1 Einleitung

Mit AID (Advanced Interactive Debugger) steht im Betriebssystem BS2000 eine leistungsstarke Dialog-Testhilfe zur Verfügung. AID V2.0A können Sie ab BS2000 V9.5 einsetzen. Fehlerdiagnose, Test und vorläufige Fehlerbehebung aller im BS2000 erstellten Programme können Sie mit AID wesentlich schneller und mit weniger Aufwand durchführen als mit anderen Mitteln, wie z.B. dem Einfügen von Testhilfe-Anweisungen im Programm. AID ist permanent verfügbar und besitzt eine hohe Anpassungsfähigkeit an die jeweilige Programmiersprache. Ein Programm, das Sie mit AID getestet haben, muß nicht erneut übersetzt werden, sondern kann sofort in den produktiven Einsatz gehen. Der Funktionsumfang von AID und seine Testsprache, die AID-Kommandos, sind primär auf die Dialoganwendung zugeschnitten. AID kann aber ebenso gut im Batch-Betrieb eingesetzt werden. AID bietet Ihnen vielfältige Möglichkeiten zur Ablaufüberwachung, Ablaufsteuerung, Ausgabe und Änderung von Speicherinhalten, Abfrage von Informationen über den Programmablauf und zur Handhabung von AID.

Mit AID können Sie sowohl auf der symbolischen Ebene der jeweiligen Programmiersprache als auch auf Maschinencode-Ebene testen. Wurden beim Übersetzen LSD-Sätze erzeugt, können Sie im Test Daten, Anweisungsmarken und Programmteile mit den Namen ansprechen, die Sie beim Programmieren vergeben haben. Anweisungen können Sie mit den Nummern oder Namen ansprechen, die vom Compiler erzeugt wurden. Wurden keine LSD-Sätze für ein Programm oder Modul erzeugt, können Sie Daten und Anweisungen mit virtuellen Adressen, über CSECT-Namen und Schlüsselwörter ansprechen.
Die BS2000-Kommandos, die in der AID-Dokumentation vorkommen, sind im SDF-Format (System Dialog Facility), EXPERT-Form beschrieben. SDF ist die Dialogschnittstelle zum BS2000. Die SDF-Kommandosprache löst die bisherige Kommandosprache im ISP-Format ab.

Zielgruppe

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

Konzept der AID-Handbücher

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

AID - Basishandbuch [1]
Im Basishandbuch finden Sie einen Überblick über AID und die Beschreibung der Sachverhalte und Operanden, die in allen Programmiersprachen gleich sind. Im Überblick wird die BS2000-Umgebung beschrieben, es werden die grundlegenden Begriffe erläutert und der AID-Kommandovorrat vorgestellt. Die anderen Kapitel beschreiben die Testvorbereitung, die Kommandoeingabe, die Operanden Subkommando, komplexe Speicherreferenz und Medium-und-Menge, die AID-Literale und die Schlüsselwörter. Das Handbuch enthält außerdem die Meldungen, die in Kommandofolgen unzulässigen BS2000-Kommandos, die nur noch in dieser Version unterstützten Operanden und eine Gegenüberstellung von AID und IDA.

AID - Testen auf Maschinencode-Ebene [2]
AID - Testen von COBOL-Programmen [3]
AID - Testen von FORTRAN-Programmen [4]
AID - Testen von ASSEMBH-Programmen
AID - Testen von PL/I-Programmen [5]
AID - Testen von C-Programmen [6]

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

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

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

Änderungen gegenüber AID V1.0C

Daten- und Anweisungsnamen können Sie in die Errechnung einer Speicherstelle einbeziehen. Dadurch sind Übergänge vom symbolischen Testen zum Testen auf Maschinencode-Ebene und umgekehrt geschaffen worden. Überall dort, wo die Vermischung der beiden Ebenen möglich ist, steht der neue Operand *kompl-speicherref* (komplexe Speicherreferenz).

Mit den Selektoren können Sie auf Adresse (%@), Länge (%L) oder Typ (%T) von Datennamen zugreifen.

Die Typ- und Längenmodifikation können Sie auf Daten- und Anweisungsnamen und alle Speicherreferenzen anwenden. Durch Typ- und Längenselektor, durch die neuen Speichertypen und die getrennte Anwendung von Typ- oder Längenmodifikation stehen Ihnen umfangreiche Modifikations-Möglichkeiten zur Verfügung.

