Deutsch

FUJITSU Software BS2000

AID V3.4B

Testen von COBOL-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

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 estellten 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 immer neu übersetzt werden, sondern kann ohne LSD oder eventuell auch mit AID-Korrekturen, in den produktiven Einsatz gehen. Der Funktionsumfang von AID und seine Testsprache, die AID-Kommandos, sind primär auf die Dialoganwendung zugeschnitten. AID kann aber ebensogut im Batch-Betrieb eingesetzt werden. AID bietet Ihnen vielfältige Möglichkeiten zur Ablaufüberwachung, Ablaufsteuerung, Ausgabe und Änderung von Speicherinhalten. Außerdem können Sie Informationen über den Programmablauf und zur Handhabung von AID abfragen.

Mit AID können Sie sowohl auf der symbolischen Ebene der jeweiligen Programmiersprache als auch auf Maschinencode-Ebene testen. Beim „symbolischen Testen“ können Sie Daten, Anweisungen und Programmteile mit den im Quellcode vereinbarten Namen ansprechen, und Anweisungen ohne Namen mit der vom Compiler erzeugten Source-Referenz.

1.1 Zielsetzung und Zielgruppen der AID-Dokumentation

AID ist die Dialogtesthilfe für alle Software-Entwickler und Diagnostiker, die im BS2000 mit den Programmiersprachen ASSEMBH, COBOL, FORTRAN, C, C++, PL/I arbeiten und Programme testen und auch korrigieren wollen. Dieses Handbuch wendet sich an die Tester von COBOL-Programmen.

1.2 Konzept der AID-Dokumentation

Die Dokumentation für AID besteht aus einem Basishandbuch und den sprachspezifischen Handbüchern für das symbolische Testen sowie dem Handbuch für das Testen auf Maschinencode-Ebene. Zusätzlich liegt für den geübten AID-Anwender ein Tabellenheft [7] vor, in dem die Syntax der Kommandos und der Operanden mit kurzen Erläuterungen aufgeführt sind. Außerdem gibt es darin die %SET-Tabellen. 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 für alle 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, das Subkommando, die Adressierung bei AID, den Operanden Medium-und-Menge, die AID-Literale und die Schlüsselwörter. Das Handbuch enthält außerdem die in Kommandofolgen unzulässigen BS2000-Kommandos.

AID-Benutzerhandbücher
In den Benutzerhandbüchern finden Sie die Kommandos in alphabetischer Reihenfolge. Alle einfachen Speicherreferenzen sind in diesen Handbüchern beschrieben. Neben dem vorliegenden Handbuch
AID - Testen von COBOL-Programmen
gibt es noch die Benutzerhandbücher
AID - Testen von FORTRAN-Programmen [3]
AID - Testen unter Posix [4]
AID - Testen von ASSEMBH-Programmen [5]
AID - Testen von C/C++-Programmen [6]

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

Die zusätzlichen Testmöglichkeiten des „maschinennahen Testens“ sind beschrieben in **„AID - Testen auf Maschinencode-Ebene“ [2]**.
Dieses Handbuch brauchen Sie beim Testen von Programmen, 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 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.

kursive kleinbuchstaben
: Im Fließtext werden Operandennamen in *kursiven kleinbuchstaben* geschrieben.

```
⎧ 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-Kommandonamen 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.

3 Voraussetzungen zum symbolischen Testen

Für das symbolische Testen benötigt AID eine "List for Symbolic Debugging" (LSD), in der die innerhalb des Programms definierten symbolischen Namen verzeichnet sind. Diese LSD-Informationen werden vom Compiler erzeugt und können beim Binden übernommen und auch mitgeladen werden. Außerdem gibt es bei AID die Möglichkeit, die LSD bei Bedarf nachzuladen, wenn sie vom Compiler in einer PLAM-Bibliothek abgelegt wurden. Diese LSD-Informationen werden vom Compiler erzeugt und vom Bindelader oder vom Statischen Binder und Starter übernommen. Die Steueranweisungen für die Erzeugung der LSD durch den COBOL85/COBOL2000-Compiler sind nachfolgend kurz beschrieben. Im AID-Basishandbuch [1] finden Sie allgemeine Informationen zu LSD-Sätzen, zum Binden, Laden und Starten.

3.1 Übersetzen

Der COBOL85-Compiler (ab Version 1.2A) und der COBOL2000-Compiler lassen sich auf zwei Arten steuern:
– durch SDF-Optionen
– durch COMOPT-Anweisungen

Die LSD-Informationen erzeugt der COBOL-Compiler, entsprechend den beiden Steuerungsmöglichkeiten, bei Angabe folgender Operanden:

SDF-Steuerung

```
/START-COBOL2000-COMPILER ...,TEST-SUPPORT = AID[(...)]
```

In der Strukturklammer nach "AID" können weitere Optionen angegeben werden, die das Testen mit AID betreffen:

Mit der STMT-REFERENCE-Option legen Sie fest, ob die Source-Referenzen aus den im Quellprogramm (Spalte 1-6) enthaltenen oder den vom Compiler vergebenen Zeilennummern gebildet werden sollen.

Die PREPARE-FOR-JUMPS-Option bestimmt, ob für alle Paragraphen- und Kapitelanfänge in der PROCEDURE DIVISION Leerbefehle generiert werden sollen, damit das AID-Kommando %JUMP verwendet werden kann.

Die SHAREABLE-CODE-Option legt fest, ob der Code der PROCEDURE DIVISION (ohne DECLARATIVES) in einen eigenen Bindemodul geschrieben wird.

COMOPT-Steuerung

```
/START-EXECUTABLE-PROGRAM $.COBOL2000
 ...
 COMOPT SYMTEST=ALL
```

Die weiteren COMOPT-Anweisungen, die Einfluss auf das Testen mit AID haben können, lauten:

COMOPT TEST-WITH-COLUMN1=YES (entspricht SDF-Option STMT-REFERENCE)
COMOPT SEPARATE-TESTPOINTS=YES (entspricht SDF-Option PREPARE-FOR-JUMPS)
COMOPT GENERATE-SHARED-CODE=YES (entspricht SDF-Option SHAREABLE-CODE)

Eine ausführliche Darstellung der entsprechenden Operanden finden Sie im COBOL2000 (BS2000)-Benutzerhandbuch [13].

Segmentierte oder mehrfachbenutzbare Programme

Normalerweise erzeugt der COBOL-Compiler aus **einem** Quellprogramm auch **einen** Bindemodul. Für segmentierte und mehrfachbenutzbare (sharable) Programme werden jedoch zusätzliche Bindemodule erzeugt.

Segmentierung
Die Einteilung der Kapitel in COBOL-Programmen wird durch ein System von Segmentnummern verwirklicht. Die Segmentnummer ist in der Kapitelüberschrift (SECTION) enthalten. Die Segmentnummern 50 bis 99 bezeichnen unabhängige Segmente. Für diese Segmente werden eigene Bindemodule erzeugt.
Als Name eines solchen Moduls wird der, falls notwendig auf 6 Zeichen verkürzte, Name der PROGRAM-ID verwendet, dem die Segmentnummer angehängt wird. Dieser Name wird in der AID-Syntax als *segmentname* bezeichnet.
Damit AID die Überlagerungsstruktur eines Programms berücksichtigt, müssen Sie %AID OV=YES eingeben.

Mehrfachbenutzbarkeit
Sollen mehrere Benutzer (Tasks) auf einzelne COBOL-Programmteile zugreifen, können diese Programmteile mehrfachbenutzbar (sharable) gemacht werden. Der COBOL-Compiler ermöglicht über die SDF- oder COMOPT-Steuerung die Erzeugung eines entsprechenden Bindemoduls.
Als Name eines solchen Moduls wird der, falls notwendig auf 7 Zeichen verkürzte, Name der PROGRAM-ID verwendet, dem das Zeichen @ angehängt wird. Dieser Name wird in der AID-Syntax als *sharename* bezeichnet.

3.2 Binden, Laden und Starten

Übersetzte Programme binden, laden und starten Sie mit den für alle Sprachen gültigen SDF-Kommandos bzw. BINDER-Anweisungen. Sie sind im AID-Basishandbuch [1] beschrieben. Dort steht auch alles zu dem Parameter, mit dem Sie veranlassen, dass die vom Compiler erzeugten LSD-Informationen an den Binder (BINDER) oder an den Dynamischen Bindelader (DBL) weitergereicht werden, damit Sie symbolisch testen können. Außerdem gibt es noch die Möglichkeit des Nachladens von LSD aus einer PLAM-Bibliothek mit Hilfe des Kommmandos %SYMLIB.

3.3 Kommandos zu Beginn einer Testsitzung

Unmittelbar nach dem Laden steht das Programm noch vor der ersten Anweisung in der PROCEDURE DIVISION und Initialisierungen sind noch nicht durchgeführt. Deshalb können einzelne Kommandos zu einer Fehlermeldung führen und bei anderen kann die Angabe von Qualifkationen erforderlich sein, die Sie vermeiden können, wenn Sie mit einem %TRACE 1 das Programm auf die erste Anweisung im Programm laufen lassen.
Haben Sie vorher ein Programm einer Programmiersprache getestet, die den Bindestrich in Namen nicht zulässt oder die Kleinbuchstaben in Namen kennt, dann sollten Sie zuerst die Kommandos %AID SYMCHARS bzw. %AID %LOW=OFF eingeben oder sich mit %SHOW %AID die aktuellen Einstellungen globaler Parameter ansehen (siehe Kapitel „AID-Kommandos“ auf Seite 31).

Um eine längere AID-Ausgabe mit der K2-Taste unterbrechen zu können, muss mit dem Kommando `/MODIFY-TERMINAL-OPTIONS`
folgende Option eingestellt sein:
`OVERFLOW-CONTROL=*USER-ACKNOWLEDGE`

4 COBOL-spezifische Adressierung

In diesem Kapitel werden nur die Speicherreferenzen beschrieben, die für das symbolische Testen von COBOL-Programmen verwendet werden. Eine allgemeine Beschreibung der Adressierung finden Sie im AID-Basishandbuch [1]. Als symbolische Speicherreferenzen (siehe Abschnitt „Symbolische Speicherreferenzen“ auf Seite 21) verwenden Sie alle in den LSD-Sätzen verzeichneten Namen von Dateien, Daten und Anweisungen aus dem Programm und die vom Compiler erzeugten Source-Referenzen. Eventuell sind davor Qualifikationen erforderlich, wie sie im Anschluss beschrieben sind.

In allen Operanden, in denen *kompl-speicherref* möglich ist, können Sie beliebig wechseln zwischen den in diesem Handbuch beschriebenen Speicherreferenzen und denen für das AID Handbuch - Testen auf Maschinencode-Ebene [2], sofern es keine expliziten Einschränkungen gibt (siehe Abschnitt „Symbolische Speicherreferenzen“ auf Seite 21).

4.1 Qualifikationen

Qualifikationen verwenden Sie, wenn ein Speicherobjekt nicht im aktuellen AID-Arbeitsbereich liegt oder dort nicht eindeutig ist oder um einen Teilbereich zu bezeichnen. Es gibt die Basisqualifikation, mit der Sie den AID-Arbeitsbereich vereinbaren und die Bereichsqualifikationen, mit denen Sie Teile des Arbeitbereichs adressieren. Außerdem beschreiben Sie durch die Verbindung von Qualifikationen den Pfad zu einem Bereich bzw. einem Speicherobjekt.

Qualifikationen werden durch Punkte getrennt. Zwischen der letzten Qualifikation und einem anschließenden Operanden muss ebenfalls ein Punkt stehen.

Basisqualifikation

E={VM|Dn}
: Die Basisqualifikation legt fest, ob der AID-Arbeitsbereich im geladenen Programm (E=VM) oder in einer Dump-Datei (E=Dn) liegen soll. Sie ist im AID-Basishandbuch [1] und bei %BASE beschrieben. Auf eine Basisqualifikation können unmittelbar Bereichsqualifikationen oder Dateiname, ein Datenname, ein Sonderregister, eine figurative Konstante, ein Anweisungsname, eine Source-Referenz, ein Schlüsselwort oder eine komplexe Speicherreferenz folgen.

Bereichsqualifikationen

Mit diesen Qualifikationen bezeichnen Sie einen Teil des Arbeitsbereiches. Endet ein Adressoperand mit einer dieser Qualifikationen, so wirkt das Kommando nur in dem Teil, der mit der letzten Qualifikation bezeichnet wurde. Mit einer Bereichsqualifikation begrenzen Sie den Wirkungsbereich eines Kommandos oder machen damit einen Daten- oder Anweisungs-Namen im Arbeitsbereich eindeutig oder Sie erreichen damit einen Namen, der an der aktuellen Unterbrechungsstelle sonst nicht ansprechbar ist.

CTX=kontext

Die CTX-Qualifikation bezeichnet einen Kontext (siehe AID-Basishandbuch [1]). Sie kann nur einer S-Qualifikation vorangehen. Nur in den Kommandos %SDUMP und %QUALIFY kann ein Adressoperand mit der CTX-Qualifikation enden. Sie brauchen diese Qualifikation, wenn sie eine Übersetzungseinheit oder CSECT ansprechen wollen, in der nicht die aktuelle Unterbrechungsstelle liegt und die in mehreren Kontexten enthalten ist. *kontext* ist der im BIND-Makro explizit vergebene Name des Kontextes oder der implizit vergebene Name LOCAL#DEFAULT. Auch mit dem DBL geladene Programme erhalten den standardmäßig vergebenen Kontextnamen LOCAL#DEFAULT. Weitere Kontexte eines Programms können durch Anschließen an ein Shared-Code-Programm entstehen.

Um die Syntax für die Adressoperanden der einzelnen Kommandos nicht mehr aufzublähen, wurde die CTX-Qualifikation dort nicht aufgenommen, Außerdem wird sie zur Zeit eher selten benutzt. Im AID-Basishandbuch [1] finden Sie weitere Informationen auch im Zusammenhang mit dem Testen auf Maschinencode-Ebene.

Beispiele

```
%CONTROL1 IN CTX=LOCAL#DEFAULT.S=HAUPT.PROC=TEIL
```

Der control-Bereich liegt hier nicht im aktuellen Kontext, indem das Programm unterbrochen wurde, sondern im Kontext LOCAL#DEFAULT.

```
%SDUMP CTX=CTXPHASE
```

Die aktuelle Unterbrechungsstelle liegt in einem anderen Kontext der Aufrufhierarchie. In diesem %SDUMP begrenzen Sie das Kommando auf den angegebenen Kontext.

```
%INSERT CTX=LOCAL#DEFAULT.S=SOURCE.PROC=UNTER.UNTER
```

Die Übersetzungseinheit SOURCE ist sowohl im aktuellen Kontext als auch im Kontext LOCAL#DEFAULT vorhanden. Um den Testpunkt vereinbaren zu können, brauchen Sie die Kontext-Qualifikation.

S=srcname

Die S-Qualifikation bezeichnet eine Übersetzungseinheit.

srcname wird beim Übersetzen aus dem Programmnamen in PROGRAM-ID eines „vollständigen“ COBOL-Programms gebildet (siehe Sprachbeschreibung, „COBOL-Compiler“ [12], Kapitel „Programmkommunikation“, bzw. Benutzerhandbuch, „COBOL-Compiler“ [13], Kapitel „Ausgaben des Compilers“.)
srcname hat bis zu 8 Stellen, wenn ein Bindemodul (OM) bezeichnet wird und bis zu 30 Stellen, wenn es sich um einen Bindelademodul (LLM) handelt. Entsteht für einen Bindemodul durch Abschneiden ein *srcname*, der mit Bindestrich endet, muss die S-Qualifikation wie folgt geschrieben werden: `S=N'srcname'`

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].

PROC=program-id [•program-id]

Die PROC-Qualifikation bezeichnet ein COBOL-Programm. Es ist ein einzelnes Programm oder das äußerste, ein äußeres oder ein inneres Programm eines geschachtelten Programms.

program-id besteht aus den maximal 30 Stellen des Namens aus der PROGRAM-ID.

Operanden, die einen Adressbereich (%CONTROLn, %TRACE) oder einen Namensraum (%SDUMP) angeben, können mit der PROC-Qualifikation enden. Der Adressbereich bzw. Namensraum umfasst dann das gesamte Programm. Ansonsten geben Sie die PROC-Qualifikation an, wenn Sie einen Namen aus den LSD-Sätzen ansprechen, der nicht im aktuellen Programm liegt oder in der Übersetzungseinheit nicht eindeutig ist, also vor einem Dateinamen, Datennamen, Anweisungsnamen oder vor einer komplexen Speicherreferenz, sofern sie mit einem Namen beginnt.

- program-id
  Wird der Name eines Programms direkt im Anschluss an eine PROC-Qualifikation wiederholt, dann bezeichnen Sie damit die Adresse der ersten ausführbaren Anweisung in diesem Programm. Liegt die aktuelle Unterbrechungsstelle im selben Programm, kann die PROC-Qualifikation entfallen. Diese Angabe können Sie in %DISASSEMBLE und %INSERT verwenden.

PROG=program-id [•program-id]
Diese Bereichsqualifikation fasst die S- und PROC-Qualifkation zusammen. Sie kann nur benutzt werden, wenn die Namen der Übersetzungseinheit und des Programms identisch sind, also für ein „äußerstes“ Programm. Dann gilt das zur PROC-Qualifikation Gesagte.
Ist *program-id* länger als 8 Zeichen können Sie die PROG-Qualifikation nicht verwenden. Diese Einschränkung gilt nur für Bindemodule (OM).

Mit den hier folgenden C-Qualifikationen wechseln Sie auf die Maschinencode-Ebene. Unmittelbar anschließend kann kein symbolischer Operand geschrieben werden (siehe Abschnitt „Symbolische Speicherreferenzen“ auf Seite 21), nur eine *kompl-speicherref* (siehe AID-Basishandbuch [1]). Trotzdem erwartet bzw. ergänzt AID im %CONTROLn oder %TRACE ein symbolisches Kriterium. Vor einer C-Qualifikation können nur eine E- und gegebenenfalls eine CTX-Qualifikation stehen.

C=segmentname

Damit bezeichnen Sie ein Segment.
segmentname setzt sich zusammen aus den ersten 6 Stellen der PROGRAM-ID und der Segmentnummer aus der Kapitelüberschrift.

Mit dieser C-Qualifikation können Sie ein Segment als Bereich im %CONTROLn, %FIND, %ON *write-ereignis* oder %TRACE festlegen oder die Anfangsadresse des Segments als *start* im %DISASSEMBLE bzw. als *testpunkt* im %INSERT vereinbaren.

C=sharename

Damit bezeichnen Sie einen Modul, der mit der SDF-Option SHAREABLE-CODE=YES übersetzt wurde. Sie bezeichnen damit einen Bindemodul.
sharename setzt sich zusammen aus den ersten 7 Stellen der PROGRAM-ID und dem Zeichen @.

Mit dieser C-Qualifikation können Sie den Bindemodul als Bereich im %CONTROLn, %FIND, %ON *write-ereignis* oder %TRACE festlegen, sofern er im Klasse-6-Speicher geladen ist. Nicht ansprechen können Sie damit Module, die im Klasse-4-Speicher geladen sind.

4.2 Symbolische Speicherreferenzen

Als symbolische Speicherreferenzen können alle in den LSD-Sätzen verzeichneten Namen von Dateien, Daten und Anweisungen aus dem Programm und die vom Compiler erzeugten Source-Referenzen verwendet werden, außerdem die AID-Schlüsselwörter.
Keine LSD-Sätze werden erzeugt für 88er-Stufen, das NATIVE-Alphabet und für Definitionen aus der REPORT-SECTION (außer LINE-COUNTER, CBL-CTR, PAGE-COUNTER). Diese Daten können Sie deshalb mit AID nicht ansprechen.

Auf alle symbolischen Speicherreferenzen können Sie die im AID-Basishandbuch [1] beschriebenen Operationen anwenden. In allen Operanden, in denen das möglich ist, finden sie den Eintrag *kompl-speicherref*. Dann können Sie mit den beschriebenen Einschränkungen zwischen den in diesem Handbuch beschriebenen Speicherreferenzen und denen für das AID Handbuch - Testen auf Maschinencode-Ebene [2] wechseln.

dateiname

ist der Name einer Datei aus einer Dateierklärung in der FILE SECTION der DATA DIVISION.
In den Kommandos %DISPLAY und %SDUMP gibt AID folgende Informationen aus: den Datei-Status und, falls die Datei eröffnet ist, den Inhalt des Datensatz-Bereiches sowie eventuell den Satz-Schlüssel. Außerdem kann auf *dateiname* der Adress- und Längenselektor angewandt werden.

Beispiel

```
<FILE(size),format> file-name
_FILE_NAME = |file-name-complete|
_OPEN_MODE = {CLOSE | OPEN-INPUT | etc.}
_RECORD    =
[|]Content-of-the-current-record-(first-<80-symbols)[|][…]
```

file-name-complete ist ein vollqualifizierter Name der Datei *file-name*. Die Länge des Datensatzes der Datei beträgt *size* Bytes. *_RECORD* enthält einen Character-String (*format*=C) oder Bytes (X) des aktuellen Datensatz (gefolgt von drei Punkten, falls die Ausgabe unvollständig ist).

datenname

steht für alle im Quellprogramm in der DATA DIVISION definierten Namen von Datenfeldern, für die COBOL-Sonderregister und die figurativen Konstanten. Datenfelder sind sowohl Datensätze, Datengruppen und Tabellen als auch deren Elemente. Sie können gekennzeichnet und indiziert werden.
datenname ist eine maximal 30-stellige alphanumerische Zeichenfolge. Sie kann in allen Kommandos zur Ausgabe und Änderung von Daten angegeben werden; das sind die Kommandos %DISPLAY, %MOVE, %SDUMP, %SET; außerdem im %FIND-Kommando (Suchen einer Zeichenfolge) und im %ON-Kommando (Schreibüberwachung).

datenname [kennzeichnung][...] [(index[,...])]

kennzeichnung

Ist *datenname* nicht eindeutig innerhalb eines Programms, kann er gekennzeichnet werden, indem er mit IN oder OF einer bestimmten Datengruppe zugeordnet wird. *datenname* muss mit so vielen Kennzeichnungen versehen sein, wie er für seine eindeutige Bezeichnung braucht. Wird er nicht gekennzeichnet, dann muss es eine Definition auf der Stufe 01 oder 77 geben, die AID dann bearbeitet, sonst folgt eine Fehlermeldung.

In komplexen Speicherreferenzen ist die Angabe von *kennzeichnung* nicht immer möglich.

kennzeichnung wird folgendermaßen angegeben:

```
{IN }
{   }    datengruppenname
{OF }
```

index
ist *datenname* der Name eines Tabellenelements, kann er wie in einer COBOL-Anweisung indiziert bzw. subskribiert werden. Mehrere Indizes müssen im AID-Kommando im Gegensatz zu COBOL durch Komma getrennt werden.

Datendefinitionen, die einem *datennamen* mit OCCURS-Klausel untergeordnet sind, müssen im %SET bzw. %SDUMP mit so vielen Indizes versehen werden, wie in einer COBOL-Anweisung zum Zugriff angegeben werden müssen. Im %DISPLAY, %FIND und %MOVE können Sie die Indexangabe zu dem Datennamen weglassen, den Sie mit *datenname* ansprechen, und brauchen dann nur Indexangaben zu übergeordneten Indexstufen angeben (siehe Beispiel). Sonst kann ein *datenname*, der selbst mit der OCCURS-Klausel erklärt wurde, im %DISPLAY, %FIND und %MOVE ohne *index* angegeben werden. Damit sprechen Sie alle Elemente zu diesem Namen an.

index wird folgendermaßen angegeben:

```
{n                          }
{index-name                 }
{datenname                  }
{TALLY                      }
{arithmetischer ausdruck    }
```

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

index-name
ist der symbolische Name, der in der INDEXED BY-Klausel für die Indizierung einer Tabellenstufe vereinbart wurde.

datenname
bezeichnet ein numerisches Datenelement (außer Gleitpunkt) aus der DATA DIVISION, das gekennzeichnet werden kann. Es muss im selben Programm wie die Tabelle liegen.

TALLY
ist das vom COBOL-Compiler für jedes Programm erzeugte Sonderregister.

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

index-name kann nur mit *n* kombiniert werden und darf nur zur Indizierung der Tabellenstufe verwendet werden, der er über die INDEXED BY-Klausel zugewiesen wurde.

Beispiel

```
01  TABELLE.
  02  GRUPPE1                          OCCURS 10.
      04  ELEMENT1    PIC X(5).
      04  ELEMENT2    PIC 9(2)         OCCURS 6.
  02  GRUPPE2         PIC 9(2)         OCCURS 12.
01  FELD               PIC X(70).
01  EINGABE-STRUKTUR.
  02  GR1.
      04  ELEM1       PIC X(5).
      04  ELEM2       PIC 9(2).
  02  GR2             PIC 9(2)         OCCURS 12.
```

Die verschiedenen Datennamen können im AID-Kommando wie folgt angesprochen werden:

```
%DISPLAY GRUPPE1
```

Alle Elemente GRUPPE1(1) bis GRUPPE1(10)werden ausgegeben.

```
%MOVE GR1 INTO GRUPPE1
```

Mit dem Kommando werden alle Elemente GRUPPE1(1) bis GRUPPE1(10) mit dem Inhalt von GR1 überschrieben.

```
%MOVE GR1 INTO GRUPPE1(1)
```

Das erste Element GRUPPE1(1) wird überschrieben.

```
%MOVE GRUPPE2 INTO GR2
```

Der gesamte Inhalt von GRUPPE2(1) bis GRUPPE2(12) werden in GR2(1) bis GR2(12) übertragen. Dagegen ist es nicht möglich folgendes Kommando zu schreiben:

```
%SET GRUPPE2 (1) INTO GR2
```

Im %SET-Kommando muss wie in COBOL-Anweisungen voll indiziert werden.

```
%SET GRUPPE2 (1) INTO GR2 (12)
%SET GR2 (ELEM2) INTO ELEMENT2 (5,ELEM2)
```

Bereich von Indizes

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

Es kann auch 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.

COBOL-Sonderregister

Sie können nur die Sonderregister angeben, die für das Programm vom COBOL-Compiler erzeugt und bereits mit aktuellen Werten versorgt wurden. Zum Beispiel können Sie die SORT-Sonderregister nur dann angeben, wenn das Programm einen Sortierteil hat.

```
LINAGE-COUNTER
RETURN-CODE
SORT-CCSN
SORT-CORE-SIZE
SORT-EOW
SORT-FILE-SIZE
SORT-MODE-SIZE
SORT-RETURN
TALLY
```

Figurative Konstanten

datenname ist einer der COBOL-Namen für figurative Konstanten oder der Name eines *symbolic character*, der im SPECIAL-NAMES-Paragraphen vereinbart wurden. HIGH-VALUE und LOW-VALUE repräsentieren immer den alphanumerischen Wert, der ihnen standardmäßig oder nach den Vereinbarungen mit der PROGRAM COLLATING SEQUENCE-Klausel entspricht.

```
ZERO
SPACE
HIGH-VALUE
LOW-VALUE
QUOTE
symbolic character
```

anweisungsname

bezeichnet eine Adresskonstante. Sie enthält die Adresse des ersten Befehls in einem Kapitel oder Paragraphen in der PROCEDURE DIVISION.

```
{L'kapitel'                        }
{L'paragraph' [IN L'kapitel']      }
```

In den Kommandos %CONTROLn, %DISASSEMBLE, %INSERT, %JUMP und %TRACE kann ein alphanumerischer Kapitel- oder Paragraphenname auch ohne L'...' angegeben werden, da in diesen Kommandos eine Verwechslung mit einem Datennamen nicht möglich ist. Folgt in einer komplexen Speicherreferenz auf *anweisungsname* ein Pointer (->), muss die Form L'...' verwendet werden. Ist ein Paragraphenname innerhalb eines Programms nicht eindeutig, dann muss er mit dem Kapitelnamen des Kapitels, in dem er definiert wurde, gekennzeichnet werden:

```
L'paragraph' IN L'kapitel'
```

In den Kommandos %DISPLAY, %FIND, %MOVE und %SET bezeichnen Sie damit die Adresse. In den Kommandos %DISASSEMBLE, %INSERT und %JUMP bezeichnen Sie damit die Speicherstelle an dieser Adresse. In den Kommandos %CONTROLn und %TRACE bezeichnen Sie damit das gesamte Kapitel bzw. den gesamten Paragraphen.

source-referenz

ist eine Adresskonstante für die vom Compiler erzeugte Bezeichnung einer Anweisung. In Abhängigkeit von der SDF-Option TEST–SUPPORT mit dem Operanden STMT–REFERENCE hat sie einen unterschiedlichen Aufbau.

STMT-REFERENCE=LINE-NUMBER
S'n[verb[m]]'

n

ist die vom Compiler vergebene Zeilennummer in der PROCEDURE DIVISION. Führende Nullen dürfen nicht angegeben werden. Hier ist die *source-referenz* innerhalb einer Übersetzungseinheit eindeutig. S'n' geben Sie nur für Zeilen mit Paragraphen- oder Kapitelnamen an, in denen kein COBOL-Verb auftritt.

verb

ist die festgesetzte Abkürzung eines COBOL-Verbs in der mit *n* bezeichneten Anweisungszeile.
S'nverb' geben Sie für Zeilen an, in denen ein COBOL-Verb steht.

m

ist eine 1stellige Nummer > 1. *m* geben Sie nur an, wenn das gleiche COBOL-Verb mehrfach in einer Zeile steht und Sie nicht das erste ansprechen möchten. Sie bezeichnen damit das m-te COBOL-Verb innerhalb der Zeile.

STMT-REFERENCE=COLUMN1-TO-6
S'xverb[m]'

x

ist der unveränderte Inhalt der Spalten 1 bis 6 einer Quellprogrammzeile. Enthaltene Leerzeichen müssen angegeben werden.
Ist eine Quellcodezeile durch x...x nicht eindeutig innerhalb der gesamten Übersetzungseinheit zu identifizieren, so ist die Source-Referenz ebenfalls nicht eindeutig. Paragraphen und Kapitel können hier nicht mit einer Source-Referenz angesprochen werden.

verb

ist die festgesetzte Abkürzung eines COBOL-Verbs in der mit *x* bezeichneten Anweisungszeile.
S'xverb' geben Sie für Zeilen an, in denen ein COBOL-Verb steht.

m
ist eine 1stellige Nummer > 1. Sie bezeichnen damit das m-te COBOL-Verb innerhalb einer Zeile, wobei hier als Zeile alle Anweisungen bis zu einer neuen Zeilennummer interpretiert werden.

In %FIND und %ON *write-ereignis* muss auf die Source-Referenz der Pointer-Operator folgen. Damit bezeichnen Sie vier Byte des Maschinencodes ab der Adresse, die in der Adresskonstanten hinterlegt ist. In den Kommandos %DISPLAY, %MOVE und %SET bezeichnen Sie damit die Adresse. In den Kommandos %DISASSEMBLE, %INSERT und %JUMP bezeichnen Sie damit die Speicherstelle an dieser Adresse. In den Kommandos %CONTROLn und %TRACE können Sie mit zwei Source-Referenzen einen Bereich festlegen.

Beispiel

```
%DISPLAY S'95ADD2'
```

Das Programm wurde mit STMT-REFERENCE=LINE-NUMBER übersetzt. Die Source-Referenz bezeichnet die Adresse, die zur zweiten ADD-Anweisung in der Zeile 95, in den LSD-Sätzen hinterlegt wurde. Es ist die Adresse der Speicherstelle, an der der erste Befehl steht, der zu dieser Anweisung generiert wurde.

Abkürzung	COBOL-Verb	Abkürzung	COBOL-Verb
ACC	ACCEPT	INI	INITIATE
ADD	ADD	INSP	INSPECT
ADDC	ADD CORRESPONDING	INV	INVOKE
ALLO	ALLOCATE	KEE	KEEP
ALT	ALTER	MOD	MODIFY
CALL	CALL	MOV	MOVE
CANC	CANCEL	MOVC	MOVE CORRESPONDING
CLO	CLOSE	MRG	MERGE
COM	COMPUTE	MUL	MULTIPLY
CON	CONNECT	OPE	OPEN
CONT	CONTIUE	PER	PERFORM oder EXIT PERFORM
DEL	DELETE	PERT	TEST OF PERFORM
DIS	DISPLAY	RAIS	RAISE
DIV	DIVIDE	REA	READ
DSC	DISCONNECT	REDY	READY
END	END-xxx	REL	REALSE
ENTR	ENTRY	RET	RETURN
ERA	ERASE	REW	REWRITE
EVAL	EVALUATE	SEA	SEARCH
EXI	EXIT [PARAGRAPH/SECTION]	SET	SET
EXIT	EXIT {PROGRAM/METHOD}	SOR	SORT
FET	FETCH	STA	START
FIN	FINISH	STO	STOP
FND	FIND	STOR	STORE
FRE	FREE	STRG	STRING
GEN	GENERATE	SUB	SUBTRACT
GET	GET	SUBC	SUBTRACT CORRESPONDING
GO	GOBACK	TER	TERMINATE
GOT	GO TO	UNST	UNSTRING
IF	IF	WRI	WRITE
INIT	INITIALIZE		

Tabelle 1: Liste der COBOL-Verben und ihrer Abkürzungen

5 AID-Kommandos

%AID

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

– Mit *CCS* treffen Sie eine Voreinstellung, in welchem Zeichensatz Zeichen interpretiert werden, falls keine explizite Angabe im %DISPLAY-Kommando gemacht wird. Unicode-Zeichensätze sind nicht erlaubt.

– 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.

– 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.

– 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 *EBCDIC* spezifizieren Sie die EBCDIC-Codierung einer C-Zeichenkette in Form eines Coded-Character-Set-Namens (CCSN). AID verwendet diesen CCSN z.B. bei Konvertierungen von und nach UTF16-/UTFE-Strings.

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

Kommando	Operand
%AID	CCS [= {<coded-character-set> | *USRDEF}]
	CHECK [= {ALL|NO}]
	REP [= {YES|NO}]
	SYMCHARS [= {STD|NOSTD}]
	OV [= {YES|NO}]
	LOW [= {ON|OFF|ALL}]
	DELIM [= {C'x'|'x'C|'x' / '|'}]
	LANG [= {D | E}]
	EBCDIC={*USRDEF | <ebcdic-coded-character-set>}
	LEV [= {ON|OFF}]

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

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

%AID verändert den Programmzustand nicht.

CCS

<coded-character-set>
: Name des Zeichensatzes (<name 1..8>), in dem AID-Daten interpretiert werden. Der angegebene Zeichensatz muss XHCS bekannt sein, andernfalls weist AID die Anweisung mit der Meldung AID0555 ab.

*USRDEF
: CCSNAME des Zeichensatzes, der der Benutzer-Id zugewiesen ist. -*USRDEF ist der Standardwert von *CCS*.

Wenn Sie den *CCS*-Operand in einem %AID-Kommando eingeben, prüft AID mit Hilfe von XHCS, ob der CCSNAME zulässig ist. Wenn XHCS den CCSNAME nicht kennt, wird das Kommando zurückgewiesen und der aktuelle *CCS*-Wert wird beibehalten.

Mit folgendem AID-Kommando können Sie eine vollständige Liste der CCSNAMEs, die XHCS unterstützt, anzeigen:

```
%SHOW %CCSN
```

CHECK

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

```
OLD CONTENT:
AAAAAAAA
NEW CONTENT:
BBBBBBBB
%  AID0274 CHANGE DESIRED? REPLY (Y = YES; N = NO) ?

