=MAX.

```
medium-u-menge-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - -

{ I  } { MIN  }
{ H  } { MAX  }
{ Fn } { XMAX }
{ P  } { XFLAT}
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

medium-u-menge ist ausführlich im AID-Basishandbuch, Kapitel 7 beschrieben.

T Terminal-Ausgabe
H Hardcopy-Ausgabe
Fn Datei-Ausgabe
P Ausgabe nach SYSLST

AID berücksichtigt die Modi XMAX und XFLAT für die Ausgabe des %OUT-Protokolls nicht. Statt dessen generiert es die Standardausgabe (T=MAX).

MAX Ausgabe mit Zusatzinformationen.

MIN Ausgabe ohne Zusatzinformationen.

XMAX Festlegung des Modus XMAX für das entsprechende Kommando %DISASSEMBLE, %DISPLAY, %HELP, %SDUMP oder %TRACE.

XFLAT Festlegung des Modus XFLAT für das entsprechende Kommando %DISASSEMBLE, %DISPLAY, %HELP, %SDUMP oder %TRACE.

Beispiele

1. `%OUT %SDUMP T=MIN,F1=MAX`
   Datenausgaben des Kommandos %SDUMP sollen auf dem Terminal in Kurzform und parallel dazu in die Datei mit dem Linknamen F1 mit Zusatzinformationen ausgegeben werden.

2. `%OUT %TRACE F1=MAX`
   Das TRACE-Protokoll mit Zusatzinformationen wird nur in die Datei mit dem Linknamen F1 ausgegeben.

3. `%OUT %TRACE`
   Für das Kommando %TRACE wird festgelegt, dass bisherige Vereinbarungen zur Ausgabe von Daten gelöscht werden und dass der Standardwert T=MAX gilt.

%OUTFILE

Mit %OUTFILE können Sie den AID-Linknamen F0 bis F7 Ausgabedateien zuweisen oder Ausgabedateien schließen. In diese Dateien können Sie die Ausgaben der Kommandos %DISASSEMBLE, %DISPLAY, %HELP, %SDUMP und %TRACE schreiben lassen, indem Sie im *medium-u-menge*-Operanden von %OUT, %DISPLAY , %HELP oder %SDUMP den entsprechenden Linknamen angeben. Falls eine Datei noch nicht existiert, wird sie durch AID katalogisiert und geöffnet.

– Mit *link* wählen Sie den Linknamen für die Datei aus, die katalogisiert und geöffnet oder geschlossen werden soll.

– Mit *datei* weisen Sie dem Linknamen einen Dateinamen zu.

Kommando	Operand
%OUTFILE	[link [= datei]]

Ohne den *datei*-Operanden veranlassen Sie AID, die mit *link* bezeichnete Datei zu schließen. So können Sie auch während des Testverlaufs einen Zwischenstand der Datei ausdrucken.

Ein %OUTFILE ohne Operanden schließt alle offenen AID-Ausgabedateien.
Wenn Sie eine AID-Ausgabedatei nicht explizit mit %OUTFILE schließen, bleibt sie geöffnet bis /LOGOFF.

Ohne Verwendung von %OUTFILE haben Sie zwei Möglichkeiten, AID-Ausgabedateien einzurichten und zuzuweisen:

1. Sie geben ein /SET-FILE-LINK-Kommando für einen noch nicht belegten Linknamen Fn. Dann eröffnet AID diese Datei beim ersten Ausgabekommando für diesen Linknamen.
2. Sie überlassen AID das Einrichten, Zuweisen und Eröffnen. Dann verwendet AID Standard-Datei-Namen mit folgendem Aufbau: AID.OUTFILE.Fn entsprechend dem Linknamen *Fn*.

%OUTFILE verändert den Programmzustand nicht.

link

bezeichnet einen der AID-Linknamen für Ausgabedateien und hat das Format:

Fn, wobei n eine Zahl mit einem Wert $0 \leq n \leq 7$ ist.

Die mit %MOVE erzeugten REPs werden stets in die Ausgabedatei mit dem Linknamen F6 geschrieben (siehe %AID und %MOVE).

datei

gibt den vollqualifizierten Dateinamen an, mit dem AID die Ausgabedatei katalogisiert und öffnet.
Mit einem %OUTFILE ohne *datei*-Operand wird die dem Linknamen Fn zugewiesene Datei geschlossen.

%QUALIFY

Mit %QUALIFY definieren Sie Qualifikationen, auf die Sie sich im Adress-Operanden eines anderen Kommandos durch Voranstellen eines Punktes beziehen können.
Diese verkürzte Schreibweise ist immer dann sinnvoll, wenn Sie mehrfach Adressen ansprechen wollen, die nicht im aktuellen AID-Arbeitsbereich liegen.

– Mit *vorqualifikation* legen Sie die Qualifikationen fest, die Sie in nachfolgenden Kommandos durch Voranstellen eines Punktes übernehmen möchten.

Kommando	Operand
%Q[UALIFY]	[vorqualifikation]

Eine mit %QUALIFY vereinbarte *vorqualifikation* gilt, bis sie durch ein %QUALIFY mit neuer *vorqualifikation* überschrieben wird, bis sie durch ein %QUALIFY ohne Operanden aufgehoben wird oder bis /LOGOFF.

Bei der Eingabe eines %QUALIFY wird das Kommando nur syntaktisch überprüft. Ob dem angegebenen Linknamen eine Dump-Datei zugewiesen bzw. ob die angegebene Programmeinheit geladen oder in den LSD-Sätzen verzeichnet ist, wird erst bei der Ausführung darauf folgender Kommandos geprüft, wenn die Angaben aus *vorqualifikation* in die Adressierung einbezogen werden.

Die Vereinbarungen des %QUALIFY werden nur von nachfolgend eingegebenen Kommandos übernommen. Auf die Subkommandos in %CONTROL, %INSERT und %ON, die vorher eingegeben wurden, hat ein neuer %QUALIFY keine Auswirkungen, auch wenn die Subkommandos erst danach ausgeführt werden.

Sowohl bei der Eingabe des %QUALIFY wie auch bei der Ersetzung in einem Adress-Operanden muss dieselbe Einstellung mit %AID LOW={ON|OFF} gelten.

%QUALIFY darf nur als Einzelkommando eingegeben werden, es darf nicht in einer Kommandofolge oder einem Subkommando stehen.

%QUALIFY verändert den Programmzustand nicht.

vorqualifikation

bezeichnet eine Basis-Qualifikation oder eine PROG-Qualifikation oder beide Qualifikationen, die dann durch einen Punkt getrennt werden müssen.

In den Adress-Operanden nachfolgender AID-Kommandos können Sie sich durch Voranstellung eines Punktes auf die im %QUALIFY definierte *vorqualifikation* beziehen.

```
vorqualifikation-OPERAND  - - - - - - - - - - - - - - - - - - - - - - - - - - -

                 ⎧VM ⎫
[E=              ⎨   ⎬[.]][PROG=program-name]
                 ⎩Dn ⎭

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

E={VM|Dn}
: geben Sie an, wenn Sie eine andere als die aktuelle Basis-Qualifikation verwenden wollen (siehe %BASE).

PROG=program-name
: bezeichnet eine Programmeinheit.

Beispiele

1.
```
%QUALIFY E=D1.PROG=SORT
%D .CFELD(1)
```

   Durch die *vorqualifikation* hat der %DISPLAY dieselbe Bedeutung wie das folgende, ausgeschriebene %DISPLAY-Kommando:

```
%D E=D1.PROG=SORT.CFELD(1)
```

2.
```
%QUALIFY PROG=SUB
%SET .A INTO .B
```

   Durch die *vorqualifikation* hat der %SET dieselbe Bedeutung wie das folgende, ausgeschriebene %SET-Kommando:

```
%SET PROG=SUB.A INTO PROG=SUB.B
```

%REMOVE

Mit %REMOVE heben Sie die Testvereinbarungen der Kommandos %CONTROLn, %INSERT oder %ON auf.

– Mit *ziel* legen Sie fest, ob AID für ein angegebenes Kommando alle wirksamen Vereinbarungen aufheben soll, oder ob nur ein bestimmter Testpunkt, ein bestimmtes Ereignis oder ein Subkommando gelöscht werden soll.

Kommando	Operand
%REM[OVE]	ziel

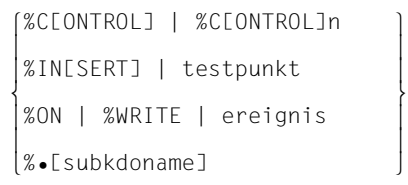
Steht ein %REMOVE in einem Subkommando, der dieses Subkommando oder die zugehörige Überwachungsbedingung (*testpunkt*, *ereignis* oder *kriterium*) löscht, werden nachfolgende *subkdo*-Kommandos nicht mehr ausgeführt. Diese Angabe ist folglich nur als letztes Kommando in einem Subkommando sinnvoll.

%REMOVE verändert den Programmzustand nicht.

ziel

bezeichnet entweder ein Kommando, für das alle Vereinbarungen gelöscht werden sollen, oder einen *testpunkt*, der gelöscht werden soll, oder ein *ereignis*, das nicht mehr überwacht werden soll, oder das zu löschende Subkommando. Liegt *ziel* in einem geschachtelten Subkommando und ist somit noch nicht eingetragen, kann es auch nicht gelöscht werden.

ziel-OPERAND



%C[ONTROL]
Die Vereinbarungen aller eingetragenen %CONTROLn werden gelöscht.

%C[ONTROL]n
Der %CONTROLn mit der angegebenen Nummer ($1 \leq n \leq 7$) wird gelöscht.

%IN[SERT]
Alle eingetragenen Testpunkte werden gelöscht.

testpunkt
Der angegebene *testpunkt* wird gelöscht. *testpunkt* wird wie bei %INSERT angegeben. Innerhalb des eigenen Subkommandos kann der Testpunkt auch mit %REMOVE %PC-> gelöscht werden, da der Befehlszähler (%PC) zu diesem Zeitpunkt die Adresse des Testpunkts enthält.

%ON
Alle eingetragenen Ereignisse werden gelöscht.

%WRITE
Das *write-ereignis* wird gelöscht.

ereignis
Das angegebene *ereignis* wird gelöscht. *ereignis* wird wie bei %ON mit einem Schlüsselwort angegeben. Die *ereignis*-Tabelle mit den Schlüsselwörtern und den Erläuterungen der einzelnen Ereignisse steht in der %ON-Beschreibung.

Für die Ereignisse %ERRFLG(zzz), %SVC(zzz) und %LPOV(zzz) gilt:

%REMOVE *ereignis(zzz)* löscht nur das Ereignis mit der angegebenen Nummer. %REMOVE *ereignis* ohne Angabe einer Nummer löscht alle Ereignisse der entsprechenden Gruppe.

%**.**[subkdoname]
löscht das Subkommando eines %CONTROLn oder %INSERT mit *subkdoname*.

%**.** ist die Kurzform für einen Subkommando-Namen, die nur innerhalb des Subkommandos verwendet werden kann. %REMOVE %. löscht folglich das gerade ausgeführte Subkommando.

Da %CONTROLn nicht gekettet werden kann, wird auch der zugehörige %CONTROLn gelöscht. Das Löschen des Subkommandos entspricht folglich einer Löschung des %CONTROLn mit Angabe der Nummer.

An einem *testpunkt* des Kommandos %INSERT können dagegen mehrere Subkommandos gekettet sein. Mit %REMOVE %.[subkdoname] löschen Sie ein einzelnes Subkommando aus einer Kette, weitere Subkommandos zum selben *testpunkt* bleiben dagegen bestehen (siehe AID-Basishandbuch, Kapitel 5). War zu dem *testpunkt* nur das Subkommando mit *subkdoname* eingetragen, so wird auch der *testpunkt* gelöscht.

Für %ON ist %REMOVE %.[subkdoname] nicht zugelassen.

Beispiele

1.
```
%C1 %CALL <CTL1: %D %.>
%REM %C1
%REM %.CTL1
```

   Beide %REMOVE-Kommandos haben dieselbe Wirkung: %C1 wird gelöscht.

2.
```
%IN L'100' <SUB1: %D I,J,IFELD(I,J)>
%IN L'100' <SUB2: %D %PC; %REM %.>
.
.
%REM L'100'
```

   Wenn der Testpunkt L’100’ erreicht wird, wird der Program Counter ausgegeben. Danach wird das Subkommando SUB2 gelöscht. Dieses Subkommando wird also nur ein einziges Mal ausgeführt. Dann werden die Indices I und J und das zugehörige Feldelement IFELD(I,J) ausgegeben, und das Programm wird fortgesetzt. Immer wenn der Programmablauf an den Testpunkt L’100’ kommt, wird Subkommando SUB1 ausgeführt. Mit %REM L’100’ wird später der Testpunkt gelöscht. Ein %REM %.SUB1 würde dasselbe bewirken, denn zum Testpunkt L’100’ ist nur noch dieses Subkommando eingetragen.

%RESUME

Mit %RESUME starten Sie das geladene Programm oder setzen es an der unterbrochenen oder der mit %JUMP vereinbarten Stelle fort. Das Programm läuft ohne Ablaufverfolgung.

Wurde das Programm während eines %TRACE angehalten, wird mit %RESUME der %TRACE abgebrochen und nicht mehr zu Ende geführt. Ein %CONTINUE hingegen führt einen unterbrochenen %TRACE fort.

Kommando	Operand
%R[ESUME]	

Steht %RESUME in einer Kommandofolge oder in einem Subkommando, werden nachfolgende Kommandos nicht mehr ausgeführt.
Steht in einem Subkommando nur das Kommando %RESUME, wird der Durchlaufzähler erhöht und ein eventuell aktiver %TRACE gelöscht.

%RESUME verändert den Programmzustand.

%SDUMP

Mit %SDUMP geben Sie einen symbolischen Dump aus: einzelne Datenelemente, alle Datenelemente der aktuellen Aufrufhierarchie oder die Programmnamen der aktuellen Aufrufhierarchie werden ausgegeben. Die aktuelle Aufrufhierarchie reicht von der Unterprogrammebene, auf der das Programm unterbrochen wurde, über die durch CALL-Anweisungen aufgerufenen Unterprogramme bis zum Hauptprogramm.
Die Ausgabe erfolgt über SYSOUT, SYSLST oder in eine katalogisierte Datei.

– Mit *dump-bereich* bezeichnen Sie die Variablen oder Felder, die AID ausgeben soll, oder Sie geben an, dass AID die Programmnamen der aktuellen Aufrufhierarchie ausgeben soll.

– Mit *medium-u-menge* geben Sie an, welche Ausgabemedien AID verwenden und ob Zusatzinformationen ausgegeben werden sollen. Mit diesem Operanden setzen Sie eine mit %OUT getroffene Vereinbarung für das aktuelle %SDUMP-Kommando außer Kraft.

Kommando	Operand
%SD[UMP]	[[dump-bereich][,...] [medium-u-menge][,...]]

Daten können mit %SDUMP erst nach der Initialisierung angesprochen werden, d.h. bei Erreichen der ersten ausführbaren Anweisung einer Programmeinheit. Dazu geben Sie die beiden folgenden Kommandos ein:

```
%INSERT PROG=program-name.program-name
%RESUME
```

Befinden sich in der Hierarchie Programmeinheiten, für die es keine LSD-Sätze gibt, auch nicht in einer PLAM-Bibliothek, so können Sie das Kommando %SDUMP nur einzeln für die Programmeinheiten geben, für die LSD-Sätze geladen wurden oder aus einer PLAM-Bibliothek nachladbar sind (siehe %SYMLIB).