Die neuen Speichertypen für Character (%C), Gleitpunkt (%D) und gepackt (%P) bieten neue Möglichkeiten für eine Redefinition von Speicherinhalten oder für die Ausgabe in einem anderen Format.

Mit den Speichertypen zur Adreßinterpretation %S und %SX können Sie Adressen in Verbindung mit nachfolgendem Pointer-Operator (->) errechnen lassen wie bei Assembler-Befehlen aus Basisregister und Distanz (%S) bzw. Indexregister, Basisregister und Distanz (%SX).

Die AID-Register %nG und %nGD bieten einen programmunabhängigen Registersatz zur freien Verwendung für den Benutzer.

Maschinennahe Qualifikationen werden nun auch wie symbolische erst zum Ausführungszeitpunkt überprüft, nicht bei der Eingabe.

Mit %HLLOC(speicherref) können Sie sich im %DISPLAY-Kommando für eine beliebige Speicherreferenz die symbolische Lokalisierungsinformation, d.h. die symbolischen Namen der Umgebung der entsprechenden Adresse ausgeben lassen.

In einem Subkommando können Sie einen Namen und/oder eine Bedingung definieren. Mit dem Namen können Sie den Durchlaufzähler des Subkommandos ansprechen oder das Subkommando löschen. Von der Bedingung können Sie die Ausführung des Subkommandos abhängig machen.

Das Kommando %CONTINUE startet oder setzt das Programm fort. Im Gegensatz zu %RESUME setzt %CONTINUE auch einen unterbrochenen %TRACE fort.

Das Kommando %MOVE kann auch auf symbolischer Ebene eingesetzt werden. Es verändert Speicherinhalte, ohne vorher die Vereinbarkeit der Speichertypen von *sender* und *empfänger* zu überprüfen und ohne numerische Werte zu konvertieren.

Das Kommando %FIND kann auch auf symbolischer Ebene innerhalb von Datennamen angewendet werden. Die Trefferadresse legt AID im AID-Register %0G, die Fortsetzungsadresse in %1G ab.

Das %AID-Kommando wurde erweitert:
Mit dem Operanden LOW legen Sie fest, ob AID Kleinbuchstaben aus Benutzereingaben in Großbuchstaben umsetzen soll oder nicht.
Mit dem Operanden DELIM legen Sie die Begrenzer von alphanumerischen Datenausgaben fest.
Mit dem Operanden OV legen Sie fest, ob AID die Überlagerungsstruktur eines Programms berücksichtigen soll oder nicht. Ab Version 2.0A beachtet AID nicht mehr standardmäßig die Überlagerungsstruktur eines Programms.
Mit dem Operanden LANG legen Sie fest, ob %HELP die Informationen in Deutsch oder Englisch ausgeben soll.

2 Voraussetzungen zum symbolischen Testen

Das Erzeugen der LSD-Sätze, die AID zum symbolischen Testen benötigt, steuern Sie durch die im folgenden beschriebenen Operanden, die beim Übersetzen, Binden und Laden angegeben werden müssen. Eine ausführliche Beschreibung dieser Operanden finden Sie im ASSEMBH-Benutzerhandbuch [9].

2.1 Übersetzen

Der Assembler ASSEMBH-XT läßt sich auf zwei Arten steuern:
- durch SDF-Optionen oder
- durch COMOPT-Anweisungen.

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

SDF-Steuerung

```
/START-PROG $ASSXT ....,TEST-SUPPORT = { NO  }
                                       { YES }
```

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

YES Der ASSEMBH-XT erzeugt LSD-Sätze. Das Programm kann mit AID symbolisch getestet werden.

COMOPT-Steuerung

```
/START-PROGRAM $ASSXTC
*...
*COMOPT = { NOISD }
          { ISD   }
```

NOISD Es werden keine LSD-Sätze erzeugt.

ISD Der ASSEMBH-XT erzeugt LSD-Sätze. Das Programm kann mit AID symbolisch getestet werden.