N

AID0342 NOTHING CHANGED
```

Nach der Eingabe **Y** wird der alte Speicherinhalt ohne weitere Meldung überschrieben. In Prozeduren im Stapelbetrieb kann AID keinen Dialog führen und nimmt immer **Y** an. Der alte bzw. neue Inhalt wird auf SYSOUT ausgegeben. Wird SYSOUT umgewiesen, dann sind diese Ausgaben am Terminal nicht zu sehen. Das gilt auch, wenn das %MOVE- bzw. %SET-Kommando mit dem CMD-Makro gegeben wurde und eine Ausgabe auf SYSOUT vereinbart ist. Dagegen geht die Meldung AID0274 und gegebenenfalls AID0342 immer auf das Medium Terminal.

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-Korrekturanweisungen im SDF-Format (REPs) erstellt. Wenn die Objekt-Strukturliste nicht zur Verfügung steht, erstellt AID keine Korrektursätze und gibt eine Fehlermeldung aus.

AID hinterlegt die Korrekturen in einer Datei mit dem Linknamen F6. Für den LMS-Lauf muss dann noch die MODIFY-ELEMENT-Anweisung eingefügt werden. Achten Sie 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), legt AID die Datei AID.OUTFILE.F6 an, in die es dann den REP schreibt. Benutzerspezifische REP-Dateien müssen mit Zugriffsmethode SAM angelegt sein. Von AID angelegte REP-Dateien werden ebenfalls mit Zugriffsmethode

SAM, Satzformat V und Eröffnungsart EXTEND angelegt.
Die Datei bleibt geöffnet, bis sie mit %OUTFILE geschlossen wird oder bis /LOGOFF bzw. /EXIT-JOB.

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.

OV

YES müssen Sie angeben, wenn Sie ein Programm mit Überlagerungsstruktur (Overlay) testen. AID überprüft dann jedes Mal, 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 Übersetzungseinheiten 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, %SDUMP und im Änderungsdialog des %SET) 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. Mit dem SDF-Kommando `MODIFY-MSG-ATTRIBUTES TASK-LANGUAGE=D` bekommen Sie auch die AID-Meldungen auf Deutsch. Der Änderungsdialog (siehe CHECK-Operand) wird dadurch nicht beeinflusst.

EBCDIC

*USERDEF
: Ist diejenige Codiertabelle, die der BS2000-Kennung zugeordnet ist. AID holt die Informationen bei der Initialisierung für einen Task oder bei Angabe von *USRDEF. Eine Änderung der Codiertabelle für die Kennung wirkt erst nach erneuter Eingabe von *USRDEF.

<ebcdic-coded-character-set>
: CCSNAME eines 1-Byte-EBCDIC-Codes, wie ihn XHCS unterstützt. Dieser Name kann auch beim BS2000-Kommando CREATE-FILE im Operanden CODED-CHARACTER-SET angegeben werden kann.
Bei der Eingabe des Kommandos prüft AID den CCSNAME über XHCS auf Zulässigkeit. Ist der CCSNAME XHCS nicht bekannt oder kein 1-Byte-EBCDIC, wird das Kommando abgewiesen und die bisherige Einstellung beibehalten.

AID verwendet die EBCDIC-Tabelle, die über das %AID-Kommando eingestellt ist, wenn eine Konvertierung zwischen einem UTFE-/UTFE16-String und einer C-Zeichenkette durchgeführt werden muss.

Die eingestellte EBCDIC-Codiertabelle wird außerdem für die Interpretation der Eingabezeichen (SYSCMD, SYSDTA) und die Zeichendarstellung bei Ausgaben (SYSOUT, SYSLST) verwendet.

Wenn den Eingabe- und Ausgabemedien keine eindeutige Codetabelle zugeordnet ist (bei CODED-CHARACTER-SET=*NONE für die entsprechende Datei oder CODED-CHARACTER-SET=7-BIT für das Terminal (TERMINAL-OPTION)), wird dem Medium standardmäßig der CODED-CHARACTER-SET der Benutzerkennung zugeordnet. Die jeweilige Zuordnung zeigt das Kommando %SH[OW] %CCSN.

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.

%AINT

Mit %AINT legen Sie fest, ob AID bei indirekter Adressierung mit 24-Bit-Adressen oder mit 31-Bit-Adressen arbeiten soll. Die Adresse vor dem Pointer-Operator (->) besteht für AID dann entsprechend aus 24 oder 31 Bits.
Der Adressierungsmodus des Testobjekts wird damit nicht beeinflusst.

– Mit *aid-mode* legen Sie die Adressinterpretation für indirekte Adressierung innerhalb eines AID-Arbeitsbereiches fest.

Kommando	Operand
%AINT	[aid-mode] [,...]

Standardmäßig interpretiert AID indirekte Adressangaben entsprechend dem aktuellen Adressierungsmodus des Testobjekts. Mit einem %AINT mit dem Schlüsselwort %MODE*n* schalten Sie diese automatische Anpassung aus. Den Adressierungsmodus des Testobjektes fragen Sie mit %DISPLAY %AMODE ab. Mit %MOVE können Sie ihn verändern. Mit %SHOW %AID oder %SHOW %BASE erhalten Sie neben anderen Informationen auch den für den aktuellen AID-Arbeitsbereich geltenden Adressierungsmodus.

Ohne die Angabe einer Qualifikation gilt %AINT für AID-Kommandos, die indirekte Adressen des aktuellen AID-Arbeitsbereichs ansprechen oder benutzen.

Ein %AINT ohne Operanden schaltet zurück auf die Standard-Adressinterpretation. Dasselbe bewirkt ein %AINT mit Basisqualifikation und ohne %MODE*n*. Sonst gilt der vereinbarte Adressierungsmodus bis /LOGOFF bzw. /EXIT-JOB.

%AINT verändert den Programmzustand nicht.

aid-mode

legt für den aktuellen oder den mit der angegebenen Basisqualifikation bezeichneten AID-Arbeitsbereich fest, wie indirekte Adressen in später folgenden AID-Kommandos interpretiert werden sollen.

Geben Sie ein Schlüsselwort für die Adressinterpretation und keine Qualifikation an, so gilt das %AINT-Kommando für die Bearbeitung des aktuellen AID-Arbeitsbereichs.

Geben Sie eine Basisqualifikation und kein Schlüsselwort für die Adressinterpretation an, gilt für den entsprechenden AID-Arbeitsbereich die AID-Standard-Adressinterpretation.

```
aid-mode-OPERAND  - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

[•][E={VM}[•]] [{%M[ODE]31}]
       {Dn}    {%M[ODE]24}

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

- Steht der Punkt an führender Stelle, ist er das Kennzeichen für eine *vorqualifikation*. Sie muss mit einem vorhergehenden %QUALIFY-Kommando definiert worden sein.
  Zwischen Basisqualifikation und dem Schlüsselwort zur Adressinterpretation muss ein Punkt gesetzt werden.

```
E={VM}
  {Dn}
```

geben Sie an, wenn die Umschaltung der Adressinterpretation nicht für den aktuellen AID-Arbeitsbereich gelten soll. Geben Sie nur eine Basisqualifikation an, so gilt für den damit angesprochenen Bereich wieder die Standard-Adressinterpretation.

```
{%M[ODE]31}
{%M[ODE]24}
```

Schlüsselwort, das angibt, wie viele Bits bei indirekter Adressierung in AID-Kommandos berücksichtigt werden sollen.

%M[ODE]31 31-Bit-Adressierung
%M[ODE]24 24-Bit-Adressierung

Beispiele

Die Adresse V’100’ hat den Inhalt: 1200000C
Register 5 hat den Inhalt: 010001A0

1. ```
   %AINT %MODE24
   %DISPLAY V'100'->
   %MOVE %5-> INTO %5G
   ```

   Mit %AINT stellen Sie auf 24-Bit-Adressinterpretation um. Die Umstellung gilt für den aktuellen AID-Arbeitsbereich.
   Der %DISPLAY gibt 4 Bytes ab Adresse V’00000C’ aus.
   Der %MOVE überträgt 4 Bytes ab Adresse V’0001A0’ in das AID-Register 5.
   ```

2.
```
%AINT %MODE31
%DISPLAY V'100'->
%MOVE %5-> INTO %5G
```

   Sie stellen die Adressinterpretation für den aktuellen AID-Arbeitsbereich auf 31-Bit-Interpretation um.
   Der %DISPLAY gibt 4 Bytes ab Adresse V’1200000C’ aus.
   Der %MOVE überträgt 4 Bytes ab Adresse V’010001A0’ in das AID-Register 5.

%BASE

Mit %BASE legen Sie die Basisqualifikation fest. Alle nachfolgend eingegebenen Speicherreferenzen ohne eigene Basisqualifikation 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 COBOL-Programmen entspricht der AID-Arbeitsbereich dem Bereich, den die Ladeeinheit 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 Basisqualifikation E=VM (Standardwert) und der AID-Arbeitsbereich entspricht dem nicht privilegierten Teil im virtuellen Speicher, der vom geladenen Programm mit allen konnektierten Subsystemen belegt ist (AID-Standard-Arbeitsbereich).

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

Unmittelbar bei der Eingabe werden alle Speicherreferenzen in einem Kommando, auch in einem Subkommando mit der aktuellen Basisqualifikation ergänzt, d.h. ein %BASE hat keine Auswirkung 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 Basisqualifikation fest. Alle nachfolgend eingegebenen Speicherreferenzen ohne eigene Basisqualifikation übernehmen die mit %BASE vereinbarte.

E=VM

Der virtuelle Speicherbereich des geladenen Programms ist als Basisqualifikation vereinbart. VM ist der Standardwert.

E=Dn

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

Bevor Sie eine Dump-Datei als Basisqualifikation 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 fort an der Unterbrechungsstelle oder der mit %JUMP vereinbarten Adresse.

Im Gegensatz zu %RESUME wird mit %CONTINUE ein aktiver %TRACE nicht beendet, sondern entsprechend den Vereinbarungen fortgesetzt.

Kommando	Operand
%CONT[INUE]	

Ein %TRACE ist aktiv, sobald er eingegeben wurde. In folgenden Fällen ist der %TRACE nur unterbrochen und kann mit %CONTINUE fortgesetzt werden:

1. Ein Subkommando wurde ausgeführt, weil eine Überwachungsbedingung aus einem %CONTROL*n*, %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 (siehe Abschnitt „Kommandos zu Beginn einer Testsitzung“ auf Seite 15).

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 %CONTROL*n* können Sie nacheinander bis zu sieben Ablaufüberwachungs-Funktionen vereinbaren, die dann gleich<zeitig wirken. Es gibt %CONTROL1 bis %CONTROL7.

– Mit *kriterium* wählen Sie verschiedene Typen von COBOL-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.

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

Mehrere %CONTROL*n* 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 %CONTROL*n* zusammen, so werden die zugehörigen Subkommandos in der Reihenfolge %C1 bis %C7 bearbeitet.

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

Mit %REMOVE löschen Sie eine bestimmte oder alle aktiven %CONTROL*n*-Vereinbarungen.

%CONTROL*n* kann nur im laufenden Programm eingesetzt werden; deshalb muss die Basisqualifikation E=VM eingestellt sein (siehe %BASE) oder explizit angegeben werden.

%CONTROL*n* verändert den Programmzustand nicht.

kriterium

Schlüsselwort, das den Typ der COBOL-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 %CONTROL*n* eine *kriterium*-Vereinbarung gültig ist.

kriterium	***subkdo* wird bearbeitet vor**
%STMT	jeder COBOL-Anweisung
%ASSGN	COBOL-Anweisungen, die den Inhalt eines Datenfeldes verändern: ADD [CORRESPONDING], COMPUTE, DIVIDE, INITIALIZE, INSPECT, MOVE [CORRESPONDING], MULTIPLY, SET, STRING, SUBTRACT [CORRESPONDING], UNSTRING
%CALL	CALL-, CANCEL-, INVOKE-, PERFORM-Anweisungen sowie vor SORT/MERGE- Anweisungen, da diese eine INPUT oder OUTPUT PROCEDURE aufrufen können.
%COND	EVALUATE-, IF-, SEARCH-Anweisungen und den bedingt durchlaufenen THEN-, ELSE-, WHEN-Anweisungszweigen.
%DB	COBOL-Anweisungen zum Aufruf einer Datenbank: CONNECT, DISCONNECT, ERASE, FETCH, FIND, FINISH, FREE, GET, KEEP, MODIFY, READY, STORE
%EXCEPTION	den bedingten Anweisungs-Zweigen bzw. ihren zulässigen Negationen: AT END, AT END OF PAGE, INVALID KEY, ON SIZE ERROR, ON OVERFLOW, ON EXCEPTION, der RAISE-Anweisung sowie vor der Ausführung einer USE PROCEDURE
%GOTO	ALTER-, CONTINUE-, GOTO-, RESUME-Anweisungen
%IO	COBOL-Anweisungen, die IO-Operationen veranlassen ACCEPT, DISPLAY, OPEN, CLOSE, DELETE, READ, REWRITE, START, WRITE, GENERATE, INITIATE, TERMINATE
%LAB	COBOL-Anweisungen, die einen Kapitel- oder Paragraphen-namen haben oder ihm unmittelbar folgen
%PROC	Programm- oder Modul-Start am Beginn der PROCEDURE DIVISION oder beim ENTRY Programm- oder Modul-Ende durch die Anweisung STOP RUN, GOBACK, EXIT METHOD oder EXIT PROGRAM
%SORT	MERGE- und SORT-Anweisungen, RELEASE- und RETURN-Anweisungen

Tabelle 2: *kriterium*-Vereinbarung für die Bearbeitung von *subkdo*

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.

control-bereich ist bei einem Programm ohne Segmentierung auf eine Übersetzungseinheit beschränkt; bei einem Programm mit Segmentierung auf ein Segment. Die Beschränkung auf ein Segment gilt nur für unabhängige Segmente (Segment-Nr. ≥ 50).

Eine *control-bereich*-Definition gilt bis zum nächsten %CONTROL*n* derselben Nummer mit neuer Definition, bis zum entsprechenden %REMOVE oder bis Programmende. Ein %CONTROL*n* ohne eigenen *control-bereich*-Operanden, übernimmt eine wirksame Bereichsdefinition. Ein wirksamer *control-bereich* muss in einem %CONTROL*n* 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 Übersetzungseinheit oder das aktuelle Segment.

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

                 ⎧[S=srcname] [[•]PROC=program-id]                          ⎫
                 ⎪                                                          ⎪
                 ⎪                    ⎧[PROC=program-id•] anweisungsname    ⎫⎪
IN  [•][E=VM•] ⎨[S=srcname•]        ⎨                                     ⎬⎬
                 ⎪                    ⎩(source-referenz:source-referenz)    ⎭⎪
                 ⎪C=segmentname                                             ⎪
                 ⎩C=sharename                                               ⎭

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

- 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 Basisqualifikation eine Dump-Datei vereinbart ist (siehe %BASE).

S=srcname

geben Sie an, wenn *control-bereich* nicht in der aktuellen Übersetzungseinheit liegen soll oder wenn eine vereinbarte Bereichseinschränkung nicht mehr gelten soll.

PROC=program-id
: geben Sie an, wenn *control-bereich* nicht im aktuellen Programm liegen soll, wenn er mit *anweisungsname* festgelegt werden soll und dieser in der Übersetzungseinheit nicht eindeutig ist
oder um eine bisher geltende *control-bereich*-Vereinbarung zu überschreiben. Endet *control-bereich* mit einer PROC-Qualifikation, so umfasst er das gesamte angegebene Programm. Diese muss zum Zeitpunkt der Eingabe des %CONTROL*n* bzw. bei Abarbeitung des Subkommandos, in dem der %CONTROL*n* steht, geladen sein.

Stimmt der *srcname* in der S-Qualifikation mit *program-id* überein, schreiben Sie an Stelle der beiden nur die PROG-Qualifikation.

Obwohl Sie mit den folgenden C-Qualifikationen auf die Maschinencode-Ebene wechseln, können Sie im Anschluss daran nur ein *kriterium* aus der vorhergehenden Tabelle „kriterium-Vereinbarung für die Bearbeitung von subkdo“ auf Seite 45 auswählen bzw. AID setzt den Standardwert `%STMT` ein.

C=segmentname
: Mit dieser Angabe erklären Sie das bezeichnete Segment zum *control-bereich*. Sie ist nur erforderlich, wenn die Unterbrechungsstelle nicht in diesem Segment liegt oder wenn eine bisher geltende Bereichseinschränkung auf Teile dieses Segmentes aufgehoben werden soll.

C=sharename
: Mit dieser Angabe erklären Sie den bezeichneten Bindemodul zum *control-bereich*. Diese Angabe ist nur dann erforderlich, wenn sich die Unterbrechungsstelle nicht in dem angegebenen Bindemodul befindet oder wenn eine für diesen Bindemodul geltende Bereichseinschränkung aufgehoben werden soll.

anweisungsname
: *control-bereich* wird durch einen Anweisungsnamen festgelegt und umfasst ein Kapitel oder einen Paragraphen in der PROCEDURE DIVISION.

```
{L'kapitel'                          }
{L'paragraph' [IN L'kapitel']        }
```

Ein alphanumerischer Kapitel- oder Paragraphenname kann auch ohne L'...' angegeben werden, da in diesem Kommando eine Verwechslung mit einem Datennamen nicht möglich ist.

Ist ein Paragraphenname innerhalb eines Programms nicht eindeutig, dann muss er mit dem Kapitelnamen des Kapitels, in dem er definiert wurde, gekennzeichnet werden: `L'paragraph' IN L'kapitel'`

(source-referenz : source-referenz)

control-bereich wird durch die Angabe einer Anfangs- und Endsourcereferenz festgelegt. Beide müssen innerhalb derselben Übersetzungseinheit liegen und es gilt: Anfangssourcereferenz ≤ Endsourcereferenz.
Soll *control-bereich* nur eine Anweisungszeile umfassen, müssen Anfangs- und Endsourcereferenz gleich sein.
Eine Begrenzung von *control-bereich* auf einzelne COBOL-Verben innerhalb einer Zeile ist nicht möglich.

source-referenz

bezeichnet die Adresse des ersten Befehls, der zu einer Anweisung in der PROCEDURE DIVISION generiert wurde, und muss in einem der folgenden Formate angegeben werden:

S'n'

für Zeilen mit Paragraphen- oder Kapitelnamen, in denen kein COBOL-Verb auftritt. Diese Angabe ist für Programme nicht möglich, die mit `STMT-REFERENCE=COLUMN1-TO-6` übersetzt wurden.

S'nverb' | S'xverb'

für Zeilen, in denen ein COBOL-Verb steht.

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 AID-Basishandbuch [1].

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

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

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

Ein Subkommando kann einen Namen, eine Bedingung und einen Kommandoteil enthalten. Zu jedem Kommando 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 %CONTROL*n* sind zusätzlich zu den Kommandos, die in keinem Subkommando zugelassen sind, die AID-Kommandos %CONTROL*n*, %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, %PROC IN (S'123':S'250GOT')<%DISPLAY ZAEHLER; %STOP>`
   `%C1 %CALL,%PROC IN (S'123':S'250GOT') <%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 Anweisungszeilen 123 bis einschließlich 250 wirken. In der Anweisungszeile 123 steht kein COBOL-Verb, in der Anweisungszeile 250 steht das COBOL-Verb GO TO.

   Tritt im Programmablauf im genannten Bereich eine COBOL-Anweisung auf, die den Kriterien %CALL oder %PROC entspricht, wird aus *subkdo* der %DISPLAY für das Feld 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- oder PERFORM-Anweisung führt AID den %DISPLAY aus *subkdo* aus und unterbricht dann das Programm auf Grund des %STOP-Kommandos.

3. `%CONTROL2 %SORT <%SDUMP %NEST P=MAX; %REMOVE %C1>`
   Bevor eine SORT-Anweisung ausgeführt wird, gibt AID die 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 <%STOP>`
   Mit dem %C3 wird vereinbart, dass AID ein %STOP-Kommando ausführen soll, bevor die erste Anweisung der PROCEDURE DIVISION oder die erste Anweisung nach einer ENTRY-Anweisung ausgeführt oder bevor der Modul verlassen oder das Programm beendet wird.

5. `%C4 %PROC <(SLF LE 10): %D TAB(1)>`
   Mit dem %C4 wird vereinbart, dass AID das erste Tabellenelement mit Namen TAB ausgeben soll vor Programm- oder Modul-Start bzw. vor Programm- oder Modul-Ende, wenn der Wert in SLF kleiner oder gleich 10 ist.

%DISASSEMBLE

Mit %DISASSEMBLE können Sie Speicherinhalte in symbolische Assembler-Notation rückübersetzen und ausgeben lassen.

– Mit *ausgabe-menge* legen Sie den Umfang der Speicherinhalte fest, die ausgegeben werden sollen.

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

Kommando	Operand
{ %DISASSEMBLE %DA }	[ausgabe-menge] [FROM start]

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 %DA-Kommando solange fortsetzen, bis Sie mit einem BS2000- oder AID-Kommando (START-EXECUTABLE-PROGRAM, LOAD-EXECUTABLE-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 *ausgabe-menge* nicht angegeben, so erzeugt AID dieselbe Menge 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 an, dann arbeitet AID mit Standard-Werten (10 für *ausgabe-menge* und V'0' für *start*). Wurde das Programm nicht ab V'0' geladen, muss *start* angegeben werden.

Mit dem Kommando %OUT können Sie steuern, wie die aufbereitete Speicherinformation dargestellt wird und ob sie auf SYSOUT, SYSLST oder in eine katalogisierte Datei ausgegeben werden soll. Der Aufbau der möglichen Ausgabezeilen ist im Anschluss an die Beschreibung des *start*-Operanden nachzulesen.

%DISASSEMBLE verändert den Programmzustand nicht.

ausgabe-menge

legt den Umfang der Speicherinhalte fest, die ausgegeben werden sollen. Wenn Sie *ausgabe-menge* nicht angeben, setzt AID beim ersten %DISASSEMBLE nach dem Laden des Programms den Standardwert 10 ein.

Bei jedem weiteren %DISASSEMBLE wird die zuletzt angegebenen *ausgabe-menge* genutzt.

```
ausgabe-menge-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - -

{anzahl }
{laenge }
{ALL    }

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

anzahl
: gibt an, wieviele Assembler-Befehle rückübersetzt und ausgegeben werden sollen.

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

laenge
: gibt die Größe des Speicherinhalts an, der innerhalb eines einzelnen, eingegebenen %DISASSEMBLE-Kommandos interpretiert und ausgegeben werden soll.

  ist eine Sedezimalzahl #'f..f' mit einem Wert:
  $1 \leq laenge \leq 2^{31}-1$

ALL
: gibt an, dass die Assembler-Befehle bis zum Ende der CSECT rückübersetzt und ausgegeben werden sollen, in der der *start*-Wert liegt. Wenn *start* nicht angegeben ist, bestimmt die aktuelle %DA-Position die CSECT.

  Wenn der *start*-Wert nicht innerhalb einer CSECT liegt, wird das Kommando mit einer Fehlermeldung abgewiesen.

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 nach dem Laden eines Programms den Standardwert V'0' ein. Wenn ein Programm nicht ab V'0' geladen wurde, gibt AID eine Fehlermeldung aus. Bei jedem weiteren %DISASSEMBLE wird hinter dem zuletzt rückübersetzten Assembler-Befehl fortgefahren.

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

                        ⎧C=segmentname     ⎫
                        ⎪C=sharename       ⎪
                        ⎪program=id        ⎪
FROM  [•][qua•][...]    ⎨                  ⎬
                        ⎪anweisungsname    ⎪
                        ⎪source=referenz   ⎪
                        ⎩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 Qualifikationen geben Sie an, wenn ein Adressoperand nicht für den aktuellen AID-Arbeitsbereich, die aktuelle Übersetzungseinheit oder das Programm gilt oder wenn er sonst nicht eindeutig ist.

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

S=srcname
: geben Sie nur an, wenn *start* nicht in der aktuellen Übersetzungseinheit liegen soll.

PROC=program-id
: geben Sie nur an, wenn *start* nicht im aktuellen Programm liegen soll (siehe Kapitel „COBOL-spezifische Adressierung“ auf Seite 17), oder mit *anweisungsname* festgelegt werden soll und dieser in der Übersetzungseinheit nicht eindeutig ist.

Stimmt der *srcname* in der S-Qualifikation mit *program-id* überein, schreiben Sie an Stelle der beiden nur die PROG-Qualifikation.

Vor die anschließend aufgeführten C-Qualifikationen können Sie nur die Basisqualifikation bzw. die CTX-Qualifikation setzen. Mit der C-Qualifikation verlassen Sie die symbolische Ebene. Unmittelbar anschließend kann kein symbolischer Operand geschrieben werden (siehe Abschnitt „Symbolische Speicherreferenzen“ auf Seite 21), nur eine *kompl-speicher-ref*.

C=segmentname
: Mit dieser Angabe setzen Sie *start* auf die Anfangsadresse des bezeichneten Segments.

C=sharename
: Mit dieser Angabe setzen Sie *start* auf die Anfangsadresse des bezeichneten Bindemoduls.

program-id
: Diese Angabe ist nach einer expliziten PROC/PROG-Qualifikation mit derselben *program-id* möglich, oder wenn die aktuelle Unterbrechungsstelle im mit *program-id* bezeichneten Programm liegt. Damit setzen Sie *start* auf die erste ausführbare Anweisung des bezeichneten Programms.

anweisungsname
: bezeichnet die Adresse des ersten Befehls in einem Kapitel oder Paragraphen in der PROCEDURE DIVISION.

  ```
  {L'kapitel'                        }
  {L'paragraph' [IN L'kapitel']      }
  ```

  Ein alphanumerischer Kapitel- oder Paragraphenname kann hier auch ohne L’...’ angegeben werden.

  Ist ein Paragraphenname innerhalb eines Programms nicht eindeutig, dann muss er mit dem Kapitelnamen des Kapitels, in dem er definiert wurde, gekennzeichnet werden: `L'paragraph' IN L'kapitel'`. Wenn Sie einen Adressversatz anschließen wollen, müssen Sie erst einen Pointer-Operator (->) schreiben.

source-referenz
: bezeichnet die Adresse des ersten Befehls, der zu einer Anweisung in der PROCEDURE DIVISION generiert wurde, und muss in einem der folgenden Formate angegeben werden:

  S’n’
  : für Zeilen mit Paragraphen- oder Kapitelnamen, in denen kein COBOL-Verb auftritt. Diese Angabe ist für Programme nicht möglich, die mit `STMT-REFERENCE=COLUMN1-TO-6` übersetzt wurden.

  S’nverb[m]’ | S’xverb[m]’
  : für Zeilen, in denen ein COBOL-Verb steht.

Wenn Sie einen Adressversatz anschließen wollen, müssen Sie erst einen Pointer-Operator (->) schreiben.

kompl-speicherref

sollte die Anfangsadresse eines Maschinenbefehls ergeben; andernfalls erhalten Sie eine unsinnige Disassemblierung. Folgende Operationen können in einer *kompl-speicherref* vorkommen (siehe AID-Basishandbuch[1]):

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

Soll ein Anweisungsname oder eine Source-Referenz als Speicherreferenz verwendet werden, muss anschließend der Pointer-Operator (->) geschrieben werden. In diesem Fall muss *anweisungsname* mit L'...' angegeben werden. Ohne den Pointer-Operator können Anweisungsname und Source-Referenz überall da verwendet werden, wo auch Sedezimalzahlen geschrieben werden können.
Beispiel: `%DISASSEMBLE L'AUSGABE'->.4`
Vom ersten Befehl, der im Kapitel AUSGABE steht, wird um 4 Bytes weiterpositioniert und ab dort wird disassembliert.

Eine Typmodifikation ist nur sinnvoll, wenn der Inhalt eines Datenelementes als Adresse eingesetzt werden kann oder wenn Sie die Adresse aus einem Register entnehmen.
Beispiel: `%1G.2%AL2->`
Die letzten beiden Bytes aus AID-Register %1G werden als Adresse benutzt.

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 Anfangsadresse im %DISASSEMBLE-Kommando bezieht sich auf das Beispiel in Abschnitt „Quellprogrammliste“ auf Seite 171.

```
/%DISASSEMBLE 8 FROM L'VORLAUF'->.4
 M0BS+9FC          UNPK  0(4,R4),12C(1,R12)     F3 30 4000 C12C
 M0BS+A02          LA    R4,28(R0,R3)           41 40 3028
 M0BS+A06          LR    R0,R0                  18 00
 M0BS+A08          L     R15,98(R0,R11)         58 F0 B098
 M0BS+A0C          BALR  R14,R15                05 EF
 M0BS+A0E          STH   R0,0(R0,R0)            40 00 0000
 M0BS+A12          DC   X'0004'  INVALID OPCODE 00 04
 M0BS+A14          DC   X'0000'  INVALID OPCODE 00 00
```

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 8 FROM L'VORLAUF'->.4
 000009FC  UNPK  0(4,R4),12C(1,R12)
 00000A02  LA    R4,28(R0,R3)
 00000A06  LR    R0,R0
 00000A08  L     R15,98(R0,R11)
 00000A0C  BALR  R14,R15
 00000A0E  STH   R0,0(R0,R0)
 00000A12  DC   X'0004'  INVALID OPCODE
 00000A14  DC   X'0000'  INVALID OPCODE
```

Beispiele

1. `%DISASSEMBLE FROM PROG=BEISPIEL.AUS2 IN AUSGABE`
   Das Kommando veranlasst die Rückübersetzung von 10 Befehlen (Standardwert) ab der Adresse des ersten ausführbaren Befehls des Paragraphen AUS2 im Kapitel AUSGABE.

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

3. `%DA FROM S'45INIT'`
   Da für *ausgabe-menge* kein Wert angegeben wurde, setzt AID entweder den Standardwert 10 ein, wenn es das erste %DISASSEMBLE für dieses Programm ist, oder übernimmt den Wert aus dem vorherigen %DISASSEMBLE. Die Rückübersetzung beginnt mit dem ersten Befehl, der zur Anweisung S’45INIT’ generiert 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 Datenfelder, deren Adressen und Längen, Anweisungen, Dateierklärungen, Register, Durchlaufzähler von Subkommandos, Systeminformationen, COBOL-Sonderregister und Figurative Konstanten. 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* im aktuellen Programm an.