%SDUMP ohne Operanden gibt alle Datenelemente der aktuellen Aufrufhierarchie aus. Daten, die mehrfach definiert sind, werden auch mehrfach ausgegeben.

%SDUMP %NEST gibt die Namen aller Programmeinheiten der aktuellen Aufrufhierarchie aus.

dump-bereich können Sie bis zu 7mal wiederholen.

Sie können mit diesem Kommando im geladenen Programm oder in einer Dump-Datei arbeiten.

%SDUMP verändert den Programmzustand nicht.

dump-bereich

beschreibt, welche Informationen AID ausgeben soll. AID kann die Programmnamen der aktuellen Aufrufhierarchie, alle Daten der aktuellen Aufrufhierarchie, alle Daten einer Programmeinheit oder einzelne Datenelemente ausgeben. Datenelemente bereitet AID entsprechend der Definition im Quellprogramm auf. Passt der Inhalt nicht zum definierten Speichertyp, wird die Ausgabe mit einer Fehlermeldung abgelehnt.

datenname, der in mehreren Programmeinheiten der aktuellen Aufrufhierarchie definiert ist, wird auch mehrmals ausgegeben, es sei denn, *dump-bereich* wurde mit einer Qualifikation eingeschränkt.
Für *datenname*, der in den LSD-Sätzen nicht verzeichnet ist, gibt AID eine Fehlermeldung aus. Nachfolgende *dump-bereiche* desselben Kommandos werden ordnungsgemäß ausgegeben.

```
dump-bereich-OPERAND  - - - - - - - - - - - - - - - - - - - - - - - - - - - -

{      {VM }     {                                  }
{[•][E={   }[•]] {[PROG=program-name[•]] [datenname]}
{      {Dn }     {                                  }
{                {%NEST                             }

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

•
: Steht der Punkt an führender Stelle, ist er das Kennzeichen für eine *vorqualifikation*. Sie muss mit einem vorhergehenden %QUALIFY definiert worden sein.
Aufeinanderfolgende Qualifikationen werden durch einen Punkt getrennt. Außerdem muss zwischen der letzten Qualifikation und dem anschließenden Operandenteil ein Punkt stehen.

E ={VM | Dn}
: Eine explizite Basis-Qualifikation geben Sie nur an, wenn die aktuelle Basis-Qualifikation für *dump-bereich* nicht gelten soll. Wenn Sie nur eine Basis-Qualifikation angeben, erhalten Sie alle Daten der entsprechenden Aufrufhierarchie.

PROG=program-name
: Eine PROG-Qualifikation ist erforderlich, wenn *dump-bereich* nur für die angegebene Programmeinheit gelten soll. Endet die Definition von *dump-bereich* mit einer PROG-Qualifikation, gibt AID alle Datenelemente dieser Programmeinheit aus.

datenname
: ist der im Quellprogramm definierte Name einer Konstanten, einer Variablen, eines Feldes oder eines Feldelements.
datenname ist eine maximal 15stellige alphanumerische Zeichenfolge.

Ist *datenname* der Name eines Felds, kann er wie in einer FORTRAN-Anweisung indiziert werden. Wenn Sie *feldname* ohne Indexliste angeben, werden alle Feldelemente ausgegeben.

feldname (index1[, index2][, ...])
index gibt die Position innerhalb eines Feldes an. Es sind so viele Indizes erforderlich, wie in einer FORTRAN-Anweisung zum Zugriff angegeben werden müssen. Mehrere Indizes müssen durch Komma getrennt werden.
index kann folgendermaßen angegeben werden:

```
{n                       }
{datenname               }
{arithmetischer ausdruck }
```

%NEST
ist ein AID-Schlüsselwort, das die Ausgabe der aktuellen Aufrufhierarchie veranlasst. Für die unterste Hierarchiestufe gibt AID den Namen der Programmeinheit und die Nummer der Anweisung aus, an der das Programm unterbrochen wurde. Für die höheren Hierarchiestufen gibt AID den Namen des aufrufenden Programms und die Nummer der CALL-Anweisung aus.

medium-u-menge

legt fest, über welches oder über welche Medien die Ausgabe erfolgen soll und ob AID Zusatzinformationen ausgeben soll. Ohne diesen Operanden und ohne eine Vereinbarung mit dem %OUT-Kommando arbeitet AID mit dem Standardwert T = MAX.

```
medium-u-menge-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - -

{T  }   {MIN  }
{H  }   {MAX  }
{Fn } = {XMAX }
{P  }   {XFLAT}

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

medium-u-menge ist ausführlich im AID-Basishandbuch, Kapitel 7 beschrieben.

T Terminal-Ausgabe
HHardcopy-Ausgabe
FnDatei-Ausgabe
PAusgabe nach SYSLST

MAX	Ausgabe mit Zusatzinformationen.
MIN	Ausgabe ohne Zusatzinformationen.

XMAX Ausgabe wie bei MAX, jedoch erweitert um Typ-Informationen:
Zusätzlich geht jedem Datenelement ein Typ-Tag voraus, das Typ, Größe und Ausgabe-Format dieses Datenelements definiert. Syntax des Typ-Tags:
`<data-type(memory-size-in-bytes),output-format>`

XFLAT Ausgabe wie bei XMAX, jedoch mit folgenden Einschränkungen:
Für strukturierte Datentypen wird nur die jeweils oberste Strukturebene ausgegeben. Bei langen Daten (z.B. langen Strings oder Arrays) werden nur die ersten Elemente ausgegeben.

Datentypen

Wenn Sie den Operandenwert XMAX oder XFLAT angegeben haben, generiert AID die Ausgabe wie bei MAX erweitert um die folgenden Typ-Tags:

```
<INT(size),D>
int-name = int-value
```

size	Speicherlänge in Bytes.
int-name	bezeichnet ein Element vom Typ Integer.
int-value	Dezimalzahl (D); Wert von *int-name*.

```
<POINTER(size),X>
pointer-name = pointer-value
```

size	Speicherlänge in Bytes.
pointer-name	bezeichnet ein Elementvom Typ Pointer.
pointer-value	Hexadezimalzahl (X); Wert von *pointer-name*.

```
<FLOAT(size),E>
float-name = float-value
```

size	Speicherlänge in Bytes.
float-name	bezeichnet ein Element vom Typ Floating Point Number.
float-value	Gleitkommazahl dargestellt als Dezimalbruch mit Exponent (E); Wert von *float-name.*

```
<CHARS(size),C>
chars-name = |string|
```

size	Speicherlänge in Bytes.
chars-name	bezeichnet ein Element vom Typ String, also Array vom Typ Character.
string	Folge von abdruckbaren Zeichen (C); Wert von *chars-name;* Nicht abdruckbare Zeichen werden als hexadezimaler Wert dargestellt.

Ist *string* länger als 80 Zeichen, dann werden bei XFLAT nur die ersten 72 Zeichen ausgegeben, gefolgt von drei Punkten ... , um die Unvollständigkeit der Ausgabe anzuzeigen. Siehe auch Hinweis 1 am Ende der Liste.

```
<UNSIGN(size),D>
unsign-name = unsign-value
```

size	Speicherlänge in Bytes.
unsign-name	bezeichnet ein Element vom Typ Integer ohne Vorzeichen (unsigned).
unsign-value	Dezimalzahl (D): Wert von *unsign-name*.

```
<ADDR(size),X>
addr-name = addr-value
```

size	Speicherlänge in Bytes.
addr-name	bezeichnet ein Element einer relativen oder absoluten Speicheradresse.
addr-value	Hexadezimalzahl (X), Wert von *addr-name*.

```
<AREA(size),X>
area-name = area-value
```

size	Speicherlänge in Bytes.
area-name	bezeichnet einen Primärspeicherbereich.
area-value	Speicherabzug im Dump-Format, Wert von *area-name*. Das Dump-Format besteht aus hexadezimaler (X) und alphanumerischer Darstellung, nicht abdruckbare Zeichen werden in der alphanumerischen Darstellung als `|.|` dargestellt. Ist die Ausgabe länger als 80 Zeichen, dann werden bei XFLAT nur die ersten 4 hexadezimalen Wörter ausgegeben (ggf. auch weniger). Die alphnumerische Darstellung enthält maximal 16 Zeichen (bei UTF16: 8 Zeichen) gefolgt vom String ETC. Siehe auch Hinweis 1 am Ende der Liste.

```
<ARRAY(size),type | STRUCT>
array-name (dimension)
(a1) value1 (a2) value2 (a3) value3 ...
```

size	Primärspeicher-Länge in Bytes.
type	Datentyp (CHARS, INT, FLOAT,...), wenn das Array aus einem bestimmten Datentyp besteht.
STRUCT	das Array besitzt eine komplexe, aus unterschiedlichen Datentypen zusammengesetzte Struktur.
array-name	bezeichnet ein Element vom Typ Array.
dimension	die Dimensionen des Arrays.

(*a1*) *value1* (*a2*) *value2* (*a3*) *value3* ...	*a1*, *a2*, *a3*, ... bezeichnet die Unterelemente des Arrays, *value1*, *value2*, *value3*, ... deren Werte. Die Darstellung der Werte hängt vom jeweiligen Datentyp ab. Bei XMAX werden alle Unterelemente ausgegeben. Bei XFLAT werden keine Unterelemente ausgegeben, siehe auch Hinweis 1. Details zu Array-Bereichen siehe Hinweis 2.

```
<STRUCT(size)>
level struct-name
sub-elements
```

size	Speicherlänge in Bytes.
level	Schachtelungstiefe der Struktur oder eines Struktur-Elementes (01, 02, 03, ...). 01 steht für die oberste Ebene.
struct-name	bezeichnet ein Element vom Typ Struktur.
sub-elements	weitere Elemente, die in der Struktur enthalten sind. Bei XMAX werden alle Elemente ausgegeben. Bei XFLAT wird nur ein Teil der Elemente, siehe Abschnitt „Strukturen mit XFLAT“. Siehe auch Hinweis 1 am Ende der Liste.

Hinweise

1. Um den gesamten Inhalt eines Strings, einer Struktur oder eines Arrays aufgeteilt auf mehrere Zeilen abzufragen, verwenden Sie folgende Syntax:

   ```
   %SDUMP name {T | H | Fn | P} = {XMAX | MAX}
   ```

2. Um den Inhalt der Array-Elemente innerhalb des bestimmten Bereichs abzufragen, verwenden Sie folgende Syntax:

   ```
   %SDUMP name [from:to] {T | H | Fn | P} = {XMAX | XFLAT | MAX}
   ```

Strukturen mit XFLAT

Für Strukturen generiert AID verschiedene XFLAT-Datenausgaben abhängig davon, ob das %SDUMP-Kommando Datenoperanden enthält oder nicht.

- %SDUMP ohne Datenoperand

  ```
  %SDUMP {T | H | Fn | P} = XFLAT
  ```

  Nur der Typ-Tag und der Name werden ausgegeben (Ebene 01). Die Ausgabe der Struktur-Elemente entfällt.

- %SDUMP mit einer Struktur als Operand

  ```
  %SDUMP structure-name {T | H | Fn | P} = XFLAT
  ```

Der Struktur-Name und die Struktur-Elemente werden ausgegeben (Ebene 02). Elemente mit elementaren Typen werden normal ausgegeben, Elemente mit Array-Typ mit Namen und Dimension, Elemente mit Struktur-Typ nur mit ihrem Namen. Dabei geht jedem Element ein Typ-Tag voraus. Der Name wird durch eine Zahl, die Schachtelungstiefe, erweitert.

- %SDUMP mit einer Unterstruktur als Operand

  ```
  %SDUMP structure-name.substruct-name {T | H | Fn |P} = XFLAT
  ```

  Gibt zusätzlich die Struktur-Elemente der Unterstruktur aus (Ebene 03)

  Es können auch weitere Schachtelungstiefen angegeben werden, indem die weiteren Unterstruktur-Namen durch einen Punkt verkettet werden:

  ```
  structure-name.substruct1-name.substruct2-name.substruct3-name. ....
  ```

Um den gesamten Inhalt einer Struktur und ihrer Unterstrukturen abzufragen, verwenden Sie XMAX statt XFLAT.

Beispiele

Die Übersetzungsliste zu beiden Beispielen finden Sie im Abschnitt „Quellprogrammliste“ auf Seite 131.

1. Im Subkommando des %INSERT wird mit %SD ein symbolischer Dump aller Programmeinheiten der aktuellen Aufrufhierarchie angefordert. Alle Datenelemente der Programmeinheiten TAUSCH, SORT und B1 werden ausgegeben. Für *medium-u-menge* wird der Standardwert T=MAX eingesetzt. Im Subkommando steht außerdem ein %STOP. Deswegen bleibt das Programm nach der Ausgabe aller Daten unterbrochen, und AID schreibt eine STOP-Meldung mit der Nummer der Anweisung und dem Namen der Programmeinheit zur aktuellen Unterbrechungsstelle.

```
/LOAD-PROG FROM-FILE=*MOD(LIB=*OMF),TEST-OPT=AID
%  BLS0001 DLL VER 823
%  BLS0517 MODULE 'B1' LOADED
/%IN PROG=TAUSCH.S'5' <%SD; %STOP>
/%R
BS2000 F O R 1 : FORTRAN PROGRAM "B1"
STARTED ON 91-06-28 AT 15:18:33
   CFELD UNSORTIERT
Berta
Hansi
Anton
Erich
Ilona
Diane
Carlo
Franz
Georg
**ITN: #'000000CB'***TSN:1114************************************************************
SRC_REF:     5  SOURCE: TAUSCH     PROC: TAUSCH  *********************************
CHAR            = |Ilona|

SRC_REF:    33  SOURCE: SORT       PROC: SORT  **********************************
CFELD( 1: 9)
(  1) |Berta|  (  2) |Hansi|  (  3) |Anton|  (  4) |Erich|  (  5) |Ilona|
(  6) |Diane|  (  7) |Carlo|  (  8) |Franz|  (  9) |Georg|

IMITTE          = |Ilona|

L( 1: 5)
(  1)           1  (  2)           0  (  3)           0  (  4)           0
(  5)           0

R( 1: 5)
(  1)           9  (  2)           0  (  3)           0  (  4)           0
(  5)           0

Z               =           0
LI              =           1
RI              =           9
I               =           5
J               =           9


SRC_REF:    11  SOURCE: B1         PROC: B1  ************************************
CFELD( 1: 9)
(  1) |Berta|  (  2) |Hansi|  (  3) |Anton|  (  4) |Erich|  (  5) |Ilona|
(  6) |Diane|  (  7) |Carlo|  (  8) |Franz|  (  9) |Georg|

K               =          10
STOPPED AT SRC_REF: 5 , SOURCE: TAUSCH , PROC: TAUSCH
```

2. Mit dem Kommando %SD %NEST wird die aktuelle Aufrufhierarchie angefordert:

   ```
   %SD %NEST
   ```

   AID gibt zuerst die Nummer der Anweisung in der Programmeinheit der untersten Hierarchiestufe aus, auf der das Programm unterbrochen wurde. Danach werden die Nummern der CALL-Anweisungen ausgegeben, mit der die Programmeinheiten höherer Hierarchiestufen (hier: Unterprogramm SORT und Hauptprogramm B1) verlassen wurden.