Beispiel

```
/START-PROG $ASSXT
//COMPILE SOURCE = QUELLE.TEST,
                   TEST-SUPPORT = YES,
                   MODULE-LIBRARY = PROGRAMLIB
```

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

```
/DELETE-SYSTEM-FILE FILE-NAME = OMF
/START-PROGRAM $ASSXTC
**COMOPT SOURCE=QUELLE.TEST
**COMOPT ISD
**COMOPT MODULE=PROGRAMLIB
**END HALT
```

2.2 Binden, Laden und Starten

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

Die vom Assembler ASSEMBH-XT erzeugten LSD-Informationen müssen an den Statischen Binder (TSOSLNK) bzw. Dynamischen Bindelader (DLL bis BS2000 V9.5, DBL ab V10.0) sowie an den Statischen Starter (ELDE) weitergereicht werden, damit Sie symbolisch testen können.

ASSEMBH-Programme binden, laden und starten Sie mit den für alle Sprachen gültigen SDF-Kommandos bzw. TSOSLNK-Anweisungen, die im AID-Basishandbuch, Kapitel 3 beschrieben sind.

3 ASSEMBH-spezifische Adressierung

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

Qualifikationen

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

E={VM|Dn}

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

PROG=program-name

Als Bereichs-Qualifikation können Sie bei ASSEMBH die PROG-Qualifikation verwenden. *program-name* bezeichnet eine Programmeinheit aus einem ASSEMBH-Programm.
program-name ist der Name, der im Quellprogramm in einer START- oder CSECT-Anweisung vergeben wurde.
Operanden, die einen Adreßbereich (%CONTROLn, %TRACE) oder einen Namensraum (%SDUMP) angeben, können mit der PROG-Qualifikation enden. Der Adreßbereich bzw. Namensraum umfaßt dann die gesamte Programmeinheit.

PROG=program-name•program-name

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

Speicherreferenzen

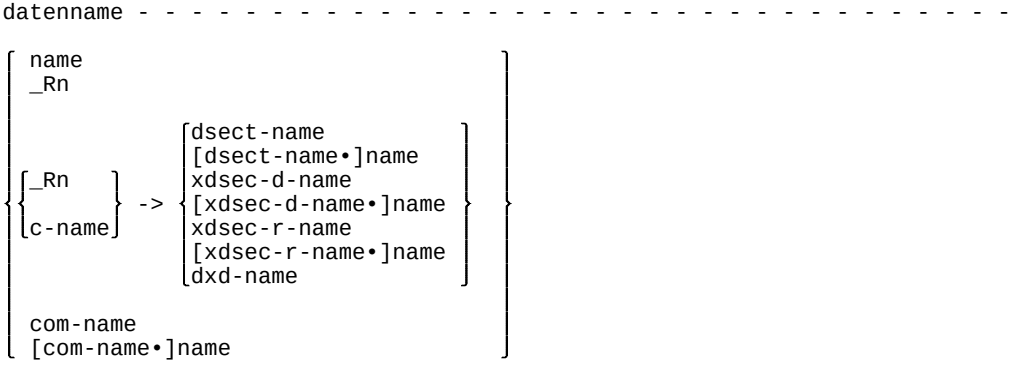
Als Speicherreferenzen können alle in den LSD-Sätzen verzeichneten Datennamen und Anweisungsnamen aus dem Programm und die vom Assembler erzeugten Statementnummern verwendet werden, und sie können alle Operationen, wie im AID-Basishandbuch, Kapitel 6 beschrieben, darauf anwenden.
In allen Operanden, in denen *kompl-speicherref* möglich ist, können Sie beliebig wechseln zwischen den in diesem Handbuch beschriebenen Speicherreferenzen und denen für das Testen auf Maschinencode-Ebene [2].

datenname
: ist der im Quellprogramm definierte Name von Konstanten, Datenfeldern, vordefinierten Mehrzweckregistern, Programmabschnitten, Pseudoabschnitten, externen Pseudoabschnitten, Pseudoregistern und gemeinsamen Hilfsabschnitten.

  Daten, die im Quellprogramm über einen Pseudoabschnitt, einen externen Pseudoabschnitt, oder ein Pseudoregister definiert sind (siehe DSECT-, XDSEC- und DXD-Anweisung, ASSEMBH, Beschreibung [10]), können beim symbolischen Testen nur mit Hilfe einer Pointer-Operation angesprochen werden.

  datenname ist der Namenseintrag einer DC-, DS-, EQU-, CSECT-, DSECT-, XDSEC-, DXD- und COM-Assembleranweisung (siehe ASSEMBH, Beschreibung [10]).