Mit einer Qualifikation können Sie *daten* in einer Dump-Datei oder in einer anderen geladenen Übersetzungseinheit oder Programmeinheit ansprechen.

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 [1]).

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

Es empfiehlt sich, das Kommando nicht unmittelbar nach dem Laden einzugeben, da noch nicht alle Einträge in der DATA DIVISION initialisiert sind (z.B. Datensatzbeschreibungen und Sonderregister).

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

%DISPLAY verändert den Programmzustand nicht.

Die folgenden „Namen“ werden vom COBOL-Compiler automatisch für jede Übersetzungseinheit angeboten:

_COMPILER	Compiler, von dem das Objekt übersetzt wurde
_COMPILATION_DATE	Datum der Übersetzung
_COMPILATION_TIME	Uhrzeit der Übersetzung
_PROGRAM_NAME	ID-Name des Objekts
_EBCDIC_CCSN	Name der EBCDIC Variante, die bei Konvertierungen zwischen alphanumerischen und nationalen Daten angenommen wird (steht erst ab COBOL2000 V1.4A zur Verfügung)

AID ab V3.4B10 unterstützt auch die Ausgabe von Daten in unterschiedlichen EBCDIC- und ASCII-Zeichensätzen. Da BS2000-Terminals nur ausgewählte EBCDIC-Zeichensätze direkt unterstützen, muss zwischen folgenden Zeichensätzen unterschieden werden:

- Datenzeichensatz: Zeichensatz, in dem die Daten vorliegen bzw. interpretiert werden
- Ausgabezeichensatz: Zeichensatz, in dem die Daten dargestellt werden

AID interpretiert die Daten in dem Zeichensatz der beim %DISPLAY-Kommando angegeben ist. Wenn dort keiner angegeben ist, wird der Zeichensatz aus dem Operanden CCS des %AID-Kommandos verwendet.

Der Ausgabezeichensatz muss vorher mit dem Kommando /MODIFY-TERMINAL-OPTIONS eingestellt werden. Es muss also ein EBCDIC-Zeichensatz sein, der vom Terminal unterstützt wird. UTFE ist nicht zulässig. Außerdem muss der Ausgabezeichensatz in der gleichen Gruppe sein, wie der Datenzeichensatz. Wenn beispielsweise der Datenzeichensatz ISO88592 ist, muss zunächst mit /MOD-TERM-OPT CODE=EDF042 der entsprechende Ausgabezeichensatz eingestellt werden (siehe Benutzerhandbuch XHCS).

```
%DISPLAY <data-start> { %C|%X }[Lddd] ['<coded-character-set>']
```

Wenn Sie das Kommando %DISPLAY mit dem Speichertyp %C oder %X eingeben, gibt AID Zeichen in Übereinstimmung mit dem explizit definierten Zeichensatz <coded-character-set> aus, oder in Übereinstimmung mit dem aktuellen, Zeichensatz CCS ('<coded-character-set>' ist nicht angegeben). %C and %X legen verschiedene Ausgabelayouts fest.

```
%DISPLAY <char-variable> ['<coded-character-set>']
```

Wenn char-Variablen ausgegeben werden sollen, gibt AID diese in Übereinstimmung mit dem explizit definierten, Zeichensatz <coded-character-set> aus, oder in Übereinstimmung mit dem aktuellen, Zeichensatz CCS aus. Das Ausgabelayout unterscheidet sich jedoch von den Layouts, die durch %C bzw. %X festgelegt sind.

Den aktuellen Zeichensatz `CCS` zeigen Sie mit folgendem AID-Kommando an:
`%SHOW %AID`

Den aktuellen Zeichensatz `CCS` können Sie mit folgendem AID-Kommando ändern:
`%AID CCS = {<coded-character-set>|*USRDEF}`

daten

beschreibt, welche Informationen AID ausgeben soll. Sie können sich Dateierklärungen, Inhalt, Adresse und Länge von Datenfeldern und Sonderregistern, figurative Konstanten und die Adressen 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.

Datenfelder bereitet AID entsprechend der Definition im Quellprogramm auf, wenn Sie nicht mit Typmodifikation einen anderen Ausgabetyp festlegen (siehe AID-Basishandbuch [1]). Passt der Inhalt nicht zum definierten Speichertyp, wird die Ausgabe mit einer Fehlermeldung abgelehnt. Sie können sich den Inhalt des Datenfeldes 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, soweit keine expliziten Einschränkungen vorhanden (siehe Abschnitt „Symbolische Speicherreferenzen“ auf Seite 21).
Geben Sie für *daten* einen Namen an, der nicht in den LSD-Sätzen verzeichnet ist, gibt AID eine Fehlermeldung aus. Die anderen *daten* desselben Kommandos werden ordnungsgemäß bearbeitet.

- 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 Qualifikationen geben Sie nur an, wenn ein Adressoperand nicht für den aktuellen AID-Arbeitsbereich gilt oder wenn eine Adresse angesprochen werden soll, die nicht in der aktuellen Übersetzungseinheit oder dem aktuellen Programm liegt.

E={VM | Dn}
: geben Sie nur an, wenn für einen Datei-, Daten-, Anweisungsnamen, eine Source-Referenz oder ein Schlüsselwort die aktuelle Basisqualifikation nicht gelten soll (siehe %BASE).

S=srcname
: geben Sie nur an, wenn *daten* nicht in der aktuellen Übersetzungseinheit enthalten ist.

PROC=program-id
: geben Sie nur an, wenn Sie einen Datei-, Daten- oder Anweisungsnamen ansprechen, der nicht im aktuellen Programm liegt (siehe Kapitel „COBOL-spezifische Adressierung“ auf Seite 17) oder der in der aktuellen Übersetzungseinheit nicht eindeutig ist. Sie brauchen sie auch für einen globalen Datennamen, der lokal verdeckt ist.

Stimmt *srcname* in der S-Qualifikation mit *program-id* überein, schreiben Sie nur die PROG-Qualifikation.

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.

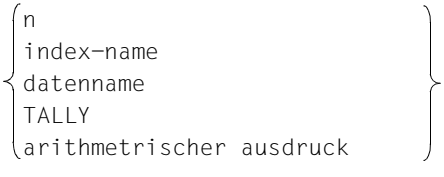
dateiname
ist der Name einer Datei aus einer Dateierklärung in der FILE SECTION der DATA DIVISION. AID gibt folgende Informationen aus:
den Datei-Status und, falls die Datei eröffnet ist, den Inhalt des Datensatz-Bereiches sowie eventuell den Satz-Schlüssel.

datenname
ist der im Quellprogramm definierte Name eines Datenfeldes oder der Name eines COBOL-Sonderregisters oder eine Figurative Konstante.
Ist *datenname* nicht eindeutig innerhalb eines Programms, kann er gekennzeichnet werden.
Ist *datenname* der Name eines Tabellenelementes, kann er wie in einer COBOL-Anweisung indiziert bzw. subskribiert werden (siehe Abschnitt „Symbolische Speicherreferenzen“ auf Seite 21 zu *datenname*).

datenname [kennzeichnung][...] [(index[,...])]

kennzeichnung
mit IN oder OF wird *datenname* einer bestimmten Datengruppe zugeordnet.
datenname muss mit so vielen Kennzeichnungen versehen sein, wie er für seine eindeutige Bezeichnung braucht.
Wird er nicht gekennzeichnet, gibt AID nur dann Daten für *datenname* aus, wenn es dafür eine Daten-Definition auf der Stufe 01 oder 77 gibt; sonst folgt eine Fehlermeldung.

index
wird wie in einer COBOL-Anweisung geschrieben, außer dass im AID-Kommando mehrere Indizes durch Komma getrennt werden müssen.
index kann folgendermaßen angegeben werden:



Es kann auch 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.

COBOL-Sonderregister[1]

```
LINAGE-COUNTER
RETURN-CODE
SORT-CCSN
SORT-CORE-SIZE
SORT-EOW
SORT-FILE-SIZE
SORT-MODE-SIZE
SORT-RETURN
TALLY
```

Figurative Konstanten

Auf die Figurativen Konstanten kann der Adressselektor nicht angewandt werden.

```
ZERO
SPACE
HIGH-VALUE
LOW-VALUE
QUOTE
symbolic character
```

anweisungsname

bezeichnet die Adresse des ersten Befehls in einem Kapitel oder Paragraphen in der PROCEDURE DIVISION.

```
{L'kapitel'                        }
{L'paragraph' [IN L'kapitel']      }
```

Ist ein Paragraphenname innerhalb eines Programms nicht eindeutig, muss er mit dem Kapitelnamen des Kapitels, in dem er definiert wurde, gekennzeichnet werden: `L'paragraph' IN L'kapitel'`

Mit nachfolgendem Pointer-Operator (->) gibt AID 4 Bytes des Programmcodes aus, der zu der ersten Anweisung in dem Kapitel oder Paragraphen generiert wurde.

[1] Die meisten COBOL-Sonderregister sind nur vorhanden, wenn die entsprechenden Sprachmittel verwendet werden.

source-referenz
: bezeichnet die Adresse des ersten Befehls, der zu einer Anweisung in der PROCEDURE DIVISION generiert wurde, und muss in einem der folgenden Formate angegeben werden:

  S'n'
  : für Zeilen mit Paragraphen- oder Kapitelnamen, in denen kein COBOL-Verb auftritt. Diese Angabe ist für Programme nicht möglich, die mit `STMT-REFERENCE=COLUMN1-TO-6` übersetzt wurden.

  S'nverb[m]' | S'xverb[m]'
  : für Zeilen, in denen ein COBOL-Verb steht.

  Mit nachfolgendem Pointer-Operator (->) gibt AID 4 Bytes des Programmcodes aus, der zu der Anweisung generiert wurde.

schlüsselwort
: Sie können alle Schlüsselwörter für Programmregister, AID-Register, Systemtabellen und das für den Durchlaufzähler oder die symbolische Lokalisierungsinformation verwenden (siehe AID-Basishandbuch [1]). Vor *schlüsselwort* können Sie nur eine Basisqualifikation angeben.

%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
%MR	alle 16 Mehrzweckregister in Tabellenform
%FR	alle 4 Gleitpunktregister mit doppelter Genauigkeit in Tabellenform aufbereitet
%PC	Befehlszähler (Program Counter)
%CC	Condition Code
%PM	Program Mask
%AMODE	Adressierungsmodus des Testobjekts
%PCB	Process Control Block
%PCBLST	Liste aller Process Control Blocks
%SORTEDMAP	Liste aller CSECTs und COMMONs des Benutzerprogramms(namens- und adresssortiert)lange Namen werden abgeschnitten
%MAP [CTX=kontext]	Liste aller CSECTs und COMMONs aller Kontexte des Benutzerprogramms oder des mit der Kontext-Qualifikation bezeichneten Kontextes; die Namen werden ungekürzt ausgegeben (weitere Operanden siehe AID-Basishandbuch [1])
%LINK	Name des zuletzt nachgeladenen Segments
%HLLOC(speicherref)	Lokalisierungsinformation auf symbolischer Ebene für eine Speicherreferenz in der PROCEDURE DIVISION (High-Level-Location)
%LOC(speicherref)	Lokalisierungsinformation auf Maschinencode-Ebene für eine Speicherreferenz in der PROCEDURE DIVISION (Low-Level-Location)
%•[subkdoname]	Durchlaufzähler
%•	Durchlaufzähler des gerade aktiven Subkommandos

kompl-speicherref

Folgende Operationen können darin vorkommen (siehe AID-Basishandbuch [1]):

- Adressversatz (•)
- indirekte Adressierung (->)
- Typmodifikation (%T(datenname), %X, %C, %E, %P, %D, %F, %A, %S, %SX, %UTF16)
- Längenmodifikation (%L(...), %L=(ausdruck), %Ln)
- Adressselektion (%@(...))
- Zeichenkonvertierungsfunktionen %C() und %UTF16()

Soll ein Anweisungsname oder eine Source-Referenz als Speicherreferenz verwendet werden, muss anschließend der Pointer-Operator (->)geschrieben werden. Ohne den Pointer-Operator können Anweisungsname und Source-Referenz überall da verwendet werden, wo auch Sedezimalzahlen geschrieben werden können. Mit der Typmodifikation können Sie sich *daten* in einer anderen Aufbereitung ausgeben lassen, da sich mit dem Speichertyp auch der Ausgabetyp ändert.

Mit der Längenmodifikation können Sie die Ausgabelänge selbst bestimmen, z.B. um nur Teile eines Datenfeldes oder ein Datenfeld in der Länge eines anderen Datenfeldes ausgeben zulassen. Mit Typ- oder Längenmodifikation dürfen die impliziten Bereichsgrenzen einer Adresse nur überschritten werden, wenn Sie mit *%@(datenname)->* auf die Maschinencode-Ebene gewechselt haben, auf der der Bereich den vom geladenen Programm belegten virtuellen Speicher umfasst.

%@(...)

Mit dem Adressselektor können Sie sich die Adresse eines Dateieintrags, eines Datenfeldes, eines Sonderregisters oder einer komplexen Speicherreferenz ausgeben lassen (siehe AID-Basishandbuch [1]).
Der Adressselektor lässt sich nicht auf Konstanten anwenden. Die dazu gehörenden Anweisungsnamen, die Source-Referenzen und die Figurativen Konstanten können Sie aber mit einem nachfolgenden Pointer ebenfalls angeben.

Beispiele

```
%D %@(L'VORLAUF'->)
%D %@(S'97MOV'->)
```

%L(...)

Mit dem Längenselektor können Sie sich die Länge eines Dateieintrags, eines Datenfeldes oder eines Sonderregisters ausgeben lassen (siehe AID-Basishandbuch [1]).

Beispiel: `%DISPLAY %L(FELD1)`
Die Länge von FELD1 wird ausgegeben.

%L=(ausdruck)

Mit der Längenfunktion können Sie sich einen Wert errechnen lassen.
ausdruck wird gebildet aus Speicherreferenzen und arithmetischen Operatoren (siehe AID-Basishandbuch [1]).
Beispiel: `%DISPLAY %L=(FELD1)`
Wenn FELD1 ganzzahlig ist (Typ %F), wird der Inhalt von FELD1 ausgegeben. Andernfalls gibt AID eine Fehlermeldung aus.

%UTF16(...) oder %C(...)

Die Funktion %UTF16() wandelt Strings von einer 1-Byte-EBCDIC-Codierung in die UTF16-Codierung um, die Funktion %C realisiert die Konvertierung umgekehrt. Weitere Informationen siehe AID-Basishandbuch [1].

AID-Literal

Alle im AID-Basishandbuch [1] beschriebenen AID-Literale können Sie angeben:.

`{C´x...x´|´x...x´| U´x...x´}`	`Character-Literal`
`{X´f...f´}`	`Sedezimal-Literal`
`{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-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - -

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

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

medium-u-menge ist ausführlich im AID-Basishandbuch [1] beschrieben.

T Terminal-Ausgabe
H Hardcopy-Ausgabe (schließt die Terminal-Ausgabe mit ein und kann nicht gemeinsam mit *T* angegeben werden)
Fn Datei-Ausgabe
P Ausgabe nach SYSLST

MAX Ausgabe mit Zusatzinformationen.

MIN Ausgabe ohne Zusatzinformationen.

XMAX Im Kommando %DISPLAY wird der Operandenwert XMAX derzeit nicht berücksichtigt, so dass das Verhalten identisch zum Standardwert MAX ist.

XFLAT Im Kommando %DISPLAY wird der Operandenwert XFLAT derzeit nicht berücksichtigt, so dass das Verhalten identisch zum Standardwert MAX ist.

Beispiele

1. Angabe mehrerer *medium-u-menge*-Operanden
   ```
   %DISPLAY DATENSATZ F1=MAX, H=MIN
   ```
2. ```
   %DISPLAY E=D1.PROG=BEISP.FCOMP3S,'DUMP-INHALT'
   ```
   Es handelt sich hier um die Auswertung eines DUMPs.

   ```
   ** D1: DUMP.BEISPIEL ***********************************************************
   FCOMP3S        =               +999456989
   DUMP-INHALT
   ```

3. ```
   %DISPLAY %L=(S'13ADD'-S'12MOV')
   ```
   AID gibt die Länge der Maschinencode-Sequenz aus, die für die Anweisung mit dem Namen 12MOV generiert wurde.

   ```
           +52
   ```

4. `%BASE`
   `%DISPLAY L'VERARBEITUNG'`
   Mit %BASE wird zurückgeschaltet auf den AID-Standard-Arbeitsbereich. Danach gibt AID die Adresse des ersten Befehls im Paragraphen VERARBEITUNG als Sedezimalzahl aus.

```
*** TID: 000801CA *** TSN: 6567 ********************************************
SRC_REF:     45INIT  SOURCE: MOBS      PROC: MOBS  ***************************
VERARBEITUNG    = 00000A84
```

5. `%DISPLAY L'VERARBEITUNG'->`
   AID gibt 4 Bytes des Maschinencodes aus, der ab der Adresse des Paragraphen VERARBEITUNG steht. Der Pointer-Operator bewirkt den Übergang zur Maschinencode-Ebene, sodass AID zusätzlich eine entsprechende Kopfzeile ausgibt.

```
CURRENT PC: 00000A04    CSECT: MOBS    ************************************
V'00000A84' = MOBS     + #00000A84''
00000A84 (00000A84) 18001800                                ....
```

6. `%DISPLAY %HLLOC(L'AUS1' IN L'AUSGABE'->)`
   AID gibt die symbolische Lokalisierungsinformation zum Paragraphen AUS1 im Kapitel AUSGABE aus.

```
V'00000C2C' = CONTEXT  :  LOCAL#DEFAULT
              SMOD     :  BEISPIEL
              PROC     :  BEISPIEL
              SECTION  :  AUSGABE
              PARAGRAPH:  AUS1
              SRC-REF  :      77
              LABEL    :  AUS1
```

7. `%DISPLAY %LOC(L'AUSGABE'->)`
   AID gibt die Lokalisierungsinformation auf Maschinencode-Ebene zum Paragraphen AUS1 im Kapitel AUSGABE aus.

```
V'00000C2C' = CONTEXT:LOCAL#DEFAULT
              LMOD :  %UNIT
              SMOD :  MOBS
              OMOD :  MOBS
              CSECT : MOBS  (00000000) + 00000CB8
```

8. Das Programm M1BS wird geladen und gestartet:

```
/LOAD-EXECUTABLE-PROGRAM M1BS,TEST-OPT=*AID
%  BLS0500 PROGRAM 'M1BS', VERSION ' ' OF '91-09-04' LOADED.
Entpackte Zahlen
12345
1234N
Gepackte Zahlen
12345
1234N
%  IDAON51 PROGRAM INTERRUPT AT LOCATION '008702 (M1BS), (CDUMP), EC=68
%  IDAON45 DUMP DESIRED? REPLY (Y = USER/AREA DUMP; Y,SYSTEM = SYSTEM ,N=NO)?
%  EXC0077 PROGRAM IS STILL LOADED AND IN 'HOLD-PROGRAM' MODE. PROGRAM RUN MAY
BE CONTINUED WITH /RESUME-PROGRAM
```

Es ist auf einen Fehler gelaufen. Nun interessiert es Sie, welche Anweisung den Fehler verursacht hat. Dazu geben Sie ein %DISPLAY %HLLOC zu der Adresse ein, an der das Programm durch den Fehler unterbrochen wurde:

```
/%DISPLAY %HLLOC(V'8702')
*** TID: 0000004D *** TSN: 4192 ***********************************************
CURRENT PC: 00008702    CSECT: UPRO      ************************************
V'00008702' = SMOD     :  UPRO
              PROC     :  UPRO
              SRC-REF  :      33COM
```

9. `%DISPLAY ALPHA-ZEICHEN(I)`
   ALPHA-ZEICHEN sei definiert wie in Beispiel 9 beschrieben, und Index I enthalte den Wert 5, es wird also das 5. Element der Tabelle ausgegeben:

```
ALPHA-ZEICHEN( 5) = |E|
```

10. `%DISPLAY ALPHA-ZEICHEN`
    Das Element ALPHA-ZEICHEN ist in einer Tabelle 26mal enthalten und in der DATA DIVISION folgendermaßen definiert:

```
01    A-Z-TAB1.
   02 ALPHA-ZEICHEN    PIC X OCCURS 26 INDEXED BY I.
```

Da im %DISPLAY kein Index angegeben wurde, gibt AID alle Elemente mit diesem Namen aus:

```
*** TID: 00010053 *** TSN: 6567 ***********************************************
SRC_REF:     45INIT  SOURCE: BEISPIEL  PROC: BEISPIEL  ***********************
ALPHA-ZEICHEN( 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|
```

11. Gegenüberstellung der AID- und COBOL-Ausgabe von Datenfeldern:

```
IDENTIFICATION DIVISION.
PROGRAM-ID.  PROG-NUM.
ENVIRONMENT DIVISION.
CONFIGURATION SECTION.
SPECIAL-NAMES.
    TERMINAL IS T.
DATA DIVISION.
WORKING-STORAGE SECTION.
01  ENTPA1 PIC 99999.
01  ENTPA2 PIC S999V99 VALUE ZERO.
01  GEPA1 PIC 99999 COMP-3.
01  GEPA2 PIC S999V99 COMP-3 VALUE ZERO.
01  GLEIT1 PIC +999.99E-99.
01  GLEIT2 COMP-1.
01  GLEIT3 COMP-1 VALUE 12.
01  GLEIT4 COMP-2.
01  GLEIT5 COMP-2 VALUE +123456789.1234567E+10.
01  BIN1 PIC 99999 BINARY.
01  BIN2 PIC S9999 BINARY VALUE ZERO.
PROCEDURE DIVISION.
ENTPA.
    DISPLAY "Entpackte Zahlen" UPON T.
    MOVE 12345 TO ENTPA1.
    DISPLAY ENTPA1 UPON T.
    MOVE -123.45 TO ENTPA2.
    DISPLAY ENTPA2 UPON T.
GEPA.
    DISPLAY "Gepackte Zahlen" UPON T.
    MOVE 12345 TO GEPA1.
    DISPLAY GEPA1 UPON T.
    MOVE ENTPA2 TO GEPA2.
    DISPLAY GEPA2 UPON T.
GLEIT.
    DISPLAY "Gleitpunkt-Zahlen" UPON T.
    MOVE 12345 TO GLEIT1.
    MOVE 12345 TO GLEIT2.
    DISPLAY GLEIT1 UPON T.
    DISPLAY GLEIT2 UPON T.
    DISPLAY GLEIT3 UPON T.
    MOVE ENTPA2 TO GLEIT4.
    DISPLAY GLEIT4 UPON T.
    DISPLAY GLEIT5 UPON T.
BIN.
    DISPLAY "Binaere-Zahlen" UPON T.
    MOVE 12345 TO BIN1.
    DISPLAY BIN1 UPON T.
    MOVE ENTPA2 TO BIN2.
    DISPLAY BIN2 UPON T.
ENDE.
    STOP RUN.
```

COBOL-Ausgabe

```
Entpackte Zahlen
12345
1234N
Gepackte Zahlen
12345
1234N
Gleitpunkt-Zahlen
+123.45E 02
+.123450E+05
+.120000E+02
-.123450000000000E+03
+.123456789123457E+19
Binaere-Zahlen
12345
012L
```

AID-Ausgabe

```
%D ENTPA1, ENTPA2
SRC_REF:     44DIS   SOURCE: UPRONUM   PROC: UPRONUM  ************************
ENTPA1          =   12345
ENTPA2          =  -123.45

%D GEPA1,GEPA2
GEPA1           =                12345
GEPA2           =               -123.45
```

```
%D GLEIT1,GLEIT2,GLEIT3,GLEIT4,GLEIT5
GLEIT1          = +.12345 E+005
GLEIT2          = +.1234500 E+005
GLEIT3          = +.1200000 E+002
GLEIT4          = -.1234499999999999 E+003
GLEIT5          = +.1234567891234566 E+019

%D BIN1, BIN2
BIN1            =                12345
BIN2            =                 -123
```

%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. `D3=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 Datenfeld oder in der PROCEDURE DIVISION 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 *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.

– Mit *find-bereich* geben Sie an, in welchem Datenfeld oder in welchem Bereich des ausführbaren Programmteils *suchbegriff* gesucht werden soll. Ohne Angabe von *find-bereich* durchsucht AID den gesamten Speicherbereich zur aktuell eingestellten Basisqualifikation (siehe %BASE).

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

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.

In einem %FIND mit eigenem *suchbegriff* und ohne weitere Operanden ergänzt AID einen Operanden ohne aktuellen Wert mit dem entsprechenden Standardwert. Hier werden also keine Operanden aus einem vorhergehenden %FIND übernommen.

Im Trefferfall erfolgt eine Ausgabe in der Länge von maximal 12 Bytes, vom Treffer an bis zum Ende von *find-bereich* auf das Medium Terminal (SYSOUT) im Ausgabetyp DUMP (Sedezimal- und Character-Darstellung). Zusätzlich zum Treffer wird seine Adresse und (soweit möglich) der Name der Übersetzungseinheit ausgegeben, in der der Treffer gefunden wurde und die relative Adresse des Treffers zum Anfang der Übersetzungseinheit.

Die Trefferadresse wird 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 Fortsetzungsadresse des letzten Treffers in %1G abgespeichert. Falls *suchbegriff* nicht gefunden wurde, bleiben die AID-Register %0G und %1G 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-, ein Sedezimal-Literal oder eine Speicherstelle. Bei Angabe einer Speicherstelle müssen Klammern in der Form (*suchbegriff*) angegeben werden.

Bei Literal-Angabe von *suchbegriff* können Wildcard-Symbole verwendet werden. Diese Symbole sind immer Treffer. Sie werden durch '%' dargestellt.

```
suchbegriff=OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

⎧C'x...x' | 'x...x'           ⎫
⎨X'f...f' |                   ⎬
⎪%C(literal) | %UTF16(literal)⎪
⎩(speicherstelle)             ⎭

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

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

Sedezimal-Literamit 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.

`%C(literal) | %UTF16(literal)`

Diese Funktionen sind zu verwenden, wenn z.B. eine UTF16-Codierung des *suchbegriffs* gewünscht wird oder wenn Zeichenfolgen in 1-Byte-Codierung gesucht werden, obwohl das Literal in UTFE-Codierung eingegeben wurde.

`(speicherstelle)`

Der *suchbegriff* wird aus *speicherstelle* entnommen. Hat *speicherstelle* den Typ %UTF16, so können maximal 160 Bytes = 80 UTF16-Zeichen gesucht werden. In allen anderen Fällen ist *suchbegriff* auf 80 Bytes begrenzt.

speicherstelle kann auch ein symbolisches Feld sein. Ein NATIONAL Feld wird dann wie eine %UTF16 *speicherstelle* behandelt.

find-bereich

legt einen Speicherbereich fest, in dem *suchbegriff* gesucht werden soll. *find-bereich* kann ein Datenfeld oder ein Teil der PROCEDURE DIVISION 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 [1]) d.h. es wird der Klasse-6-Speicher zur aktuell eingestellten Basisqualifikation (siehe %BASE) durchsucht.

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

                      ⎧datenname             ⎫
                      ⎪anweisungsname ->     ⎪
IN [•][qua•] [•]      ⎨                      ⎬
                      ⎪source-referenz ->    ⎪
                      ⎩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 Qualifikationen geben Sie nur an, wenn ein Adressoperand nicht für den aktuellen AID-Arbeitsbereich gilt oder wenn eine Adresse angesprochen werden soll, die nicht in der aktuellen Übersetzungseinheit oder im aktuellen Programm liegt.

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

S=srcname
: geben Sie nur an, wenn *find-bereich* nicht in der aktuellen Übersetzungseinheit liegt (siehe Kapitel „COBOL-spezifische Adressierung“ auf Seite 17).

PROC=program-id
: geben Sie nur an, wenn *find-bereich* nicht im aktuellen Programm liegt (siehe Kapitel „COBOL-spezifische Adressierung“ auf Seite 17) oder mit *datenname* oder *anweisungsname* vereinbart wird, der in der Übersetzungseinheit nicht eindeutig ist.

Stimmt *srcname* in der S-Qualifikation mit *program-id* überein, schreiben Sie nur die PROG-Qualifikation.

Vor die anschließend aufgeführten C-Qualifikationen können Sie nur die Basisqualifikation bzw. die CTX-Qualifikation setzen. Mit der C-Qualifikation verlassen Sie die symbolische Ebene. Unmittelbar anschließend kann kein symbolischer Operand geschrieben werden (siehe Abschnitt „Symbolische Speicherreferenzen“ auf Seite 21), nur eine *kompl-speicherref*.

C=segmentname
: Ohne Längenmodifikation geben Sie das ganze Segment als *find-bereich* an.

C=sharename
: Ohne Längenmodifikation geben Sie den gesamten Bindemodul als *find-bereich* an.

datenname

ist der im Quellprogramm definierte Name eines Datenfeldes oder der Name eines COBOL-Sonderregisters.
Ist *datenname* nicht eindeutig innerhalb eines Programms, kann er gekennzeichnet werden.
Ist *datenname* der Name eines Tabellenelementes, kann er wie in einer COBOL-Anweisung indiziert bzw. subskribiert werden (siehe Abschnitt „Symbolische Speicherreferenzen“ auf Seite 21).

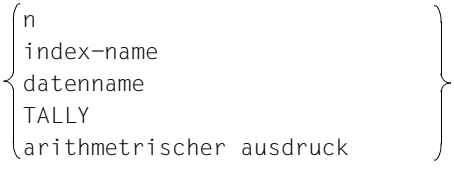
datenname [kennzeichnung][...] [(index[,...])]

kennzeichnung

mit IN oder OF kann *datenname* einer bestimmten Datengruppe zugeordnet wird. *datenname* muss mit so vielen Kennzeichnungen versehen sein, wie er für seine eindeutige Bezeichnung braucht. Wird er nicht gekennzeichnet, bearbeitet AID *datenname* nur dann, wenn es dafür eine Daten-Definition auf der Stufe 01 oder 77 gibt; sonst folgt eine Fehlermeldung.

index

wird wie in einer COBOL-Anweisung geschrieben außer, dass im AID-Kommando mehrere Indizes durch Komma getrennt werden müssen.
index kann folgendermaßen angegeben werden:



COBOL-Sonderregister

```
LINAGE-COUNTER
RETURN-CODE
SORT-CCSN
SORT-CORE-SIZE
SORT-EOW
SORT-FILE-SIZE
SORT-MODE-SIZE
SORT-RETURN
TALLY
```

```
{anweisungsname }
{               }->
{source-referenz}
```

bezeichnet 4 Bytes des Programmcodes ab der Adresse, die in der Adresskonstanten hinterlegt ist. Soll eine andere Anzahl von Bytes durchsucht werden, müssen Sie eine entsprechende Längenmodifikation angeben.

anweisungsname
: bezeichnet die Adresse des ersten Befehls in einem Kapitel oder Paragraphen in der PROCEDURE DIVISION.

  ```
  {L'kapitel'                          }
  {L'paragraph' [IN L'kapitel']        }
  ```

  Ist ein Paragraphenname innerhalb eines Programms nicht eindeutig, muss er mit dem Kapitelnamen des Kapitels, in dem er definiert wurde, gekennzeichnet werden: `L'paragraph' IN L'kapitel'`

source-referenz
: bezeichnet die Adresse des ersten Befehls, der zu einer Anweisung in der PROCEDURE DIVISION generiert wurde, und muss in einem der folgenden Formate angegeben werden:

  S'n'
  : für Zeilen mit Paragraphen- oder Kapitelnamen, in denen kein COBOL-Verb auftritt. Diese Angabe ist für Programme nicht möglich, die mit `STMT-REFERENCE=COLUMN1-TO-6` übersetzt wurden.

  S'nverb[m]' | S'xverb[m]'
  : für Zeilen, in denen ein COBOL-Verb steht.

kompl-speicherref
: Folgende Operationen können in *kompl-speicherref* vorkommen (siehe AID-Basishandbuch [1]):
  – Adressversatz (.)
  – indirekte Adressierung (->)
  – Typmodifikation (%A, %S, %SX)
  – Längenmodifikation (%L(...), %L=(ausdruck), %Ln)
  – Adressselektion (%@(...))

  Beginnt *kompl-speicherref* mit Anweisungsname oder Source-Referenz, muss anschließend der Pointer-Operator (->)geschrieben werden. In diesem Fall muss *anweisungsname* mit L'...' angegeben werden. Ohne den Pointer-Operator können Anweisungsname und Source-Referenz überall da verwendet werden, wo auch Sedezimalzahlen geschrieben werden können. *kompl-speicherref* bezeichnet einen Bereich von 4 Bytes ab der errechneten Adresse. Soll eine andere Anzahl von Bytes durchsucht werden, muss eine entsprechende Längenmodifikation angefügt

werden. Bei der Längenmodifikation von Datenfeldern müssen Sie die Bereichsgrenzen beachten oder mit %@(datenname)-> auf Maschinencode-Ebene wechseln.

alignment

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

```
alignment-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

          ⎧1⎫
ALIGN [=] ⎨2⎬
          ⎩4⎭
          ⎩8⎭

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

suchbegriff wird gesucht ab

1 Byte-Grenze (Standardwert)

2 Halbwort-Grenze

4 Wort-Grenze

8 Doppelwort-Grenze

Beispiele

1. `%FIND X'F0' IN DATA`
   Das Sedezimal-Literal X’F0’ wird im Datenfeld DATA gesucht. Ein Treffer wird auf SYSOUT ausgegeben.