   ```
   SRC_REF:     5  SOURCE: TAUSCH  PROC: TAUSCH  ***********************************
   SRC_REF:    33  SOURCE: SORT    PROC: SORT  *************************************
   SRC_REF:    11  SOURCE: B1      PROC: B1  ***************************************
   ```

3. `%SDUMP [..] {T | H | Fn | P} = XMAX`

   AID generiert die vollständige Ausgabe, die der Ausgabe mit XFLAT ähnelt. Zusätzlich umfasst die Ausgabe mit XMAX den vollständigen Inhalt von langen Strings, Arrays und Strukturen.

4. `%SD[UMP] T=XFLAT` generiert beispielsweise folgende Ausgabe. Wenn große Datenmengen oder komplexe Datentypen ausgegeben werden, kürzt das Kommando bestimmte Datenwerte oder Datenelemente oder entfernt diese.

   ```
   <INT(4),D> int_var = 5

   <FLOAT(8),E> float_var = -0.123456789

   <CHARS(1),C> dollar = |$|

   <CHARS(72),C> char_array =
   |Dies ist ein 72 Zeichen langer String.............................|

   <CHARS(100),C> char_string =
   |Dies sind die ersten 75 Zeichen eines 100 Zeichen langen
   Strings.........|

   <STRUCT(128)> person

   <ARRAY(1280),STRUCT> person_array(1:10)

   <BITINT(1),D> bit_integer = 5
   ```

Beispiel zur selektiven Auswahl von Komponenten komplexer Datenstrukturen:

```
struct { struct {int x; char y;} inner[100];
             struct {float u, v;}  *pointer;
             } outer [10];

%AID low=on
%OUT %SDUMP T=XFLAT

%SD
<ARRAY(8040),STRUCT> outer( 0: 9)

%SD outer
<ARRAY(8040),STRUCT> outer( 0: 9)

%SD outer[5]
<STRUCT(804)> 01 outer( 5)
<ARRAY(800),STRUCT>  02 inner( 0:99)
<POINTER(4),X>       02 pointer = 0103B700

%SD outer[5].inner[15]
<STRUCT(5)> 02 outer.inner( 5, 15)
<INT(4),D>    03  x = -2130463928
<CHARS(1),C>  03  y = |.|

%SD outer[5].inner[15] T=XMAX
<STRUCT(5)> 02 outer.inner( 5, 15)
<INT(4),D>    03  x = -2130463928
<CHARS(1),C>  03  y = |.|

%SD outer[5].inner[15:16]
<ARRAY(16),STRUCT> outer.inner ( 5) ( 15: 16)

%SD outer[5].inner[15:16] T=XMAX
<ARRAY(16),STRUCT> outer.inner ( 5) ( 15: 16)
<STRUCT(5)> 02 inner(15)
<INT(4),D>    03  x = -2130463928
<CHARS(1),C>  03  y = |.|
<STRUCT(5)> 02 inner(16)
<INT(4),D>    03  x = 0
<CHARS(1),C>  03  y = |%|
```

%SET

Mit %SET übertragen Sie Speicherinhalte oder AID-Literale auf Speicherstellen im geladenen Programm. Vor der Übertragung werden die Speichertypen von *sender* und *empfänger* auf Verträglichkeit geprüft. Der Inhalt von *sender* wird in den Speichertyp von *empfänger* konvertiert.

– Mit *sender* bezeichnen Sie eine Variable oder ein Feldelement, einen logischen Wert, eine Länge, eine Adresse, einen Durchlaufzähler, ein AID-Register oder ein AID-Literal. *sender* kann im virtuellen Speicher des geladenen Programms oder in einer Dump-Datei liegen.

– Mit *empfänger* bezeichnen Sie eine Variable oder ein Feldelement, einen Durchlaufzähler oder ein AID-Register, das überschrieben werden soll. *empfänger* kann nur im virtuellen Speicher des geladenen Programms liegen.

Kommando	Operand
%S[ET]	sender INTO empfänger

Im Gegensatz zum %MOVE überprüft AID beim %SET vor der Übertragung, ob der Speichertyp von *empfänger* mit dem von *sender* verträglich ist und ob der Inhalt von *sender* zu seinem Speichertyp passt. Andernfalls lehnt AID die Übertragung ab und gibt eine Fehlermeldung aus.

Ist *sender* länger als *empfänger*, wird er entsprechend seinem Speichertyp links oder rechts abgeschnitten, und AID gibt eine Warnung aus. *sender* und *empfänger* können sich überlappen. Bei der numerischen Übertragung wird *sender* bei Bedarf in den Speichertyp von *empfänger* konvertiert, und der Inhalt von *sender* wird werterhaltend in *empfänger* abgelegt. Passt der Wert nicht vollständig in *empfänger*, wird eine Warnung ausgegeben.

Die Übertragung mit %SET entspricht insoweit den Regeln der FORTRAN-Zuweisungs-Anweisung. Folgende Besonderheiten sind jedoch zu berücksichtigen:
Datenelemente vom Typ COMPLEX können mit dem %SET-Kommando nur verändert werden, indem mit *datenname* ._REAL bzw. *datenname* ._IMAG gezielt Real- bzw. Imaginärteil der komplexen Zahl verändert werden.
Wird im %SET-Kommando ein REAL*4-Datenelement als *sender* und ein REAL*8-Datenelement als *empfänger* verwendet, dann werden die rechten 4 Bytes des REAL*8-Datenelements mit binären Nullen aufgefüllt. Dies kann ebenso wie alle anderen Konvertierungen numerischer Werte zu Ungenauigkeiten führen.

Mit einem %SET-Kommando können Sie die FOR1-Anweisung ASSIGN nicht nachvollziehen. %SET wirkt stets wie eine Zuweisungs-Anweisung, auch wenn *empfänger* eine Markenvariable ist.

Welche Speichertypen miteinander verträglich sind und wie übertragen wird, können Sie der Tabelle am Ende der %SET-Beschreibung entnehmen.

Es empfiehlt sich, das Kommando nicht unmittelbar nach dem Laden einzugeben, da Sie Daten und Anweisungen erst dann ohne explizite Qualifikation ansprechen können, wenn das Programm vor der ersten ausführbaren Anweisung steht. Dies erreichen Sie mit der folgenden Kommandofolge:

```
%INSERT PROG=program-name.program-name
%RESUME
```

Neben den hier beschriebenen Operandenwerten können Sie auch die im Handbuch für das Testen auf Maschinencode-Ebene beschriebenen Operandenwerte einsetzen.

Mit `%AID CHECK = ALL` können Sie zur Kontrolle einen Änderungsdialog einschalten, der Ihnen vor Durchführung der Übertragung den alten und neuen Inhalt von *empfänger* zeigt und Ihnen die Möglichkeit zum Abbruch des %SET gibt.

%SET verändert den Programmzustand nicht.

sender INTO empfänger

Für *sender* oder *empfänger* können Sie eine Variable, ein Feldelement oder eine komplexe Speicherreferenz, einen Durchlaufzähler oder ein Register angeben. Symbolische Konstanten, Adressen und Längen von Datenelementen, logische Werte und AID-Literale können Sie nur als *sender* einsetzen. *sender* kann sowohl im virtuellen Speicherbereich des geladenen Programms als auch in einer Dump-Datei liegen; *empfänger* kann dagegen nur im virtuellen Speicherbereich des geladenen Programms liegen.

```
sender-OPERAND  - - - - - - - - - - - - - - - - empfänger-OPERAND - - - - - - - - -

|           {datenname        }     |
|           {L'n'             }     |
|[•][qua•] {S'n'              }     |
|           {schlüsselwort    }     |
|           {kompl-speicherref}     |
|                                   |
|                                   {datenname        }
{{%@}           {datenname        }} INTO [•][qua•] {schlüsselwort    }
{   }([•][qua•]{                  })                {kompl-speicherref}
{%L}           {kompl-speicherref}|
|                                   |
|%L=(ausdruck)                      |
|                                   |
|AID-literal                        |

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

- Steht der Punkt an führender Stelle, ist er das Kennzeichen für eine *vorqualifikation*. Sie muss mit einem vorhergehenden %QUALIFY definiert worden sein.

Aufeinanderfolgende Qualifikationen werden durch einen Punkt getrennt. Außerdem muss zwischen der letzten Qualifikation und dem anschließenden Operandenteil ein Punkt stehen.

qua
Eine Qualifikation geben Sie nur an für Speicherobjekte, die nicht im aktuellen AID-Arbeitsbereich liegen.

E={VM | Dn} für *sender*
E=VM für *empfänger*
geben Sie nur an, wenn für einen Daten- oder Anweisungsnamen oder für eine Source-Referenz oder ein Schlüsselwort die aktuelle Basis-Qualifikation nicht gelten soll (siehe %BASE).
sender kann sowohl im virtuellen Speicher als auch in einer Dump-Datei liegen.
empfänger kann dagegen nur im virtuellen Speicher liegen.

PROG=program-name
geben Sie nur an, wenn Sie einen Daten- oder Anweisungsnamen oder eine Source-Referenz ansprechen, die nicht in der aktuellen Programmeinheit liegt (siehe Kapitel 3).

NESTLEV= level-nummer

level-nummer Nummer einer Ebene in der aktuellen Aufrufhierarchie
Auf *level-nummer* muss *datenname* folgen.
NESTLEV= *level-nummer* geben sie an, wenn Sie einen Datennamen in einer bestimmten Ebene der aktuellen Aufrufhierachie ansprechen wollen. Diese Qualifikation kann nur mit E= kombiniert werden, nicht mit anderen Qualifikationen.

datenname
ist der im Quellprogramm definierte Name einer Konstanten, einer Variablen oder eines Feldelementes. Konstanten können nur als *sender* eingesetzt werden.
datenname ist eine maximal 15stellige alphanumerische Zeichenfolge.

Ein Feld als Gesamtes können Sie weder übertragen noch überschreiben. Sie können nur einzelne Feldelemente übertragen bzw. überschreiben. Um ein Feldelement anzusprechen, indizieren Sie den Namen des Felds wie in einer FORTRAN-Anweisung.

feldname (index1[, index2][, ...])
index gibt die Position innerhalb eines Felds an. Es sind so viele Indizes erforderlich, wie in einer FORTRAN-Anweisung zum Zugriff angegeben werden müssen.
Mehrere Indizes müssen durch Komma getrennt werden.
index kann folgendermaßen angegeben werden:

```
{n                      }
{datenname              }
{arithmetischer ausdruck}
```

L'n'
: ist ein Anweisungsname und bezeichnet die Adresse der ersten ausführbaren FORTRAN-Anweisung nach einer Anweisungsmarke.
n ist eine maximal 5stellige Anweisungsmarke. Führende Nullen dürfen nicht angegeben werden.

S'n'
: ist eine Source-Referenz und bezeichnet die Adresse einer ausführbaren FORTRAN-Anweisung.
n ist die Nummer einer Anweisung; siehe Spalte STMT der Übersetzungsliste.

Anweisungsnamen und Source-Referenzen sind Adresskonstanten und können daher nur als *sender* angegeben werden. Die mit *L'n'* bzw. *S'n'* bezeichnete Adresse wird übertragen.

Beispiel

```
%SET S'5' INTO %2G
```

Die Adresse der Anweisung mit der Nummer 5 wird in das AID-Register %2G geschrieben.

Mit *L'n'->* bzw. *S'n'->* bezeichnen Sie 4 Bytes des an der entsprechenden Adresse stehenden Maschinencodes (siehe AID-Basishandbuch, Abschnitt 6.4).
Die Maschinenbefehle können Sie sich mit %DISASSEMBLE ausgeben lassen, um eventuell eine Längenmodifikation vorzunehmen.
Bei *empfänger* können Sie Anweisungsnamen und Source-Referenzen nur in Verbindung mit dem Pointer-Operator (->) verwenden.

schlüsselwort
: ist ein logischer Wert, ein Durchlaufzähler, der Befehlszähler oder ein Register. Im AID-Basishandbuch, Kapitel 9 sind die impliziten Speichertypen der Schlüsselwörter angegeben.
Die beiden Schlüsselwörter für .TRUE. und .FALSE. können Sie nur als *sender* verwenden. Sie können sie in jede logische Variable des Quellprogramms übertragen.

Vor *schlüsselwort* können Sie nur eine Basis-Qualifikation angeben.

```
%TRUE          ogischer Wert für .TRUE.
%FALSE         logischer Wert für .FALSE.
%•subkdoname   Durchlaufzähler
%•             Durchlaufzähler des gerade aktiven Subkommandos
%PC            Befehlszähler (Program Counter)
%n             Mehrzweckregister, 0 ≤ n ≤ 15
%nD|E          Gleitpunktregister, n = 0,2,4,6
%nQ            Gleitpunktregister, n = 0,4
%nG            AID-Mehrzweckregister, 0 ≤ n ≤ 15
%nDG           AID-Gleitpunktregister, n = 0,2,4,6
```

kompl-speicherref

Folgende Operationen können darin vorkommen (siehe AID-Basishandbuch, Kapitel 6):

- Adressversatz (**.**)
- indirekte Adressierung (->)
- Typmodifikation (%T(datenname), %X, %C, %E, %D, %P, %F, %A)
- Längenmodifikation (%L(...), %L=(ausdruck), %Ln)
- Adressselektion (%@(...))

Mit einer expliziten Typ- oder Längenmodifikation können Sie die Speichertypen von *sender* und *empfänger* einander anpassen. Mit einem Speichertyp unvereinbare Speicherinhalte jedoch werden von AID auch bei der Typmodifikation abgelehnt (siehe AID-Basishandbuch, Abschnitt 6.8).
Nach Adressversatz (**.**) oder Pointer-Operation (->) gehen impliziter Speichertyp und implizite Länge der Ausgangsadresse verloren. An der errechneten Adresse gilt der Speichertyp %X in der Länge 4, falls Sie nicht Typ und Länge explizit angeben.
Für jeden Operanden in einer komplexen Speicherreferenz darf der zugeordnete Speicherbereich durch einen Adressversatz oder eine Längenmodifikation nicht überschritten werden, sonst führt AID das Kommando nicht aus und schreibt eine Fehlermeldung.
Durch die Verbindung von Adressselektion (%@) mit Pointer-Operator (->) verlassen Sie die symbolische Ebene. Nun können Sie die Adresse eines Datenelements verwenden, ohne auf dessen Bereichsgrenzen achten zu müssen.

Beispiel

Die Variablen CFELD und CFELD1 sind vom Typ Character und belegen je 5 Bytes.

Die letzten 2 Bytes von CFELD sowie die 3 anschließenden Bytes sollen nach CFELD1 übertragen werden.
Das folgende Kommando würde AID wegen Bereichsverletzung von CFELD ablehnen:

```
%SET CFELD.3%CL5 INTO CFELD1
```

Richtig müsste es dagegen heißen:

```
%SET %@(CFELD)->.3%CL5 INTO CFELD1
```

%@(...)

Mit dem Adressselektor können Sie die Adresse eines Datenelements oder einer komplexen Speicherreferenz als *sender* verwenden (siehe AID-Basishandbuch, Abschnitt 6.11). Der Adressselektor liefert als Ergebnis eine Adresskonstante.

%L(...)

Mit dem Längenselektor können Sie die Länge eines Datenelements oder einer komplexen Speicherreferenz als *sender* verwenden (siehe AID-Basishandbuch, Abschnitt 6.11). Der Längenselektor liefert als Ergebnis eine Ganzzahl.

Beispiel