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



name
ist der Namenseintrag einer DC-, DS-, EQU- oder CSECT-Anweisung.

- Mit Namen von DC- oder DS-Anweisungen sprechen Sie den jeweiligen Speicherinhalt an. AID gibt den Speicherinhalt in dem Datentyp und der Länge aus, wie sie im Quellprogramm definiert wurden.
- Mit Namen von EQU-Anweisungen sprechen Sie entweder den zugewiesenen Wert oder den Speicherinhalt an der betreffenden Adresse an. Die Ausgabe erfolgt entsprechend dem Längenmerkmal des EQU-Namens in der Form, die dem Typenmerkmal entspricht.
- Mit Namen von CSECT-Anweisungen sprechen Sie die Anfangsadresse oder Fortsetzungsadresse eines Programmabschnitts an.

_Rn
ist der vordefinierte Name für ein Mehrzweckregister. Bei Angabe dieses Namens gibt AID den Inhalt des betreffenden Registers aus. _Rn entspricht dem AID-Schlüsselwort %n.

n ist eine Zahl mit $0 \leq n \leq 15$

Pseudoabschnitt (DSECT-Anweisung)

```
{_Rn    } -> {dsect-name       }
{c-name }    {[dsect-name•]name}
```

_Rn
gibt das Basisadreßregister des Pseudoabschnitts an.
Falls der Pseudoabschnitt über eine Q-Konstante referenziert wurde, gibt *_Rn*, die Anfangsadresse des Pseudoregistervektors an (siehe ASSEMBH, Beschreibung [10]).

c-name
ist der Name einer A-, Y- oder V-Konstanten, deren Inhalt die Basisadresse des Pseudoabschnitts ist.
Falls der Pseudoabschnitt über eine Q-Konstante referenziert wurde, gibt *c-name*, die Anfangsadresse des Pseudoregistervektors an (siehe ASSEMBH, Beschreibung [10]).

dsect-name
ist der Name des Pseudoabschnitts.

Damit sprechen Sie alle benannten Felder des Pseudoabschnitts entsprechend ihrem Typ und nach aufsteigenden Adressen sortiert an.

[dsect-name.]name
dsect-name ist der Name des Pseudoabschnitts.

name
ist der Name eines einzelnen Feldes innerhalb des Pseudoabschnitts.
AID interpretiert das Feld entsprechend dem Typ und dem Längenmerkmal.

Definition eines externen Pseudoabschnitts (XDSEC D)

```
{_Rn     } -> {xdsec-d-name          }
{c-name  }    {[xdsec-d-name•]name  }
```

_Rn
gibt das Basisadreßregister des externen Pseudoabschnitts an.

c-name
ist der Name einer A-, Y- oder V-Konstanten, deren Inhalt die Basisadresse des externen Pseudoabschnitts ist (siehe ASSEMBH, Beschreibung [10]).

xdsec-d-name
ist der Name des externen Pseudoabschnitts.

Damit sprechen Sie alle benannten Felder des Pseudoabschnitts entsprechend ihrem Typ und nach aufsteigenden Adressen sortiert an.

[xdsec-d-name•]name
xdsec-d-name ist der Name des externen Pseudoanschnitts.

name
ist der Name eines einzelnen Feldes innerhalb des externen Pseudoabschnitts.
AID interpretiert das Feld entsprechend dem Typ und dem Längenmerkmal.

Referenz eines externen Pseudoabschnitts (XDSEC R)

```
{_Rn     } -> {xdsec-r-name          }
{c-name  }    {[xdsec-r-name•]name  }
```

_Rn
gibt das Basisadreßregister des externen Pseudoabschnitts an.

c-name
ist der Name einer A-, Y- oder V-Konstanten, deren Inhalt die Basisadresse des externen Pseudoabschnitts ist (siehe ASSEMBH, Beschreibung [10]).

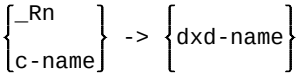
xdsec-r-name
ist der Name des externen Pseudoabschnitts.

Damit sprechen Sie nur die Felder des Pseudoabschnitts (in beliebiger Reihenfolge) an, die im Programm referenziert wurden.