2. `%F X'D2' IN S'12MOV'->%L=(S'13ADD'-S'12MOV') ALIGN=2`
   Im für die Anweisung 12MOV generierten Maschinencode wird an einer Halbwortgrenze das Sedezimal-Literal X’D2’ gesucht.

3. `%F`
   Die Suche wird mit den Parametern des letzten %FIND-Kommandos hinter dem letzten Treffer fortgesetzt.

4. Das Eingabemedium hat den CCSN UTFE:
   `%FIND  %UTF16('[{Ö')  IN V'xxx'`
   Das Kommando sucht die Zeichenfolge ’[{Ö’ in ihrer UTF16-Codierung ab der Speicherstelle V'xxx'.
   Ohne die Angabe der Funktion %UTF16() würde AID die UTFE-Codierung X’BB-FB9EB6’ von ’[{Ö’ im Speicher suchen.
   Durch die Verwendung der Funktion %UTF16() wird seine UTF16-Codierung X’005B007B00D6' im Speicher gesucht.

5. Das Eingabemedium hat den CCSN UTFE.
   ```
   %FIND  %C('Ä')  IN V'xxx'
   ```

   1. `%AID EBCDIC=EDF03DRV` (deutscher Zeichensatz)
      Das Kommando sucht die deutsche Codierung von Ä (entspricht X’BB’) ab Adresse V’xxx’. Ohne die Angabe %C() würde AID X’9E9F’ (= UFTE-Codierung von ’Ä’) im Speicher suchen.

   2. `%AID EBCDIC=EDF03IRV`
      Das Kommando sucht für das Zeichen ’Ä’, das im Zeichensatz EDF03IRV ungültig ist, das Ersatzzeichen ’.’. Dabei meldet AID, dass ein Ersatzzeichen bei der %C()-Konvertierung aufgetreten ist.

%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 Kommando 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 AID-Meldungen haben im Meldungsschlüssel den Aufbau AID0n, die AIDSYS-Meldungen den Aufbau IDA0n. Beide werden mit /HELP abgefragt. Daneben können Sie in der aktuellen AID-Version wie bisher mit dem AID-Kommando %HELP die AID-Meldungen mit In abfragen.

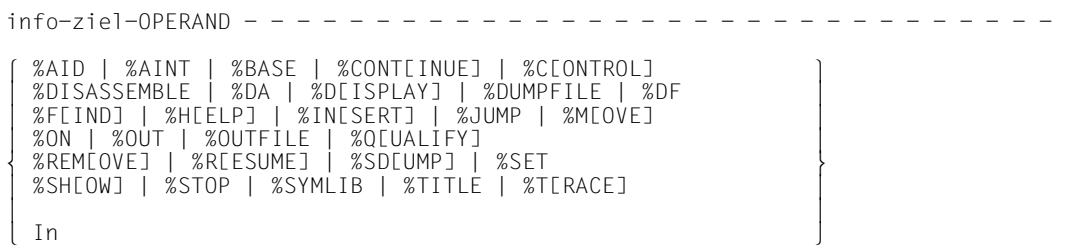
%HELP darf nur als Einzelkommando eingegeben werden, es darf nicht in einer Kommandofolge oder in einem Subkommando stehen.

%HELP verändert den Programmzustand nicht.

info-ziel

bezeichnet ein Kommando oder eine Meldungsnummer, über die Informationen ausgegeben werden sollen.
Ohne diesen Operanden wird eine Übersicht über die AID-Kommandos mit einer Kommando-Kurzbeschreibung und den AID-Meldungsnummernkreis ausgegeben.

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 %? angeben.



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

In bezeichnet den alten Meldungsschlüssel einer Meldung 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.

Ohne diesen Operanden und ohne eine Vereinbarung mit dem %OUT-Kommando arbeitet AID mit dem Standardwert T=MAX. Die Angabe {MIN | MAX | XMAX | XFLAT} hat bei %HELP keine Auswirkungen, eine der Angaben ist aber syntaktisch erforderlich.

medium-u-menge ist ausführlich im AID-Basishandbuch [1] beschrieben.

T Terminal-Ausgabe
H Hardcopy-Ausgabe (schließt die Terminal-Ausgabe mit ein und kann nicht gemeinsam mit *T* angegeben werden)
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 dessen Ausführung AID den Programmablauf unterbrechen soll, um *subkdo* zu bearbeiten.

– Mit *subkdo* definieren 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 Basisqualifikation 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 COBOL-Anweisung generiert 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 - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                  {C=segmentname     }
                  {C=sharename       }
                  {program-id        }
 [•][qua•][...] {                    }
                  {anweisungsname    }
                  {source-referenz   }
                  {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 Qualifikationen geben Sie an, wenn ein Adressoperand nicht für den aktuellen AID-Arbeitsbereich, die aktuelle Übersetzungseinheit oder das aktuelle Programm gilt oder wenn er sonst nicht eindeutig ist.

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

S=srcname
: geben Sie nur an, wenn *testpunkt* nicht in der aktuellen Übersetzungseinheit liegen soll.

PROC=program-id
: geben Sie nur an, wenn ein Anweisungsname nicht im aktuellen Programm liegt oder wenn er in der aktuellen Übersetzungseinheit nicht eindeutig ist (siehe Kapitel „COBOL-spezifische Adressierung“ auf Seite 17).

Stimmt *srcname* in der S-Qualifikation und *program-id* überein, schreiben Sie nur die PROG-Qualifikation.

Vor die anschließend aufgeführten C-Qualifikationen können Sie nur die Basisqualifikation bzw. die CTX-Qualifikation setzen. Mit der C-Qualifikation verlassen Sie die symbolische Ebene. Unmittelbar anschließend kann kein symbolischer Operand geschrieben werden (siehe Abschnitt „Symbolische Speicherreferenzen“ auf Seite 21), nur eine *kompl-speicher-ref*.

C=segmentname
: Mit dieser Angabe setzen Sie *testpunkt* auf die Anfangsadresse des bezeichneten Segments.

C=sharename
: Mit dieser Angabe setzen Sie *testpunkt* auf die Anfangsadresse des bezeichneten, mehrfach benutzbaren Bindemoduls.

program-id
: Diese Angabe ist nach einer expliziten PROC/PROG-Qualifikation möglich oder wenn die aktuelle Unterbrechungsstelle in dem Programm liegt, dass mit *program-id* bezeichnet wird. Damit setzen Sie *testpunkt* auf die erste ausführbare Anweisung des bezeichneten Programms.

anweisungsname
: bezeichnet die Adresse des ersten Befehls in einem Kapitel oder Paragraphen in der PROCEDURE DIVISION.

  ```
  {L'kapitel'                        }
  {L'paragraph' [IN L'kapitel']      }
  ```

  Ein alphanumerischer Kapitel- oder Paragraphenname kann auch ohne L’...’ angegeben werden, da in diesem Kommando eine Verwechslung mit einem Datennamen nicht möglich ist.

  Ist ein Paragraphenname innerhalb eines Programms nicht eindeutig, dann muss er mit dem Kapitelnamen des Kapitels, in dem er definiert wurde, gekennzeichnet werden: `L'paragraph' IN L'kapitel'`

source-referenz
: bezeichnet die Adresse des ersten Befehls, der zu einer Anweisung in der PROCEDURE DIVISION generiert wurde, und muss in einem der folgenden Formate angegeben werden (siehe Kapitel „COBOL-spezifische Adressierung“ auf Seite 17):

  S’n’
  : für Zeilen mit Paragraphen- oder Kapitelnamen, in denen kein COBOL-Verb auftritt. Diese Angabe ist für Programme nicht möglich, die mit `STMT-REFERENCE=COLUMN1-TO-6` übersetzt wurden.

  S’nverb[m]’ | S’xverb[m]
  : für Zeilen, in denen ein COBOL-Verb steht.

kompl-speicherref

Das Ergebnis von *kompl-speicherref* muss die Anfangsadresse eines ausführbaren Maschinenbefehls sein. *kompl-speicherref* kann folgende Operationen enthalten (siehe AID-Basishandbuch [1]):

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

Beginnt eine *kompl-speicherref* mit Anweisungsname oder Source-Referenz, muss anschließend der Pointer-Operator (->)geschrieben werden. In diesem Fall muss *anweisungsname* mit L'...' angegeben werden. Ohne den Pointer-Operator können Anweisungsname und Source-Referenz überall da verwendet werden, wo auch Sedezimalzahlen geschrieben werden können.

Beispiel: `%INSERT L'AUSGABE' ->.4`

testpunkt wird auf den zweiten Befehl nach dem Paragraphen AUSGABE gesetzt. Der erste Befehl war 4 Bytes lang.

Eine Typmodifikation ist nur sinnvoll, wenn der Inhalt eines Datenfeldes als Adresse eingesetzt werden kann oder wenn Sie die Adresse aus einem Register entnehmen.

Beispiel: `%INSERT %1G.2 %AL2 ->`

Die letzten beiden Bytes aus AID-Register %1G werden als Adresse benutzt.

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 AID-Basishandbuch [1].

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

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

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

Ein 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. Im *subkdo* eines %ON oder %INSERT sind die Kommandos %JUMP, %CONTROL*n*, %INSERT und %ON zugelassen. Mit den letzten drei Kommandos können Sie eine Schachtelung bis zu 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
$n = 2^{31}-1$ und K ein.

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

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

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

n ist eine Zahl mit dem Wert $1 \le n \le 2^{31}-1$.
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'48MOV'`
   testpunkt wird mit einer Source-Referenz angegeben und wird auf die Speicherstelle gesetzt, an der der Befehlscode steht, der zum MOVE in der Anweisungszeile 48 generiert wurde.

2. `%IN ST3 <%DISPLAY PERSNR> ONLY 10 S`
   testpunkt wird mit dem Paragraphen-Namen ST3 bezeichnet. D.h. immer wenn der Programmablauf zur ersten Anweisung im Paragraphen ST3 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.

3. ```
   %IN ST2 <%DISPLAY TEXTDAT, 'ST2'>
   %IN ST3 <%DISPLAY 'INSERT1', TEXTDAT; %IN AUSGABE<%D 'INSERT2', -
   I,J,K, ANZAHL-TABELLE; %IN S'172' <%D 'INSERT3' ,I,J; %REMOVE AUSGABE>>>
   ```

   Mit dem ersten %INSERT wird als *testpunkt* der Paragraph ST2 festgelegt. Kommt nach der Beendigung der Kommandoeingabe der Programmablauf zu ST2, wird das Subkommando ausgeführt. Es besteht aus einem %DISPLAY für den Datennamen TEXTDAT und dem Literal 'ST2'. Anschließend läuft das Programm weiter.

   Mit dem zweiten %INSERT wird der *testpunkt* ST3 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 zum Paragraphen ST3, wird das zugehörige *subkdo* ausgeführt, d.h. das %DISPLAY-Kommando für das Literal 'INSERT1' und das Datenfeld TEXTDAT wird ausgeführt und der *testpunkt* AUSGABE gesetzt.
   Das *subkdo* zum *testpunkt* AUSGABE ist noch nicht wirksam. Im zu testenden Programm sind also bis zu dieser Stelle des Programmablaufs drei *testpunkte* gesetzt: ST2, ST3 und AUSGABE.

   Da auch das *subkdo* zum *testpunkt* ST3 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 AUSGABE, wird nun der `%D 'INSERT2', I, J, K, ANZAHL-TABELLE` ausgeführt. Außerdem enthält *subkdo* wieder ein %INSERT-Kommando, dessen *testpunkt* diesmal mit der *source-referenz* S'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 Datenfelder I und J aus. Mit dem zweiten Kommando in diesem *subkdo*, dem %REMOVE AUSGABE, wird der *testpunkt* AUSGABE gelöscht. Das ist z.B. dann erforderlich, wenn ein *testpunkt* in einer Schleife liegt und es dadurch zur unerwünschten Kettung geschachtelter *subkdos* kommen würde. Ohne das %REMOVE-Kommando würde beim zweiten Durchlaufen von AUSGABE zum *testpunkt* S'172' folgendes *subkdo* entstehen:

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

4.
```
%OUT %DISPLAY P=MAX
%IN S'73SET' <%D 'I GE 10',I,ZEICHEN(I),K,ANZ-Z(I,K)>
%IN S'73SET' <(I LT 10): %D 'I LT 10',I,ZEICHEN(I); %CONT>
```

   Zunächst wird vereinbart, dass alle Ausgaben des Kommandos %DISPLAY nach SYSLST erfolgen.
   Durch die danach eingegebenen beiden %INSERTs entsteht am *testpunkt* S'73SET' das folgende Subkommando:

```
<(I LT 10): %D 'I LT 10',I,ZEICHEN(I); %CONT; %D 'I GE 10',I,ZEICHEN(I),
-K,ANZ-Z(I,K)>
```

   Jedes Mal, wenn der Programmablauf an die Anweisung mit dem Namen 73SET 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 ZEICHEN(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 ZEICHEN(I) auch die Werte von Index K und dem Tabellenelement ANZ-Z(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 COBOL-Anweisung sein.

Kommando	Operand
%JUMP	fortsetzung

%JUMP kann nur für Programme verwendet werden, die mit dem COBOL-Compiler übersetzt wurden. Bei der Übersetzung müssen Sie die SDF-Option `PREPARE-FOR-JUMPS=YES` bzw. die COMOPT-Anweisung `SEPARATE-TESTPOINTS=YES` angeben.

Die Fortsetzungsadresse muss im selben Programm liegen wie die aktuelle Unterbrechungsstelle, sonst führt das Kommando zum Fehler, weil erforderliche Initialisierungen nicht durchgeführt wurden.
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-EXECUTABLE-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.

%JUMP verändert den Programmzustand nicht.

fortsetzung

definiert die Stelle, an der das Programm fortgesetzt werden soll. *fortsetzung* muss die Adresse einer ausführbaren Anweisung im aktuellen Programm sein. Steht der %JUMP in einem Subkommando, so muss *fortsetzung* eine Anweisung in dem Programm bezeichnen, in dem die aktuelle Unterbrechungsstelle zu *testpunkt* oder *ereignis* eingetreten ist.

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

{anweisungsname                 }
{source-referenz                }

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

anweisungsname
: bezeichnet die Adresse des ersten Befehls in einem Kapitel oder Paragraphen in der PROCEDURE DIVISION.

  ```
  {L'kapitel'                     }
  {L'paragraph' [IN L'kapitel']   }
  ```

  Ein alphanumerischer Kapitel- oder Paragraphenname kann auch ohne L'...' angegeben werden, da in diesem Kommando eine Verwechslung mit einem Datennamen nicht möglich ist.

  Ist ein Paragraphenname innerhalb eines Programms nicht eindeutig, dann muss er mit dem Kapitelnamen des Kapitels, in dem er definiert wurde, gekennzeichnet werden: `L'paragraph' IN L'kapitel'`

source-referenz
: *fortsetzung* kann nur die Adresse des ersten Befehls in einem Kapitel oder Paragraphen in der PROCEDURE DIVISION sein und kann deshalb nur mit folgender Source-Referenz angegeben werden:

  S'n'
  : für Zeilen mit Paragraphen- oder Kapitelnamen, wenn kein COBOL-Verb auftritt. Das bedeutet, dass für ein Programm, das übersetzt wurde mit `STMT-REFERENCE=COLUMN1-TO-6` kein %JUMP *source-referenz* möglich ist.

Beispiel

```
%JUMP S'67'
%JUMP AUSGABE
```

Beide Kommandos beziehen sich auf das Programmbeispiel in Abschnitt „Quellprogrammliste“ auf Seite 171.
In der Anweisungszeile 67 steht nur der Paragraphenname AUSGABE. Mit beiden Kommandos wird also dieselbe Fortsetzungsadresse vereinbart, nämlich die erste ausführbare Anweisung im Paragraphen AUSGABE.

%MOVE

Mit %MOVE übertragen Sie Speicherinhalte oder AID-Literale auf Speicherstellen im geladenen Programm. Die Übertragung erfolgt linksbündig ohne Überprüfung und ohne Anpassung des Speichertyps von Sender an Empfänger. Für eine typgerechte Übertragung wie in der COBOL-MOVE-Anweisung brauchen Sie das %SET-Kommando.

– Mit *sender* bezeichnen Sie ein Datenfeld, eine Länge, eine Adresse, einen Durchlaufzähler, ein AID-Register, ein COBOL-Sonderregister, eine Figurative Konstante oder ein AID-Literal.
sender kann im virtuellen Speicher des geladenen Programms oder in einer Dump-Datei liegen.

– Mit *empfänger* bezeichnen Sie ein Datenfeld, einen Durchlaufzähler, ein AID-Register oder ein COBOL-Sonderregister, 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 die globale Einstellung (siehe %AID-Kommando) für das aktuelle %MOVE-Kommando außer Kraft.

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

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

Der *sender* bestimmt die Länge der Übertragung. Eine Längenmodifikation beim *empfänger* ist wirkungslos. Wenn die Übertragung über das Ende des *empfängers* hinaus führt, weist AID die Übertragung mit einer Fehlermeldung ab.

Es empfiehlt sich, das Kommando nicht unmittelbar nach dem Laden einzugeben, da noch nicht alle Einträge in der DATA DIVISION initialisiert sind (z.B. Datensatzbeschreibungen und Sonderregister).

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 ein Datenfeld, ein COBOL-Sonderregister, einen Durchlaufzähler, ein Register oder eine komplexe Speicherreferenz angeben. Anweisungsnamen, Sourcereferenzen, Figurative Konstanten, Adressen und Längen von Datenfeldern 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; dagegen kann *empfänger* kann nur im virtuellen Speicherbereich des geladenen Programms liegen. Übertragen bzw. überschreiben Sie Programmbereiche mit Befehlscode, kann das zu unerwünschten Ergebnissen führen, wenn Adressen betroffen sind, die zu einem *control*- oder *trace-bereich* gehören oder auf die mit %INSERT ein Testpunkt gesetzt wurde (siehe AID-Basishandbuch [1]).

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.

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

                 ┌C=segmentname          ┐ ┐
                 │C=sharename            │ │
                 │datenname              │ │
[•][qua•]        │anweisungsname         │ │
                 │source-referenz        │ │
                 │schlüsselwort          │ │                    ┌C=segmentname     ┐
                 └kompl-speicherref      ┘ │                    │datenname         │
┌                                          ┘  INTO  [•][qua•]  │                  │
│┌%@┐                ┌dateiname          ┐                      │schlüsselwort     │
└└  ┘┘([•][qua•])    │datenname          │)                     └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 Qualifikationen geben Sie an, wenn ein Adressoperand nicht für den aktuellen AID-Arbeitsbereich gilt oder wenn eine Adresse angesprochen werden soll, die nicht in der aktuellen Übersetzungseinheit oder im aktuellen Programm liegt.

E={VM | Dn} für *sender*
E=VM für *empfänger*
geben Sie nur an, wenn für einen Datei-, Daten-, Anweisungsnamen, eine Source-Referenz oder ein Schlüsselwort die aktuelle Basisqualifikation 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.

S=srcname
geben Sie an, wenn *sender* oder *empfänger* nicht in der aktuellen Übersetzungs einheit liegt.

PROC=program-id
geben Sie an, wenn Sie einen Datei-, Daten- oder Anweisungsnamen ansprechen, der nicht im aktuellen Programm liegt oder der in der aktuellen Übersetzungseinheit nicht eindeutig ist (siehe Kapitel „COBOL-spezifische Adressierung“ auf Seite 17). Sie brauchen sie auch für einen globalen Datennamen, der lokal verdeckt ist.

Stimmt *srcname* in der S-Qualifikation mit *program-id* überein, schreiben Sie nur die PROG-Qualifikation.

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.

Vor die anschließend aufgeführten C-Qualifikationen können Sie nur die Basisqualifikation bzw. die CTX-Qualifikation setzen. Mit der C-Qualifikation verlassen Sie die symbolische Ebene. Unmittelbar anschließend kann kein symbolischer Operand geschrieben werden (siehe Abschnitt „Symbolische Speicherreferenzen“ auf Seite 21), nur eine *kompl-speicherref*.

C=segmentname
Ohne Längenmodifikation geben Sie das ganze Segment als *sender* oder *empfänger* an. Ist das Segment länger als 3900 Byte, können Sie es nur mit mehreren %MOVEs übertragen.

C=sharename
Ohne Längenmodifikation geben Sie den gesamten Bindemodul als *sender* oder *empfänger* an. Ist er länger als 3900 Byte, kann er nur mit mehreren %MOVEs übertragen werden.

datenname

ist der im Quellprogramm definierte Name eines Datenfeldes. Das sind sowohl einzelne Datenelemente, als auch Datengruppen und Tabellen als auch deren Elemente oder der Name eines COBOL-Sonderregisters. Figurative Konstanten können nur als *sender* eingesetzt werden.
Ist *datenname* nicht eindeutig innerhalb eines Programms, kann er gekennzeichnet werden.

Ist *datenname* der Name eines Tabellenelementes, kann er wie in einer COBOL-Anweisung indiziert bzw. subskribiert werden.

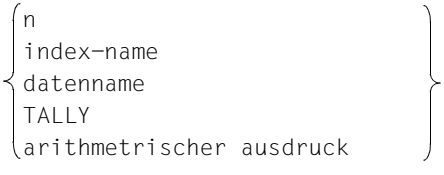
datenname [kennzeichnung][...] [(index[,...])]

kennzeichnung

ist *datenname* nicht eindeutig innerhalb eines Programms, kann er gekennzeichnet werden, indem er mit IN oder OF einer bestimmten Datengruppe zugeordnet wird. *datenname* muss mit so vielen Kennzeichnungen versehen sein, wie er für seine eindeutige Bezeichnung braucht. Wird er nicht gekennzeichnet, bearbeitet AID *datenname* nur dann, wenn es dafür eine Daten-Definition auf der Stufe 01 oder 77 gibt; sonst folgt eine Fehlermeldung.

index

wird wie in einer COBOL-Anweisung geschrieben, außer dass mehrere Indizes durch Komma getrennt werden müssen. Geben Sie den Namen eines Tabellenelementes ohne Index an bedeutet dies bei *sender*, dass die gesamte Tabelle übertragen wird. Geben Sie bei *empfänger* einTabellenelement ohne Index an, wird die Tabelle beginnend bei der Anfangsadresse in der Länge von *sender* überschrieben, ohne dass die Unterteilung in Tabellenelemente beachtet wird.
index kann folgendermaßen angegeben werden:



COBOL-Sonderregister

```
LINAGE-COUNTER
RETURN-CODE
SORT-CCSN
SORT-CORE-SIZE
SORT-EOW
SORT-FILE-SIZE
```

```
SORT-MODE-SIZE
SORT-RETURN
TALLY
```

Figurative Konstanten können nur als *sender* angegeben werden, der Adressselektor kann auf sie nicht angewandt werden. Die Figurativen Konstanten HIGH-VALUE und LOW-VALUE repräsentieren immer den alphanumerischen Wert, der ihnen standardmäßig oder nach den Vereinbarungen mit der PROGRAM COLLATING SEQUENCE-Klausel entspricht. Im Gegensatz zur COBOL MOVE-Anweisung wird bei der Verwendung einer *Figurativen Konstanten* im AID-Kommando %MOVE immer nur ein Zeichen übertragen.

```
ZERO
SPACE
HIGH-VALUE
LOW-VALUE
QUOTE
symbolic character
```

anweisungsname

bezeichnet die Adresse des ersten Befehls in einem Kapitel oder Paragraphen in der PROCEDURE DIVISION.

```
{L'kapitel'                      }
{L'paragraph' [IN L'kapitel']    }
```

Ist ein Paragraphenname innerhalb eines Programms nicht eindeutig, dann muss er mit dem Kapitelnamen des Kapitels, in dem er definiert wurde, gekennzeichnet werden: `L'paragraph' IN L'kapitel'`

Anweisungsnamen sind Adresskonstanten. Sie können nur als *sender* angegeben werden. Es wird die damit bezeichnete Adresse übertragen.
Mit nachfolgendem Pointer-Operator (*anweisungsname*->) bezeichnen Sie 4 Bytes des Programmcodes, der zu der ersten Anweisung in einem Kapitel oder Paragraphen generiert wurde. Für Zweibyte- bzw. Sechsbyte-Befehle müssen Sie eine entsprechende Längenmodifikation angeben. *anweisungsname* -> können Sie als *sender* und als *empfänger* verwenden. Siehe Beispiele.

source-referenz

bezeichnet die Adresse des ersten Befehls, der zu einer Anweisung in der PROCEDURE DIVISION generiert wurde, und muss in einem der folgenden Formate angegeben werden:

S'n'

für Zeilen mit Paragraphen- oder Kapitelnamen, in denen kein COBOL-Verb auftritt. Diese Angabe ist für Programme nicht möglich, die übersetzt wurden mit `STMT-REFERENCE=COLUMN1-TO-6`

S'nverb[m]' | S'xverb[m]'
für Zeilen, in denen ein COBOL-Verb steht.

Source-Referenzen sind Adresskonstanten. Sie können nur als *sender* angegeben werden. Es wird die damit bezeichnete Adresse übertragen.
Mit nachfolgendem Pointer-Operator (*source-referenz*->) bezeichnen Sie 4 Bytes des Programmcodes, der zu der Anweisung generiert wurde. Für Zweibyte- bzw. Sechsbyte-Befehle müssen Sie eine entsprechende Längenmodifikation angeben. *source-referenz* -> können Sie als *sender* und als *empfänger* verwenden. Siehe Beispiele.

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

```
%•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 [1]):
- Adressversatz (•)
- indirekte Adressierung (->)
- Typmodifikation (%A, %E, %S, %SX)
- Längenmodifikation (%L(...), %L=(ausdruck), %Ln)
- Adressselektion (%@(...))
- Zeichenkonvertierungsfunktionen %C() und %UTF16() (nur für sender)

Beginnt eine *kompl-speicherref* mit Anweisungsname oder Source-Referenz, muss anschließend der Pointer-Operator (->)geschrieben werden. In diesem Fall muss *anweisungsname* mit L'...' angegeben werden. Ohne den Pointer-Operator können Anweisungsname und Source-Referenz überall da verwendet werden, wo auch Sedezimalzahlen geschrieben werden können. 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: `%0G.2%AL2->`
Die letzten beiden Bytes des AID-Registers %0G werden als Adresse benutzt.

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 Typ und Länge nicht explizit angeben.
Trotzdem bleiben die Bereichsgrenzen der Ausgangsadresse (z.B. CSECT, *datenname*, Schlüsselwort etc.) wirksam. Sie dürfen weder durch Adressversatz noch Längenmodifikation überschritten werden. Sonst gibt AID eine Fehlermeldung aus. Erst durch die Verbindung von Adressselektion (%@) mit dem Pointer-Operator (->) wechseln Sie auf die Maschinencode-Ebene, auf der der Bereich den vom geladenen Programm belegten virtuellen Speicher umfasst.

Beispiel: `%MOVE CFELD.3%L5 INTO CFELD1`
Dieses Kommando wird von AID wegen Bereichsverletzung von CFELD abgelehnt. Die Datenfelder CFELD und CFELD1 belegen je 5 Bytes. Die letzten 2 Bytes von CFELD sowie die 3 anschließenden Bytes sollen nach CFELD1 übertragen werden. Richtig müsste das Kommando lauten:
`%MOVE %@(CFELD)->.3%L5 INTO CFELD1`

%@(...)
Mit dem Adressselektor können Sie die Anfangsadresse eines Dateieintrags, eines Datenfeldes, eines Sonderregisters oder einer komplexen Speicherreferenz als *sender* verwenden. Der Adressselektor liefert als Ergebnis eine Adresskonstante (siehe AID-Basishandbuch [1]).
Der Adressselektor lässt sich nicht auf Konstanten anwenden. Dazu gehören auch die Anweisungsnamen, die Source-Referenzen und die Figurativen Konstanten.

%L(...)
Mit dem Längenselektor können Sie die Länge eines Dateieintrags, eines Datenfeldes oder eines Sonderregisters als *sender* verwenden. Der Längenselektor liefert als Ergebnis eine Ganzzahl (siehe AID-Basishandbuch [1]).
Beispiel: `%MOVE %L(FELD1) INTO %OG`
Die Länge von FELD1 wird übertragen.

%L=(ausdruck)
Mit der Längenfunktion können Sie sich als *sender* einen Wert errechnen lassen. *ausdruck* wird gebildet aus dem Inhalt von Speicherreferenzen, Konstanten, Ganzzahlen und arithmetischen Operatoren. Nur ganzzahlige Speicherreferenz-Inhalte (Typ %F oder %A) sind zugelassen. Die Längenfunktion liefert als Ergebnis eine Ganzzahl (siehe AID-Basishandbuch [1]).
Beispiel: `%MOVE %L=(FELD1) INTO %OG`
Der Inhalt von FELD1 wird übertragen, wenn er ganzzahlig ist (Typ %F). Sonst gibt AID eine Fehlermeldung aus.

AID-Literal

Die folgenden AID-Literale (siehe AID-Basishandbuch [1]) können mit %MOVE übertragen werden:

`{C'x...x'| 'x...x'| U'x...x'}`	Character-Literal
`{X'f...f'}`	Sedezimal-Literal
`{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 die globale Einstellung (siehe %AID-Kommando) für dieses Kommando 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-Korrekturanweisungen (REPs) im SDF-Format erstellt. Wenn die Objekt-Strukturliste nicht zur Verfügung steht, erstellt AID keine Korrekturanweisungen und gibt eine Fehlermeldung aus.
Auch wenn *empfänger* nicht vollständig innerhalb einer CSECT liegt, oder *sender* länger als 3900 Byte ist, gibt AID eine Fehlermeldung aus und schreibt keinen REP. Um dennoch REP-Sätze zu erhalten, müssen Sie die Übertragung auf mehrere %MOVE-Kommandos verteilen.

AID hinterlegt die Korrekturen mit den nötigen LMS-Korrektur-Anweisungen in einer Datei mit dem Linknamen F6. Für den LMS-Lauf muss dann noch die MODIFY-ELEMENT-Anweisung eingefügt werden. 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 COBOL-Programm sind die folgenden Felder und Tabellen definiert:

```
01    ANZAHL-TAB-1.
   02 ZAHL-1 PIC 999 OCCURS 10 INDEXED BY I.

01    ANZAHL-TAB-2.
   02 ZAHL-2 PIC S9(6) OCCURS 50 INDEXED BY J.

01    FESTPUNKT-TAB.
   02 FEST-ZAHL PIC S999V99 OCCURS 26.

01    ZEICHEN PIC X(4).
01    GANZ-ZAHL PIC S9(7) BINARY.
```

1. `%MOVE ZAHL-1 INTO ZAHL-2`
   Bei beiden Tabellenelementen ist kein Index angegeben; AID überträgt daher die gesamte Tabelle ANZAHL-TAB-1 ohne Beachtung der Unterteilung in Tabellenelemente sedezimal linksbündig nach ANZAHL-TAB-2.

2. `%MOVE 20 INTO GANZ-ZAHL`
   In das Datenfeld GANZ-ZAHL, das auch im COBOL-Programm 4 Bytes belegt, schreibt AID ein Wort, das den Wert 20 (X’00000014’) enthält.

3. `%MOVE 20 INTO FEST-ZAHL(5)`
   Achtung: Wie im Beispiel 2 wird auch hier ein Wort mit dem Inhalt X’00000014’ nach FEST-ZAHL(5) geschrieben, was natürlich bei einem Tabellenelement vom Typ Festpunktzahl keinen Sinn ergibt. Um den Wert 20 nach FEST-ZAHL(5) zu übertragen, müssten Sie ein %SET-Kommando eingeben (siehe %SET), das vor der Übertragung eine Konvertierung durchführt.

4. `%MOVE S'12MOV' INTO %0G`
   Die Adresse der Anweisung mit dem Namen 12MOV wird in das AID-Register %0G geschrieben.

%ON

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

– Mit *write-ereignis* beschreiben Sie das Ereignis des schreibenden Zugriffs auf einen Speicherbereich. Immer wenn das Programm schreibend auf den angegebenen Speicherbereich zugreift, soll AID den Programmablauf unterbrechen, und *subkdo* bearbeiten.

– Mit *ereignis* beschreiben Sie eins der anderen Ereignisse (normale oder abnormale Programmbeendigung, einen Supervisor-Call (SVC), einen 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. Das 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 Basisqualifikation E=VM eingestellt sein (siehe %BASE).