```
%SET %L(FELD1) INTO %2G
```

Die Länge von FELD1 wird übertragen.

%L=(ausdruck)

Mit der Längenfunktion können Sie sich den Wert von *ausdruck* berechnen und in *empfänger* abspeichern lassen (siehe AID-Basishandbuch, Abschnitte 6.9 und B6.10). In *ausdruck* können Sie Speicherreferenzen und ganze Zahlen mit den arithmetischen Operatoren (+,-,*,/) verknüpfen.
Die Längenfunktion liefert als Ergebnis eine Ganzzahl.

Beispiel

```
%SET %L=(FELD1) INTO %2G
```

Der Inhalt von FELD1 wird übertragen. FELD1 muss vom Typ Integer sein, sonst gibt AID eine Fehlermeldung aus.

AID-literal

Alle im AID-Basishandbuch, Kapitel 8 beschriebenen AID-Literale können Sie angeben. Bitte beachten Sie die dort beschriebenen Konvertierungen der AID-Literale an den jeweiligen *empfänger*:

```
{C'x...x' | 'x...x'C | 'x...x'} Character-Literal
{X'f...f' | 'f...f'X}      Sedezimal-Literal
{B'b...b' | 'b...b'B}      Binär-Literal
[{±}]n                     Ganzzahl
#'f...f'                   Sedezimalzahl
[{±}]n.m                   Dezimalpunktzahl
[{±}]mantisseE[{±}]exponent Gleitpunktzahl
```

%SET-Tabelle

Die folgende Tabelle gibt eine Übersicht über die zulässigen Kombinationen von Sende- und Empfänger-Typen.

Sender	**Empfänger**				
	INTEGER REAL *complex*._REAL *complex*._IMAG %F %P %A %D	COMPLEX	CHARACTER %C	LOGICAL	%X
INTEGER REAL *complex*._REAL *complex*._IMAG %F %P %A %D ±n #'f...f'	num	*	–	–	bin
±n.m ±mantE±exp	num	*	–	–	–
COMPLEX	*	*	–	–	–
CHARACTER %C C'x...x'	num(1)	–	char	–	bin
LOGICAL %TRUE %FALSE	–	–	–	bin	–
%X X'f...f' B'b...b'	bin	–	bin	bin	bin

bin Binäre Übertragung
linksbündig
sender < *empfänger*: rechts wird mit binären Nullen aufgefüllt.
sender > *empfänger*: rechts wird abgeschnitten.
Ein numerisches Literal (nur Ganzzahl erlaubt) entspricht bei der Übertragung in den Speichertyp %X einem Integerwert mit Vorzeichen in der Länge 4 Bytes (%FL4), die binär übertragen werden.

char Character-Übertragung
linksbündig
sender < *empfänger*: rechts wird mit Leerzeichen (X'40') aufgefüllt.

sender > *empfänger*: rechts wird abgeschnitten.

num — Numerische Übertragung
werterhaltend
sender wird bei Bedarf dem Speichertyp von *empfänger* angepasst.

num(1) — Wenn ein Sender vom Typ Character nur Ziffern enthält und höchstens 18 Stellen lang ist, und wenn der Empfänger vom Typ numerisch ist, führt AID eine numerische Übertragung durch. Alle übrigen Sender vom Typ Character können nicht in numerische Empfänger übertragen werden.

\- — keine Übertragung
AID meldet die Unverträglichkeit der Speichertypen.

* — keine Übertragung
AID führt diese Übertragung im Gegensatz zu FOR1 nicht aus.
Komplexe Werte können nur in Realteil (*datenname*._REAL) und Imaginärteil (*datenname*._IMAG) getrennt übertragen werden.

Beispiele

Für die folgenden Beispiele wurde mit `%AID CHECK=ALL` der Änderungsdialog eingeschaltet. So sehen Sie den Inhalt des Empfangsfelds vor und nach der Ausführung des %SET:

1. `%SET #'061' INTO ZAEHLER`

```
OLD CONTENT:
          1
NEW CONTENT:
         97
%  IDA0129 CHANGE? (Y=YES;N=NO)?
Y
```

Zum selben Ergebnis führt folgendes Kommando:

```
%SET 97 INTO ZAEHLER
```

2. `%QUALIFY PROG=SORT`
`%SET .LI INTO .L(Z)`

```
OLD CONTENT:
          0
NEW CONTENT:
         10
%  IDA0129 CHANGE? (Y=YES;N=NO)?
Y
```

3. `%SET 'ABCDEFG' INTO CHARVAR`

```
OLD CONTENT:
|1234567890|
NEW CONTENT:
|ABCDEFG   |
%  IDA0129 CHANGE? (Y=YES;N=NO)?
Y
```

4. `%SET 0.12345E-03 INTO COMPLVAR ._REAL`

```
I390 WARNING: SOURCE TRUNCATED
OLD CONTENT:
+.0000000 E+000
NEW CONTENT:
+.1234499 E-003
%  IDA0129 CHANGE? (Y=YES;N=NO)?
Y
```

5. `%AID SYMCHARS=NOSTD`
   `%SET ARRAY(I*J-K,L) INTO CARRAY ._IMAG(M+3)`

   Das %AID-Kommando bewirkt, dass AID den Index I*J-K richtig berechnet. Andernfalls würde AID den Ausdruck J-K als Namen einer Variablen auffassen.

```
OLD CONTENT:
+.7000000000000000 E+003
NEW CONTENT:
+.8765429999999999 E-003
%  IDA0129 CHANGE? (Y=YES;N=NO)?
Y
```

%STOP

Mit %STOP veranlassen Sie AID, das Programm anzuhalten, in den Kommandomodus zu gehen und eine STOP-Meldung auszugeben. Dieser Meldung können Sie entnehmen, an welcher Anweisung und in welcher Programmeinheit das Programm unterbrochen wurde.

Wird das Kommando am Terminal oder aus einer Prozedurdatei eingegeben, so wird der Programmzustand nicht verändert, da das Programm ja bereits steht. In diesen Fällen können Sie das Kommando anwenden, um mit der STOP-Meldung Lokalisierungsinformation über die Programmunterbrechungsstelle zu erhalten.

Kommando	Operand
%STOP	

Steht %STOP in einer Kommandofolge oder in einem Subkommando, werden nachfolgende Kommandos nicht mehr ausgeführt.

Wenn Sie als Basisqualifikation mit %BASE eine Dump-Datei eingestellt haben und dann ein %STOP-Kommando eingeben, gibt AID eine STOP-Meldung aus. Diese Meldung enthält die Lokalisierungsinformation zu der Adresse, an der das Programm unterbrochen war als der Dump geschrieben wurde.

Wenn das Programm durch Drücken der K2-Taste unterbrochen wurde, muss die Programmunterbrechungsstelle nicht unbedingt im Benutzerprogramm liegen, sondern das Programm kann auch in den Routinen des Laufzeitsystems stehen.

%STOP verändert den Programmzustand.

Ein %STOP in einem Subkommando bezieht sich stets auf das geladene Programm.

Beispiel

```
/%IN PROG=SORT.S'20' <%D CFELD; %STOP>
/%RESUME

CFELD( 1: 9)
(  1) |Berta|  (  2) |Hansi|  (  3) |Anton|  (  4) |Erich|  (  5) |Ilona|
(  6) |Diane|  (  7) |Carlo|  (  8) |Franz|  (  9) |Georg|
STOPPED AT SRC_REF: 20 , SOURCE: SORT , PROC: SORT
```

Mit %INSERT wird ein Testpunkt auf die Anweisung mit der Nummer 20 gesetzt. Das Subkommando enthält die Kommandos %DISPLAY und %STOP. Nach der Ausgabe von CFELD hält AID das Programm an und schreibt eine STOP-Meldung mit Anweisungsnummer und Programmeinheit der aktuellen Unterbrechungsstelle.

%SYMLIB

Mit %SYMLIB veranlassen Sie AID, PLAM-Bibliotheken zu öffnen oder zu schließen. Auf geöffnete PLAM-Bibliotheken greift AID zu, wenn Sie in einem Kommando symbolische Speicherreferenzen ansprechen, die in einer Programmeinheit liegen, zu der AID keine LSD-Sätze geladen hat.

– Mit *qualifikation-u-lib* melden Sie eine oder mehrere Bibliotheken an oder ab, in denen Bindemoduln mit den zugehörigen LSD-Sätzen abgespeichert sind. Sie können jede Bibliothek dem aktuellen Programm oder einer Dump-Datei zum Nachladen der LSD-Sätze zuordnen, indem Sie die entsprechende Basis-Qualifikation dazu angeben.

Kommando	Operand
%SYMLIB	[qualifikation-u-lib][,...]

Bei Ausführung dieses Kommandos stellt AID nur fest, ob die angegebene Bibliothek geöffnet werden kann; es überprüft nicht, ob der Inhalt einer Bibliothek zu dem Programm passt, das gerade bearbeitet wird. Vorbereitend können Sie also die Bibliotheken anmelden, die Sie während eines Testlaufs benötigen. Erst beim Zugriff auf die nachgeladenen LSD-Sätze überprüft AID, ob der Bindemodul des angesprochenen Programms mit dem der PLAM-Bibliothek übereinstimmt.

Sind zu einer Basis-Qualifikation mehrere Bibliotheken angemeldet, so durchsucht AID sie in der Reihenfolge, in der sie im %SYMLIB-Kommando angegeben wurden.
Verläuft die Suche von AID nicht erfolgreich oder ist keine Bibliothek angemeldet, so können Sie nach der entsprechenden Meldung mit einem neuen %SYMLIB-Kommando die richtige Bibliothek zuweisen und dann das Kommando wiederholen, zu dessen Ausführung die LSD-Sätze fehlten.

Eine Bibliothek bleibt solange angemeldet, bis sie durch ein neues %SYMLIB-Kommando zu derselben Basis-Qualifikation oder durch ein %SYMLIB ohne Operand abgemeldet wird oder bis /LOGOFF. Enthält ein neues Kommando neue Dateinamen, dann werden diese Bibliotheken angemeldet und geöffnet.

%SYMLIB verändert den Programmzustand nicht.

qualifikation-u-lib

ist eine Basis-Qualifikation und/oder der Dateiname einer PLAM-Bibliothek.

– Geben Sie eine Basis-Qualifikation und einen Dateinamen an, meldet AID die angegebene Bibliothek zu dieser Basis-Qualifikation an und öffnet sie. Bisher angemeldete Bibliotheken zu derselben Basis-Qualifikation werden abgemeldet.
– Geben Sie nur einen Dateinamen an, meldet AID die Bibliothek zur gerade eingestellten Basis-Qualifikation an (siehe %BASE) und öffnet sie. Alle zur aktuellen Basis-Qualifikation angemeldeten Bibliotheken werden abgemeldet.
– Geben Sie nur eine Basis-Qualifikation an, werden alle dazu angemeldeten Bibliotheken abgemeldet.

AID kann maximal 15 Bibliotheks-Anmeldungen verwalten. Dabei zählt eine Bibliothek, die gleichzeitig mit verschiedenen Basis-Qualifikationen angemeldet ist, so oft, wie sie angegeben wird.

```
qualifikation-u-lib-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - -

                 ⎧VM ⎫
[•][E=           ⎨   ⎬•][dateiname]
                 ⎩Dn ⎭

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

•
Steht der Punkt an führender Stelle, ist er das Kennzeichen für eine *vorqualifikation*. Sie muss mit einem vorhergehenden %QUALIFY definiert worden sein und kann nur für eine Basis-Qualifikation stehen.

E=VM
%SYMLIB gilt für das geladene Programm (siehe %BASE).

E=Dn
%SYMLIB gilt für einen Speicherabzug in einer Dump-Datei mit dem Linknamen *Dn* (siehe %BASE).

dateiname
ist der BS2000-Katalogname einer PLAM-Bibliothek. Sie wird zu der explizit oder mit *vorqualifikation* angegebenen Basis-Qualifikation angemeldet. Ohne die Angabe einer Qualifikation wird sie zur gerade eingestellten Basis-Qualifikation angemeldet.

Beispiel

```
%SYMLIB        E=D5.PLAMLIB,FOR1OUTPUT
```

Wenn AID für die Bearbeitung eines Speicherabzugs in der Dump-Datei mit dem Linknamen D5 LSD-Sätze benötigt, versucht es, diese aus der Bibliothek PLAMLIB zu laden. Die Bibliothek FOR1OUTPUT wird zur aktuell eingestellten Basis-Qualifikation angemeldet. Wurde bisher kein %BASE gegeben, verwendet AID diese Bibliothek zum Nachladen von LSD-Sätzen zum geladenen Programm.

%TITLE

Mit %TITLE definieren Sie einen eigenen Seitenkopf-Text. Diesen verwendet AID, wenn die Kommandos %DISASSEMBLE, %DISPLAY, %HELP, %SDUMP und %TRACE in die System-Datei SYSLST schreiben.

– Mit *seitenkopf* geben Sie den Text der Kopfzeile an, veranlassen AID, den Seitenzähler auf 1 zu setzen und vor der nächsten Druckzeile SYSLST auf Seitenanfang zu positionieren.

Kommando	Operand
%TITLE	[seitenkopf]

Mit einem %TITLE ohne *seitenkopf*-Operanden wechseln Sie wieder zur AID-Standard-Überschrift. AID setzt den Seitenzähler wieder auf 1 und positioniert SYSLST vor der nächsten Druckzeile auf Seitenanfang.

Eine mit %TITLE vereinbarte Seitenüberschrift gilt bis zu einem neuen %TITLE oder bis Programmende.

%TITLE verändert den Programmzustand nicht.

seitenkopf

gibt den variablen Teil der Seitenüberschrift an. Er wird von AID mit der Uhrzeit, dem Datum und dem Seitenzähler ergänzt.

seitenkopf
ist ein Character-Literal in der Form {C'x...x' | 'x...x'C | 'x...x'} und kann maximal 80 Zeichen lang sein. Ein längeres Literal wird mit einer Fehlermeldung abgewiesen, in der aber nur die ersten 52 Stellen des Literals protokolliert werden.

Auf eine Druckseite werden außer der Seitenüberschrift bis zu 58 Zeilen gedruckt.

%TRACE

Mit %TRACE schalten Sie die AID-Ablaufverfolgung ein und starten das Programm oder setzen es an der unterbrochenen oder mit %JUMP vereinbarten Stelle fort.

– Mit *anzahl* legen Sie fest, wieviele FORTRAN-Anweisungen maximal verfolgt, d.h. ausgeführt und protokolliert werden sollen.

– Mit *fortsetzung* legen Sie fest, ob das Programm nach Beendigung des %Trace anhält oder ohne Protokollierung weiterläuft.

– Mit *kriterium* wählen Sie verschiedene Typen von FORTRAN-Anweisungen aus, die AID protokollieren soll. Die Protokollierung erfolgt vor der Ausführung der ausgewählten Anweisungen.

– Mit *trace-bereich* legen Sie den Programmbereich fest, in dem *kriterium* berücksichtigt werden soll.

Kommando	Operand
%T[RACE]	[anzahl] [fortsetzung] [kriterium][,...] [IN trace-bereich]

Wird der Programmablauf während eines %TRACE unterbrochen, so kann der %TRACE mit %CONTINUE fortgesetzt werden. Dies trifft auf folgende Fälle zu:

– Ein Subkommando, das ein %STOP-Kommando enthält, wurde ausgeführt.