[xdsec-r-name•]name
xdsec-d-name ist der Name des externen Pseudoabschnitts.

name
ist der Name eines einzelnen Feldes innerhalb des externen Pseudoabschnitts. Angesprochen werden können nur Felder, die im Programm referenziert wurden.
AID interpretiert das Feld entsprechend dem Typ und dem Längenmerkmal.

Pseudoregister (DXD)



_Rn
gibt das Register an, das mit der Anfangsadresse des Pseudoregistervektors geladen ist.

c-name
ist der Name einer A-, Y- oder V-Konstanten, deren Inhalt die Anfangsadresse des Pseudoregistervektors ist (siehe ASSEMBH, Beschreibung [10]).

dxd-name
ist der Name des Pseudoregisters.

com-name
ist der Name eines gemeinsamen Hilfsabschnitts (siehe COM-Anweisung, ASSEMBH, Beschreibung [10]).

Damit sprechen Sie alle benannten Felder des gemeinsamen Hilfsabschnitts an.

Vor *com-name* brauchen Sie nur eine Basis-Qualifikation (keine PROG-Qualifikation) angeben, falls *com-name* nicht im aktuellen AID-Arbeitsbereich liegt.

[com-name•]name
com-name ist der Name des gemeinsamen Hilfsabschnitts.

name
Damit sprechen sie den Namen eines einzelnen Feldes innerhalb des gemeinsamen Hilfsabschnitts an.

L’name’

ist ein Anweisungsname und bezeichnet die Adresse einer ausführbaren Assembler-Instruktion oder eines Aufrufs für einen vordefinierten Makro (@-Makro; siehe ASSEMBH, Beschreibung [10]).
name ist der maximal 64-stellige Namenseintrag (im Quellprogramm) einer Assembler-Instruktion oder eines Aufrufs für einen vordefinierten Makro.
name wird von AID auf 32 Stellen verkürzt.

L’name’ kann in allen Operanden angegeben werden, die entweder eine Adresse im ausführbaren Teil des Programms bezeichnen (%DISASSEMBLE, %FIND, %INSERT) oder die zur Ausgabe und Änderung von Speicherstellen angegeben werden (%DISPLAY, %MOVE, %SET).

S’stmt-nr’

ist eine Source-Referenz über die eine Assembler-Instruktion mit Namen oder der Aufruf eines vordefinierten Makros (@-Makro) angesprochen werden kann.
stmt-nr ist die Statementnummer, die vom Assembler vergeben wird und in der Übersetzungsliste der Spalte STMNT zu entnehmen ist (siehe Quellprogramm, Kapitel 6).

stmt-nr ist eine Ganzzahl zwischen 1 und 2^{31}-1.

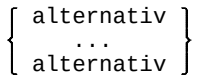
S’stmt-nr’ kann in allen Operanden angegeben werden, die einen Bereich (%CONTROLn, %TRACE) oder eine Adresse (%DISASSEMBLE, %FIND, %INSERT) im ausführbaren Teil des Programms bezeichnen oder die zur Ausgabe und Änderung von Speicherstellen angegeben werden (%DISPLAY, %MOVE, %SET).

4 Metasyntax

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

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

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



`{alternativ | ... | alternativ}`
: 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.

Im Fließtext werden Operanden in *kursiven kleinbuchstaben* dargestellt.

5 AID-Kommandos

%AID

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

- Mit *CHECK* legen Sie fest, ob vor der Ausführung von %MOVE oder %SET ein Änderungsdialog durchgeführt werden soll.
- Mit *REP* legen Sie fest, ob die Speicheränderungen eines %MOVE-Kommandos als REP abgelegt werden sollen.
- Mit *SYMCHARS* legen Sie fest, ob AID den Bindestrich "-" in Programm-, Daten- und Anweisungsnamen als Bindestrich oder als Minuszeichen interpretieren soll. Damit der Bindestrich entsprechend den Assembler-Konventionen stets als Minuszeichen interpretiert wird, muß SYMCHARS = NOSTD gesetzt werden.
- Mit *OV* legen Sie fest, ob AID die Überlagerungsstruktur eines Programms (Overlay) berücksichtigen soll.
- Mit *LOW* legen Sie fest, ob AID Kleinbuchstaben aus Character-Literalen und Namen in Großbuchstaben konvertieren oder als Kleinbuchstaben interpretieren soll. Voreinstellung ist OFF.
- Mit *DELIM* legen Sie die Begrenzer (Delimiter) für die AID-Ausgabe alphanumerischer Daten fest. Der senkrechte Strich ist der Standard-Begrenzer.
- Mit *LANG* legen Sie fest, ob AID die Informationen des %HELP-Kommandos auf Deutsch oder Englisch ausgeben soll.