%ON verändert den Programmzustand nicht.

write-ereignis

Mit dem Schlüsselwort %WRITE schalten Sie die Schreibüberwachung ein. Dahinter setzen Sie in Klammern den zu überwachenden Speicherbereich. Wenn das Programm ein Byte innerhalb des festgelegten Bereichs verändert, wird der Programmablauf unterbrochen und *subkdo* ausgeführt. Die Unterbrechung erfolgt nach dem Befehl, der die Speicherstelle verändert hat; sie kann auch in einer Laufzeitroutine auftreten.

Es kann immer nur ein *write-ereignis* definiert sein. Die Eingabe von einem neuen *write-ereignis* überschreibt ein bestehendes. Andere Ereignisse können jedoch gleichzeitig angemeldet sein. 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 %CONTROL*n* oder ein %TRACE mit maschinennahem *kriterium* angemeldet ist, wird die Eingabe von %ON *write-ereignis* mit einer Fehlermeldung abgewiesen.

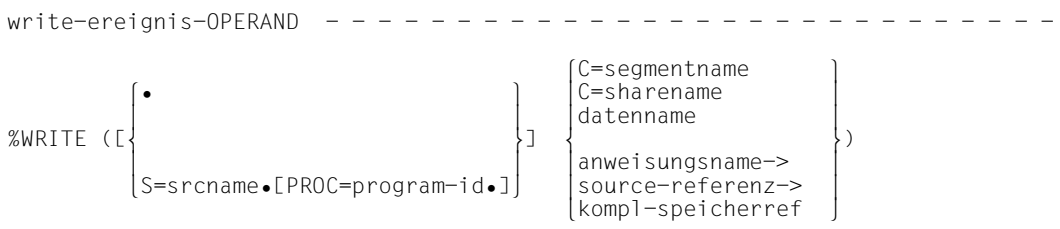
– Wenn ein Maschinenbefehl durch einen %CONTROL*n* oder %TRACE mit symbolischem *kriterium* mit der AID-internen Markierung (X'0A81') überschrieben wurde, bemerkt AID den schreibenden Zugriff dieses Befehls nicht.

– 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 %CONTROL*n*- 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. Der Bereich kann maximal 64KB lang sein, 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.



- 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 PROC-Qualifikation und dem anschließenden Operandenteil ein Punkt stehen.

S=srcname
: geben Sie nur an, wenn *write-ereignis* nicht für die aktuelle Übersetzungseinheit vereinbart werden soll.

PROC=program-id
: geben Sie nur an, wenn Sie einen Daten- oder Anweisungsnamen ansprechen, der nicht im aktuellen Programm liegt (siehe Kapitel „COBOL-spezifische Adressierung“ auf Seite 17) oder in der aktuellen Übersetzungseinheit nicht eindeutig ist.

Stimmen *srcname* in der S-Qualifikation und *program-id* überein, schreiben Sie an Stelle der beiden nur die PROG-Qualifikation.

Vor die anschließend aufgeführten C-Qualifikationen können Sie keine Qualifikation setzen. Mit der C-Qualifikation verlassen Sie die symbolische Ebene. Unmittelbar anschließend kann kein symbolischer Operand geschrieben werden (siehe Abschnitt „Symbolische Speicherreferenzen“ auf Seite 21), nur eine *kompl-speicherref*.

C=segmentname
: Der zu überwachende Speicherbereich umfasst mit dieser Angabe das bezeichnete Segment.

C=sharename
: Der zu überwachende Speicherbereich umfasst mit dieser Angabe den bezeichneten Bindemodul.

datenname
: ist der im Quellprogramm definierte Name eines Datenfeldes oder eines COBOL-Sonderregisters. Er kann wie im COBOL-Programm gekennzeichnet und indiziert werden (siehe Abschnitt „Symbolische Speicherreferenzen“ auf Seite 21 *datenname*).

  datenname ist eine maximal 30stellige alphanumerische Zeichenfolge.

  datenname [kennzeichnung][...][(index[,...])]

kennzeichnung

Ist *datenname* nicht eindeutig innerhalb eines Programms, wird er mit IN oder OF einer bestimmten Datengruppe zugeordnet. *datenname* muss mit so vielen Kennzeichnungen versehen sein, wie er für seine eindeutige Bezeichnung braucht. Wird er nicht gekennzeichnet, bearbeitet AID *datenname* nur dann, wenn es dafür eine Daten-Definition auf der Stufe 01 oder 77 gibt; sonst folgt eine Fehlermeldung.

index

Ist *datenname* der Name eines Tabellenelementes, kann er indiziert bzw. subskribiert werden; die Schreibweise unterscheidet sich gegenüber COBOL nur darin, dass mehrere Indizes durch Komma getrennt werden müssen. Wenn Sie den Namen eines Tabellenelementes ohne Index angeben, sprechen Sie die gesamte Tabelle an.
index kann folgendermaßen angegeben werden:

```
{ n                          }
{ index-name                 }
{ datenname                  }
{ TALLY                      }
{ arithmetrischer ausdruck   }
```

COBOL-Sonderregister

```
LINAGE-COUNTER
RETURN-CODE
SORT-CCSN
SORT-CORE-SIZE
SORT-EOW
SORT-FILE-SIZE
SORT-MODE-SIZE
SORT-RETURN
TALLY
```

```
{anweisungsname  } ->
{source-referenz }
```

bezeichnet 4 Bytes des Programmcodes ab der Adresse, die in der Adresskonstanten hinterlegt ist. Soll eine andere Anzahl von Bytes überwacht werden, müssen Sie eine entsprechende Längenmodifikation angeben.

anweisungsname
: muss in einem der folgenden Formate angegeben werden:

```
{L'kapitel'                      }
{L'paragraph' [IN L'kapitel']    }
```

Ist ein Paragraphenname innerhalb eines Programms nicht eindeutig, muss er mit dem Kapitelnamen des Kapitels, in dem er definiert wurde, gekennzeichnet werden: `L'paragraph' IN L'kapitel'`

source-referenz
: muss in einem der folgenden Formate angegeben werden:

S'n'
: für Zeilen mit Paragraphen- oder Kapitelnamen, in denen kein COBOL-Verb auftritt. Diese Angabe ist für Programme nicht möglich, die mit `STMT-REFERENCE=COLUMN1-TO-6` übersetzt wurden.

S'nverb[m]' | S'xverb[m]'
: für Zeilen, in denen ein COBOL-Verb steht.

kompl-speicherref
: Folgende Operationen können in *kompl-speicherref* vorkommen (siehe AID-Basishandbuch [1]):
  – Adressversatz (•)
  – indirekte Adressierung (->)
  – Typmodifikation (%A, %S, %SX)
  – Längenmodifikation (%L(...), %L=(ausdruck), %Ln)
  – Adressselektion (%@(...))

  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 *datenname* müssen Sie die Bereichsgrenzen beachten oder mit %@(datenname)-> auf Maschinencode-Ebene wechseln. Beginnt eine *kompl-speicherref* mit Anweisungsname oder Source-Referenz, muss anschließend der Pointer-Operator (->)geschrieben werden. In diesem Fall muss *anweisungsname* mit L'...' angegeben werden. Ohne den Pointer-Operator können Anweisungsname und Source-Referenz überall da verwendet werden, wo auch Sedezimalzahlen geschrieben werden können.

ereignis

Ein Schlüsselwort legt fest, bei welchem Ereignis (Programmfehler, abnormale Programmbeendigung, Supervisor-Call etc.) AID das angegebene *subkdo* bearbeiten soll. Auf einen Ereigniscode, der mit einer STXIT-Routine bearbeitet wurde, kann anschließend nicht noch mit einem zu diesem *ereignis* vereinbarten *subkdo* reagiert werden. Wird ein Subkommando zum Ereignis %ANY ausgeführt, entfällt bei der anschließenden Beendigung des Programms die Abfrage, ob ein Dump ausgegeben werden soll. Bei Bedarf müssen Sie im Subkommando mit `/CREATE-DUMP` selbst die Ausgabe des Dump anstoßen.

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 der Subkommandos).
Wenn ein *write-ereignis* gleichzeitig mit einem anderen *ereignis* eintritt, wird erst das Subkommando zum *write-ereignis* abgearbeitet. Zur Auswahl der SVC-Nummern und Ereigniscodes siehe „Makroaufrufe an den Ablaufteil“ [7].

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 privilegierten Operation und vor Abbruch des Programms.
%ARTHCHK	nach Auftreten eines Datenfehlers, Divisionsfehlers, Exponentenüberlaufs 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) %SVC	vor Ausführung des Systemaufrufs (SVC) mit der angegebenen Nummer vor Ausführung eines beliebigen Systemaufrufs (SVC).

ereignis	***subkdo* wird bearbeitet:**
%LPOV(x...x) %LPOV	nach dem Laden des Segmentes mit dem angegebenen Namen. nach dem Laden eines beliebigen Segmentes (der Name wird mit %D %LINK ausgegeben).
%TERM(N[ORMAL]) %TERM(A[BNORMAL]) %TERM(D[UMP]) %TERM(S[TEP]) %TERM	vor normaler Beendigung eines Programms. vor abnormaler Beendigung eines Programms, jedoch nach der Ausgabe eines Speicherabzugs. vor Ausgabe eines Speicherabzugs mit anschließender Programmbeendigung. vor Beendigung eines Programms mit anschließender Verzweigung innerhalb von Prozeduren. vor Beendigung eines Programms durch alle vorher beschriebenen %TERM-Ereignisse.
%ANY	vor der Beendigung eines Programms auf Grund eines Programmfehlers bzw. durch die oben beschriebenen %TERM-Ereignisse.

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 die angegebene Nummer des Ereigniscodes oder die SVC-Nummer sinnvoll oder zulässig ist.

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 AID-Basishandbuch [1].

```
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 hier auch die Kommandos %CONTROL*n*, %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 'MONA12 GELADEN'; %STOP>`
   Jedes Mal, nachdem das Segment MONA12 geladen wurde, gibt AID den Text 'MONA12 GELADEN' aus und hält das Programm an.

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

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

4. `%ON %WRITE(PROG=HPROG.TABELLE) <%D %HLLOC(%PC ->),TABELLE F1=MAX>`
   Immer, wenn Daten in TABELLE im Hauptprogramm HPROG überschrieben wurden, werden die symbolischen Lokalisierungsinformationen zum aktuellen Befehlszählerstand und der Inhalt von TABELLE ausgegeben. Die Ausgabe erfolgt in die Datei, die dem Linknamen F1 zugeordnet wurde. Dann wird das Programm fortgesetzt. Anschließend kann in dieser Datei gesucht werden, wann TABELLE überschrieben wurde.

%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 den Kommandos %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 bzw. /EXIT-JOB.

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

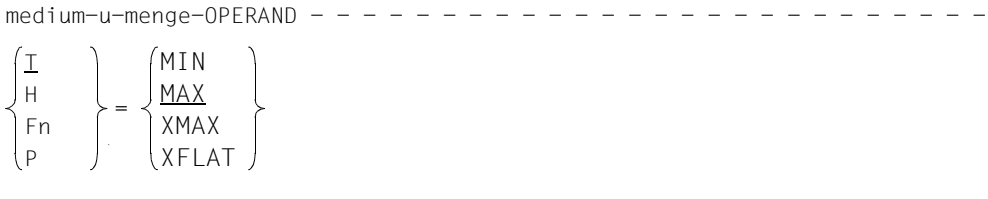
bezeichnet das Kommando, für das die Vereinbarungen gelten sollen. Jeweils eines der folgenden Kommandos kann hier angegeben werden:

```
{ %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 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 ist ausführlich im AID-Basishandbuch [1] beschrieben.

T Terminal-Ausgabe
H Hardcopy-Ausgabe (schließt die Terminal-Ausgabe mit ein und kann nicht gemeinsam mit *T* angegeben werden)
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.

Bei der Ausgabe von Informationen, die in UTF16/ UTFE vorliegen, berücksichtigt AID den CCSN des Ausgabemediums und führt eine entsprechende Konvertierung durch. Als CCSN sind UTFE und alle 1-Byte-EBCDIC-Codierungen zugelassen, die von XHCS unterstützt werden. Mit %SHOW %CCSN können die aktuell zugewiesenen OUTFILEs mit den von AID verwendeten CCSNs ausgegeben werden.

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

– Mit *datei* bezeichnen Sie die Ausgabedatei.

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 bzw. /EXIT-JOB.

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

1. Sie geben ein /ADD-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.

Hat eine Datei selbst keinen CCSN, so verwendet AID den über %AID EBCDIC eingestellten CCSN, wenn eine Zeichenkonvertierung notwendig ist. Ist der CCSN der Datei für AID unzulässig und ist eine Zeichenkonvertierung notwendig, z.B. weil das Eingabemedium vom Typ UTFE ist, wird keine Ausgabe durchgeführt.

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). Achten Sie deshalb darauf, dass Sie in die Datei mit dem Linknamen F6 keine anderen Ausgaben schreiben lassen.

datei

gibt den vollqualifizierten Dateinamen an, mit dem AID die Ausgabedatei katalogisiert und geöffnet.
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 Adressoperanden 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 einen %QUALIFY mit neuer *vorqualifikation* überschrieben wird, bis sie durch ein %QUALIFY ohne Operanden aufgehoben wird oder bis /LOGOFF bzw. /EXIT-JOB.

Bei der Eingabe eines %QUALIFY wird das Kommando nur syntaktisch überprüft. Ob dem angegebenen Linknamen eine Dump-Datei zugewiesen bzw. ob das angegebene Programm 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*n*, %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 Adressoperanden 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

besteht aus einer einzelnen oder einer Folge von Qualifikationen, die dann durch einen Punkt getrennt werden müssen.

In den Adressoperanden nachfolgender AID-Kommandos können Sie sich durch Voran Stellen eines Punktes auf die im %QUALIFY definierte *vorqualifikation* beziehen.

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

    ⎧VM ⎫                          ⎧PROC=program-id ⎫
[E=⎨   ⎬] [[•] S=srcname] [[•] ⎨C=segmentname    ⎬]
    ⎩Dn ⎭                          ⎩C=sharename     ⎭

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

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

S=srcname
: *srcname* bezeichnet eine Übersetzungseinheit.

PROC=program-id
: bezeichnet ein Programm.

Stimmen *srcname* in der S-Qualifikation und *program-id* überein, schreiben Sie nur die PROG-Qualifikation.

C=segmentname
: *segmentname* setzt sich zusammen aus der Angabe im PROGRAM-ID-Paragraphen und der Segmentnummer. Nur die ersten 6 Stellen der PROGRAM-ID werden zur Bildung des Segmentnamen herangezogen.

C=sharename
: *sharename* setzt sich zusammen aus den ersten 7 Stellen der PROGRAM-ID-Angabe und dem Zeichen @.

Beispiele

1.
```
%QUALIFY E=D1.PROG=SORT
%D .TAB(1)
```
Durch die *vorqualifikation* hat der %DISPLAY dieselbe Bedeutung wie das folgende, ausgeschriebene %DISPLAY-Kommando:
```
%D E=D1.PROG=SORT.TAB(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
```

3.
```
%QUALIFY PROG=SUB
%D .TAB(I)
%D .L'AUS1' IN L'AUSGABE'
```
In den beiden %DISPLAY-Kommandos wird wie in Beispiel 1 und 2 vor den Punkt die PROG-Qualifikation aus dem %QUALIFY geschrieben.
Im ersten %DISPLAY sprechen Sie damit nicht nur das Tabellenelement TAB aus der Programmeinheit SUB an, sondern auch der Index I wird in der Programmeinheit SUB gesucht.
Dasselbe gilt im zweiten %DISPLAY für die Kennzeichnung des Paragraphen: die PROG-Qualifikation bezieht sich sowohl auf den Paragraphen AUS1 als auch auf das kennzeichnende Kapitel AUSGABE.

%REMOVE

Mit %REMOVE heben Sie die Testvereinbarungen der Kommandos %CONTROL*n*, %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

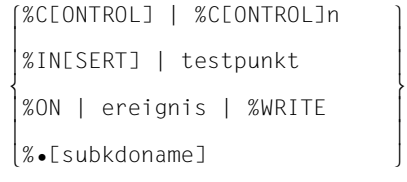
Steht ein %REMOVE in einem Subkommando, der dieses Subkommando oder die zugehörige Überwachungsbedingung (*testpunkt*, *ereignis* oder *kriterium*) löscht, werden nachfolgende Kommandos in *subkdo* nicht mehr ausgeführt. Diese Angabe ist deshalb nur als letztes Kommando in einem Subkommando sinnvoll.

%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 %CONTROL*n* werden gelöscht.

%C[ONTROL]*n*
: Der %CONTROL*n* 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.

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

%WRITE
: Das *write-ereignis* wird gelöscht.

%.[subkdoname]
: löscht das Subkommando eines %CONTROL*n* oder %INSERT mit *subkdoname*.

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

  Da %CONTROL*n* nicht gekettet werden kann, wird auch der zugehörige %CONTROL*n* gelöscht. Das Löschen des Subkommandos entspricht folglich einer Löschung des %CONTROL*n* 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 [1]). 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 <CALL: %D %.>
%REM %C1
%REM %.CALL
```

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

2.
```
%IN S'58SEA' <SUB1: %D ZEICHEN, ZAHL>
%IN S'58SEA' <SUB2: %D ERGEB; %REM %.>
%R
...
%REM S'58SEA'
```

   Wenn der Testpunkt S’58SEA’ erreicht wird, wird ERGEB ausgegeben. Danach wird das Subkommando SUB2 gelöscht. Dieses Subkommando wird also nur ein einziges Mal ausgeführt. Dann werden ZEICHEN und ZAHL ausgegeben, und das Programm wird fortgesetzt. Immer wenn danach der Programmablauf an den Testpunkt S’58SEA’ kommt, wird Subkommando SUB1 ausgeführt. Mit `%REM S'58SEA'` wird später der Testpunkt gelöscht. Ein `%REM %.SUB1` würde dasselbe bewirken, denn zum Testpunkt S’58SEA’ 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.

%RESUME beendet alle aktiven %TRACE-Kommandos, %CONTINUE hingegen hat keine Auswirkungen auf %TRACE.

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 Datenfelder oder Dateierklärungen, alle Datenfelder oder Dateierklärungen der aktuellen Aufrufhierarchie oder die Programmnamen der aktuellen Aufrufhierarchie werden ausgegeben. Die aktuelle Aufrufhierarchie reicht von der Unterprogramm-Ebene, auf der das Programm unterbrochen wurde, über die Folge der CALL-Anweisungen bis zum äußersten Programm. Die Ausgabe erfolgt über SYSOUT, SYSLST oder in eine katalogisierte Datei.

– Mit *dump-bereich* bezeichnen Sie die Datenfelder oder Dateierklärungen 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 Ausgabe-Medien 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][,...]]

Befinden sich in der Hierarchie Übersetzungseinheiten, für die es keine LSD-Sätze gibt, auch nicht in einer PLAM-Bibliothek, so geben Sie die Übersetzungseinheiten einzeln an, für die LSD-Sätze geladen wurden oder für die sie aus einer PLAM-Bibliothek nachladbar sind (siehe %SYMLIB). *dump-bereich* können Sie bis zu 7mal wiederholen.

%SDUMP ohne Operanden gibt alle Datenfelder der aktuellen Aufrufhierarchie aus, soweit AID auf die zugehörigen LSD-Sätze zugreifen kann. Daten, die mehrfach definiert sind, werden auch mehrfach ausgegeben.

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

Es empfiehlt sich das Kommando nicht unmittelbar nach dem Laden einzugeben, da noch nicht alle Einträge in der DATA DIVISION initialisiert sind (z.B. Datensatzbeschreibungen und Sonderregister) und es auch zu einer Fehlermeldung kommen kann.

Geben Sie für *dump-bereich* einen Namen an, der nicht in den LSD-Sätzen verzeichnet ist, gibt AID eine Fehlermeldung aus. Die anderen *dump-bereiche* desselben Kommandos werden ordnungsgemäß bearbeitet.

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 eines Programms oder einzelne Datenfelder bzw. Dateierklärungen ausgeben.
Datenfelder bereitet AID entsprechend der Definition im Quellprogramm auf. Passt der Inhalt nicht zum definierten Speichertyp, wird die Ausgabe mit einer Fehlermeldung abgelehnt.

Ein *datenname* oder *dateiname*, der in mehreren DATA DIVISIONs der aktuellen Aufrufhierarchie definiert ist, wird auch mehrmals ausgegeben, es sei denn, *dump-bereich* wurde mit einer Qualifikation eingeschränkt oder *datenname* ist gekennzeichnet. Enthält eine auszugebende Datengruppe oder DATA DIVISION Redefinitionen, so werden sie mit ausgegeben.

Bei einem %SDUMP, mit dem ganze DATA DIVISIONs ausgegeben werden, sind alle vom Compiler erzeugten Datenfelder enthalten. Die Ausgabe enthält auch Informationen über die im Programm definierten Dateien.

```
dump-area-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

                  ┌   ┌filename┐   ┐
                  │ [ │        │ ] │
[•][qua[•]][      │   └dataname┘   │ ]
                  │                │
                  │   %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.

qua

Eine oder mehrere Qualifikationen geben Sie an, wenn die Unterbrechungsstelle nicht im Gültigkeitsbereich des adressierten Objekts liegt oder wenn das Speicherobjekt an der Unterbrechungsstelle nicht sichtbar ist. Sie geben nur die Qualifikationen an, die zur eindeutigen Adressierung genügen.

E ={VM | Dn}

geben Sie nur an, wenn die aktuelle Basisqualifikation für *dump-bereich* nicht gelten soll. Wenn Sie nur eine Basisqualifikation angeben, erhalten Sie alle Daten der entsprechenden Aufrufhierarchie.

S=srcname

geben Sie nur an, wenn *dump-bereich* nicht in der aktuellen Übersetzungseinheit liegen soll. Diese muss in der Aufrufhierarchie liegen.

PROC=program-id

geben Sie an, wenn *dump-bereich* nur für das angegebene Programm gelten soll. Es muss in der Aufrufhierarchie liegen. Endet *dump-bereich* mit einer PROC-Qualifikation, gibt AID alle Daten dieses Programms aus.

Stimmen *srcname* in der S-Qualifikation und *program-id* überein, schreiben Sie nur die PROG-Qualifikation.

NESTLEV= level-nummer

level-nummer Nummer einer Ebene in der aktuellen Aufrufhierarchie

Auf *level-nummer* kann nur *datenname* folgen.
Das %SDUMP-Kommando gibt entweder einen symbolischen Dump für alle Daten aus, die auf der angegebenen Ebene definiert wurden, oder den Datennamen *datenname*, der auf der angebenen Ebene der Aufrufhierarchie definiert wurde.

dateiname

ist der Name einer Datei aus einer Dateierklärung in der FILE SECTION der DATA DIVISION. AID gibt folgende Informationen aus:
den Datei-Status und, falls die Datei geöffnet ist, den Inhalt des Datensatz-Bereiches sowie eventuell den Satzschlüssel.

datenname

ist der im Quellprogramm definierte Name eines Datenfeldes, der Name eines COBOL-Sonderregisters oder einer Figurativen Konstanten.
datenname ist eine maximal 30stellige alphanumerische Zeichenfolge.

datenname [kennzeichnung][...][(index[,...])]

kennzeichnung

ist *datenname* nicht eindeutig innerhalb eines Programms, kann er gekennzeichnet werden, indem er mit IN oder OF einer bestimmten Datengruppe zugeordnet wird.
datenname muss mit so vielen Kennzeichnungen versehen sein, wie er für seine eindeutige Bezeichnung braucht. Wird er nicht gekennzeichnet, gibt AID nur dann Daten für *datenname* aus, wenn es dafür eine Daten-Definition auf derStufe 01 oder 77 gibt; sonst folgt eine Fehlermeldung.

index

ist *datenname* der Name eines Tabellenelements, kann er wie in einer COBOL-Anweisung indiziert bzw. subskribiert werden; die Schreibweise unterscheidet sich gegenüber COBOL nur darin, dass mehrere Indizes durch Komma

getrennt werden müssen. Wenn Sie den Namen eines Tabellenelements ohne Index angeben, wird die gesamte Tabelle ausgegeben.
index kann folgendermaßen angegeben werden:

```
{ n                          }
{ index-name                 }
{ datenname                  }
{ TALLY                      }
{ arithmetrischer ausdruck   }
```

COBOL-Sonderregister

```
LINAGE-COUNTER
RETURN-CODE
SORT-CCSN
SORT-CORE-SIZE
SORT-EOW
SORT-FILE-SIZE
SORT-MODE-SIZE
SORT-RETURN
TALLY
```

Figurative Konstanten

```
ZERO
SPACE
HIGH-VALUE
LOW-VALUE
QUOTE
symbolic character
```

%NEST

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

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

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

medium-u-menge ist ausführlich im AID-Basishandbuch [1] beschrieben.

ITerminal-Ausgabe
HHardcopy-Ausgabe (schließt die Terminal-Ausgabe mit ein und kann nicht gemeinsam mit *T* angegeben werden)
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 am Ende der Liste.

```
<PACKED(size),D>
packed-name = packed-value
```

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

```
<ZONED(size),D>
zoned-name = zoned-value
```

size	Speicherlänge in Bytes
zoned-name	bezeichnet ein Element vom Typ Zoned Decimal (ungepackte Dezimalzahl)
zoned-value	Dezimalzahl (D); Wert von *zoned-name*.

```
<BINARY(size),D>
binary-name = binary-value
```

size	Speicherlänge in Bytes.
binary-name	bezeichnet ein Element vom Typ Binary (Binärzahl mit fester Kommastelle).
binary-value	Dezimalbruch mit fester Kommastelle; Wert von *binary-name*.

```
<DECIMAL(size),D>
decimal-name = decimal-value
```

size	Speicherlänge in Bytes.
decimal-name	bezeichnet ein Element vom Typ Decimal (gepackte Dezimalzahl mit fester Kommastelle).
decimal-value	Dezimalbruch mit fester Kommastelle; Wert von *decimal-name*.

```
<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*.

```
<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 am Ende der Liste.

```
<CLASS(size),S>
class-name = class-value
```

size	Speicherlänge in Bytes.
class-name	bezeichnet ein Element vom Typ CLASS.
class-value	symbolische Konstante (S), Wert von *class-name*.

```
<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 . Details zu Array-Bereichen siehe Hinweis .

```
<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 „Datentyp FILE bei XMAX und XFLAT“. Siehe auch Hinweis 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.

Datentyp FILE bei XMAX und XFLAT

Der Datentyp FILE ist aus mehreren elementaren Datentypen zusammengesetzt und besitzt mehrere Schlüsselwörter, die mit einem Unterstrich beginnen.

```
<FILE(size)>
f-name
<CHARS (file-name-length), C>
_FILE_NAME = |file-name|
<CLASS(size),S>
_OPEN_MODE = status
<CHARS (record-size), C>
_RECORD =
|record-content|
   [
<INT (4), D>
_RECORD_NO = record-number
   ]
   [
<CHARS (key-length), C>
_RECORD_KEY = |key-value|
   ]
```

f-name	bezeichnet ein Element vom Typ File.
size	Speicherlänge in Bytes.
file-name-length	Länge von *file-name*.
file-name	vollqualifizierter Dateiname
status	aktueller Status der Datei *file-name*, z.B. OPEN-OUTPUT, CLOSE, ...).
record-size	Länge des aktuellen Satzes, der in _RECORD ausgegeben wird.
record-content	Inhalt des aktuellen Satzes eingeschlossen in '|'. Ist der Satz länger als 80 Zeichen, dann werden bei XFLAT nur die ersten 72 Zeichen ausgegeben, gefolgt von drei Punkten, die die Unvollständigkeit der Ausgabe anzeigen.
record-number	relative Satznummer bei relativer Dateiorganisation.
key-length	Länge des Primärschlüssels bei indizierter Dateiorganisation.
key-value	Wert des Primärschlüssels bei indizierter Dateiorganisation.

Beispiele

1. %SDUMP

   Mit diesem Kommando wird ein symbolischer Dump aller DATA DIVISIONs der aktuellen Aufrufhierarchie angefordert. Der Wert für *medium-u-menge* ist T=MAX. Die Übersetzungsliste zu dieser SDUMP-Ausgabe finden sie im Abschnitt „Quellprogrammliste“ auf Seite 171.

```
 SRC_REF:     57SEA   SOURCE: MOBS      PROC: MOBS  ***************************
_COMPILER          =  |COBOL2000 V01.4A02|
_COMPILATION_DATE =  |2006-06-23|
_COMPILATION_TIME =  |09:01:33|
_PROGRAM_NAME    =   |MBOS|
_EBCDIC-CCSN     =   |EDF03IRV|
 ZERO         =        0
 HIGH-VALUE   = FF
 LOW-VALUE    = 00  .
 SPACE        = | |
 QUOTE        = |"|
 01_LAST_EXCEPTION
   02_EXCEPTION_NAME  = |      |
 TALLY          =      +0
 RETURN         =     +0
```

   Die %SDUMP-Ausgabe beginnt mit einer Kopfzeile. Sie enthält die Source-Referenz der Anweisung, bei der das Program unterbrochen wurde und den Namen des aktuellen Programms.

Es folgen Angaben zum Testobjekt, figurativen Konstanten und Sonderregister.

```
TEXTDAT
_FILE_NAME  = |M.EIN                                                          |
_OPEN_MODE  = OPEN-INPUT
_RECORD     =
|DIES IST EINE DATEI, DIE ALS EINGABE DIENT FUER EIN PROGRAMM,.................|
|..............................................................................|
|..............................................................................|
|.........................|
```

Datei-Information zur Datei TEXTDAT.

```
01      SATZ
 02     FELD(1:61)
         (  1) |D| (  2) |I| (  3) |E| (  4) |S| (  5) | | (  6) |I|
         (  7) |S| (  8) |T| (  9) | | ( 10) |E| ( 11) |I| ( 12) |N|
         ( 13) |E| ( 14) | | ( 15) |D| ( 16) |A| ( 17) |T| ( 18) |E|
         ( 19) |I| ( 20) |,| ( 21) | | ( 22) |D| ( 23) |I| ( 24) |E|
         ( 25) | | ( 26) |A| ( 27) |L| ( 28) |S| ( 29) | | ( 30) |E|
         ( 31) |I| ( 32) |N| ( 33) |G| ( 34) |A| ( 35) |B| ( 36) |E|
         ( 37) | | ( 38) |D| ( 39) |I| ( 40) |E| ( 41) |N| ( 42) |T|
         ( 43) | | ( 44) |F| ( 45) |U| ( 46) |E| ( 47) |R| ( 48) | |
         ( 49) |E| ( 50) |I| ( 51) |N| ( 52) | | ( 53) |P| ( 54) |R|
         ( 55) |O| ( 56) |G| ( 57) |R| ( 58) |A| ( 59) |M| ( 60) |M|
         ( 61) |,|
```

SATZ heißt die Datensatz-Definition für die Datei TEXTDAT. Der Inhalt hat Tabellenform und wird mit einem fest zugeordneten Index indiziert. Die Elemente der Tabelle sind vom Typ alphanumerisch. Deshalb wird der Element-Inhalt von senkrechten Strichen begrenzt dargestellt. Jedem Tabellenwert ist der zugehörige Indexwert in runden Klammern vorangestellt.

```
K    =              +1
SLF  =    61
VERARB-SCHALTER=|0|
```

Zu Datenelementen der Stufe 77 oder 01 wird keine Stufennummer ausgegeben.

```
01      A-Z-TAB
 02                  = |ABCDEFGHIJKLMNOPQRSTUVWXYZ|

01      ABC-TAB
 02     ZEICHEN(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|

 I                    = +1

 01      ANZAHL-TAB
  02     ZAHL(1:26)
         (  1)      0 (  2)      0 (  3)      0 (  4)      0 (  5)      0
         (  6)      0 (  7)      0 (  8)      0 (  9)      0 ( 10)      0
         ( 11)      0 ( 12)      0 ( 13)      0 ( 14)      0 ( 15)      0
```

```
          ( 16)      0 ( 17)      0 ( 18)      0 ( 19)      0 ( 20)      0
          ( 21)      0 ( 22)      0 ( 23)      0 ( 24)      0 ( 25)      0
          ( 26)      0

 J           =     +1
```

Datengruppen A-Z-TAB, ABC-TAB und ANZAHL-TAB. Die Datengruppen ABC-TAB und ANZAHL-TAB haben Tabellenstruktur. Sie bestehen jeweils aus 26 Elementen. ABC-TAB ist alphanumerisch definiert und wird mit dem Index I indiziert, ANZAHL-TAB ist vom Typ numerisch und wird mit J indiziert. Beide Indizes haben den Wert 1.

```
         ANZ-SUMME      =         +1

         PROZ-SUMME     =      +0.00

 01      AUS-KOPF
  02                    = |BUCHSTABE ANZAHL PROZENT|

 01      AUS-ZEILE
  02     BUCHSTABE      = |.|
  02                    = |         |
  02     ANZAHL         = |.......|
  02                    = |   |
  02     PROZENT        = |......|

 01      AUS-FUSS
  02                    = |GESAMT:    |
  02     A-SUMME        = |......|
  02                    = |   |
  02     P-SUMME        = |......|
```

Definitionen von Feldern der Kopf- und Fußzeile.