– Ein %INSERT endet mit einer Programmunterbrechung, weil der *steuerung*-Operand K oder S lautet.

– Die K2-Taste wurde gedrückt. Dazu muss am Terminal die SDF-Option OVERFLOW-CONTROL = USER-ACKNOWLEDGE eingestellt sein (Kommando /MODIFY-TERMINAL-OPTIONS).

– Das Programm wurde durch die FORTRAN-Anweisung PAUSE angehalten.

Beendet wird der %TRACE dagegen durch folgende Ereignisse:

– Die maximale Anzahl der zu überwachenden Anweisungen wurde erreicht.

– Ein Subkommando wurde ausgeführt, das ein %RESUME- oder %TRACE-Kommando enthält.

– Nach einer der oben beschriebenen Programmunterbrechungen wird mit %RESUME fortgefahren.

Die Operandenwerte eines %TRACE gelten so lange, bis sie durch Angaben aus einem späteren %TRACE überschrieben werden oder bis Programmende. In einem neuen %TRACE setzt AID also für einen nicht angegebenen Operanden den Wert aus dem vorhergehenden %TRACE ein. Beim *trace-bereich*-Operanden ist dies nur der Fall, wenn die aktuelle Unterbrechungsstelle in dem zu übernehmenden *trace-bereich* liegt. Gibt es keine zu übernehmenden Werte, setzt AID die Standardwerte ein. Für *anzahl* ist dies 10; für *trace-bereich* wird die Programmeinheit eingesetzt, in der die aktuelle Unterbrechungsstelle liegt.

Mit %OUT können Sie steuern, welche Informationen eine Protokollzeile enthält und auf welches Ausgabemedium das Protokoll ausgegeben werden soll.

Steht %TRACE in einer Kommandofolge oder in einem Subkommando, werden nachfolgende Kommandos nicht mehr ausgeführt.

trace-bereich kann nur im geladenen Programm liegen, deshalb muss die Basis-Qualifikation E=VM eingestellt sein (siehe %BASE) oder explizit angegeben werden.

%TRACE verändert den Programmzustand.

anzahl

gibt an, wieviele FORTRAN-Anweisungen vom Typ *kriterium* maximal ausgeführt und protokolliert werden sollen.

anzahl
ist eine Ganzzahl mit $1 \leq anzahl \leq 2^{31}-1$. Standardwert ist 10. Er wird von AID in ein %TRACE-Kommando ohne *anzahl*-Operanden eingesetzt, wenn es keinen Wert aus einem vorhergehenden %TRACE gibt.

Nachdem die vorgegebene *anzahl* von Anweisungen überwacht wurde, gibt AID über SYSOUT eine Meldung aus, das Programm wird angehalten, und Sie können wieder AID- oder BS2000-Kommandos eingeben. Der Meldung können Sie entnehmen, an welcher Anweisung und in welcher Programmeinheit das Programm angehalten wurde.

fortsetzung

gibt an, ob AID das Programm nach Beendigung des %TRACE anhalten oder fortsetzen soll.
Der Operand gilt solange, bis in einem neuen %TRACE ein anderer Operandenwert angegeben wird oder bis Programmende.

```
fortsetzung-OPERAND - - - - - - - - - - - - - - - - - - - - - - - -

{S | R}

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

S

Das Programm wird angehalten. AID gibt eine STOP-Meldung aus, die Lokalisierungsinformationen über die Unterbrechungsstelle enthält. S ist Standardwert.

R

Das Programm wird ohne Ausgabe einer Meldung fortgesetzt.

kriterium

ist ein Schlüsselwort, das den Typ der Anweisungen festlegt, die beim Ablauf überwacht werden sollen. Sie können mehrere Schlüsselwörter gleichzeitig angeben, die dann gemeinsam wirken. Zwischen zwei Schlüsselwörtern muss ein Komma stehen.
Wird kein *kriterium* vereinbart, arbeitet AID mit dem Standardwert %STMT, wenn nicht noch aus einem vorhergehenden %TRACE eine *kriterium*-Vereinbarung gültig ist.

kriterium	**Protokolliert wird vor der Ausführung von:**
%STMT	jeder FORTRAN-Anweisung, die durchlaufen wird
%ASSGN	Zuweisungs-Anweisungen
%CALL	SUBROUTINE-Aufrufen (CALL-Anweisungen)
%COND	IF(...) THEN-,ELSE IF(...) THEN-,ELSE- und IF(...)-Anweisungen
%GOTO	GOTO-Anweisungen
%IO	Ein-/Ausgabe-Anweisungen
%LAB	jeder Anweisung mit einer Anweisungsmarke
%PROC	STOP-, END- und RETURN-Anweisungen sowie der ersten ausführbaren Anweisung nach SUBROUTINE und FUNCTION

trace-bereich

legt den Programmbereich fest, in dem die Ablaufverfolgung stattfinden soll. Nur innerhalb dieses Bereiches werden die mit *kriterium* ausgewählten Anweisungen überwacht und protokolliert. Außerhalb dieses Bereiches ist der %TRACE inaktiv und wird erst bei Rückkehr in den Bereich wieder aktiv.

Eine *trace-bereich*-Definition ist wirksam bis zu einem neuen %TRACE mit eigenem *trace-bereich*-Operanden, einem %TRACE, der außerhalb dieses Bereiches eingegeben wird oder bis zum Programmende. Wird *trace-bereich* nicht angegeben, wird die Bereichsdefinition aus einem vorhergehenden %TRACE übernommen, wenn die aktuelle Unterbrechungsstelle in diesem Bereich liegt. Sonst setzt AID den Standardwert ein, das ist die Programmeinheit, in der die aktuelle Unterbrechungsstelle liegt.

Die Fortsetzungsadresse für den Programmablauf kann mit %TRACE nicht beeinflusst werden, dazu müssen Sie das Kommando %JUMP verwenden.

```
trace-bereich-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                      ⎧PROG=program-name                      ⎫
IN  [•][E=VM•] ⎨                                              ⎬
                      ⎩[PROG=program-name•]( S'n' : S'n' )    ⎭

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
```

•
: Steht der Punkt an führender Stelle, so ist er das Kennzeichen für eine *vorqualifikation*. Sie muss mit einem vorhergehenden %QUALIFY-Kommando definiert worden sein. Aufeinanderfolgende Qualifikationen werden durch einen Punkt getrennt. Außerdem muss zwischen der letzten Qualifikation und dem anschließenden Operandenteil ein Punkt stehen.

E=VM
: Da *trace-bereich* nur im virtuellen Speicher des geladenen Programms liegen kann, geben Sie *E=VM* nur an, wenn als aktuelle Basis-Qualifikation eine Dump-Datei vereinbart ist (siehe %BASE).

PROG=program-name
: *program-name* ist der maximal 7stellige Name einer Programmeinheit.
Diese Programmeinheit muss zum Zeitpunkt der Eingabe des %TRACE geladen sein.

  Eine PROG-Qualifikation ist nur erforderlich, wenn ein Lademodul aus mehreren Programmeinheiten entstanden ist und sich der %TRACE nicht auf die aktuelle Programmeinheit bezieht oder um eine bisher geltende *trace-bereich*-Vereinbarung zu überschreiben.

Endet *trace-bereich* mit einer PROG-Qualifikation, so umfasst er die gesamte angegebene Programmeinheit.

(S'n' : S'n')
trace-bereich wird durch die Angabe einer Anfangs- und einer Endadresse festgelegt. Beide müssen in derselben Programmeinheit liegen, und es gilt:
Anfangsadresse ≤ Endadresse.
n ist die Nummer einer Anweisung; siehe Spalte STMT der Übersetzungsliste.

Soll *trace-bereich* nur eine Anweisung umfassen, müssen Anfangs- und Endadresse gleich sein.

Ausgabe des %TRACE-Protokolls

Das %TRACE-Protokoll wird standardmäßig in ausführlicher Form (%OUT-Operandenwert T=MAX) über SYSOUT ausgegeben. Mit %OUT können Sie die Ausgabe-Medien und den Informationsumfang für die Ausgabe festlegen (siehe AID-Basishandbuch, Kapitel 7).

Ein %TRACE-Protokoll mit Zusatzinformationen (T=MAX) enthält die Nummer und den Typ der Anweisung, die ausgeführt wurde. Ist eine Anweisungsmarke vorhanden, wird auch diese ausgegeben.

In einem %TRACE-Protokoll mit verkürzter Information (T=MIN) wird der Typ der Anweisung nicht ausgegeben.

AID berücksichtigt die Modi XMAX und XFLAT für die Ausgabe des %TRACE-Protokolls nicht. Statt dessen generiert es die Standardausgabe (T=MAX).

Beispiele

1.

```
/%OUT %TRACE    T=MAX
/%T 3
       49          33       STMT
       50                   ASSIGN
       51                   ASSIGN
STOPPED AT SRC_REF:  51,  SOURCE: BEISPIEL, PROC: BEISPIEL
```

Mit %OUT wurde die Ausgabe auf Terminal zurückgeschaltet und festgelegt, dass der maximale Informationsumfang ausgegeben werden soll.
Das %TRACE-Kommando soll drei FORTRAN-Anweisungen verfolgen. Nach der dritten Anweisung kommt die Abschlussmeldung für diesen %TRACE: Der Programmablauf wurde bei Anweisung 51 unterbrochen, die Anweisung 51 steht in der Programmeinheit BEISPIEL, der Lademodul hat denselben Namen.

2.

```
/%OUT %T T=MIN
/%T 3
       49          33
       50
       51
STOPPED AT SRC_REF:  51,  SOURCE: BEISPIEL, PROC: BEISPIEL
```

Mit dem %OUT-Kommando wird der Informationsumfang für das Kommando %TRACE reduziert. Der danach eingebene %TRACE gibt das Protokoll mit verkürztem Informationsumfang aus.

3. `%TRACE 5 R %INSTR`

Fünf Befehle des Programms werden ausgeführt und protokolliert. Danach wird das Programm ohne Protokollierung fortgesetzt.

4. `%C1 %CALL IN S=TESTPROG <%TRACE 1 R>`

Alle Unterprogrammaufrufe der Programmeinheit TESTPROG werden protokolliert. Das Programm wird jeweils nach Ausführung und Protokollierung der CALL-Anweisung fortgesetzt.

6 Anwendungsbeispiel

In diesem Kapitel wird eine AID-Testsitzung für ein kleines FORTRAN-Programm gezeigt. Anhand dieser Testsitzung können Sie die Anwendung und Wirkung einiger AID-Kommandos nachvollziehen; die Vorgehensweise ist bewusst einfach gehalten. Zunächst ist die Übersetzungsliste des FORTRAN-Programms abgebildet; der Testablauf ist dann anschließend dargestellt.

6.1 Quellprogrammliste

```
PROGRAM UNIT: B1

DO/IF SEG  STMT  I/H LINE      SOURCE-TEXT
------------------------------------------------------------------------------
      1/1    1          1 |    PROGRAM B1
                        2 |*
                        3 |*       SORTIEREN EINES CHARACTER - FELDES
                        4 |*
       1     2          5 |    IMPLICIT INTEGER (A-Z)
       1     3          6 |    PARAMETER (DIM=9)
       1     4          7 |    COMMON  /CB/ CFELD
       1     5          8 |    CHARACTER * 5 CFELD(DIM)
       1     6          9 |    DATA CFELD /'Berta','Hansi','Anton','Erich','Ilona',
       1               10 |   &            'Diane','Carlo','Franz','Georg'/
       1     7         11 |    WRITE (2,*) '   CFELD UNSORTIERT'
       1     8         12 |    DO 10 K=1,DIM
  1    2     9         13 |       WRITE (2,*) CFELD(K)
  1    3    10         14 | 10 CONTINUE
       4    11         15 |    CALL SORT
                       16 |*
                       17 |*  KONTROLLAUSGABE
                       18 |*
       4    12         19 |    WRITE (2,*) '   CFELD SORTIERT'
       4    13         20 |    DO 20 K=1,DIM
  1    5    14         21 |       WRITE (2,*) CFELD(K)
  1    6    15         22 | 20 CONTINUE
       7    16         23 |    END
```

```
PROGRAM UNIT: SORT

DO/IF SEG  STMT  I/H LINE      SOURCE-TEXT
                          1 |*
      1/1    1            2 |      SUBROUTINE SORT
                          3 |*
                          4 |*  CFELD WIRD SORTIERT
                          5 |*
       1     2            6 |      IMPLICIT INTEGER (A-Z)
       1     3            7 |      PARAMETER (DIM=9)
       1     4            8 |      COMMON /CB/CFELD
       1     5            9 |      CHARACTER * 5 CFELD(DIM), IMITTE
       1     6           10 |      DIMENSION L(5) ! LINKE INTERVALLGRENZEN
       1     7           11 |      DIMENSION R(5) ! RECHTE INTERVALLGRANZEN
                         12 |*
       1     8           13 |      Z=1            ! ANZAHL DER ZU SORTIERENDEN
       1                 14 |                     ! TEILINTERVALLE
       1     9           15 |      L(1)=1         ! AUSGANG IST DAS GESAMTE INTERVALL
       1    10           16 |      R(1)=DIM
                         17 |*
       1    11           18 |    1 CONTINUE   ! SIMULATION EINER "REPEAT-SCHLEIFE"
       1                 19 |                 ! DIE SCHLEIFE WIRD SOLANGE DURCHLAUFEN,
       1                 20 |                 ! BIS ALLE TEILINTERVALLE BEZUEGLICH
       1                 21 |                 ! DER INTERVALLMITTE SORTIERT SIND.
                         22 |*
                         23 |*
                         24 |*   SORTIERUNG EINES EINZELNEN TEILINTERVALLES
                         25 |*   BZGL. DER INTERVALL-MITTE
                         26 |*
       2    12           27 |      LI=L(Z)    ! LINKE UND RECHTE INTERVALLGRENZE          (10)
       2    13           28 |      RI=R(Z)    ! DES AKTUELLEN SORTIERINTERVALLES
       2    14           29 |      Z=Z-1      ! ANZAHL DER NOCH ZU SORTIERENDEN
       2                 30 |                 ! INTERVALLE ERNIEDRIGEN
       2    15           31 |      IMITTE=CFELD(INT(LI+RI)/2)    ! INTERVALLMITTE
                         32 |*
                         33 |* ALGORITHMUS:
                         34 |*   DAS ZU SORTIERENDE FELD WIRD ALS INTERVALL
                         35 |*   BETRACHTET UND DIE INTERVALLMITTE BESTIMMT. NUN
                         36 |*   WIRD VON LINKS IN DER LINKEN INTERVALLHAELFTE EIN
                         37 |*   ELEMENT GROESSER ALS DER INTERVALLMITTENWERT
                         38 |*   GESUCHT, EBENSO WIRD VON RECHTS EIN ELEMENT
                         39 |*   KLEINER ALS DER INTERVALLMITTENWERT GESUCHT.
                         40 |*   DIESE BEIDEN WERTE WERDEN VERTAUSCHT.
                         41 |*   IST VON LINKS BZW. VON RECHTS DIE
                         42 |*   INTERVALLMITTE NOCH NICHT ERREICHT,
                         43 |*   SO WIRD DIESES VERFAHREN WEITER DURCHGEFUEHRT.
                         44 |*
                         45 |*   IST EINE INTERVALLGRENZE ERREICHT, SO WIRD
                         46 |*   GEPRUEFT, OB DAS ELEMENT DER INTERVALLMITTE
                         47 |*   MIT EINEM ELEMENT DER LINKEN ODER RECHTEN
                         48 |*   INTERVALLHAELFTE ZU VERTAUSCHEN IST.
                         49 |*
                         50 |*   DIESES VERFAHREN WIRD NUN BEI DEM LINKEN UND
                         51 |*   RECHTEN TEILINTERVALL GETRENNT DURCHGEFUEHRT,
                         52 |*   USW..
                         53 |*   DAS VERFAHREN BRICHT AB, WENN ALLE INTERVALLE
                         54 |*   BEARBEITET WURDEN.
                         55 |*
       2    16           56 |      I=LI  ! LAUFINDEX FUER LINKE INTERVALLHAELFTE SETZEN
       2    17           57 |      J=RI  ! LAUFINDEX FUER RECHTE INTERVALLHAELFTE SETZEN
                         58 |*
       2    18           59 |    2 CONTINUE   ! SIMULATION EINER "REPEAT-SCHLEIFE"
       2                 60 |                 ! BEARBEITUNG DES AKTUELLEN INTERVALLES
                         61 |*
       2    19           62 |    3  CONTINUE  ! SIMULATION EINER "DO-LOOP-SCHLEIFE"
       2                 63 |                 ! ZU VERTAUSCHENDES ELEMENT LINKS SUCHEN
       3    20           64 |       IF (CFELD(I) .GE. IMITTE) GOTO 31
```