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

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

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

%AID verändert den Programmzustand nicht.

CHECK

ALL

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

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

N

I342 NOTHING CHANGED
```

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

NO

%MOVE und %SET werden ohne Änderungsdialog ausgeführt.

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

REP

YES

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

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

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

NO
Es werden keine REPs erstellt.

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

SYMCHARS

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

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

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

OV

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

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

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

LOW

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

OFF
: Alle Kleinbuchstaben aus Benutzereingaben werden in Großbuchstaben umgesetzt.

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

DELIM

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

|
-
: Der Standard-Begrenzer ist der senkrechte Strich.

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

LANG

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

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

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

%BASE

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

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

Kommando	Operand
%BASE	[basis]

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

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

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

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

%BASE verändert den Programmzustand nicht.

`basis`

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

```
basis-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -

E = { VM }
    {    }
    { Dn }

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

E=VM

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

E=Dn

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

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

%CONTINUE

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

Kommando	Operand
%CONT[INUE]	

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

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

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

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

%CONTINUE verändert den Programmzustand.

%CONTROLn

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

%CONTROL kann nur für strukturierte Assembler-Programme mit Aufrufen der vordefinierten Makros verwendet werden. Zusätzlich dürfen diese Programme nur aus einem Programmabschnitt (CSECT) bestehen. Für Assembler-Programme, die nicht mit vordefinierten Makros erstellt sind und/oder die mehr als einen Programmabschnitt enthalten, ist %CONTROL nicht zulässig. Für solche Assembler-Programme muß das %CONTROL-Kommando auf Maschinencode-Ebene verwendet werden (siehe AID, Testen auf Maschinencode-Ebene [2]).

– Mit *kriterium* wählen Sie verschiedene Typen von Assembler-Instruktionen aus. Steht eine Instruktion des gewählten Typs zur Ausführung an, unterbricht AID das Programm und bearbeitet *subkdo*.
– Mit *control-bereich* legen Sie den Programmbereich fest, in dem *kriterium* überwacht werden soll.
– Mit *subkdo* definieren Sie ein Kommando oder eine Kommandofolge und eventuell eine Bedingung. Bei zutreffendem *kriterium* und erfüllter Bedingung wird *subkdo* ausgeführt. *subkdo* muß angegeben werden.

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

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

Der einzelne Operandenwert eines %CONTROLn gilt solange, bis Sie ihn durch neue Angaben in einem späteren %CONTROLn mit derselben Nummer überschreiben, bis Sie den %CONTROLn löschen oder bis zum Programmende.
Mit %REMOVE können Sie einen einzelnen oder alle aktiven %CONTROL-Vereinbarungen löschen.

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

%CONTROLn verändert den Programmzustand nicht.

`kriterium`

Schlüsselwort, das die Assembler-Instruktionen festlegt, vor deren Ausführung AID *subkdo* bearbeiten soll.
Sie können mehrere Schlüsselwörter gleichzeitig angeben, die dann gemeinsam wirken. Zwischen zwei Schlüsselwörtern muß ein Komma stehen.
Wird kein *kriterium* vereinbart, arbeitet AID mit dem Standardwert %STMT, falls nicht noch aus einem vorhergehenden %CONTROLn eine *kriterium*-Vereinbarung gültig ist.

kriterium	*subkdo* wird ausgeführt vor
%CALL	dem vordefinierten Makro @PASS (Assembler-Prozeduraufruf)
%COND	den vordefinierten Makros für Auswahl-Strukturblöcke @IF, @THEN, @ELSE, @CASE, @BEGI, @CAS2, @OF, @OFRE
%GOTO	den vordefinierten Makros @BREA und @EXIT
%PROC	dem vordefinierten Makro @ENTR (Assembler-Prozeduranfang)
%STMT	jedem vordefinierten Makro, der ausgeführt wird.

`control-bereich`

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

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

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

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

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

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

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

PROG=program-name
: *program-name* ist der Name einer Programmeinheit.
  Diese Programmeinheit muß zum Zeitpunkt der Eingabe des %CONTROLn bzw. bei der Abarbeitung des Subkommandos, in dem der %CONTROLn enthalten ist, geladen sein.