2. `%SDUMP %NEST`

   Die aktuelle Aufrufhierarchie soll ausgegeben werden.

```
SRC_REF: 75EXI   SOURCE: UNTER     PROC: UNTER ******************************
SRC_REF: 41CALL  SOURCE: BEISP     PROC: BEISP ******************************
```

Das Programm wurde an der Anweisung mit dem Namen 75EXI in der Programmeinheit UNTER unterbrochen. Die zweite Zeile nennt die Programmeinheit BEISP, aus der UNTER mit der Anweisung CALL aufgerufen wurde. Der CALL steht in der Anweisungszeile 41. Die aktuelle Aufrufhierarchie hat zwei Stufen.

3. Beispiele für XMAX und XFLAT

   Folgendes COBOL-Programm soll getestet werden:

```
IDENTIFICATION DIVISION. PROGRAM-ID. X-COB22. ENVIRONMENT DIVISION.
CONFIGURATION SECTION. OBJECT-COMPUTER.
PROGRAM COLLATING SEQUENCE IS MYSEQ.

SPECIAL-NAMES.
ALPHABET MYSEQ IS 'A' 'B' 'Y' 'Z'. INPUT-OUTPUT SECTION.
FILE-CONTROL.
SELECT RDAT         ASSIGN TO DAT
ACCESS DYNAMIC      RELATIVE KEY RELKY
RELATIVE            FILE STATUS FS, FSEXT.

DATA DIVISION. FILE SECTION.
FD RDAT GLOBAL.
01 RSATZ GLOBAL.
  02 FSEXT11      PIC S9(4) COMP.
  02 FSEXT21      PIC X(146). WORKING-STORAGE SECTION.
01 FSEXT GLOBAL.
  02 FSEXT1       PIC S9(2) COMP.
  02 FSEXT2       PIC X(4).
01 FS GLOBAL.
  02 FS1          PIC X.
  02 FS2          PIC X.
01 DAT PIC X(8) VALUE "DATA.001".
01 RELKY PIC 9(2) VALUE 0.

01 DATABOX.
  02 BNR USAGE BINARY.
  02 INX USAGE INDEX.
  02 PPP USAGE POINTER.
  02 C0      PIC 9(9)       COMP    VALUE -12345.67890.
  02 C0S     PIC S9(9)      COMP    VALUE -12345.67890.
  02 C5      PIC 9(9)       COMP-5  VALUE -12345.67890.
  02 C5S     PIC S9(9)      COMP-5  VALUE -12345.67890.
  02 C0V     PIC 9(9)V9(5)  COMP    VALUE -12345.67890.
  02 C0SV    PIC S9(9)V9(5) COMP    VALUE -12345.67890.
  02 C5V     PIC 9(9)V9(9)  COMP-5  VALUE -12345.67890.
  02 C5SV    PIC S9(9)V9(9) COMP-5  VALUE -12345.67890.
  02 C3      PIC 9(9)       COMP-3  VALUE -12345.67890.
  02 C3S     PIC S9(9)      COMP-3  VALUE -12345.67890.
  02 C3V     PIC 9(9)V9(9)  COMP-3  VALUE -12345.67890.
  02 C3SV    PIC S9(9)V9(9) COMP-3  VALUE -12345.67890.
  02 C1                     COMP-1  VALUE -12345.67890.
  02 C2                     COMP-2  VALUE -12345.67890.
```

```
  02 Z20     PIC 9(20)              VALUE -98765432101234567890.
  02 Z16S    PIC S9(16)     SIGN IS LEADING SEPARATE
                                    VALUE -98765432101234567890.
  02 Z16T    PIC S9(16)     SIGN IS TRAILING
                                    VALUE -98765432100123456789.
  02 D20C    DISPLAY                VALUE "-98765432101234567890".
  02 NAT     PIC N(46)
               VALUE N"1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890".
  02 G REDEFINES NAT group-usage national.


    03 G1 pic N(10).
    03 G2 pic N(26).
    03 G3 pic N(10).

********** 1,2,3 -DIMENSION TABLEs      **********
01 TXT      PIC X(48)
               VALUE "1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890AB".
01 TAB1     REDEFINES TXT.
  02 ITEM1       PIC A(4)   OCCURS 12.

01 TAB2     REDEFINES TXT.
  02 TAB2-1
    03 TAB2-2

01 TAB3     REDEFINES TXT.
  02 TAB3-1                 OCCURS 2.
    03 TAB3-2               OCCURS 3.
      04 TAB3-3  PIC A(1)   OCCURS 8.

PROCEDURE DIVISION. START-RUN SECTION.
FIRST-PARAGRAPH.
  OPEN OUTPUT RDAT
    IF FS NOT = "00" GO TO END-RUN
  END-IF
  MOVE "ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890" TO FSEXT21.
END-RUN.
STOP RUN.
END PROGRAM      X-COB22.
```

Nach dem Laden des COBOL-Programms werden folgende AID-Kommandos eingegeben:

```
%AID LOW=OFF
%AID SYMCHARS=STD
%INSERT L'END-RUN'
%RESUME
```

Die folgenden Varianten zeigen die Wirkung verschiedener Angaben bei XFLAT und XMAX:

- XFLAT ohne Daten-Operand
- XFLAT mit Struktur DATABOX als Operand
- XMAX mit Substruktur G in DATABOX als Operand
- XFLAT mit Struktur TAB3 als Operand
- XFLAT und XMAX mit einem Array als Operand

XFLAT ohne Daten-Operand

Wenn Sie XFLAT ohne Operand angeben, werden von den Strukturen nur die oberste Ebene 01 ausgegeben, lange Strings, Arrays und Areas werden in verkürzter Form ausgegeben.

```
%SDUMP T=XFLAT
SRC_REF:     92STO     SOURCE: X-COB22     PROC: X-COB22
<CHARS(21),C>
_COMPILER     = |COBOL2000     V01.4B00     |

<CHARS(10),C>
_COMPILATION_DATE = |2015-03-30|

<CHARS(8),C>
_COMPILATION_TIME = |13:59:22|

<CHARS(30),C>
_PROGRAM_NAME    = |X-COB22     |

<CHARS(8),C>
_EBCDIC_CCSN     = |EDF03IRV|

<UNSIGN(4),D>
ZERO             =     0

<AREA(1),X>
HIGH-VALUE       = FF     ~

<AREA(1),X>
LOW-VALUE        = C1     A


<CHARS(1),C>
SPACE      = | |

<CHARS(1),C>
QUOTE      = |"|
```

```
<STRUCT(1)>
01     _LAST_EXCEPTION

<AREA(256),X>
MYSEQ      = 04050607 08090A0B 0C0D0E0F 10111213   ................    ETC

<FILE(1)> RDAT
<CHARS(54),C>
_FILE_NAME = |:RZV0:$PBELY.FILE.COBOL.DATA.001     |
<CLASS(1),S>
_OPEN_MODE = OPEN-OUTPUT
<CHARS(148),C>
_RECORD    =
|..ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890     | ...
<INT(4),D>
_RECORD_NO =     0

<STRUCT(148)>
01     RSATZ

<INT(4),D>
TALLY       =     +0

<INT(4),D>
RETURN-CODE =     +0

<STRUCT(6)>
01     FSEXT

<STRUCT(2)>
01     FS

<CHARS(8),C>
DAT         = |DATA.001|

<ZONED(2),D>
RELKY       =     0

<STRUCT(268)>
01     DATABOX

<CHARS(48),C>
TXT         = |1234567890ABCDEFGHIJKLMNOPQRSTUVWXYZ1234567890AB|

<STRUCT(48)>
01     TAB1

<STRUCT(48)>
```

```
01     TAB2

<STRUCT(48)>
01     TAB3
```

XFLAT mit Struktur DATABOX als Operand

Wenn Sie bei XFLAT eine Struktur angeben (hier DATABOX), wird zusätzlich die Ebene 02 der Struktur ausgegeben.

```
/%SDUMP databox t=xflat
SRC_REF:   92STO     SOURCE: X-COB22  PROC: X-COB22 ***********
<STRUCT(268)>
01        DATABOX
<INT(2),D>
02        BNR         =        +0
<INT(4),D>
02        INX         =        +0
<POINTER(4),X>
02        PPP         = FFFFFFFF
<UNSIGN(4),D>
02        C0          =        12345
<INT(4),D>
02        C0S         =        -12345
<UNSIGN(4),D>
02        C5          =        12345
<INT(4),D>
02        C5S         =        -12345
<BINARY(8),D>
02        C0V         =        12345.67890
<BINARY(8),D>
02        C0SV        =        -12345.67890
<BINARY(8),D>
02        C5V         =        12345.678900000
<BINARY(8),D>
02        C5SV        =        -12345.678900000
<PACKED(5),D>
02        C3          =        12345
<PACKED(5),D>
02        C3S         =        -12345
<DECIMAL(10),D>
02        C3V         =        12345.678900000
<DECIMAL(10),D>
02        C3SV        =        -12345.678900000
<FLOAT(4),E>
02        C1          = -.1234567 E+005
<FLOAT(8),E>
02        C2          = -.1234567889999999 E+005
```

```
<ZONED(20),D>
02        Z20         =         98765432101234567890
<ZONED(17),D>
02        Z16S        =         -5432101234567890
<ZONED(16),D>
02        Z16T        =         -5432100123456789
<CHARS(21),C>
02        D20C        = |-98765432101234567890|
<AREA(92),X>
02        NAT         = 00310032 00330034 00350036        123456        ETC
<STRUCT(92)>
02        G
```

XMAX mit Substruktur G in DATABOX als Operand

Mit XMAX für die Substruktur G in DATABOX wird die komplette Substruktur von G (Areas im Dump-Format) in voller Länge ausgegeben:

```
/%SD g in databox t=xmax
SRC_REF:    92STO  SOURCE: X-COB22  PROC: X-COB22  *********
<STRUCT (92) >
02          DATABOX.G
<AREA(20),X>
 03         G1              =
            00310032 00330034 00350036 00370038   12345678
            00390030   90
<AREA(52),X>
 03         G2              =
            00410042 00430044 00450046 00470048   ABCDEFGH
            0049004A 004B004C 004D004E 004F0050   IJKLMNOP
            00510052 00530054 00550056 00570058   QRSTUVWX
            0059005A   YZ
<AREA(20),X>
 03         G3              =
            00310032 00330034 00350036 00370038   12345678
            00390030   90
```

XFLAT mit Struktur TAB3 als Operand

```
/%SD tab3 t=xflat
SRC_REF:  92STO  SOURCE: X-COB22  PROC: X-COB22


<STRUCT(48)>
01          TAB3
<ARRAY(48),STRUCT>
 02         TAB3-1( 1: 2)
```

XFLAT und XMAX mit einem Array als Operand

XFLAT für einzelnes Array-Element:

```
/%SD tab3-1(2)  t=xflat
SRC_REF:  92STO  SOURCE: X-COB22  PROC: X-COB22  ***
<STRUCT(24)>
02       TAB3-1( 2)
<ARRAY(24),STRUCT>
 03      TAB3-2( 1: 3)
```

XFLAT für 2-dimensionales Array-Element:

```
/%SD tab3-2(2,3)  t=xflat
SRC_REF:  92STO  SOURCE: X-COB22  PROC: X-COB22  ***
<STRUCT(8)>
03  TAB3-1.TAB3-2( 2, 3)
<ARRAY(8),CHARS>
 04  TAB3-3( 1: 8)
```

XMAX für 3-dimensionales Array-Element mit Angabe eines Bereichs:

```
/%SD tab3-3(2,3,5:8)t=xmax
SRC_REF:  92STO  SOURCE: X-COB22  PROC: X-COB22  ***
<ARRAY(4),CHARS>
TAB3-1.TAB3-2.TAB3-3( 5: 8)
( 5) |9|  ( 6) |0|  ( 7) |A|  ( 8) |B|
```

%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, sodass die %SET-Anweisung bis auf später genannte Ausnahmen wie die COBOL-MOVE-Anweisung arbeitet.

– Mit *sender* bezeichnen Sie ein Datenfeld, eine Länge, eine Adresse, einen Durchlaufzähler, ein AID-Register, ein COBOL-Sonderregister, eine Figurative Konstante oder ein AID-Literal.
sender kann im virtuellen Speicher des geladenen Programms oder in einer Dump-Datei liegen.

– Mit *empfänger* bezeichnen Sie ein Datenfeld, einen Durchlaufzähler, ein AID-Register oder ein COBOL-Sonderregister, 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.

sender und *empfänger* können auch in der FILE SECTION oder SUB-SCHEMA SECTION definiert sein. Liegen sie in der LINKAGE SECTION, muss diese in der aktuellen Aufrufhierarchie enthalten sein.

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 noch nicht alle Einträge in der DATA DIVISION initialisiert sind (z.B. Datensatzbeschreibungen und Sonderregister).

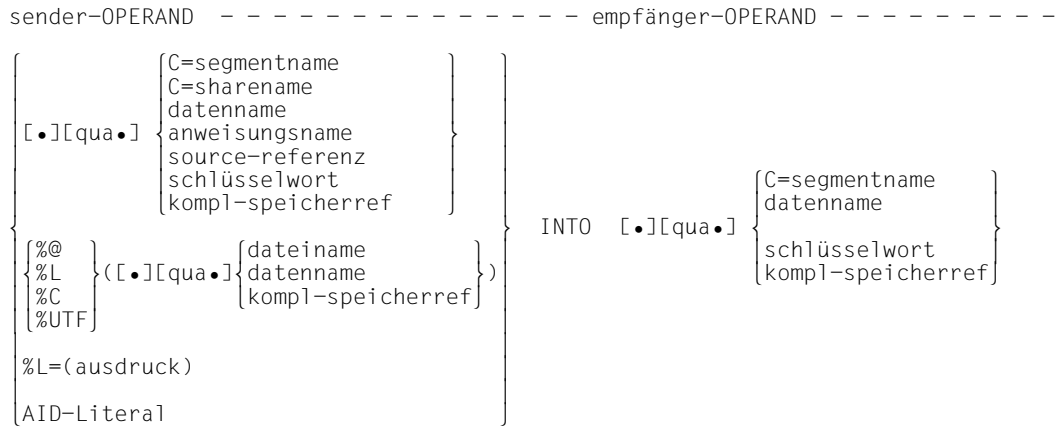
Neben den hier beschriebenen Operandenwerten können Sie auch die im Handbuch für das Testen auf Maschinencode-Ebene beschriebenen Operandenwerte einsetzen.

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 Datenfelder, COBOL-Sonderregister, Durchlaufzähler, Register oder eine komplexe Speicherreferenz angeben. Nur als *sender* können Sie einsetzen: Anweisungsnamen, Source-Referenzen, Figurative Konstanten, AID-Literale, sowie Adressen und Längen von Datenfeldern. *sender* kann sowohl im virtuellen Speicherbereich des geladenen Programms als auch in einer Dump-Datei liegen. *empfänger* dagegen kann nur im virtuellen Speicherbereich des geladenen Programms liegen.
Übertragen bzw. überschreiben Sie Programmbereiche mit Befehlscode, kann das zu unerwünschten Ergebnissen führen, wenn Adressen betroffen sind, die zu einem *control-* oder *trace-bereich* gehören oder auf die mit %INSERT ein Testpunkt gesetzt wurde (siehe AID-Basishandbuch [1]).



- 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 Qualifikationen geben Sie nur an, wenn ein Adressoperand nicht für den aktuellen AID-Arbeitsbereich gilt oder wenn eine Adresse angesprochen werden soll, die nicht in der aktuellen Übersetzungseinheit oder dem aktuellen Programm liegt.

E=VM für *empfänger*
geben Sie nur an, wenn für *sender* oder *empfänger* die aktuelle Basisqualifikation 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.

S=srcname
geben Sie nur an, wenn *sender* oder *empfänger* nicht in der aktuellen Übersetzungseinheit liegt.

PROC=program-id
geben Sie nur an, wenn Sie einen Datei-, Daten- oder Anweisungsnamen ansprechen, der nicht im aktuellen Programm liegt oder in der Übersetzungseinheit nicht eindeutig ist siehe Kapitel „COBOL-spezifische Adressierung“ auf Seite 17). Sie brauchen sie auch für einen globalen Datennamen, der lokal verdeckt ist.

Stimmt *srcname* in der S-Qualifikation mit *progam-id* überein, schreiben Sie nur die PROG-Qualifikation.

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.

Vor die anschließend aufgeführten C-Qualifikationen können Sie nur die Basisqualifikation bzw. die CTX-Qualifikation setzen. Mit der C-Qualifikation verlassen Sie die symbolische Ebene. Unmittelbar anschließend kann kein symbolischer Operand geschrieben werden (siehe Abschnitt „Symbolische Speicherreferenzen“ auf Seite 21), nur eine *kompl-speicherref*.

C=segmentname
Ohne Längenmodifikation geben Sie das ganze Segment als *sender* oder *empfänger* an.

C=sharename
Ohne Längenmodifikation geben Sie den gesamten Bindemodul als *sender* oder *empfänger* an.

datenname
ist der im Quellprogramm definierte Name einer Datengruppe oder eines Datenelements oder der Name eines COBOL-Sonderregisters. Figurative Konstanten können nur als *sender* eingesetzt werden.
datenname ist eine maximal 30stellige alphanumerische Zeichenfolge.

Datenelemente überträgt AID entsprechend den COBOL-MOVE-Regeln und berücksichtigt dabei die Definitionen aus dem Quellprogramm.
Datengruppen können mit %SET nur bearbeitet werden, wenn sowohl *sender* als auch *empfänger* als Datengruppen definiert sind. AID führt eine alphanumerische Übertragung durch und berücksichtigt dabei weder Struktur noch Datentyp-Definition.
Numerische und alphanumerische Empfangsfelder mit Druckaufbereitung können nur mit einem AID-Zeichen-Literal (C'...', X'...' oder B'...') modifiziert werden, dessen Inhalt bereits entsprechend druckaufbereitet ist.

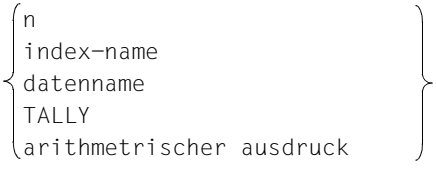
datenname [kennzeichnung][...][(index[,...])]

kennzeichnung

Ist *datenname* nicht eindeutig innerhalb eines Programms, kann er gekennzeichnet werden, indem er mit IN oder OF einer bestimmten Datengruppe zugeordnet wird. *datenname* muss mit so vielen Kennzeichnungen versehen sein, wie er für seine eindeutige Bezeichnung braucht.
Wird er nicht gekennzeichnet, gibt AID nur dann Daten für *datenname* aus, wenn es dafür eine Datendefinition auf der Stufe 01 oder 77 gibt; sonst folgt eine Fehlermeldung.

index

Ist *datenname* der Name eines Tabellenelementes, kann er wie in einer COBOL-Anweisung indiziert bzw. subskribiert werden; die Schreibweise unterscheidet sich gegenüber COBOL nur darin, dass mehrere Indizes durch Komma getrennt werden müssen. Wenn Sie den Namen eines Tabellenelementes ohne Index oder mit unvollständigem Index angeben, lehnt AID die Übertragung ab.
index kann folgendermaßen angegeben werden:



COBOL-Sonderregister

LINAGE-COUNTER
RETURN-CODE
SORT-CCSN
SORT-CORE-SIZE
SORT-EOW
SORT-FILE-SIZE

```
SORT-MODE-SIZE
SORT-RETURN
TALLY
```

Figurative Konstanten
können nur als *sender* angegeben werden, der Adressselektor kann auf sie nicht angewandt werden. Die Figurativen Konstanten HIGH-VALUE und LOW-VALUE repräsentieren immer den alphanumerischen Wert, der ihnen standardmäßig oder nach den Vereinbarungen mit der PROGRAM COLLATING SEQUENCE-Klausel entspricht. Im Gegensatz zur COBOL-MOVE-Anweisung wird bei der Verwendung einer *Figurativen Konstanten* im AID-Kommando %SET immer nur ein Zeichen übertragen.

```
ZERO
SPACE
HIGH-VALUE
LOW-VALUE
QUOTE
symbolic character
```

anweisungsname
bezeichnet die Adresse des ersten Befehls in einem Paragraphen oder Kapitel in der PROCEDURE DIVISION.

```
{L'kapitel'                          }
{L'paragraph' [IN L'kapitel']        }
```

Ist ein Paragraphenname innerhalb eines Programms nicht eindeutig, dann muss er mit dem Kapitelnamen des Kapitels, in dem er definiert wurde, gekennzeichnet werden: `L'paragraph' IN L'kapitel'`

Anweisungsnamen sind Adresskonstanten. Sie können nur als *sender* angegeben werden. Es wird die damit bezeichnete Adresse übertragen.
Mit nachfolgendem Pointer-Operator (*anweisungsname* ->) bezeichnen Sie 4 Bytes des Programmcodes, der zu der Anweisung generiert wurde. Für Zweibyte- bzw. Sechsbyte-Befehle müssen Sie eine entsprechende Längenmodifikation angeben. *anweisungsname* -> können Sie als *sender* und als *empfänger* verwenden. Siehe Beispiel 6 und 7.

source-referenz
bezeichnet die Adresse des ersten Befehls, der zu einer Anweisung in der PROCEDURE DIVISION generiert wurde, und muss in einem der folgenden Formate angegeben werden:

S'n'
: für Zeilen mit Paragraphen- oder Kapitelnamen, in denen kein COBOL-Verb auftritt. Diese Angabe ist für Programme nicht möglich, die übersetzt wurden mit: `STMT-REFERENCE=COLUMN1-TO-6`

S'nverb[m]' | S'xverb[m]'
: für Zeilen, in denen ein COBOL-Verb steht. *m* geben Sie nur an, wenn das gleiche COBOL-Verb mehrfach in einer Zeile steht.

Source-Referenzen sind Adresskonstanten. Sie können nur als *sender* angegeben werden. Es wird die damit bezeichnete Adresse übertragen.
Mit nachfolgendem Pointer-Operator (*source-referenz* ->) bezeichnen Sie 4 Bytes des Programmcodes, der zu der Anweisung generiert wurde. Für Zweibyte- bzw. Sechsbyte-Befehle müssen Sie eine entsprechende Längenmodifikation angeben. *source-referenz* -> können Sie als *sender* und als *empfänger* verwenden. Siehe Beispiel 6 und 7.

schlüsselwort
: ist ein Durchlaufzähler, der Befehlszähler oder ein Register. Vor *schlüsselwort* können Sie nur eine Basisqualifikation angeben.
Im AID-Basishandbuch [1], sind die impliziten Speichertypen der Schlüsselwörter angegeben.

```
%•[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 [1]):
– Adressversatz (•)
– indirekte Adressierung (->)
– Typmodifikation (%T(datenname), %X, %C, %E, %D, %P, %F, %A, %S, %SX, %UTF16)
– Längenmodifikation (%L(...), %L=(ausdruck), %Ln)
– Adressselektion (%@(...))
– Zeichenkonvertierungsfunktionen %C() und %UTF16()

Mit einer expliziten Typ- oder Längenmodifikation können Sie die Speichertypen von *sender* und *empfänger* einander anpassen. Eine Typmodifikation mit einem Speichertyp, der mit dem Speicherinhalt nicht vereinbar ist, wird von AID abgelehnt.
Beginnt eine *kompl-speicherref* mit *anweisungsname* oder *source-referenz*, muss

anschließend der Pointer-Operator (->) geschrieben werden. In diesem Fall muss *anweisungsname* mit L'...' angegeben werden. Ohne den Pointer-Operator (->) können *anweisungsname* und *source-referenz* überall da verwendet werden, wo auch Sedezimalzahlen geschrieben werden können.

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. Trotzdem bleiben die Bereichsgrenzen einer Ausgangsadresse (CSECT, *datenname*, Schlüsselwort etc.) wirksam. Sie dürfen für keinen Operanden in einer komplexen Speicherreferenz durch einen Adressversatz oder eine Längenmodifikation überschritten werden, sonst gibt AID eine Fehlermeldung aus. Erst durch die Verbindung von Adressselektor (%@) mit Pointer-Operator (->) wechseln Sie auf die Maschinencode-Ebene, auf der der Bereich den vom geladenen Programm belegten virtuellen Speicher umfasst.
Beispiel: `%SET CFELD.3%L5 INTO CFELD1`
Der Bereich von CFELD ist fünf Byte lang. Nach dem Adressversatz würde mit der Längenmodifikation %L5 der Bereich von CFELD um drei Byte überschritten. Das ist nicht zugelassen. Sollen mit dem %SET noch drei Byte im Anschluss an CFELD in CFELD1 übertragen werden, muss der %SET wie folgt beschrieben werden:
`%SET %@(CFELD)->.3%L5 INTO CFELD`

%@(...)

Mit dem Adressselektor können Sie die Anfangsadresse eines Dateieintrags, eines Datenfeldes, eines Sonderregisters oder einer komplexen Speicherreferenz als *sender* verwenden. Der Adressselektor liefert als Ergebnis eine Adresskonstante (siehe AID-Basishandbuch [1]).
Der Adressselektor lässt sich nicht auf Konstanten anwenden, dazu gehören auch die Anweisungsnamen, die Source-Referenzen und die Figurativen Konstanten.

%L(...)

Mit dem Längenselektor können Sie die Länge eines Dateieintrags, eines Datenfeldes oder eines Sonderregisters als *sender* verwenden.
Der Längenselektor liefert als Ergebnis eine Ganzzahl (siehe AID-Basishandbuch [1]).
Beispiel: `%SET %L(FELD1) INTO %OG`
Die Länge von FELD1 wird übertragen.

%L=(ausdruck)

Mit der Längenfunktion können Sie sich als *sender* einen Wert errechnen lassen. *ausdruck* wird gebildet aus dem Inhalt von Speicherreferenzen, Konstanten, Ganzzahlen und arithmetischen Operatoren. Nur ganzzahlige Speicherreferenz-Inhalte (Typ %F oder %A) sind zugelassen.
Die Längenfunktion liefert als Ergebnis eine Ganzzahl (siehe AID-Basishandbuch [1]).

Beispiel: `%SET %L=(FELD1) INTO %OG`
Der Inhalt von FELD1 wird übertragen, wenn er ganzzahlig ist (Typ %F). Sonst gibt AID eine Fehlermeldung aus.

%C(...) oder %UTF16(...)

Die Funktion wandelt Strings von einer 1-Byte-EBCDIC-Codierung in die UTF16-Codierung um oder umgekehrt.
Weitere Informationen siehe AID-Basishandbuch [1].

AID-Literal

Alle im AID-Basishandbuch [1] 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'| U'x...x'}`	Character-Literal
`X'f...f'`	Sedezimal-Literal
`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*								
	Festpkt., ext.Gleitpunktzahl, Subskript, Index, Sonderregister, %D	int. Gleitpunktzahl	Index[1]	alphabetisch	alphanumerisch %C	druckaufbereitet	%X	%UTF16, NATIONAL	stark typisiert
Festpunktzahl, externe Gleitpunktzahl, Subskript, Index, Sonderregister, %D	num	num	num	–	–	–*	–	–	–
interne Gleitpunktzahl ZERO, %F, %P, %A [{±}]n,#' f...f'	num	num	num	–	–	–*	bin	–	–
[{±}]n.m [{±}]mantE[{±}]exponent	num	num	–	–	–	–*	–	–	–
druckaufbereitet (alphabetisch, alpanumerisch)	–	–	–	char[a]	char	–*	–	char[e]	–
alphanumerisch, SPACE,QUOTE, %C	num[n]	num[n]	num[n]	char[a]	char	–*	bin	char[e]	–
numerisch druckaufber., HIGH- / LOW-VALUE	–	–	–	–	char	–	–	char[e]**	–
symbolischer character	num[n]	num[n]	num[n]	char[a]	char	–*	–	char[e]	–
C' x...x'	num[n]	num[n]	num[n]	char[a]	char	char	bin	char[e]	–
X' f...f' , B' b...b'	bin	bin	–	bin	bin	bin	bin	bin	bin
%X	–	bin	–	–	bin	–	bin	bin	–
%UTF16, U'x...x', NATIONAL	num[n]	num[n]	num[n]	char[ae]	char[e]***	–*	bin	char[g]	–
stark typisiert	–*	–*	–	–*	–*	–*	–	–	char[t]

* AID weist diese Übertragung mit einer Meldung ab; COBOL führt sie durch.

** Bei der Übertragung von HIGH-VALUE/LOW-VALUE findet eine Konvertierung nach NATIONAL statt; dies ist nicht COBOL-konform.

*** Die Übertragung ist in COBOL verboten.

bin Binäre Übertragung; linksbündig

sender < *empfänger* rechts wird mit binären Nullen aufgefüllt

sender > *empfänger* rechts wird abgeschnitten

Ganzzahlige numerische Literale entsprechen bei der Übertragung in %X einem Integerwert mit Vorzeichen in der Länge 4 Byte (%FL4), die binär übertragen werden.

char Character-Übertragung; linksbündig bzw. rechtsbündig, falls die Klausel JUSTIFIED RIGHT bei *empfänger* angegeben ist.

sender < *empfänger* gemäß JUSTIFIED-Klausel wird mit Leerzeichen '(...)' aufgefüllt

sender > *empfänger* gemäß JUSTIFIED-Klausel wird abgeschnitten

a Übertragen wird nur, wenn der Inhalt von *sender* alphabetisch ist.

e Konvertierung von/nach National/%UTF16.
Falls *sender* bzw. *empfänger* nicht vom Typ NATIONAL/%UTF16 sind, wird bei symbolischen COBOL-Feldern der EBCDIC-Codeset, der im COBOL-Programm vereinbart wurde, verwendet.
Bei allen anderen Feldern (von alten COBOL-Programmen, anderen Programmiersprachen oder Typ %C) wird die EBCDIC-Einstellung aus dem %AID-Kommando verwendet. Ist ein Zeichen im *sender*-Coded-Character-Set oder im *empfänger*-Coded-Character-Set unzulässig, so wird in den *empfänger* an der entsprechenden Zeichenposition das Ersatzzeichen '.' (Punkt) im Coded-Character-Set des *empfängers* übertragen, ohne dass eine AID-Meldung ausgegeben wird.

g Besitzt eine Gruppe das Attribut GROUP-USAGE NATIONAL, so verhält sich die Gruppe wie ein NATIONAL Feld.

t Nur wenn der *empfänger* vom gleichen Typ ist.

num Numerische Übertragung; werterhaltend

sender wird bei Bedarf in den Speichertyp von *empfänger* konvertiert.
Die Klausel SIGN LEADING/TRAILING [SEPARATE] wird berücksichtigt.

n Wenn *sender* vom Typ Character nur Ziffern enthält und höchstens 31 Stellen lang ist, führt AID eine numerische Übertragung durch. Enthält *sender* Zeichen vom Typ Character ungleich Ziffern, wird keine Übertragung ausgeführt.

1 In Index können nur Werte > 0 übertragen werden. AID führt die erforderliche Umrechnung von Tabellenplatznummer in Tabellenelement-Distanz und umgekehrt durch.

– keine Übertragung

AID meldet die Unverträglichkeit der Speichertypen.

Beispiele

In einem COBOL-Programm sind die folgenden Felder und Tabellen definiert:

```
01     ANZAHL-TAB.
   02  ZAHL              PIC S9(6) OCCURS 50 INDEXED BY J.