```
                                65 |*
                                66 |*   GOTO 31: ELEMENT MUSS VERTAUSCHT WERDEN ODER
                                67 |*            INTERVALLMITTE ERREICHT
                                68 |*
          4     22              69 |       I=I+1
                                70 |*        NAECHSTES ELEMENT PRUEFEN
          4     23              71 |       GOTO 3
          4     24              72 |  31 CONTINUE
                                73 |*
          4     25              74 |    4 CONTINUE  ! SIMULATION EINER "DO-LOOP-SCHLEIFE"
          5     26              75 |       IF (CFELD(J).LE.IMITTE) GOTO 41                        (1)
                                76 |*
                                77 |*   GOTO 41: ELEMENT MUSS VERTAUSCHT WERDEN ODER
                                78 |*            INTERVALLMITTE ERREICHT
                                79 |*
          6     28              80 |       J=J-1
          6     29              81 |       GOTO 4
          6     30              82 |  41 CONTINUE
                                83 |*
          7     31              84 |    IF (I .GE. J)GOTO 21  ! EXIT LOOP                         (2)
                                85 |*
                                86 |*   GOTO 21: KEINE ELEMENTE SIND ZU VERTAUSCHEN
                                87 |*             BZW. ALLE NOTWENDIGEN VERTAUSCHUNGEN
                                88 |*             SIND DURCHGEFUEHRT.
                                89 |*
                                90 |*
                                91 |*   ELEMENTE AUS UNTEREM MIT ELEMENT AUS OBEREM
                                92 |*   INTERVALLGEBIET VERTAUSCHEN
                                93 |*
          8     33              94 |       CALL TAUSCH(CFELD(I),CFELD(J))
          8     34              95 |       I=I+1
          8     35              96 |       J=J-1
          8     36              97 |       IF (I .LT. J) GOTO 2  ! UNTIL LOOP
                                98 |*
                                99 |*   GOTO 2 * AUF WEITERE VERTAUSCHUNGEN PRUEFEN
                               100 |*
                               101 |*   ALLE ELEMENTE IM INTERVALL-INNEREN VERTAUSCHT
                               102 |*
          8     38              103 |  21 CONTINUE
          9     39              104 |    IF (LI .LT. J) THEN  ! IMMER ERFUELLT                     (3)
                               105 |*      TEILINTERVALL FESTSTELLEN UND GEGEBENENFALLS MERKEN
    1    10     40              106 |       IF (LI .EQ. J-1) THEN                                  (4)
                               107 |*   TEILINTERVALL BESTEHT AUS NUR ZWEI ELEMENTEN
    1    10                    108 |*
    2    11     41              109 |       IF (CFELD(LI).GT.CFELD(J)) THEN
                               110 |*
                               111 |*   ELEMENT DES INTERVALLRANDES VERTAUSCHEN
                               112 |*
    3    12     42              113 |         CALL TAUSCH (CFELD(J),CFELD(LI))
    3    12     43              114 |       ENDIF
    2    12     44              115 |    ELSE
                               116 |*
                               117 |*   NOCH NICHT BEARBEITETES TEILINTERVALL MERKEN
                               118 |*
    2    13     45              119 |         Z=Z+1                                                (5)
    2    13     46              120 |         L(Z)=I                                               (6)
    2    13     47              121 |         R(Z)=J                                               (7)
    2    13     48              122 |      ENDIF
    1    13     49              123 |    ENDIF
         14     50              124 |    IF (I .LT. RI) THEN                                       (8)
                               125 |*   TEILINTERVALL FESTSTELLEN UND GEGEBENENFALLS MERKEN
    1    15     51              126 |       IF (I .EQ. RI-1) THEN
                               127 |*   TEILINTERVALL BESTEHT AUS NUR ZWEI ELEMENTEN
    2    16     52              128 |         IF (CFELD(I) .GT. CFELD(RI)) THEN
                               129 |*
                               130 |*   ELEMENT DES INTERVALLRANDES VERTAUSCHEN
                               131 |*
    3    17     53              132 |           CALL TAUSCH (CFELD(I),CFELD(RI))
```

```
  3   17    54        133 |          ENDIF
  2   17    55        134 |        ELSE
                      135 |*
                      136 |*   NOCH NICHT BEARBEITETES TEILINTERVALL MERKEN
                      137 |*
  2   18    56        138 |           Z=Z+1
  2   18    57        139 |           L(Z)=I
  2   18    58        140 |           R(Z)=RI
  2   18    59        141 |        ENDIF
  1   18    60        142 |      ENDIF
      19    61        143 |    IF (Z .NE. 0) GOTO 1  ! UNTIL SIMULATION                (9)
                      144 |*
                      145 |*   INTERVALL-FELD (=STACK-ERSATZ) ABGEARBEITET
                      146 |*
      20    63        147 |    RETURN
      20    64        148 |    END
```

```
PROGRAM UNIT: TAUSCH

DO/IF SEG  STMT  I/H LINE      SOURCE-TEXT
──────────────────────────────────────────────────────────────────────────────
                        1 |*
     1/1    1           2 |    SUBROUTINE TAUSCH (CHAR1,CHAR2)
      1     2           3 |      CHARACTER * 5 CHAR1,CHAR2,CHAR
                        4 |*
                        5 |*   ELEMENTE WERDEN VERTAUSCHT
                        6 |*
      1     3           7 |       CHAR=CHAR1
      1     4           8 |       CHAR1=CHAR2
      1     5           9 |       CHAR2=CHAR
      1     6          10 |       RETURN
      1     7          11 |    END
```

6.2 Testablauf

1. Schritt

Das FORTRAN-Quellprogramm B1 in der Datei QSORT wird mit FOR1 übersetzt. Durch Angabe der SDF-Option TOOL-SUPPORT = AID wird von FOR1 LSD-Information erzeugt, die symbolisches Testen ermöglicht. Um mit AID problemlos testen zu können, wird zunächst ohne Optimierung übersetzt (SDF-Option OPTIMIZATION = NO; siehe Kap. 2). Die Übersetzung ergibt keine Fehler.
Eingaben sind im Beispiel zur besseren Lesbarkeit fettgedruckt.

```
/START-FOR1-COMPILER SOURCE=QSORT,OPTIMIZATION=NO,-
                     TEST-SUPPORT=PARAMETER(TOOL-SUPPORT=AID),-
                     LISTING=PARAMETER(OUTPUT=LF.QSORT),-
                     SOURCE-PROPERTIES=PARAMETER(LINE-END-COMMENTS='!')
%  BLS0500 PROGRAM 'FOR1', VERSION '2.1A00' OF '91-04-29' LOADED.
  FOR1:   V2.1A00 READY, GIVE COMPILER OPTION
  FOR1: LIST FILE REPLACED = LF.QSORT
  FOR1: NO ERRORS DURING COMPILATION OF P.U. B1
  FOR1: NO ERRORS DURING COMPILATION OF P.U. SORT
  FOR1: NO ERRORS DURING COMPILATION OF P.U. TAUSCH

 END OF  F O R 1  COMPILATION; CPU TIME USED: 3.904 SEC.
```

2. Schritt

Das Programm läuft ebenfalls fehlerfrei ab. Das Ergebnis des Sortieralgorithmus ist allerdings nicht korrekt: die Namensliste wird nicht in alphabetisch sortierter Reihenfolge ausgegeben.

```
/SET-TASKLIB LIBRARY=$FOR1MODLIBS
/START-FOR1-PROGRAM FROM-FILE=*MODULE(LIBRARY=*OMF)
 %  BLS0001 DLL VER 823
 %  BLS0517 MODULE 'B1' LOADED
 BS2000  F O R 1 : FORTRAN PROGRAM "B1"
 STARTED ON 91-04-29 AT 16:10:53
    CFELD UNSORTIERT
 Berta
 Hansi
 Anton
 Erich
 Ilona
 Diane
 Carlo
 Franz
 Georg
    CFELD SORTIERT
 Berta
 Hansi
 Anton
 Erich
 Georg
 Diane
 Carlo
 Franz
 Ilona
 BS2000  F O R 1 : FORTRAN PROGRAM "B1           " ENDED PROPERLY AT 16:11:04
 CPU - TIME USED:      0.0937 SECONDS
 ELAPSED TIME   :     11.4430 SECONDS
```

3. Schritt

Um das Programm symbolisch mit AID testen zu können, wird es mit der SDF-Option TEST-OPTIONS=AID erneut geladen. Nach dem LOAD-PROGRAM-Kommando können AID-Kommandos eingegeben werden.

```
/LOAD-PROGRAM FROM-FILE=*MODULE(LIBRARY=*OMF),-
              TEST-OPTIONS=AID
%  BLS0001 DLL VER 823
%  BLS0517 MODULE 'B1' LOADED
```

4. Schritt

```
/%C1 %CALL IN PROG=SORT <%D I,J,IMITTE,CFELD,RI; %STOP>
```

Mit dem %CONTROL-Kommando wird als *kriterium* die CALL-Anweisung vereinbart; *kriterium* soll nur im Unterprogramm SORT überwacht werden. Jedesmal vor der Ausführung des CALL-Aufrufs soll das Subkommando ausgeführt werden. Im Subkommando sollen die

Laufindizes I für die linke bzw. J für die rechte Intervallgrenze, der Wert des mittleren Feldelements IMITTE, das zu sortierende Feld CFELD und die rechte Intervallgrenze RI des aktuellen Sortierintervalls ausgegeben werden. Nach der Ausgabe soll der Programmablauf unterbrochen werden, so dass AID-Kommandos eingegeben werden können.

```
/%IN PROG=SORT.S'18' <%D I,J,IMITTE,CFELD,LI,RI; %STOP>
```

Mit dem %INSERT-Kommando wird ein Testpunkt auf die Anweisung mit der Nummer 18 gesetzt, die die Bearbeitung des aktuellen Sortierintervalls einleitet. Vor Ausführung der CONTINUE-Anweisung sollen jedesmal die Datenelemente I, J, IMITTE, CFELD, LI und RI ausgegeben werden.

5. Schritt

Mit dem %RESUME-Kommando wird das geladene Programm gestartet. AID meldet als Unterbrechungsstelle die IF-Anweisung mit der Nummer 20 und nicht die Anweisung mit der im %INSERT-Kommando angegebenen Nummer 18, da die CONTINUE-Anweisung hier nur als Leeranweisung dient.

```
/%R
BS2000  F O R 1 : FORTRAN PROGRAM "B1"
STARTED ON 91-04-29 AT 09:43:28
   CFELD UNSORTIERT
Berta
Hansi
Anton
Erich
Ilona
Diane
Carlo
Franz
Georg
** ITN: #'000000DF' *** TSN: 1627 ***********************************************
SRC_REF:     20 SOURCE: SORT      PROC: SORT ***********************************
I               =            1
J               =            9
IMITTE          = |Ilona|
CFELD ( 1: 9)
(  1) |Berta|  (  2) |Hansi|  (  3) |Anton|  (  4) |Erich|  (  5) |Ilona|
(  6) |Diane|  (  7) |Carlo|  (  8) |Franz|  (  9) |Georg|
LI              =            1
RI              =            9

STOPPED AT SRC_REF: 20 , SOURCE: SORT, PROC: SORT
```

6. Schritt

Mit den AID-Kommandos %R "1" bis %R "4" wird das Programm jeweils fortgesetzt. Die Anführungszeichen stehen für Anfang bzw. Ende eines Kommentars. Das Programm prüft jedesmal, ob das Feldelement CFELD(I) größer oder gleich dem Wert der Intervallmitte IMITTE ist. Nach dem Vergleich wird der Laufindex I für die linke Intervallhälfte jeweils um 1 erhöht. Nach Ausführung des AID-Kommandos %R "4" hat I den Wert 5.

```
/%R "1"
 I                = 2
 J                = 9
 IMITTE           = |Ilona|
 CFELD ( 1: 9)
 (  1) |Berta|  (  2) |Hansi|  (  3) |Anton|  (  4) |Erich|  (  5) |Ilona|
 (  6) |Diane|  (  7) |Carlo|  (  8) |Franz|  (  9) |Georg|
 LI               = 1
 RI               = 9

 STOPPED AT SRC REF: 20 , SOURCE: SORT, PROC: SORT

/%R "2"
 I                = 3
 J                = 9
 IMITTE           = |Ilona|
 CFELD ( 1: 9)
 (  1) |Berta|  (  2) |Hansi|  (  3) |Anton|  (  4) |Erich|  (  5) |Ilona|
 (  6) |Diane|  (  7) |Carlo|  (  8) |Franz|  (  9) |Georg|
 LI               = 1
 RI               = 9

 STOPPED AT SRC REF: 20 , SOURCE: SORT, PROC: SORT

/%R "3"
 I                = 4
 J                = 9
 IMITTE           = |Ilona|
 CFELD ( 1: 9)
 (  1) |Berta|  (  2) |Hansi|  (  3) |Anton|  (  4) |Erich|  (  5) |Ilona|
 (  6) |Diane|  (  7) |Carlo|  (  8) |Franz|  (  9) |Georg|
 LI               = 1
 RI               = 9

 STOPPED AT SRC REF: 20 , SOURCE: SORT, PROC: SORT

/%R "4"
 I                = 5
 J                = 9
 IMITTE           = |Ilona|
 CFELD ( 1: 9)
 (  1) |Berta|  (  2) |Hansi|  (  3) |Anton|  (  4) |Erich|  (  5) |Ilona|
 (  6) |Diane|  (  7) |Carlo|  (  8) |Franz|  (  9) |Georg|
 LI               = 1
 RI               = 9

 STOPPED AT SRC REF: 20 , SOURCE: SORT, PROC: SORT
```

7. Schritt

Der Programmablauf wird mit %R "5" fortgesetzt. Der Vergleich (CFELD(I).GE.IMITTE) führt zur Anweisung mit der Anweisungsmarke 31, der Vergleich (CFELD(J).LE.IMITTE) führt zur Anweisung mit der Anweisungsmarke 41 und zum Aufruf des Unterprogramms TAUSCH mit der Anweisungsnummer 33. Vor Ausführung dieser CALL-Anweisung wird das Programm wegen des %C1-Kommandos unterbrochen und das zugehörige Subkommando ausgeführt.