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

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

(S'stmt-nr' : S'stmt-nr')
: ist eine Source-Referenz über die jeder Aufruf eines vordefinierten Makro (@-Makro) angesprochen werden kann.
  stmt-nr ist die Statementnummer aus dem Übersetzungsprotokoll; siehe Spalte STMNT.

  control-bereich wird festgelegt durch die Angabe einer Anfangs- und einer Ende-*stmt-nr* und umfaßt damit einen Teil des Quellprogramms.

  Die Anfangs-*stmt-nr* muß kleiner sein als die Ende-*stmt-nr*.

  Soll *control-bereich* nur eine Zeile umfassen, müssen Anfangs- und Ende-*stmt-nr* gleich sein.

subkdo

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

subkdo muß angegeben werden, da AID bei %CONTROLn kein <%STOP> einsetzt.

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

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

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

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

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

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

Im *subkdo* eines %CONTROLn sind zusätzlich zu den Kommandos, die in allen Subkommandos nicht zugelassen sind, die AID-Kommandos %CONTROLn, %INSERT 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'250') <%DISPLAY ZAEHLER;%STOP>`
   `%C1 %CALL,%PROC IN(S'123':S'250') <%D ZAEHLER;%STOP>`

   Die beiden AID-Kommandos unterscheiden sich nur in der Schreibweise.
   Das erste Beispiel ist voll ausgeschrieben und enthält unterschiedlich viele Leerzeichen an den zulässigen Stellen, das zweite ist abgekürzt.

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

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

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

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

4. `%C3 %PROC <%STOP>`
   Mit dem %C3 wird vereinbart, daß AID das Programm anhalten soll, bevor die erste Instruktion einer Assembler-Prozedur (@ENTR) ausgeführt wird.

%DISASSEMBLE

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

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

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

Kommando	Operand
%DISASSEMBLE %DA	[anzahl] [FROM start]

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

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

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

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

%DISASSEMBLE verändert den Programmzustand nicht.

`anzahl`

gibt an, wieviele Assemblerbefehle ausgegeben werden sollen.
Wird für *anzahl* kein Wert angegeben und ist auch aus einem vorherigen %DISASSEMBLE kein Wert gültig, so setzt AID den Standardwert 10 ein.

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

`start`

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

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

                ⌈program-name      ⌉
                |L'name'            |
FROM  [•][qua•] {S'stmt-nr'         }
                |                   |
                ⌊kompl-speicherref  ⌋

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

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

qua
: Eine oder mehrere Qualifikationen geben Sie an, wenn *start* nicht im aktuellen AID-Arbeitsbereich liegt.

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

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

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

L'name'
: ist ein Anweisungsname und bezeichnet die Adresse einer ausführbaren Assembler-Instruktion oder eines Aufrufs für einen vordefinierten Makro.
name ist der Namenseintrag einer Assembler-Instruktion oder eines Aufrufs für einen vordefinierten Makro (@-Makro).

  Mit dieser Angabe legen Sie *start* auf den Maschinencode, der zu einer Assembler-Instruktion generiert wurde.

  name kann auch ohne L'...' angegeben werden, da in diesem Kommando eine Verwechslung mit einem Datennamen nicht möglich ist.

S'stmt-nr'
: ist eine Source-Referenz über die jede ausführbare Assembler-Instruktion mit Namen und jeder Aufruf eines vordefinierten Makro angesprochen werden kann.
stmt-nr ist die Statementnummer aus dem Übersetzungsprotokoll; siehe Spalte STMNT.

  Mit dieser Angabe legen Sie *start* auf den Maschinencode, der zu einer Assembler-Instruktion generiert wurde.

kompl-speicherref
: sollte die Anfangsadresse eines Maschinenbefehls sein, andernfalls erhalten Sie eine unsinnige Disassemblierung. Folgende Operationen können darin vorkommen (siehe AID-Basishandbuch Kapitel 6):
  - Adreßversatz (•)
  - indirekte Adressierung (->)
  - Typmodifikation (%A)
  - Längenmodifikation (%Ln)
  - Adreßselektion (%@(...))