01     ANZ-SUMME         PIC S9(6).
01     PROZ-SUMME        PIC S999V99.
01     ZEICHEN           PIC X(10).
01     NATIONAL-Zeichen PIC N(10)
```

Für die folgenden Beispiele wurde mit `%AID CHECK=ALL` der Änderungs-Dialog eingeschaltet. So sehen Sie den Inhalt des Empfangsfelds vor und nach der Ausführung des %SET:

1. `%SET #'061' INTO ANZ-SUMME`

```
OLD CONTENT:
          1
NEW CONTENT:
         97
%  AID9274 Change desired? (Y=YES; N=NO)?
Y
```

   Zum selben Ergebnis führt folgendes Kommando:

   ```
   %SET 97 INTO ANZ-SUMME
   ```

2. `%QUALIFY PROG=UPRONUM`

   ```
   %SET .ANZ-SUMME INTO .ZAHL(16)
   ```

```
OLD CONTENT:
          0
NEW CONTENT:
         10
%  AID9274 Change desired? (Y=YES; N=NO)?
Y
```

3. `%SET 'ABCDEFG' INTO ZEICHEN`

```
OLD CONTENT:
|1234567890|
NEW CONTENT:
|ABCDEFG   |
%  AID9274 Change desired? (Y=YES; N=NO)?
Y
```

4. `%SET 123.45 INTO PROZ-SUMME`

```
OLD CONTENT:
   +0.00
NEW CONTENT:
 +123.45
%  AID9274 Change desired? (Y=YES; N=NO)?
Y
```

5. `%SET 123.45 INTO ZAHL(5)`

```
I390 WARNING: SOURCE TRUNCATED
OLD CONTENT:
     0
NEW CONTENT:
   123
%  AID9274 Change desired? (Y=YES; N=NO)?
Y
```

6. `%SET L'AUSGABE' INTO %0G`
   Die Adresse des ersten Befehls, der ab dem Paragraphen AUSGABE steht, wird in das AID-Register %0G geschrieben.

7. `%DA 5 FROM L'AUSGABE'->`
   `%SET L'AUSGABE'->%L2 INTO %1G`
   Mit dem %DISASSEMBLE lassen Sie sich den Befehlscode rückübersetzen, der ab der Adresse steht, die zum Paragraphen AUSGABE hinterlegt wurde. Der erste Befehl ist ein 2-Byte-Befehl.
   Dieser erste Befehl wird mit dem %SET in das AID-Register %1G geschrieben.

8. `%SET  ZEICHEN  INTO NATIONAL-ZEICHEN`
   `%SET  '{ä}'    INTO NATIONAL-ZEICHEN`
   Im ersten Fall wird die EBCDIC-Zeichenfolge aus Feld `ZEICHEN` in die UTF16-Codierung konvertiert (entspricht dem COBOL-Datentyp NATIONAL). Die konvertierte Zeichenfolge wird in das Feld `NATIONAL-ZEICHEN` übertragen. Der EBCDIC-Zeichensatz für `ZEICHEN` wird aus dem COBOL-Programm verwendet. Dadurch ist sichergestellt, dass AID und das COBOL-Programm die gleichen Konvertierungen durchführen.

   Im zweiten Fall wird das Literal '{ä}' nach der UTF16-Konvertierung in das Feld `NATIONAL-ZEICHEN` übertagen. Die Eingabe des Literals '{ä}' ist nur dann möglich, wenn die Terminal-Emulation den Coded-Character-Set UTFE unterstützt.

9. `%SET %UTF16(V'00' %CL3) INTO NATIONAL-ZEICHEN`
   `%SET ZEICHEN INTO NATIONAL-ZEICHEN`
   Die Funktion %UTF16() kann nur auf EBCDIC-Zeichenfolgen angewendet werden. Die Typmodifikation mit %C stellt sicher, dass die Speicheradresse V'00' auch als solche interpretiert wird.
   Beide %SET Kommandos konvertieren eine EBCDIC-Zeichenfolge, die im Speicher liegt, in eine UTF16-Zeichenfolge. Diese wird jeweils in `NATIONAL-ZEICHEN` gespeichert.
   AID verwendet beim Operand %UTF16(V'00' %CL3) den durch %AID EBCDIC eingestellten Zeichensatz. Beim Operand ZEICHEN verwendet AID den von COBOL2000 vorgegebenen Zeichensatz.
   Überprüfen Sie daher die eingestellten Zeichen mit %SHOW %AID. Der aktuell eingestellte EBCDIC-Zeichensatz wird angezeigt.
   %D _EBCDIC_CCSN zeigt den für das COBOL-Programm gültigen Zeichensatz.

10. `%SET NATIONAL-ZEICHEN INTO ZEICHEN`
    `%SET %C(V'00'%UTF16L6) INTO ZEICHEN`
    Beide %SET Kommandos konvertieren eine UTF16-Zeichenfolgen, die im Speicher liegt, in eine EBCDIC-Zeichenfolge und speichern sie in `ZEICHEN` ab.
    Im ersten Fall bestimmt das COBOL2000-Objekt den EBCDIC-Zeichensatz des Zielfeldes. Im zweiten Fall bestimmt das %AID Kommando den EBCDIC-Zeichensatz des Zielfeldes.

%SHOW

Mit %SHOW können Sie sich informieren über die aktuellen Vereinbarungen zu einzelnen AID-Kommandos, feststellen wie die letzte Eingabe eines Kommandos aussah und welches Kommando zuletzt eingegeben wurde. Außerdem können Sie über den Subkommadonamen das Kommando anfordern, in dem es definiert wurde oder sich eine Liste aller eingetragenen Subkommandonamen mit dem zugehörigen Kommandotyp ausgeben lassen. Entsprechend der Vereinbarung zur Groß-/Kleinschreibung im %AID-Kommando wird die Orginaleingabe des Kommandos wiedergegeben oder der Eingabestring ist in Großbuchstaben umgesetzt.

– Mit *show-ziel* geben Sie ein Kommando, einen Subkommandonamen oder ein AID-Schlüsselwort für alle aktuellen Subkommandos an.

```
Kommando          Operand
-------------------------------------------------------------------------------

%SH[OW]           [show-ziel]

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

%SHOW ohne Operand gibt Ihnen das unmittelbar vorher eingegebene AID-Kommando aus. Wurde für die Task noch kein AID-Kommando gegeben, erhalten Sie eine Fehlermeldung. Ein %SHOW für eins der nicht vorgesehenen Kommandos führt zum Syntaxerror. Das Kommando ist in Kommando- und Subkommandofolgen zugelassen.

%SHOW verändert den Programmzustand nicht.

show-ziel

bezeichnet ein AID-Kommando, ein bestimmtes Subkommando oder alle eingetragenen Subkommandos. In *show-ziel* können Sie die für dieses Kommando zugelassenen Kommandos auch in ihrer Kurzform angeben.

Kommando oder Subkommando	**Information**
%AID	Die geltenden aktuellen Einstellungen, der Kommandos %AID, %AINT, %BASE und die Version des geladenen AID.
%BASE	Die aktuellen Einstellungen für %BASE, %AINT und %SYMLIB, die TSN, TID sowie die Version des Betriebssystems der Typ des Rechners werden ausgegeben.

Kommando oder Subkommando	Information
%CCSN	Die Ausgabe des Kommandos geht immer auf SYSOUT und enthält folgende Informationen: – Character-Code-Set-Namen der Systemdateien – Character-Code-Set-Namen der aktivierten Ausgabedateien – alle aktuell gültigen Character-Code-Set-Namen im System
%C[ONTROL]	Für jeden angemeldeten %CONTROL wird der Eingabestring ausgegeben.
%D[IS]A[SSEMBLE]	Die aktuelle *anzahl* und *start*-Adresse (V'...') wird ausgegeben.
%F[IND]	Das eingegebene Kommando und gegebenenfalls die virtuelle Adresse des letzten Treffers werden ausgegeben.
%IN[SERT] [testpunkt]	Ohne die Angabe *testpunkt* werden alle aktiven Testpunkte ausgegeben. Sonst zeigt AID das eingegebene Kommando, wo *testpunkt* vereinbart wurde.
%ON	Für jedes aktive %ON-Kommando wird der Eingabestring ausgegeben.
%OUT	Die geltenden *medium-u-menge*-Werte für die über %OUT steuerbaren Kommandos werden ausgegeben.
%OUTFILE	Alle implizit oder explizit angemeldeten Ausgabedateien werden mit ihren Linknamen aufgelistet.
%QUALIFY	Das letzte %QUALIFY-Kommando wird ausgegeben.
%SYMLIB	Die angemeldeten Bibliotheken werden mit der zugehörigen Basisqualifikation und der TSN ausgegeben.
%TRACE	Die Defaultwerte der %TRACE-Operanden werden ausgegeben. Dabei wird berücksichtigt, ob der letzte %TRACE symbolisch oder auf Maschinencode-Ebene war. In Folgezeilen gibt AID aus, wieviel Befehle oder Anweisungen schon mit dem aktuellen %TRACE bearbeitet wurden und wie das letzte aktuelle %TRACE-Kommando aussah.
%.*	Die Namen aller aktiven Subkommandos werden mit dem Typ des AID-Kommandos ausgegeben, in dem sie definiert wurden
%.subkdoname	Das Kommando, in dem *subkdoname* definiert wurde, wird ausgegeben.

%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 auf welcher Stufe der Aufrufhierarchie das Programm unterbrochen wurde.

Wird das Kommando am Terminal oder aus einer Prozedurdatei eingegeben, so wird der Programmzustand nicht verändert, da das Programm bereits angehalten ist. 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'20EXI' <%D TAB; %STOP>
/%RESUME

TAB( 1: 9)
(  1) |Berta|  (  2) |Hansi|  (  3) |Anton|  (  4) |Erich|  (  5) |Ilona|
(  6) |Diane|  (  7) |Carlo|  (  8) |Franz|  (  9) |Georg|
STOPPED AT SRC_REF: 20EXI , SOURCE: SORT , PROC: SORT
```

Mit %INSERT wird ein Testpunkt auf die Anweisung EXIT aus der Zeile 20 gesetzt. Das Subkommando enthält die Kommandos %DISPLAY und %STOP. Nach der Ausgabe von TAB hält AID das Programm an und schreibt eine STOP-Meldung mit Source-Referenz und Programm 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 einem Programm 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 Basisqualifikation dazu angeben.

Kommando	Operand
%SYMLIB	[qualifikation-u-lib][,...]

Bei Ausführung dieses Kommandos stellt AID nur fest, ob die angegebene Bibliothek eröffnet 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 Basisqualifikation 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 mit %DUMPFILE 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 abgemeldet wird, durch:
– ein neues %SYMLIB-Kommando zur selben Basisqualifikation
– ein %SYMLIB ohne Operanden
– ein %DUMPFILE-Kommando mit dem die Datei geschlossen wird

oder durch /LOGOFF bzw. /EXIT-JOB.

Enthält ein neues Kommando neue Dateinamen, dann werden diese Bibliotheken angemeldet und geöffnet.

%SYMLIB verändert den Programmzustand nicht.

qualifikation-u-lib

Ist eine Basisqualifikation und/oder der Dateiname einer PLAM-Bibliothek.

– Geben Sie eine Basisqualifikation und einen Dateinamen an, meldet AID die angegebene Bibliothek zu dieser Basisqualifikation an und öffnet sie. Bisher angemeldete Bibliotheken zu derselben Basisqualifikation werden abgemeldet.
– Geben Sie nur einen Dateinamen an, meldet AID die Bibliothek zur gerade eingestellten Basisqualifikation an (siehe %BASE) und öffnet sie. Alle zur aktuellen Basisqualifikation angemeldeten Bibliotheken werden abgemeldet.
– Geben Sie nur eine Basisqualifikation an, werden alle dazu angemeldeten Bibliotheken abgemeldet.

AID kann maximal 15 Bibliotheks-Anmeldungen verwalten. Dabei zählt eine Bibliothek, die gleichzeitig mit verschiedenen Basisqualifikationen 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 Basisqualifikation 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, %DUMPFILE).

dateiname
: ist der BS2000-Katalogname einer PLAM-Bibliothek. Sie wird zu der explizit oder mit *vorqualifikation* angegebenen Basisqualifikation angemeldet. Ohne die Angabe einer Qualifikation wird sie zur gerade eingestellten Basisqualifikation angemeldet.

Beispiel

```
%SYMLIB E=D5.PLAMLIB,COBOLOUTPUT
```

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 COBOLOUTPUT wird zur aktuell eingestellten Basisqualifikation 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'} 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, wie viele COBOL-Anweisungen maximal verfolgt, d.h. vor ihrer Ausführung protokolliert werden sollen.

– Mit *fortsetzung* legen Sie fest, ob das Programm nach Beendigung des %Trace anhält oder ohne Protokollierung weiterläuft.

– Mit *kriterium* wählen Sie die Typen von COBOL-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-Kommmando enthält, wurde ausgeführt.
– Ein %INSERT endet mit einer Programmunterbrechung, weil der *steuerung*-Operand K oder S lautet.
– Die K2-Taste wurde gedrückt (siehe Abschnitt „Kommandos zu Beginn einer Testsitzung“ auf Seite 15).

Beendet wird der %TRACE dagegen durch foldende 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 10 für *anzahl* und das Programm, in dem die aktuelle Unterbrechungsstelle liegt, für *trace-bereich* ein.

Mit %OUT können Sie steuern, welche Informationen eine Protokollzeile enthalten 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 verändert den Programmzustand.

anzahl

gibt an, wie viele COBOL-Anweisungen vom Typ *kriterium* maximal vor ihrer Ausführung 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 welchem Programm die aktuelle Unterbrechungstelle liegt.

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	***subkdo* wird bearbeitet vor**
%STMT	jeder COBOL-Anweisung
%ASSGN	COBOL-Anweisungen, die den Inhalt eines Datenfeldes verändern: ADD [CORRESPONDING], COMPUTE, DIVIDE, INITIALIZE, INSPECT, MOVE [CORRESPONDING], MULTIPLY, SET, STRING, SUBTRACT [CORRESPONDING], UNSTRING
%CALL	CALL-, CANCEL-, INVOKE-, PERFORM-Anweisungen sowie vor SORT/MERGE- Anweisungen, da diese eine INPUT oder OUTPUT PROCEDURE aufrufen können.
%COND	EVALUATE-, IF-, SEARCH-Anweisungen und den bedingt durchlaufenen THEN-, ELSE-, WHEN-Anweisungszweigen.
%DB	COBOL-Anweisungen zum Aufruf einer Datenbank: CONNECT, DISCONNECT, ERASE, FETCH, FIND, FINISH, FREE, GET, KEEP, MODIFY, READY, STORE
%EXCEPTION	den bedingten Anweisungs-Zweigen bzw. ihren zulässigen Negationen: AT END, AT END OF PAGE, INVALID KEY, ON SIZE ERROR, ON OVERFLOW, ON EXCEPTION, der RAISE-Anweisung sowie vor der Ausführung einer USE PROCEDURE
%GOTO	ALTER-, CONTINUE-, GOTO-, RESUME-Anweisungen
%IO	COBOL-Anweisungen, die IO-Operationen veranlassen ACCEPT, DISPLAY, OPEN, CLOSE, DELETE, READ, REWRITE, START, WRITE, GENERATE, INITIATE, TERMINATE
%LAB	COBOL-Anweisungen, die einen Kapitel- oder Paragraphen-namen haben oder ihm unmittelbar folgen
%PROC	Programm- oder Modul-Start am Beginn der PROCEDURE DIVISION oder beim ENTRY Programm- oder Modul-Ende durch die Anweisung STOP RUN, GOBACK, EXIT METHOD oder EXIT PROGRAM

Tabelle 3: *kriterium*-Vereinbarung für die Bearbeitung von *subkdo*

kriterium	*subkdo* **wird bearbeitet vor**
%SORT	MERGE- und SORT-Anweisungen, RELEASE- und RETURN-Anweisungen

Tabelle 3: *kriterium*-Vereinbarung für die Bearbeitung von *subkdo*

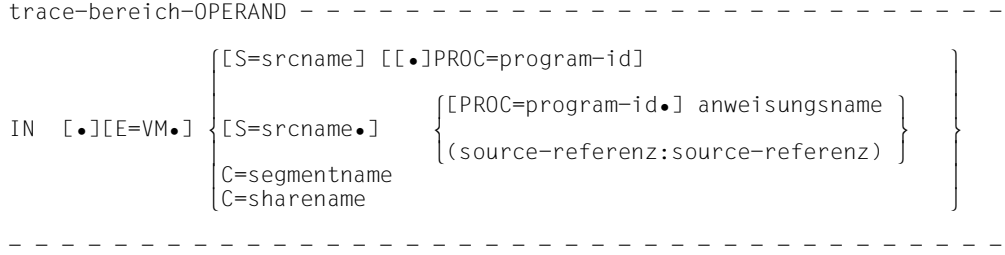
trace-bereich

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. *trace-bereich* kann nur im geladenen Programm liegen und ein angegebenes Programm muss zum Zeitpunkt der Eingabe des %TRACE bzw. bei Abarbeitung des Subkommandos, in dem der %TRACE enthalten ist, geladen sein.

trace-bereich ist bei einem Programm ohne Segmentierung auf eine Übersetzungseinheit beschränkt, bei einem Programm mit Segmentierung auf ein Segment. Die Beschränkung auf ein Segment gilt nur für unabhängige Segmente (Segment-Nr. ≥ 50).

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 Übersetzungseinheit bzw. das Segment, in der die aktuelle Unterbrechungsstelle liegt.

Die Fortsetzungsadresse für den Programmablauf kann mit %TRACE nicht beeinflusst werden, dazu müssen Sie ein %JUMP-Kommando verwenden.



- 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 Basisqualifikation eine Dump-Datei vereinbart ist (siehe %BASE).

S=srcname

geben Sie an, wenn *trace-bereich* nicht in der aktuellen Übersetzungseinheit liegen soll.

PROC=program-id

geben Sie nur an, wenn *trace-bereich* nicht im aktuellen Programm liegen soll, wenn er mit *anweisungsname* festgelegt werden soll und dieser in der Übersetzungseinheit nicht eindeutig ist oder um eine bisher geltende *trace-bereich*-Vereinbarung zu überschreiben.
Endet *trace-bereich* mit einer PROC/PROG-Qualifikation, so umfasst er das gesamte angegebene Programm. Diese muss zum Zeitpunkt der Eingabe des %TRACE bzw. bei Abarbeitung des Subkommandos, in dem der %TRACE steht, geladen sein.

Stimmt der Name in der S-Qualifikation mit *program-id* überein, schreiben Sie nur die PROG-Qualifikation.

Obwohl Sie mit den folgenden C-Qualifikationen auf die Maschinencode-Ebene wechseln, können Sie im Anschluss daran nur ein *kriterium* aus der vorhergehenden Tabelle auswählen bzw. AID setzt den Standardwert %STMT ein.

C=segmentname

Mit dieser Angabe erklären Sie das bezeichnete Segment zum *trace-bereich*. Sie ist nur erforderlich, wenn die Unterbrechungsstelle nicht in diesem Segment liegt oder wenn eine bisher geltende Bereichseinschränkung auf Teile dieses Segmentes aufgehoben werden soll.

C=sharename

Mit dieser Angabe erklären Sie den bezeichneten Bindemodul zum *trace-bereich*. Diese Angabe ist nur dann erforderlich, wenn sich die Unterbrechungsstelle nicht in dem angegebenen Bindemodul befindet oder wenn eine für diesen Bindemodul geltende Bereichseinschränkung aufgehoben werden soll.

anweisungsname

trace-bereich wird durch einen Anweisungsnamen festgelegt und umfasst einen Paragraphen oder ein Kapitel in der PROCEDURE DIVISION.

Ein alphanumerischer Kapitel- oder Paragraphenname kann auch ohne L'...' angegeben werden, da in diesem Kommando eine Verwechslung mit einem Datennamen nicht möglich ist.

Ist ein Paragraphenname innerhalb eines Programms nicht eindeutig, dann muss er mit dem Kapitelnamen des Kapitels, in dem er definiert wurde, gekennzeichnet werden: `L'paragraph' IN L'kapitel'`

(source-referenz : source-referenz)

trace-bereich wird durch die Angabe einer Anfangs- und Endadresse festgelegt. Beide müssen innerhalb derselben Übersetzungseinheit liegen, und es gilt: Anfangsadresse ≤ Endadresse.
Soll *trace-bereich* nur eine Anweisungszeile umfassen, müssen Anfangs- und Endadresse gleich sein.
Eine Begrenzung von *trace-bereich* auf einzelne COBOL-Verben innerhalb einer Zeile ist nicht möglich.

source-referenz

bezeichnet die Adresse des ersten Befehls, der zu einer Anweisung in der PROCEDURE DIVISION generiert wurde, und muss in einem der folgenden Formate angegeben werden:

S'n'

für Zeilen mit Paragraphen- oder Kapitelnamen, in denen kein COBOL-Verb auftritt. Diese Angabe ist für Programme nicht möglich, die mit `STMT-REFERENCE=COLUMN1-TO-6` übersetzt wurden.

S'nverb[m]' | S'xverb[m]'

für Zeilen, in denen ein COBOL-Verb steht.

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 [1]).

Ein %TRACE-Protokoll mit Zusatzinformationen (T=MAX) enthält die Nummer und den Typ der Anweisung, die ausgeführt wurde. Außerdem werden Kapitel- und Paragraphennamen 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          I2       LABEL
       50MOV                ASSIGN
       51ADD                ASSIGN
STOPPED AT SRC_REF:  51ADD,  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 COBOL-Anweisungen verfolgen. Nach der dritten Anweisung kommt die Abschlussmeldung für diesen %TRACE: Die Anweisung 51ADD steht in der Programmeinheit BEISPIEL, der Ladeeinheit hat denselben Namen.

2.

```
/%OUT %T T=MIN
/%T 3
       49          I2
       50MOV
       51ADD
STOPPED AT SRC_REF:  51ADD,  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 COBOL-Programm gezeigt. Anhand dieser Testsitzung können Sie die Anwendung und Wirkung einiger AID-Kommandos nachvollziehen; die Vorgehensweise ist bewusst einfach gehalten.

Der Compiler wurde mit dem folgenden SDF-Kommando aufgerufen:

```
/START-COBOL2000-COMPILER SOURCE=COB.S.SRCDAT,TEST-SUPPORT=AID-
/(PREPARE-FOR-JUMPS=YES),-
/LISTING=PARAM(NAME-INFORMATION=YES(SUPPRESS-GENERAT=AT-SEVERE-ERR),-
/OUTPUT=*SYSLST),COMPILER-ACTION=MODULE-GEN(MOD-FORM=OM,-
/SUPPR-GEN=AT-SEVERE-ERR),-
/RUNTIME-OPTIONS=PARAM(ACCEPT-STMT-INPUT=UPPERCASE-CONVERTED)
```

6.1 Quellprogrammliste

```
00001        IDENTIFICATION DIVISION.
00002        PROGRAM-ID.  MOBS.
00003        ENVIRONMENT DIVISION.
00004        CONFIGURATION SECTION.
00005        SPECIAL-NAMES.
00006            TERMINAL IS T.
00007        INPUT-OUTPUT SECTION.
00008        FILE-CONTROL.
00009             SELECT TEXTDAT ASSIGN TO "EINDAT".
00010        DATA DIVISION.
00011        FILE SECTION.
00012        FD TEXTDAT
00013                RECORD VARYING FROM 1 TO 256 DEPENDING ON SLF.
00014        01 SATZ.
00015             02 FELD PIC X OCCURS 1 TO 256 DEPENDING ON SLF
00016                INDEXED BY K.
00017        WORKING-STORAGE SECTION.
00018        77 SLF PIC 999 COMP.
00019        77 VERARB-SCHALTER PIC X.
00020           88 VERARB-ENDE VALUE "1".
00021        01 A-Z-TAB.
00022           02 FILLER PIC X(26) VALUE "ABCDEFGHIJKLMNOPQRSTUVWXYZ".
00023        01 ABC-TAB REDEFINES A-Z-TAB.
00024           02 ZEICHEN PIC X OCCURS 26 INDEXED BY I.
00025        01 ANZAHL-TAB.
00026           02 ZAHL PIC 9999 OCCURS 26 INDEXED BY J.
00027        77 ANZ-SUMME PIC S9(6) VALUE ZERO.
00028        77 PROZ-SUMME PIC S999V99 VALUE ZERO.
```

```
00029        01 AUS-KOPF.
00030           02 FILLER PIC X(24) VALUE "BUCHSTABE ANZAHL PROZENT".
00031        01 AUS-ZEILE.
00032           02 BUCHSTABE PIC X.
00033           02 FILLER PIC X(9) VALUE SPACE.
00034           02 ANZAHL PIC Z(5)9.
00035           02 FILLER PIC X(2) VALUE SPACE.
00036           02 PROZENT PIC ZZ9.99.
00037        01 AUS-FUSS.
00038           02 FILLER PIC X(10) VALUE "GESAMT:   ".
00039           02 A-SUMME PIC Z(5)9.
00040           02 FILLER PIC X(2) VALUE SPACE.
00041           02 P-SUMME PIC ZZ9.99.
00042        PROCEDURE DIVISION.
00043        VORLAUF.
00044            INITIALIZE ANZAHL-TAB.
00045            MOVE "0" TO VERARB-SCHALTER.
00046            OPEN INPUT TEXTDAT.
00047        VERARBEITUNG.
00048            READ TEXTDAT
00049               AT END DISPLAY "DATEI IST LEER" UPON T
00050               NOT AT END
00051                  PERFORM WITH TEST BEFORE UNTIL VERARB-ENDE
00052                     PERFORM WITH TEST BEFORE VARYING K FROM 1 BY 1 UNTIL
00053                        K > SLF
00054                        IF FELD(K) NOT = SPACE
00055                           THEN ADD 1 TO ANZ-SUMME
00056                           SET I TO 1
00057                           SEARCH ZEICHEN VARYING J
00058                              WHEN FELD(K) = ZEICHEN(I) ADD 1 TO ZAHL(J)
00059                           END-SEARCH
00060                        END-IF
00061                     END-PERFORM
00062                     READ TEXTDAT
00063                        AT END SET VERARB-ENDE TO TRUE
00064                     END-READ
00065                  END-PERFORM
00066               END-READ.
00067        AUSGABE.
00068            CLOSE TEXTDAT.
00069            DISPLAY AUS-KOPF UPON T.
00070            PERFORM WITH TEST BEFORE
00071               VARYING I FROM 1 BY 1 UNTIL I > 26
00072               MOVE ZEICHEN(I) TO BUCHSTABE
00073               SET J TO I
00074               MOVE ZAHL(J) TO ANZAHL
00075               COMPUTE PROZENT = ZAHL(J) * 100/ANZ-SUMME
00076               DISPLAY AUS-ZEILE UPON T
00077               COMPUTE PROZ-SUMME = PROZ-SUMME + ZAHL(J) * 100/ANZ-SUMME
00078            END-PERFORM.
00079            MOVE PROZ-SUMME TO P-SUMME.
00080            MOVE ANZ-SUMME TO A-SUMME.
00081            DISPLAY AUS-FUSS UPON T.
00082            STOP RUN.
00083
```

6.2 Inhalt der Eingabedatei

```
DIES IST EINE DATEI, DIE ALS EINGABE DIENT FUER EIN PROGRAMM,
DAS DIE HAEUFIGKEIT VON BUCHSTABEN BESTIMMT.
DIE GESAMTANZAHL IST DIE ANZAHL ALLER VON EINEM LEERZEICHEN VERSCHIEDENEN ZEICHEN
(EINSCHLIESSLICH ZIFFERN UND SONDERZEICHEN).
ABCDEFGHIJKLMNOPQRSTUVWXYZ
NUN NOCH EIN PAAR NORMALE SAETZE
ANABELL WAR AUCH IN DER ALHAMBRA
BABETTE BEMALTE BEIM BAECKER DIE BALUSTRADE
CAESAR CHECKTE SICH EIN NACH CHICAGO
```

6.3 Testablauf

1. Schritt

```
/SET-FILE-LINK LINK-NAME=EINDAT, FILE-NAME=EIN
/START-EXECUTABLE-PROGRAM FROM-FILE = MOBS
/MOD-MSG-ATTR TASK-LANG=D
%  BLS0517 MODULE 'MOBS' GELADEN
9089 UNTERBRECHUNGS-CODE= 60  BEI PC= 00000C46 , UEBERSETZUNGSEINHEIT MOBS
%  EXC0732 ABNORMALE PROGRAMM-BEENDIGUNG. FEHLERCODE 'NRT0101': /HELP-MSG NRT0101
```

Die Eingabedatei EIN wird dem Programm zugewiesen. Das Programm MOBS wird gestartet. Wegen eines Datenfehlers (EC=60) wird der Programmlauf abgebrochen.

2. Schritt

```
/SET-FILE-LINK LINK-NAME=BLSLIB00,FILE-NAME=$.SYSLNK.CRTE
/SET-FILE-LINK LINK-NAME=EINDAT,FILE-NAME=EIN
/LOAD-EXE (BIBLIO,MOBS),DBL-PARA=(RESOLUTION=(ALT-LIB=YES)),TEST-OPTIONS=*AID
%  BLS0517 MODULE 'MOBS' GELADEN
```

Das Programm wird wieder geladen; diesmal mit LSD-Informationen. Es wird nicht sofort gestartet, damit erst AID-Kommandos eingegeben werden können.

```
/%ON %ANY;%RESUME
```

Mit dem %ON %ANY soll sichergestellt werden, dass vor Beendigung des Programms noch AID-Kommandos eingegeben werden können. Anschließend wird das Programm mit %RESUME gestartet.

```
9089 UNTERBRECHUNGS-CODE= 60  BEI PC= 00000C46 , UEBERSETZUNGSEINHEIT MOBS
STOPPED AT V'F4DCBC' = IT0TRM@ + #'2C'
                             , EVENT: TERM (ABNORMAL,STEP,NODUMP)
```

Das Programm läuft auf den bekannten Fehler.

```
/%display %hlloc(v'c46')
*** TID: 00070132 *** TSN: 6DMA ********************************************
CURRENT PC: 00F4DCBC    CSECT: IT0TRM@  ************************************
V'00000C46' = CONTEXT  :  LOCAL#DEFAULT
              SMOD     :  MOBS
              PROC     :  MOBS
              SECTION  :
              PARAGRAPH:  VERARBEITUNG
              SRC-REF  :      58ADD
```

Mit dem %DISPLAY wird festgestellt, an welcher symbolischen Adresse der Datenfehler auftritt. Es ist die Sourcereferenz S’58ADD’.

```
/%display zahl(j)
%  AID0379 S- und PROC-Qualifikation ist notwendig oder die LSD-Information fehlt
```

Im Programm soll hier, nachdem das Zeichen aus dem Eingabesatz in A-Z-TAB entsprechend zugeordnet werden konnte, der Zähler für diesen Buchstaben um 1 erhöht werden. Ein %DISPLAY auf den in der COBOL-Anweisung ADD indiziert angesprochenen Tabellenplatz weist AID ab. Wie schon aus der vorhergehenden Ausgabe zur Highlevellocation (%HLLOC) der aktuellen Unterbrechungsstelle zu sehen war, liegt diese im Modul IT0TRM@ des Runtimesystems. Deshalb müssen symbolische Adressen im Programm MOBS in AID-Kommandos qualifiziert werden. AID braucht hier die S- und PROC-Qualifikation bzw. für die kurze PROGRAMM-ID ’MOBS’ kann die PROG-Qualifikation verwendet werden.

```
/%d prog=mobs.zahl(j)
%  AID0400 Dimension 01 des Feldes ZAHL gibt es nicht oder das Feld hat kein Elem
```

Aus der neuen Fehlermeldung ist zu erkennen, dass der Index J für Zahl einen falschen Wert hat. Auch AID kann mit dieser Adresse nicht arbeiten.

```
/%qualify prog=mobs
```

Um in weiteren AID-Kommandos nicht immer wieder die Qualifikation schreiben zu müssen, wird die Vorqualifikation vereinbart. In nachfolgenden Kommandos braucht nun vor eine symbolische Adresse an Stelle der Qualifkation nur der Punkt gesetzt zu werden.

```
/%d .j
J                =        +42
```

AID gibt den Inhalt von J aus. Der Index darf maximal den Wert 26 haben; in J steht aber 42. Das führte zum Datenfehler.

```
/%d .feld(k)
FELD( 6)         = |I|
```

Der Datenfehler trat bei der Bearbeitung des sechsten Zeichens aus dem Eingabesatz auf.

```
/%d .k,.i
K                =        +6
I                =        +9
```

Index I hat den Wert 9. Er steht richtig auf dem Platz des Buchstaben ’I’ im Alphabet.

```
/%d .satz
01       SATZ
 02      FELD(1:61)
          (  1) |D| (  2) |I| (  3) |E| (  4) |S| (  5) | | (  6) |I|
          (  7) |S| (  8) |T| (  9) | | ( 10) |E| ( 11) |I| ( 12) |N|
          ( 13) |E| ( 14) | | ( 15) |D| ( 16) |A| ( 17) |T| ( 18) |E|
          ( 19) |I| ( 20) |,| ( 21) | | ( 22) |D| ( 23) |I| ( 24) |E|
          ( 25) | | ( 26) |A| ( 27) |L| ( 28) |S| ( 29) | | ( 30) |E|
          ( 31) |I| ( 32) |N| ( 33) |G| ( 34) |A| ( 35) |B| ( 36) |E|
          ( 37) | | ( 38) |D| ( 39) |I| ( 40) |E| ( 41) |N| ( 42) |T|
          ( 43) | | ( 44) |F| ( 45) |U| ( 46) |E| ( 47) |R| ( 48) | |
          ( 49) |E| ( 50) |I| ( 51) |N| ( 52) | | ( 53) |P| ( 54) |R|
          ( 55) |O| ( 56) |G| ( 57) |R| ( 58) |A| ( 59) |M| ( 60) |M|
          ( 61) |,|
```

Der ganze Eingabesatz wird mit %DISPLAY ausgegeben. AID bereitet ihn entsprechend der Definition im Quellprogramm als Tabelle auf.