```
/%R "5"
SRC_REF:     33 SOURCE: SORT      PROC: SORT ************************************
 I               =           5
 J               =           9
 IMITTE          = |Ilona|
 CFELD ( 1: 9)
 (  1) |Berta|  (  2) |Hansi|  (  3) |Anton|  (  4) |Erich|  (  5) |Ilona|
 (  6) |Diane|  (  7) |Carlo|  (  8) |Franz|  (  9) |Georg|
 RI              =           9

 STOPPED AT SRC REF: 33 , SOURCE: SORT, PROC: SORT
```

8. Schritt

Nach Fortsetzung des Programms mit %R "6" wird I um 1 erhöht und J um 1 erniedrigt. Mit diesen Werten wird der Vergleich mit der Intervallmitte erneut durchgeführt. Bei den folgenden %RESUME-Kommandos %R "7" bis %R "9" wird I jeweils um 1 erhöht.

```
/%R "6"
SRC_REF:     20 SOURCE: SORT      PROC: SORT ************************************
 I               =           6
 J               =           8
 IMITTE          = |Ilona|
 CFELD ( 1: 9)
 (  1) |Berta|  (  2) |Hansi|  (  3) |Anton|  (  4) |Erich|  (  5) |Ilona|
 (  6) |Diane|  (  7) |Carlo|  (  8) |Franz|  (  9) |Georg|
 LI              =           1
 RI              =           9

 STOPPED AT SRC REF: 20 , SOURCE: SORT, PROC: SORT

/%R "7"
 I               =           7
 J               =           8
 IMITTE          = |Ilona|
 CFELD ( 1: 9)
 (  1) |Berta|  (  2) |Hansi|  (  3) |Anton|  (  4) |Erich|  (  5) |Ilona|
 (  6) |Diane|  (  7) |Carlo|  (  8) |Franz|  (  9) |Georg|
 LI              =           1
 RI              =           9

 STOPPED AT SRC REF: 20 , SOURCE: SORT, PROC: SORT

/%R "8"
 I               =           8
 J               =           8
 IMITTE          = |Ilona|
 CFELD ( 1: 9)
 (  1) |Berta|  (  2) |Hansi|  (  3) |Anton|  (  4) |Erich|  (  5) |Ilona|
```

```
 (  6) |Diane|  (  7) |Carlo|  (  8) |Franz|  (  9) |Georg|
 LI              =           1
 RI              =           9

 STOPPED AT SRC REF: 20 , SOURCE: SORT, PROC: SORT

/%R "9"
 I               =           9
 J               =           8
 IMITTE          = |Ilona|
 CFELD ( 1: 9)
 (  1) |Berta|  (  2) |Hansi|  (  3) |Anton|  (  4) |Erich|  (  5) |Ilona|
 (  6) |Diane|  (  7) |Carlo|  (  8) |Franz|  (  9) |Georg|
 LI              =           1
 RI              =           9

 STOPPED AT SRC REF: 20 , SOURCE: SORT, PROC: SORT
```

9. Schritt

Für I=9 ist CFELD(I)=IMITTE. Mit dem %TRACE-Kommando sollen die nächsten 8 Anweisungen ausgeführt und protokolliert werden. Ausgegeben werden die Anweisungsnummern, evtl. vorhandene Anweisungsmarken und der Typ der Anweisung. Anhand des %TRACE-Protokolls und der Quellprogrammliste kann der Programmablauf verfolgt werden.

```
/%T 8 %STMT IN PROG=SORT
    26  4                   IF                    (1)
    31  41                  IF                    (2)
    39  21                  IF                    (3)
    40                      IF                    (4)
    45                      THEN/ELSE, ASSIGN     (5)
    46                      ASSIGN                (6)
    47                      ASSIGN                (7)
    50                      IF                    (8)

 STOPPED AT SRC REF: 50, SOURCE: SORT, PROC: SORT
```

Das Programm verzweigt zur nächsten IF-Anweisung (1) mit der Abfrage (CFELD(J).LE.IMITTE), von dort zur IF-Anweisung (2) mit der Abfrage (I.GE.J). I ist größer J, sodass zur CONTINUE-Anweisung mit der Anweisungsmarke 21 verzweigt wird. Im TRACE-Protokoll wird die Anweisungsmarke 21 und die Anweisungsnummer der darauffolgenden IF-Anweisung ausgegeben (siehe (3)). Das Programm verzweigt danach zur IF-Anweisung (4) mit der Abfrage (LI.EQ.J-1) und durchläuft die Zuweisungsanweisungen Z=Z+1, L(Z)=I und R(Z)=J (siehe (5), (6), (7)). Als letzte Anweisung wird die IF-Anweisung (8) mit der Abfrage (I.LT.RI) protokolliert.

10. Schritt

Das AID-Kommando %T 2 entspricht dem Kommando %T 2 %STMT IN PROG=SORT. Die Voreinstellung für den Typ der Anweisung *kriterium* ist %STMT, die Voreinstellung für *trace-bereich* ist die Programmeinheit, in der die aktuelle Unterbrechungsstelle liegt.
Das Kommando %T2 führt zur Protokollierung der Anweisung mit der Nummer 61 (siehe (9)) und zum erneuten Durchlaufen des Sortieralgorithmus mit den aktualisierten Grenzen den Sortierintervalls (siehe (10)).

```
/%T 2
    61                          IF                      (9)
    12  1                       ASSIGN                  (10)

 STOPPED AT SRC REF: 12, SOURCE: SORT, PROC: SORT
```

11. Schritt

Mit dem %DISPLAY-Kommando werden die Anzahl Z der zu sortierenden Teilintervalle und die Grenzen des noch nicht bearbeiteten Teilintervalls ausgegeben:

```
/%D Z
SRC_REF:     12 SOURCE: SORT      PROC: SORT ***********************************
 Z                =           1

/%D L(1),R(1)
 L( 1)            =           9
 R( 1)            =           8
```

Nach diesen Zwischenergebnissen ist die linke Intervallgrenze größer als die rechte. Die Anweisung mit der Anweisungsnummer 46 muss statt L(Z)=I richtig L(Z)=LI heißen.

12. Schritt

Die Korrektur der falschen Anweisung kann mit dem %SET-Kommando ohne erneuten Übersetzungslauf durchgeführt werden.
Dazu wird das Programm erneut geladen:

```
/LOAD-PROGRAM FROM-FILE=*MODULE(LIBRARY=*OMF),-
              TEST-OPTIONS=AID
%  BLS0001 DLL VER 823
%  BLS0517 MODULE 'B1' LOADED
```

13. Schritt

Der Testpunkt des %INSERT-Kommandos wird auf die Anweisung mit der Nummer 47 gesetzt, damit nach der Ausführung der falschen Anweisung L(Z)=I das Element L(Z) mit dem richtigen Wert LI überschrieben wird. Bei einem Testpunkt auf S'46' würde das Programm vor Ausführung der Anweisung das Subkommando ausführen und den richtigen Wert LI einsetzen; anschließend würde die falsche Anweisung L(Z)=I ausgeführt.
Die Korrektur mit dem %SET-Kommando führt zum korrekten Sortierergebnis des Programms:

```
/%INSERT PROG=SORT.S'47' <%SET LI INTO L(Z); %RESUME>
/%R
BS2000  F O R 1 : FORTRAN PROGRAM "B1"
STARTED ON 91-04-29 at 09:47:47
   CFELD UNSORTIERT
Berta
Hansi
Anton
Erich
Ilona
Diane
Carlo
Franz
Georg
   CFELD SORTIERT
Anton
Berta
Carlo
Diane
Erich
Franz
Georg
Hansi
Ilona
BS2000  F O R ! : FORTRAN PROGRAM "B1" ENDED PROPERLY AT 09:47:58
CPU - TIME USED :       0.2067 SECONDS
ELAPSED TIME    :      11.7150 SECONDS
```

Fachwörter

Ablaufüberwachung

l%CONTROLn, %INSERT und %ON sind Kommandos zur Ablaufüberwachung. Kommt der Programmablauf an eine Anweisung der gewählten Gruppe (%CONTROLn) oder an die vereinbarte Programmadresse (%INSERT) oder tritt das ausgewählte Ereignis ein (%ON), wird der Programmablauf unterbrochen, und AID bearbeitet das vereinbarte Subkommando.

Ablaufverfolgung

%TRACE ist das Kommando zur Ablaufverfolgung. Mit ihm vereinbaren Sie, welche und wieviele Anweisungen protokolliert werden sollen. Im Standardfall können Sie den Programmablauf am Bildschirm mitverfolgen.

Adress-Operand

ist ein Operand, mit dem Sie eine Speicherstelle oder einen Speicherbereich adressieren. Sie können virtuelle Adressen, Datennamen, Anweisungsnamen, Source-Referenzen, Schlüsselwörter, komplexe Speicherreferenzen oder eine PROG-Qualifikation angeben. Die Speicherstelle oder der Speicherbereich liegen entweder im geladenen Programm oder in einem Speicherabzug in einer Dump-Datei.
Wenn Sie ein Datenelement, einen Anweisungsnamen oder eine Source-Referenz ansprechen wollen, die nicht in der aktuellen Programmeinheit liegen, so können Sie die entsprechende Speicherstelle über eine Qualifikation adressieren.

Änderungsdialog

Mit dem Kommando %AID CHECK=ALL schalten Sie den Änderungsdialog ein. Er wird bei der Ausführung von %MOVE oder %SET wirksam. AID fragt im Dialog nach, ob die Änderung des Speicherinhalts wirklich durchgeführt werden soll. Wird als Antwort ein N eingegeben, unterbleibt die Änderung; wird ein Y eingegeben, führt AID die Übertragung aus.

AID-Arbeitsbereich

ist der Adressraum, in dem Sie Adressen ohne Angabe von Qualifikationen ansprechen können.
Beim symbolischen Testen ist der AID-Arbeitsbereich die aktuelle Programmeinheit. Nur die Daten- und Anweisungsnamen und die Source-Referenzen, die innerhalb der aktuellen Programmeinheit liegen, sind ohne Angabe einer Qualifikation ansprechbar. Im geladenen Programm ist die aktuelle Programmeinheit die, in der sich gerade der Programmablauf befindet. In einem Speicherabzug ist die aktuelle Programmeinheit die, in der sich der Programmablauf befand, als der Speicherabzug erstellt wurde.
In einem Kommando können Sie vom AID-Arbeitsbereich abweichen, indem Sie im Adress-Operanden eine Qualifikation angeben. Mit dem Kommando %BASE können Sie den AID-Arbeitsbereich vom geladenen Programm in einen Speicherabzug verlegen oder umgekehrt.

AID-Ausgabedateien

sind die Dateien, in die Sie sich die Ausgaben der Kommandos %DISASSEMBLE, %DISPLAY, %HELP, %SDUMP und %TRACE schreiben lassen können. Die Dateien werden in den Ausgabekommandos über ihre Linknamen F0 bis F7 angesprochen (siehe %OUT und %OUTFILE).
In die Datei, die dem Linknamen F6 zugewiesen wurde, werden die REP-Sätze geschrieben (siehe %AID REP=YES und %MOVE).
Es gibt drei Wege, eine Ausgabedatei anzulegen:

1. /%OUTFILE-Kommando mit dem Link- und Dateinamen
2. /FILE-Kommando mit dem Link- und Dateinamen
3. Für einen Linknamen, dem noch kein Dateiname zugewiesen ist, setzt AID einen FILE-Makro mit dem Dateinamen AID.OUTFILE.Fn ab.

Eine AID-Ausgabedatei hat stets das Format FCBTYPE=SAM, RECFORM=V und OPEN=EXTEND.

AID-Eingabedateien

sind Dateien, die AID zur Ausführung von AID-Funktionen benötigt, im Unterschied zu Eingabedateien, die das Programm benutzt. AID verarbeitet nur Platten-Dateien. AID-Eingabedateien sind:

1. Dump-Dateien, in denen sich Speicherabzüge befinden (%DUMPFILE)
2. PLAM-Bibliotheken, in denen sich Bindemodule befinden. Wird die Bibliothek mit %SYMLIB zugewiesen, kann AID die LSD-Sätze nachladen.

AID-Literale

AID stellt Ihnen Zeichen-Literale und numerische Literale zur Verfügung (siehe AID-Basishandbuch, Kapitel 8):