3. Schritt

```
/LOAD-P *MOD(BIBLIO,MOBS,RUN-MODE=ADVANCED(ALT-LIB=YES)),TEST-OPTIONS=AID
%  BLS0500 PROGRAM 'MOBS' GELADEN
/%TRACE IN VERARBEITUNG
    47        VERARBEITUNG
    48REA                          I-O-ACCESS
    51PER                          EXCEPT.DEP, THEN     , CALL     , LOOP INIT
    52PER                          CALL     , LOOP INIT
    54IF                           IF
    55ADD                          THEN     , ASSIGN
    56SET                          ASSIGN
    57SEA                          CASE       , LOOP INIT
    58ADD                          WHEN/OTHERS, ASSIGN
    54IF                           IF

STOPPED AT SRC_REF: 54IF    SOURCE: MOBS     PROC: MOBS
```

Das Programm wird neu geladen. Der %TRACE im Paragraphen VERARBEITUNG zeigt, in welchem Zusammenhang die ADD-Anweisung steht.

4. Schritt

```
/%control1 %assgn in Verarbeitung <con1: %d feld(k),i,j>
```

Wenn im Paragraphen VERARBEITUNG der Inhalt eines Datenfeldes verändert wird, soll das Subkommando CON1 ausgeführt werden. Dann wird das zu bearbeitende Zeichen aus dem Eingabesatz und der Stand der Indizes I und J ausgegeben.

```
/%in s'54if' <ins1: (%.con1 gt 10): %stop>
```

Nachdem das Subkommando CON1 10 mal ausgeführt wurde, soll der Ablauf unterbrochen werden.

```
/%in s'58add' <ins2: (j gt 26): %stop>
/%r
```

Bevor die Addition in ZAHL(J) ausgeführt wird, überprüft AID, ob der Index J einen zulässigen Wert hat. Wird er zu hoch, unterbricht AID das Programm.
Nach Eingabe des %INSERT wird das Programm gestartet.

```
*** TID: 0009027E *** TSN: 634R ************************************************
SRC_REF:     55ADD   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 1)        = |D|
I               =        +1
J               =        +1
SRC_REF:     56SET   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 1)        = |D|
I               =        +1
J               =        +1
SRC_REF:     58ADD   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 1)        = |D|
I               =        +4
J               =        +4
SRC_REF:     55ADD   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 2)        = |I|
I               =        +4
J               =        +4
SRC_REF:     56SET   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 2)        = |I|
I               =        +4
J               =        +4
SRC_REF:     58ADD   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 2)        = |I|
I               =        +9
J               =       +12
SRC_REF:     55ADD   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 3)        = |E|
I               =        +9
J               =       +12
SRC_REF:     56SET   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 3)        = |E|
I               =        +9
J               =       +12
SRC_REF:     58ADD   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 3)        = |E|
I               =        +5
J               =       +16
SRC_REF:     55ADD   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 4)        = |S|
I               =        +5
J               =       +16
SRC_REF:     56SET   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 4)        = |S|
I               =        +5
J               =       +16

STOPPED AT SRC REF: 58ADD    SOURCE: MOBS     PROC: MOBS
(INS2)
```

Aus dem AID-Protokoll des Subkommandos CON1 ist zu sehen, dass die Bearbeitung des ersten Zeichens aus der Eingabedatei richtig läuft. Die Indizes I und J laufen parallel. Vom zweiten Zeichen an beginnt J schneller zu wachsen. Beim vierten Buchstaben steigt der Index auf 34. Bevor es bei der Sourcereferenz S’58ADD’ wieder zum Datenfehler kommt, wird das bedingte Subkommando INS2 ausgeführt, durch das der Ablauf unterbrochen wird. Für den Buchstaben ’S’ steht der Index I richtig auf Platz 19 der Tabelle A-Z-TAB. Aus dem Verlauf ist zu sehen, dass für die Bearbeitung eines Zeichens der Index I auf den Anfangswert 1 zurückgesetzt wird, der Index J aber nicht.

```
/%set i into j
/%r
```

Der Index J wird auf den Inhalt von I gesetzt und das Programm fortgesetzt.

```
 FELD( 4)        = |S|
 I               =       +19
 J               =       +19

STOPPED AT SRC_REF: 54IF    SOURCE: MOBS    PROC: MOBS
(INS1)
```

Das Subkommando CON1 ist noch wirksam, und der zugehörige Durchlaufzähler wurde nicht verändert, und deshalb wurde der %STOP im Subkommando INS1 wieder ausgeführt.

```
/%remove %.ins1;%resume
```

Das Subkommando INS1 wird gelöscht und das Programm mit %RESUME fortgesetzt.

```
SRC_REF:     55ADD   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 6)        = |I|
I               =       +19
J               =       +19
SRC_REF:     56SET   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 6)        = |I|
I               =       +19
J               =       +19

STOPPED AT SRC_REF: 58ADD    SOURCE: MOBS     PROC: MOBS
(INS2)
```

Wieder war die Bedingung (J größer 26) für das Subkommando INS2 erfüllt und das Programm wurde angehalten.

```
/%d feld(k),i,j
SRC_REF:     58ADD   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 6)        = |I|
I               =        +9
J               =       +27
```

Der Index J ist wieder zu hoch. Vor Ausführung der COBOL-Anweisung SEARCH wird zwar der Index I wieder auf 1 gesetzt, aber es wurde vergessen auch den Index J mit dem Anfangswert zu versehen.

```
/%insert s'56set' <ins3: %set 1 into j>
/%set i into j;%r
```

Bevor die SEARCH-Anweisung S’57SEA’ ausgeführt wird, soll jetzt durch den neuen %INSERT auch der Index J mit dem richtigen Anfangswert versehen werden. Damit die Anweisung S’58ADD’, vor der das Programm durch den %STOP im Subkommando INS2 unterbrochen wurde, richtig ausgeführt wird, wird mit dem einzelnen %SET der Wert von Index I in J übertragen.

```
FELD( 6)        = |I|
I               =        +9
J               =        +9
SRC_REF:     55ADD   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 7)        = |S|
I               =        +9
J               =        +9
SRC_REF:     56SET   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 7)        = |S|
I               =        +9
J               =        +1
SRC_REF:     58ADD   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 7)        = |S|
I               =       +19
J               =       +19
SRC_REF:     55ADD   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 8)        = |T|
I               =       +19
J               =       +19
SRC_REF:     56SET   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 8)        = |T|
I               =       +19
J               =        +1
SRC_REF:     58ADD   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 8)        = |T|
I               =       +20
J               =       +20
SRC_REF:     55ADD   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 10)       = |E|
I               =       +20
J               =       +20
SRC_REF:     56SET   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 10)       = |E|
I               =       +20
J               =        +1
SRC_REF:     58ADD   SOURCE: MOBS      PROC: MOBS  ****************************
FELD( 10)       = |E|
I               =        +5
J               =        +5
K2-EVENT HAPPENS DURING TERMINAL DISPLAY OF WROUT OUTPUT
```

Da schon zu erkennen ist, dass das Programm jetzt richtig läuft, wird die Ausgabe mit der K2-Taste unterbrochen.

```
/%show %insert
> CTX: LOCAL#DEFAULT  SRC-REF:     58ADD    SOURCE: MOBS  PROC: MOBS
(INS2                          )
> CTX: LOCAL#DEFAULT  SRC-REF:     56SET    SOURCE: MOBS  PROC: MOBS
(INS3                          )
/%remove s'58add'
/%show %control
%CONTROL1 %ASSGN IN VERARBEITUNG <CON1: %D FELD(K),I,J>
/%rem %.con1
/%r
```

Mit %SHOW wird nachgeschaut, welche %INSERT- und %CONTROL-Kommandos noch wirksam sind. Das Subkommando INS2 ist jetzt überflüssig und wird gelöscht. Auch die Ausgaben des Subkommandos CON1 werden nicht mehr gebraucht. Nur die Korrektur des dritten %INSERT ist notwendig. Das Programm wird fortgesetzt.

```
BUCHSTABE ANZAHL PROZENT
A             34    9.68
B             13    3.70
C             18    5.12
D             15    4.27
E             56   15.95
F              5    1.42
G              6    1.70
H             18    5.12
I             31    8.83
J              1    0.28
K              4    1.13
L             16    4.55
M             11    3.13
N             29    8.26
O              8    2.27
P              4    1.13
Q              1    0.28
R             17    4.84
S             18    5.12
T             16    4.55
U              8    2.27
V              4    1.13
W              2    0.56
X              1    0.28
Y              1    0.28
Z              8    2.27
GESAMT:      351   98.12
```

Das Programm läuft jetzt bis zum Ende durch und gibt die Ergebnisliste aus. Da zu Beginn des Programmlaufs noch mit falschen Indizes gearbeitet wurde, können einige Ergebnisse noch nicht stimmen. Das Programm muss nochmal mit der AID-Korrektur ablaufen.

5. Schritt

```
/LOAD-PROGRAM *M(BIBLIO,MOBS,R-M=A(A-L=V)),T-O=AID
%  BLS0500 PROGRAM 'MOBS' GELADEN
/%IN S'56SET' <%SET 1 INTO J>
/%R
BUCHSTABE ANZAHL PROZENT
A             34    9.68
B             13    3.70
C             18    5.12
D             15    4.27
E             57   16.23
F              5    1.42
G              6    1.70
H             18    5.12
I             32    9.11
J              1    0.28
K              4    1.13
L             15    4.27
M             11    3.13
N             29    8.26
O              8    2.27
P              3    0.85
Q              1    0.28
R             17    4.84
S             18    5.12
T             16    4.55
U              8    2.27
V              4    1.13
W              2    0.56
X              1    0.28
Y              1    0.28
Z              8    2.27
GESAMT:      351   98.12
```

Das Programm wird erneut geladen. Mit %INSERT setzen Sie einen Testpunkt auf die SET-Anweisung in Zeile 56. Jedes Mal, wenn der Programmablauf an diesen Testpunkt kommt, wird J auf 1 gesetzt.
Das Programm wird gestartet und gibt die gewünschte Tabelle aus.

7 Testen von speziellen COBOL-Sprachmitteln

7.1 Testen von geschachtelten Programme

7.1.1 Setzen von Testpunkten

– Paragraphen und Kapitel des inneren Programms, in dem die Unterbrechungsstelle liegt, können ohne Qualifikation angesprochen werden.

– Auf Kapitel und Paragraphen in einem anderen Programm, das auch in einer anderen Übersetzungseinheit liegen kann, wird mit der S- und PROC-Qualifikation zugegriffen:

```
%INSERT [S=program-id.]PROC=program-id-innen.paragraph [IN kapitel]
```

– Die S-Qualifikation muss immer dann angegeben werden, wenn der Testpunkt in einem anderen getrennt übersetzten Programm gesetzt werden soll.

– Ein Testpunkt am Beginn der Procedure Division des äußersten Programms kann mit einer PROG-Qualifikation gesetzt werden:

```
%INSERT PROG=program-id.program-id
```

oder ausgeschrieben:

```
%INSERT S=program-id.PROC=program-id.program-id
```

Dieses Vorgehen ist nur dann sinnvoll, wenn program-id nicht länger als 8 Zeichen ist oder ein LLM generiert wurde, da sonst der Source-Name, nicht aber der Procedure-Name auf 8 Zeichen abgeschnitten wird.

– Ein Testpunkt auf den Anfang eines inneren Programms kann, da S und PROC verschieden sind, nicht mit einer PROG-Qualifikation gesetzt werden, sondern muss wie folgt angegeben werden:

```
%INSERT [S=program-id.]PROC=program-id-innen.program-id-innen
```

– Namen in der aktuellen Übersetzungseinheit, die dort eindeutig sind, können auch ohne Qualifizierung angesprochen werden.

7.1.2 Zugriff auf Daten

– Mit %D werden die Daten des aktuellen geschachtelten Programms gefunden sowie Daten mit dem GLOBAL-Attribut, die nicht lokal verdeckt sind; d.h. es kann auf die gleichen Daten zugegriffen werden, auf die auch das Programm selbst an dieser Stelle zugreifen kann.

– Mit %SD kann man die Daten aller dynamisch umgebenden Programme erhalten, entsprechend der aktuellen Aufrufhierarchie.

– Mit der S- und PROC-Qualifikation kann man gezielt auf ein Datum eines anderen Programms zugreifen:

```
%D PROC=program-id-innen.datenfeld
```

Dies ist auch mit %SD ohne Qualifikation möglich, sofern das Datum in einem rufenden Programm liegt.

Sowohl beim Zugriff auf Testpunkte als auch auf Daten gilt, dass die PROC-Qualifikation entsprechend der Programmverschachtelung mehrfach wiederholt werden kann.

7.1.3 Ablaufverfolgung

Das %TRACE-Kommando protokolliert alle durchlaufenen Anweisungen der aktuellen CSECT; d.h. auch Anweisungen der gerufenen inneren Programme werden protokolliert, nicht aber die Anweisungen in getrennt übersetzten Programmen.

Sofern beim Trace die Anweisungstypen angezeigt werden, meldet AID, wegen intern generierter Paragraphen, gelegentlich zusätzliche LABEL-Angaben.

7.2 Testen von objektorientierten Programmen

7.2.1 Adressierung

– **Klassen** werden durch eine Source-Qualifizierung angesprochen: S=<class>.
<class> ist der Name, der im CLASS-ID Paragraphen angegeben ist.

– **Methoden** werden durch eine Procedure-Qualifizierung angesprochen, wobei
<method> der Name ist, der im METHOD-ID Paragraphen angegeben ist:
`PROC={FACTORY | OBJECT}.PROC=<method>`

Eine Source-Qualifizierung ist dann notwendig, wenn der aktuelle Programmpunkt nicht in (einer Methode) der Klasse liegt.
Procedure-Qualifizierungen sind nur soweit nötig, wie dies für die eindeutige Identifizierung erforderlich ist. So kann PROC={FACTORY | OBJECT} für Methoden grundsätzlich entfallen, da der Methodenname in der Klasse eindeutig sein muss.

7.2.2 Kommandos

7.2.2.1 Setzen von Testpunkten

Das Setzen von Testpunkten in Methoden ist mit der Source- und Procedure-Qualifikation möglich:
`%INSERT [S=<class>.] [PROC=<method>.] srcref`

Auf eine Objektreferenz kann eine Schreibüberwachung gesetzt werden:
%ON %WRITE(objref). Die Anzeige einer durch NEW veränderten Objektreferenz ist aber erst nach Rückkehr an die Aufrufstelle möglich.

7.2.2.2 Ablaufverfolgung

Bei %TRACE können Klassen und Methoden als Trace-Bereich angegeben werden:
`%TRACE <n> IN S=<class>.[PROC={FACTORY | OBJECT}.PROC=<method>]`

7.2.2.3 Anzeigen von Daten

%DISPLAY

Daten eines Objektes sind nur sichtbar, wenn sich die Unterbrechungsstelle in einer Methode dieses Objektes befindet. In diesem Fall wird keine Qualifikation angegeben.

Daten in einer Methode sind nur innerhalb dieser Methode sichtbar.

Eine Objektreferenz wird wie folgt angezeigt:

```
<level> objref
        <level+1> FACTORY | OBJECT | NULL
        <level+1> class-name
```

Die erste Komponente gibt an, ob die Referenz auf das Factory-Objekt oder ein normales Objekt verweist oder eine Nullreferenz ist. Die zweite Komponente zeigt den Namen der Klasse des aktuell referenzierten Objektes an; für eine Nullreferenz entfällt sie.

%SD

%SD zeigt die Daten in der aktuellen dynamischen Aufruf-Verschachtelung von Programmen und Methoden an. Für Methoden werden nur die lokalen Daten der Methode, nicht aber die Daten des umgebenden Objektes ausgegeben.
Zusätzlich werden pro Klasse source-modul-globale Daten, wie z.B _COMPILATION_DATE ausgegeben.

7.2.2.4 Ändern von Daten

%SET, %MOVE

Eine high-level Zuweisung an eine Objektreferenz wird von AID mit einer Fehlermeldung (Types are not convertible...) abgewiesen. Ein low-level Zugriff auf Objektreferenzen ist möglich, liegt aber vollständig in der Verantwortung des Benutzers.

7.3 Testen von benutzerdefinierten Typen und typbezogenen Zeigern

AID unterstützt die TYPEDEF-Klausel sowie den typbezogenen Zeiger von COBOL2000 Programmen (siehe auch Kapitel „Testhilfen für den Programmablauf“ im Handbuch COBOL-Compiler [11]).

Ein Dereferenzierungs-Operator und ein Adress-Operator ergänzen die existierenden AID-Operatoren.

7.3.1 Der Dereferenzierungs-Operator

Als Dereferenzierungs-Operator dient der '*'. Er erlaubt den Zugriff auf ein Datum, das über einen Zeiger adressiert wird. Der '*' wird dem Zeiger vorangestellt und kann mit der COBOL- Qualifikation (IN, OF) und der COBOL-Subskribierung kombiniert werden.

Der Dereferenzierungs-Operator kann nur auf typbezogenen Zeiger angewandt werden.

Beispiele

1. `/%DISPLAY *ZEIGER`
2. `/%DISPLAY FELD IN *ZEIGER`

Im Beispiel 1 gibt AID die Daten aus, die über `ZEIGER` adressiert werden.

Im Beispiel 2 gibt AID das Element `FELD` aus, das in der Datenstruktur liegen muss, die über `ZEIGER` adressiert wird. Dieses Beispiel zeigt auch, dass '*' stärker bindet als die Qualifikation mit IN/OF.

7.3.2 Der Adressselektor (Adress-Operator)

Wie COBOL bietet AID den Adressselektor `ADDRESS OF` an. Dieser ist in AID nur bei der Einstellung `SYMCHARS=STD` und genau in dieser Form reserviert. Im Gegensatz zu COBOL2000 ist `ADDRESS` alleine oder z. B. in Verbindung mit `ADDRESS IN` in AID nicht reserviert.

Kompatibilität zu COBOL85-Programmen

Ein COBOL-Datenfeld `ADDRESS` kann weiterhin problemlos mit AID referenziert werden. Allerdings ist eine Qualifikation nur noch über `ADDRESS IN` und nicht mehr über `ADDRESS OF` möglich.

Beispiel

```
%SET ADDRESS OF FELD INTO ZEIGER
```

Die Adresse von `FELD` wird in `ZEIGER` übertragen.

7.3.3 Typverträglichkeit bei Vergleichen und Zuweisungen (%SET)

Vergleiche bzw. Zuweisungen auf typbezogene Zeiger lässt AID nur dann zu, wenn beide Zeiger den gleichen Referenztyp haben (und damit die gleiche TYPEDEF-Klausel zu Grunde liegt). Vergleiche und Zuweisungen von Zeigern mit Daten ungleicher Typen sind daher generell nicht zugelassen.

Bei einem Vergleich oder einer Zuweisung ist auch der Adressselektor `ADDRESS OF` erlaubt. Dem Adressselektor wird implizit ein entsprechender Referenztyp zugeordnet, der bei der Typverträglichkeit analog geprüft wird.

Vergleichen und Zuweisen von Datenstrukturen mit einer TYPE-Klausel

Vergleiche bzw. Zuweisungen sind wie bei Datenstrukturen ohne TYPE-Klausel Operationen, bei denen die gesamte Datenstruktur als ein hexadezimaler String aufgefasst wird. Die TYPE-Klausel bewirkt jedoch, dass AID den Referenztyp prüft (nur bei %SET nicht bei %MOVE) und die Operation gegebenenfalls abweist. Die Prüfung geschieht analog wie bei typbezogenen Zeigern.

Ein Vergleich bzw. eine Zuweisung auf Lowlevel-Ebene, z.B. durch eine Typmodifikation mit %X, ist jedoch immer möglich.

Fachwörter

Ablaufüberwachung

%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.

Adressoperand

ist ein Operand, mit dem Sie eine Speicherstelle oder einen Speicherbereich adressieren. Sie können virtuelle Adressen, Datennamen, Anweisungsnamen, Source-Referenzen, Schlüsselwörter, komplexe Speicherreferenzen, eine C-Qualifikation (maschinennahes Testen) oder eine PROG-Qualifikation (symbolisches Testen) angeben. Die Speicherstelle bzw. der Speicherbereich liegen entweder im geladenen Programm oder in einem Speicherabzug in einer Dump-Datei.
Wenn ein Name in Ihrem Programm mehrfach vergeben wurde und somit kein eindeutiger Bezug auf eine Adresse gewährleistet ist, können Sie ihn mit Bereichsqualifikationen oder über eine *kennzeichnung* eindeutig der gewünschten Adresse zuordnen.

Adressselektor

Der Adressselektor liefert für ein Speicherobjekt die korrespondierende Adresse. Er kann in COBOL mit ADDRESS OF oder als Low-Level-Funktion in der Form %@(...) angegeben werden.

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 einer Basisqualifikation ansprechen können. Er umfasst den nicht-privilegierten Teil des virtuellen Speichers in Ihrer Task, der vom Programm samt allen konnektierten Subsystemen belegt ist oder dem entsprechenden Bereich in einem Speicherabzug.
Mit dem Kommando %BASE können Sie den AID-Arbeitsbereich vom geladenen Programm in einen Speicherabzug verlegen oder umgekehrt. In einem Kommando können Sie vom AID-Arbeitsbereich abweichen, indem Sie im Adressoperanden Qualifikationen angeben.

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 bzw. eine Ausgabedatei zuzuweisen:

1. %OUTFILE-Kommando mit dem Link- und Dateinamen
2. ADD-FILE-LINK-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 wird mit OPEN=EXTEND geöffnet.

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:

– Dump-Dateien, in denen sich Speicherabzüge befinden (%DUMPFILE)
– PLAM-Bibliotheken, in denen sich Bindemodule bzw. LLMs befinden. Wird die Bibliothek mit %SYMLIB zugewiesen, kann AID die LSD-Sätze nachladen.

AID-Literale

AID stellt Ihnen Character-Literale und numerische Literale zur Verfügung (siehe AID-Basishandbuch [1]): :

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

AID-Standard-Arbeitsbereich

Das ist der nicht-privilegierte Teil des virtuellen Speichers in Ihrer Task, der vom Programm samt allen seinen konnektierten Subsystemen belegt ist.
Ohne Vereinbarung mit %BASE und ohne Angabe einer Basisqualifikation 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 die Programmeinheit, in der die Übersetzungseinheit unterbrochen wurde. Die STOP-Meldung gibt ihren Namen aus.

Aktuelle Übersetzungseinheit

ist die Übersetzungseinheit in der die aktuelle Unterbrechungsstelle liegt.

Anweisungsname

bezeichnet die Adresse des ersten Befehls in einem Paragraphen oder Kapitel in der PROCEDURE DIVISION und wird angegeben mit:

```
{L'kapitel'                    }
{L'paragraph' [IN L'kapitel']  }
```

Ist keine Verwechslung mit einem Datennamen möglich, kann ein alphanumerischer Kapitel- oder Paragraphenname auch ohne L'...' angegeben werden.
Ist ein Paragraphenname innerhalb eines Programms nicht eindeutig, kann er mit IN L'kapitel' gekennzeichnet werden.
Anweisungsnamen sind 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.
Adress-Konstanten 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 [1] sind die AID-spezifischen Speichertypen samt zugehörigen Ausgabetypen aufgelistet. Für die in COBOL verwendeten Datentypen gilt eine entsprechende Zuordnung. Eine Typmodifikation bei %DISPLAY und %SDUMP bewirkt eine Änderung des Ausgabetyps.

Basisqualifikation

ist die Qualifikation, mit der Sie das geladene Programm oder einen Speicherabzug in einer Dump-Datei bezeichnen. Sie wird mit E={VM | Dn} angegeben. Die Basisqualifikation können Sie global mit %BASE vereinbaren oder im Adressoperanden 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 PROC/PROG-Qualifikationen ergeben sich die Bereichsgrenzen aus Anfangs- und Endadresse der Programmeinheit (siehe AID-Basishandbuch [1]).

Bereichsqualifikation

Mit diesen Qualifikationen bezeichnen Sie einen Teil des Arbeitsbereiches. Endet ein Adressoperand mit einer dieser Qualifikationen, so wirkt das Kommando nur in dem Teil, der mit der letzten Qualifikation bezeichnet wurde. Mit einer Bereichsqualifikation begrenzen Sie den Wirkungsbereich eines Kommandos oder machen damit einen Daten- oder Anweisungs-Namen im Arbeitsbereich eindeutig oder Sie erreichen damit einen Namen, der an der aktuellen Unterbrechungsstelle sonst nicht ansprechbar wäre.

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.

Datenfeld

ist ein Sammelbegriff für alle in der DATA DIVISION definierten Daten. Er umfasst Datengruppen und Tabellen, sowie deren Elemente.

Datenname

steht für alle Namen, die im Quellprogramm für Daten vergeben wurden. Mit einem Datennamen sprechen Sie Datenfelder beim symbolischen Testen an. Keine LSD-Sätze werden für Definitionen aus der REPORT-SECTION, für 88er-Stufen, für Systemschalter im SPECIAL-NAMES-Paragraphen und das NATIVE-Alphabet erzeugt. Diese Daten können Sie deshalb mit AID nicht ansprechen.
Ist ein Datenname nicht eindeutig innerhalb einer Programmeinheit, kann er gekennzeichnet werden, indem er mit IN oder OF einer bestimmten Datengruppe zugeordnet wird.
Elemente von Tabellen können Sie wie in COBOL über einen Index ansprechen.

Datentyp

Gemäß dem im Quellprogramm deklarierten Datentyp ordnet AID allen Datenfeldern einen AID-Speichertyp zu:

- Binärstring (≙ %X)
- Character (≙ %C oder %UTF16)
- numerisch (≙ %D, %F, %P)

Nicht alle Datentypen, die in COBOL numerisch behandelt werden, sind auch für AID vom Speichertyp numerisch (siehe %SET-Tabelle).
Der zugeordnete Speichertyp bestimmt, wie das Datenfeld 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 Adressoperanden. Mit einem Index wird die Position eines Tabellenelementes bestimmt. Er kann wie in COBOL angegeben werden (mehrere Indizes müssen allerdings im Gegensatz zu COBOL durch Komma getrennt werden) oder durch einen arithmetischen Ausdruck, aus dem AID den Wert des Index errechnet. Dieser AID-spezifische Index beinhaltet sowohl den Zugriff auf ein Tabellenelement mit einem Subskript als auch mit dem COBOL-spezifischen Index aus der INDEXED BY-Klausel.

index-name

ist der symbolische Name, der in der INDEXED BY-Klausel für die Indizierung einer Tabellenstufe vereinbart wurde. *index-name* darf nicht zur Indizierung einer anderen Tabelle verwendet werden.
Soll der Index von AID aus einem arithmetischen Ausdruck berechnet werden, kann *index-name* nur mit ganzen Zahlen, nicht aber mit anderen Datenfeldern oder dem COBOL-Sonderregister TALLY verknüpft werden.

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 des AID-Basishandbuch [1] 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 Figurativen Konstanten, die Ergebnisse von Längenselektion, Längenfunktion und Adressselektion sowie die Anweisungsnamen und die Source-Referenzen.
Eine Adresskonstante repräsentiert eine Adresse. Adresskonstanten 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 [1]).

Lokalisierungsinformation

Mit %DISPLAY %HLLOC(*speicherref*) für die symbolische Ebene und mit %DISPLAY %LOC(*speicherref*) für die Maschinencode-Ebene gibt Ihnen AID die statische Programmverschachtelung zur 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 Progmrammeinheit in den LSD-Sätzen verzeichneten Dateinamen, Datennamen, Sonderregister und Figurativen Konstanten.

Objekt-Strukturliste

Auf Basis des ESD (External Symbol Dictionary) erstellt der Binder die Objekt-Strukturliste, wenn die Binderoption TEST-OPTIONS=AID gilt.

Programmeinheit

ist die Untermenge eines vollständigen COBOL-Programms, die einen eigenen Namen in der PROGRAM-ID erhalten hat, z.B. das Hauptprogramm oder jedes mit CALL aufgerufene Unterprogramm. Sie kann mit einer PROC- oder C-Qualifikation (Segment, Sharedcode-Modul) angesprochen werden.

Programmzustand

AID unterscheidet drei Programmzustände, in denen sich das zu testende Programm befinden kann:

– Das Programm steht.
  %STOP oder K2-Taste unterbrechen ein laufendes Programm. Außerdem wird das Programm unterbrochen, wenn ein %TRACE abgearbeitet ist. Die Task befindet sich im Kommandomodus. Sie können Kommandos eingeben.

– 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.

– 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 ein Speicherobjekt ansprechen, das nicht im AID-Arbeitsbereich liegt oder darin keinen eindeutigen Namen hat. Die Basisqualifikation gibt an, ob die Adresse im geladenen Programm oder in einem Speicherabzug liegt. Die S-Qualifikation gibt an, in welcher Übersetzungseinheit das Speicherobjekt liegt. Die PROC- bzw. C-Qualifikation gibt an, in welcher Programmeinheit bzw. in welchem Segment die Adresse liegt. 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 PROC-Qualifikation zu einem Datenfeld der aktuellen Programmeinheit angegeben wird, außer im %SDUMP.

Sonderregister

Der COBOL-Compiler stellt Ihnen zu jedem Programm Sonderregister zur Verfügung:

```
LINAGE-COUNTER
RETURN-CODE
SORT-CCSN
SORT-CORE-SIZE
SORT-EOW
SORT-FILE-SIZE
SORT-MODE-SIZE
SORT-RETURN
TALLY
```

Zu jedem Programm wird ein TALLY-Sonderregister angelegt. Dagegen gibt es das Sonderregister RETURN-CODE nur einmal für die gesamte Übersetzungseinheit. Die SORT-Sonderregister werden nur erzeugt, wenn das Programm einen Sortierteil enthält.

Source-Referenz

bezeichnet eine ausführbare Anweisung. Sie wird angegeben mit
S'n[verb[m]] | S'xverb[m]
Source-Referenzen werden vom Compiler erzeugt und in LSD-Sätzen hinterlegt.
n | x ist die Zeilennummer, die vom Programmierer oder vom Compiler vergeben wurde, entsprechend der bei der Übersetzung geltenden SDF-Option STMT–REFERENCE.
verb ist die festgesetzte Abkürzung eines COBOL-Verbs (siehe Abschnitt „Symbolische Speicherreferenzen“ auf Seite 21).
m ist eine Nummer, die Sie nur angeben müssen, wenn in einer Anweisungszeile das gleiche COBOL-Verb mehrmals vorkommt. *m* bezeichnet dann das m-te dieser gleichen Verben.
Source-Referenzen sind 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 maschinennahe Speicherreferenzen sind virtuelle Adressen und CSECTs. Symbolische Speicherreferenzen sind alle in den LSD-Informationen verzeichneten Namen von Dateien, Daten und Anweisungen aus dem Programm, die vom Compiler erzeugten Source-Referenzen und die AID-Schlüsselwörter. Mit der komplexen Speicherreferenz 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, %UTF16, %S, %SX
(siehe %SET und AID-Basishandbuch [1]).

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 Manual AID-Basishandbuch [1] 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.

Übersetzungseinheit

besteht aus einem einzelnen oder einer Folge von Quellprogrammen. Sie wird mit der S-Qualifikation angesprochen werden.

Unterbrechungsstelle

Die Adresse, an der ein Programm unterbrochen wurde, wird Unterbrechungsstelle genannt. Aus der STOP-Meldung können Sie entnehmen, an welcher Adresse und in welchem Programmteil die Unterbrechungsstelle liegt. Dort wird das Programm fortgesetzt. Für COBOL-Programme können Sie mit %JUMP eine andere Fortsetzungsadresse vereinbaren.

Zeichenkonvertierungsfunktionen

AID stellt zwei Funkionen zur Zeichenkonvertierung %C() und %UTF16() zur Verfügung.
Die Funktion %UTF16() wandelt Strings von einer 1-Byte-EBCDIC-Codierung in die UTF16-Codierung um, die Funktion %C realisiert die Konvertierung umgekehrt.

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)
Testen auf Maschinencode-Ebene
Benutzerhandbuch

[3] **AID** (BS2000)
Advanced Interactive Debugger
Testen von FORTRAN-Programmen
Benutzerhandbuch

[4] **AID** (BS2000)
Advanced Interactive Debugger
Testen unter Posix
Benutzerhandbuch

[5] **AID** (BS2000)
Advanced Interactive Debugger
Testen von ASSEMBH-Programmen
Benutzerhandbuch

[6] **AID** (BS2000)
Testen von C/C++-Programmen
Benutzerhandbuch

[7] **AID** (BS2000)
Advanced Interactive Debugger
Tabellenheft
Benutzerhandbuch

[8] **BS2000 OSD/BC**
Makroaufrufe an den Ablaufteil
Benutzerhandbuch

[9] **BS2000 OSD/BC**
Programmiersystem
Technische Beschreibung

[10] **COBOL85** (BS2000)
COBOL-Compiler
Beschreibung

[11] **COBOL85** (BS2000)
COBOL-Compiler
Benutzerhandbuch

[12] **COBOL2000** (BS2000))
COBOL-Compiler
Sprachbeschreibung

[13] **COBOL2000** (BS2000)
COBOL-Compiler
Benutzerhandbuch

[14] **XHCS**
8-bit-Code- und Unicode-Unterstützung im BS2000
Benutzerhandbuch

Stichwörter