```
{C'x...x' | 'x...x'C | 'x...x'} Character-Literal
{X'f...f' | 'f...f'X} Sedezimal-Literal
{B'b...b' | 'b...b'B}Binär-Literal
[{±}]n              Ganzzahl
#'f...f'            Sedezimalzahl
[{±}]n.m            Dezimalpunktzahl
[{±}]mantisseE[{±}]exponentGleitpunktzahl
```

AID-Standard-Arbeitsbereich

Beim Testen auf Maschinencode-Ebene ist das der nicht-privilegierte Teil des virtuellen Speichers in Ihrer Task, der vom Programm samt allen seinen konnektierten Subsystemen belegt ist.
Beim symbolischen Testen ist es die aktuelle Programmeinheit des geladenen Programms. Ohne Vereinbarung mit %BASE und ohne Angabe einer Basis-Qualifikation gilt der AID-Standard-Arbeitsbereich.

Aktuelle Aufrufhierarchie

ist der Stand der Unterprogrammverschachtelung an der Unterbrechungsstelle. Sie reicht von der Unterprogrammebene, auf der das Programm unterbrochen wurde über die durch CALL-Anweisungen verlassenen Unterprogramme mittlerer Hierarchiestufen bis zum Hauptprogramm.
Sie wird mit %SDUMP %NEST ausgegeben.

Aktuelles Programm

ist das Programm, das in der Task geladen ist, in der Sie AID-Kommandos eingeben.

Aktuelle Programmeinheit

ist die Programmeinheit, in der das Programm unterbrochen wurde. Die STOP-Meldung gibt ihren Namen aus.

Anweisungsname

bezeichnet die erste ausführbare FORTRAN-Anweisung nach einer Anweisungsmarke. Er wird mit *L'n'* angegeben. *n* ist eine maximal 5stellige Anweisungsmarke des Quellprogramms, die vom Programmierer vergeben wird. Führende Nullen dürfen nicht angegeben werden.
Anweisungsnamen *L'n'* sind genauso wie Source-Referenzen *S'n'* Adresskonstanten.

Attribute

Jedes Speicherobjekt hat bis zu sechs Attribute:
Adresse, Name (opt), Inhalt, Länge, Speichertyp, Ausgabetyp
Mit Selektoren können Sie auf Adresse, Länge und Speichertyp zugreifen. Über den Namen findet AID in den LSD-Sätzen alle zugehörigen Attribute, um damit zu arbeiten.
Adresskonstanten und Konstanten aus dem Quellprogramm haben nur bis zu fünf Attribute:
Name (opt), Wert, Länge, Speichertyp, Ausgabetyp
Sie haben keine Adresse. Beim Ansprechen einer Konstanten greift AID nicht auf ein Speicherobjekt zu, sondern setzt nur den dafür vorgemerkten Wert ein.

Ausgabetyp

Attribut eines Speicherobjekts, das bestimmt, wie der Speicherinhalt von AID ausgegeben wird. Jedem Speichertyp ist ein Ausgabetyp zugeordnet. Im AID-Basishandbuch, Kapitel 9 sind die AID-spezifischen Speichertypen samt zugehörigen Ausgabetypen aufgelistet. Für die in FORTRAN verwendeten Datentypen gilt eine entsprechende Zuordnung. Eine Typmodifikation bei %DISPLAY und %SDUMP bewirkt eine Änderung des Ausgabetyps.

Basis-Qualifikation

ist die Qualifikation, mit der Sie den AID-Arbeitsbereich in das geladene Programm oder in einen Speicherabzug in einer Dump-Datei legen. Sie wird mit E={VM | Dn} angegeben.
Die Basis-Qualifikation können Sie global mit %BASE vereinbaren oder im Adress-Operanden für eine einzelne Speicherreferenz angeben.

Bereichsgrenzen

Jedem Speicherobjekt ist ein bestimmter Bereich zugeordnet, der bei Datennamen und Schlüsselwörtern durch die Attribute Adresse und Länge festgelegt ist. Bei virtuellen Adressen liegen die Bereichsgrenzen zwischen V'0' und der letzten Adresse des virtuellen Speichers (V'7FFFFFFF'). Bei PROG-Qualifikationen ergeben sich die Bereichsgrenzen aus Anfangs- und Endadresse der Programmeinheit (siehe AID-Basishandbuch, Abschnitt 6).

Bereichsüberprüfung

AID überprüft bei Adressversatz, Längenmodifikation und bei *empfänger* in einem %MOVE, ob die Bereichsgrenzen der angesprochenen Speicherobjekte überschritten werden und gibt im Fehlerfall eine entsprechende Meldung aus.

Benutzerbereich

ist der Bereich des virtuellen Speichers, der vom geladenen Programm samt allen seinen konnektierten Subsystemen belegt ist. Er entspricht dem Bereich, der durch das Schlüsselwort %CLASS6 bzw. %CLASS6ABOVE und %CLASS6BELOW repräsentiert wird.

CSECT-Informationen

stehen in der Objekt-Strukturliste.

datenname

steht für alle Namen, die im Quellprogramm für Daten vergeben wurden. Mit *datenname* sprechen Sie Variablen, Konstanten und Felder beim symbolischen Testen an. Elemente von Feldern können Sie wie bei FORTRAN über einen Index ansprechen.

Datenelement

ist ein Sammelbegriff für alle in FORTRAN definierbaren Daten.

Datentyp

Gemäß dem im Quellprogramm deklarierten Datentyp ordnet AID allen Datenelementen einen AID-Speichertyp zu:

– Binärstring (≙ %X)
– Character (≙ %C)
– numerisch (≙ %F, %D)

Der zugeordnete Speichertyp bestimmt, wie das Datenelement von %DISPLAY ausgegeben, von %SET übertragen bzw. überschrieben und wie es in der Bedingung eines Subkommandos verglichen wird.

Eingabepuffer

AID hat einen internen Eingabepuffer. Reicht er für die Aufnahme der Eingabe eines Kommandos nicht aus, wird das Kommando mit einer Fehlermeldung als zu lang abgewiesen. Geben Sie weniger der wiederholbaren Operanden an, wird das Kommando angenommen.

ESD

External Symbol Dictionary ist das Verzeichnis der Externbezüge eines Moduls. Es wird vom Compiler erstellt. Hierin sind unter anderem Informationen über CSECTs, DSECTs und COMMONs enthalten. Der Binder greift auf dieses Verzeichnis zu, wenn er die Objekt-Strukturliste erzeugt.

globale Einstellungen

AID stellt Ihnen Kommandos zur Verfügung, mit denen Sie das Verhalten von AID Ihren Testerfordernissen anpassen können, die Ihnen die Adressierung erleichtern oder Schreibarbeit ersparen. Die Voreinstellungen gelten während der gesamten Testsitzung (siehe %AID, %AINT, %BASE und %QUALIFY).

Index

ist ein Teil eines Adress-Operanden. Mit einem Index wird die Position eines Feldelements bestimmt. Er kann wie in FORTRAN angegeben werden oder durch einen arithmetischen Ausdruck, aus dem AID den Wert des Index errechnet.

Kommandofolge

Mehrere Kommandos werden mit Semikolon (;) zu einer Folge verbunden, die von links nach rechts abgearbeitet wird. Wie im Subkommando darf eine Kommandofolge AID- und BS2000-Kommandos enthalten. In Kommandofolgen nicht zugelassen sind die AID-Kommandos %AID, %BASE, %DUMPFILE, %HELP, %OUT, %QUALIFY und die im Anhang aufgelisteten BS2000-Kommandos.
Enthält eine Kommandofolge eines der Kommandos zur Ablaufsteuerung, wird die Kommandofolge an der Stelle abgebrochen und das Programm gestartet (%CONTINUE, %RESUME, %TRACE) oder angehalten (%STOP). Nachfolgende Kommandos aus der Kommandofolge werden nicht mehr ausgeführt.

Kommandomodus

Mit Kommandomodus wird in den AID-Handbüchern der EXPERT-Modus der SDF-Kommandosprache bezeichnet. Falls Sie gerade in einem anderen Modus
(`GUIDANCE={MAXIMUM| MEDIUM| MINIMUM| NO}`) arbeiten, sollten Sie mit Kommando `MODIFY-SDF-OPTIONS GUIDANCE=EXPERT` in den EXPERT-Modus umschalten, wenn Sie AID-Kommandos eingeben wollen. AID-Kommandos verfügen nicht über eine SDF-Syntax:

- Operanden werden nicht über Menüs abgefragt.
- Im Fehlerfall gibt AID eine Fehlermeldung aus, führt aber keinen Korrekturdialog.

Im EXPERT-Modus fordert Sie das System mit "/" zur Kommandoeingabe auf.

Konstante

Eine Konstante repräsentiert einen Wert, der nicht über eine Adresse im Programmspeicher hinterlegt ist.
Zu den Konstanten gehören die im Quellprogramm definierten symbolischen Konstanten, die Ergebnisse von Längenselektion, Längenfunktion und Adressselektion sowie die Anweisungsnamen und die Source-Referenzen.
Eine Adresskonstante repräsentiert eine Adresse. Das sind Anweisungsnamen,
Source-Referenzen und das Ergebnis einer Adressselektion. In Verbindung mit einem Pointer-Operator (->) können Sie damit die entsprechende Speicherstelle adressieren.

LIFO

Last In First Out; Treffen an einem Testpunkt (%INSERT) oder bei Auftreten eines Ereignisses (%ON) Anweisungen aus verschiedenen Eingaben zusammen, so werden die zuletzt eingegebenen zuerst abgearbeitet (siehe AID-Basishandbuch, Abschnitt 5.4).

Lokalisierungsinformation

Mit %DISPLAY %HLLOC(speicherref) für die symbolische Ebene und mit %DISPLAY %LOC(speicherref) für die Maschinencode-Ebene gibt Ihnen AID die statische Programmverschachtelung zu der angegebenen Speicherstelle aus.
Im Gegensatz dazu erhalten Sie mit %SDUMP %NEST die dynamische Programmverschachtelung, die Aufrufhierarchie zur aktuellen Programmunterbrechungsstelle.

LSD

List for Symbolic Debugging ist ein Verzeichnis der im Modul definierten Daten- und Anweisungsnamen. Ebenso sind dort die vom Compiler erzeugten Source-Referenzen hinterlegt. Die LSD-Sätze werden vom Compiler erzeugt. AID holt sich hieraus die Informationen zur symbolischen Adressierung.

Namensraum

umfasst alle zu einer Programmeinheit in den LSD-Sätzen verzeichneten Datennamen.

Objekt-Strukturliste

Auf Basis des ESD (External Symbol Dictionary) erstellt der Binder die Objekt-Strukturliste, wenn die Standardeinstellung SYMTEST=MAP gilt bzw. wenn Sie SYMTEST=ALL angegeben haben.

Programmeinheit

Ein FORTRAN-Programm wird aus Programmeinheiten aufgebaut. Eine Programmeinheit ist eine Folge von Programmzeilen, die mit der END-Anweisung abgeschlossen wird. Man unterscheidet bei den Programmeinheiten zwischen Haupt- und Unterprogrammen. In einem Unterprogramm ist die erste Anweisung eine SUBROUTINE-, FUNCTION- oder BLOCK DATA-Anweisung, wobei BLOCK DATA-Programmeinheiten nicht mit AID-Kommandos angesprochen werden können. In einem Hauptprogramm ist die erste Anweisung in der Regel eine PROGRAM-Anweisung; jede beliebige andere FORTRAN-Anweisung ist jedoch ebenfalls als erste Anweisung zulässig.

Programmzustand

AID unterscheidet drei Programmzustände, in denen sich das zu testende Programm befinden kann:

1. Das Programm steht.
   %STOP, K2-Taste und eine PAUSE-Anweisung unterbrechen ein laufendes Programm. Außerdem wird das Programm unterbrochen, wenn ein %TRACE abgearbeitet ist. Die Task befindet sich im Kommandomodus. Sie können Kommandos eingeben.
2. Das Programm läuft ohne Ablaufverfolgung.
   %RESUME startet ein Programm oder setzt es fort. %CONTINUE bewirkt dasselbe; ist allerdings ein %TRACE noch nicht ganz abgearbeitet, so setzt %CONTINUE das Programm mit Ablaufverfolgung fort.
3. Das Programm läuft mit Ablaufverfolgung.
   %TRACE startet ein Programm oder setzt es fort. Der Programmablauf wird entsprechend der Vereinbarungen im %TRACE protokolliert. %CONTINUE bewirkt dasselbe, wenn noch ein %TRACE aktiv ist.

Qualifikation

Mit einer Qualifikation können Sie eine Adresse ansprechen, die nicht im aktuellen AID-Arbeitsbereich liegt. Die Basis-Qualifikation gibt an, ob die Adresse im geladenen Programm oder in einem Speicherabzug liegt. Die PROG-Qualifikation bezeichnet die Programmeinheit, in der die Adresse enthalten ist.
Wenn ein Operand durch eine Qualifikation überbestimmt ist (d.h. die Qualifikation ist überflüssig oder widersprüchlich), wird die Qualifikation ignoriert. Das ist z.B. der Fall, wenn eine PROG-Qualifikation für ein Datenelement der aktuellen Programmeinheit angegeben wird.

Source-Referenz

bezeichnet eine ausführbare Anweisung. Sie wird mit *S'n'* angegeben. *n* ist die Nummer einer Anweisung, die vom Compiler erzeugt wird und in der Übersetzungsliste der Spalte STMT zu entnehmen ist.
Source-Referenzen *S'n'* sind genauso wie Anweisungsnamen *L'n'* Adresskonstanten.

Speicherobjekt

ist eine bestimmte Anzahl von zusammenhängenden Bytes im Speicher. Auf Programmebene sind das die Daten des Programms, sofern ihnen ein Speicherbereich zugewiesen ist, und der Befehlscode. Außerdem gehören alle Register, der Befehlszähler sowie alle anderen Bereiche, die nur über Schlüsselwörter angesprochen werden können, ebenfalls zu den Speicherobjekten. Keine Speicherobjekte hingegen sind alle im Programm definierten Konstanten, die Anweisungsnamen, Source-Referenzen, die Ergebnisse von Adressselektion, Längenselektion und Längenfunktion und die AID-Literale. Sie repräsentieren einen Wert, der nicht verändert werden kann.

Speicherreferenz

Mit einer Speicherreferenz sprechen Sie ein Speicherobjekt an. Es gibt einfache und komplexe Speicherreferenzen. Einfache Speicherreferenzen sind virtuelle Adressen, Namen, zu denen AID sich die Adresse aus den LSD-Informationen holen kann, und Schlüsselwörter. Anweisungsnamen und Source-Referenzen sind in den AID-Kommandos %CONTROLn, %DISASSEMBLE, %INSERT, %JUMP und %REMOVE als Speicherreferenz erlaubt, obwohl es nur Adresskonstanten sind.
Mit den komplexen Speicherreferenzen geben Sie AID eine Vorschrift an, wie die gewünschte Adresse errechnet werden soll und welcher Typ und welche Länge gelten sollen. Folgende Operationen können in einer komplexen Speicherreferenz vorkommen: Adressversatz, indirekte Adressierung, Typmodifikation, Längenmodifikation und Adressselektion.

Speichertyp

ist entweder der Datentyp, der im Quellprogramm festgelegt wurde oder der durch Typmodifikation gewählte. AID kennt die Speichertypen %X, %C, %E, %P, %D, %F, %A (siehe %SET und AID-Basishandbuch, Kapitel 6 und 9).

Subkommando

ist ein Operand der Überwachungskommandos %CONTROLn, %INSERT oder %ON. Ein Subkommando kann einen Namen, eine Bedingung und einen Kommandoteil enthalten. Der Kommandoteil kann aus einem einzelnen Kommando oder aus einer Kommandofolge bestehen. Er kann AID- und BS2000-Kommandos enthalten. Jedes Subkommando hat einen Durchlaufzähler. Wie eine Ausführungsbedingung formuliert wird, wie Name und Durchlaufzähler vergeben

und angesprochen werden, und welche Kommandos innerhalb von Subkommandos nicht erlaubt sind, ist im AID-Basishandbuch, Kapitel 5 beschrieben. Der Kommandoteil des Subkommandos wird dann ausgeführt, wenn die Überwachungsbedingung des entsprechenden Kommandos (*kriterium*, *testpunkt*, *ereignis*) zutrifft und die eventuell definierte Ausführungsbedingung erfüllt ist.

Unterbrechungsstelle

Die Adresse, an der ein Programm unterbrochen wurde, wird Unterbrechungsstelle genannt. Aus der STOP-Meldung können Sie die Anweisung und die Programmeinheit der Unterbrechungsstelle entnehmen. Dort wird das Programm fortgesetzt. Mit %JUMP können Sie eine andere Fortsetzungsadresse vereinbaren (nur FOR1 ab V2.1A und COBOL85).

Literatur

Die Handbücher finden Sie im Internet unter *http://manuals.ts.fujitsu.com*. Handbücher, die mit einer Bestellnummer angezeigt werden, können Sie auch in gedruckter Form bestellen.

[1] **AID** (BS2000)
Advanced Interactive Debugger
Basishandbuch
Benutzerhandbuch

[2] **AID** (BS2000)
Advanced Interactive Debugger
Testen auf Maschinencode-Ebene
Benutzerhandbuch

[3] **AID** (BS2000)
Advanced Interactive Debugger
Testen von COBOL-Programmen
Benutzerhandbuch

[4] **AID** (BS2000)
Advanced Interactive Debugger
Testen unter Posix
Benutzerhandbuch

[5] **AID** (BS2000)
Advanced Interactive Debugger
Testen von ASSEMBH-Programmen
Benutzerhandbuch

[6] BS2000
Makroaufrufe an den Ablaufteil
Benutzerhandbuch

[7] BS2000
Programmiersystem
Technische Beschreibung

[8] **FOR1** (BS2000)
FORTRAN-Compiler
Benutzerhandbuch

[9] **FOR1** (BS2000)
FORTRAN-Compiler

Stichwörter

W

Z