  Einen Anweisungsnamen *L'name'* oder eine Source-Referenz *S'stmt-nr'* können Sie innerhalb von *kompl-speicherref* nur in Verbindung mit dem Pointer-Operator benutzen, z.B. L'name' ->.4

Eine Typmodifikation ist nur sinnvoll, wenn der Inhalt eines Datenfelds als Adresse eingesetzt werden kann oder wenn Sie die Adresse aus einem Register entnehmen, z.B. %3G.2 %AL2 ->

Ausgabe des %DISASSEMBLE-Protokolls

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

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 Assemblerformat als Sedezimalzahlen dargestellt werden,
- für Speicherinhalt, der nicht mit einem gültigen Operationscode beginnt, wird eine Assembleranweisung DC im Sedezimalformat mit der Länge von 2 Bytes aufgebaut, dem der Hinweis INVALID OPCODE folgt,
- sedezimale Darstellung des Speicherinhalts (Maschinencode).

Beispiel zum Zeilenaufbau mit T=MAX

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

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

Beispiel zum Zeilenaufbau mit T=MIN

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

Beispiele

1. `%DISASSEMBLE 20 FROM AID_DISPLAY`
   Das Kommando veranlaßt die Rückübersetzung von 20 Befehlen ab der Adresse des ersten ausführbaren Befehls nach dem Namenseintrag: AID_DISPLAY.

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

3. `%DA FROM S'123'`
   Da für *anzahl* kein Wert angegeben wurde, setzt AID entweder den Standardwert 10 ein, wenn es der erste %DISASSEMBLE für dieses Programm ist, oder übernimmt den Wert aus dem vorherigen %DISASSEMBLE.
   Die Rückübersetzung beginnt mit dem ersten Befehl, der zur Instruktion mit der Statementnummer 123 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 oder Längen, Anweisungen, Register, Durchlaufzähler von Subkommandos und Systeminformationen, Sie definieren AID-Literale, oder Sie steuern den Vorschub nach SYSLST.
- Mit *medium-u-menge* geben Sie an, welche Ausgabe-Medien AID verwenden soll und ob Zusatzinformationen ausgegeben werden sollen. Mit diesem Operanden setzen Sie eine mit %OUT getroffene Vereinbarung für das aktuelle %DISPLAY-Kommando außer Kraft.

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

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

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

Mit %DISPLAY %SORTEDMAP erhalten Sie eine nach Namen und Adressen sortierte Liste aller CSECTs des Programms.
Neben den hier beschriebenen Operandenwerten können Sie auch die im Handbuch für das Testen auf Maschinencode-Ebene beschriebenen Operandenwerte einsetzen.

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

%DISPLAY verändert den Programmzustand nicht.

`daten`

beschreibt, welche Informationen AID ausgeben soll. Sie können sich Inhalt, Adresse und Länge von Konstanten und Datenfeldern sowie die Adresse von Assembler-Instruktionen oder Aufrufen für vordefinierte Makros ausgeben lassen. Den Inhalt von Registern und Durchlaufzählern sowie für Ihr Programm relevante Systeminformationen können Sie über Schlüsselwörter adressieren. Register können Sie auch über im Quellprogramm vordefinierte Namen ansprechen. Um die Protokolle Ihrer Tests übersichtlicher zu gestalten, können Sie AID-Literale definieren oder den Vorschub nach SYSLST steuern.

Daten bereitet AID entsprechend der Definition im Quellprogramm auf, wenn Sie nicht mit Typmodifikation einen anderen Ausgabetyp festlegen (siehe AID-Basishandbuch, Abschnitt 6.8). Paßt der Inhalt nicht zum definierten Speichertyp, wird die Ausgabe mit einer Fehlermeldung abgelehnt. Sie können sich den Inhalt des Datenfeldes trotzdem ansehen, z.B. indem Sie sich durch die Typmodifikation %X den Inhalt sedezimal aufbereiten lassen.
Die Änderung des Ausgabetyps mit dem Operanden AS {BIN/CHAR/DEC/DUMP/HEX} wird nur noch in dieser Version unterstützt und ist im AID-Basishandbuch, Anhang beschrieben.

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

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

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

⎡               ⎧datenname         ⎫          ⎤
⎢               ⎪L’name’           ⎪          ⎥
⎢  [•][qua•]    ⎨S’stmt-nr’        ⎬          ⎥
⎢               ⎪schlüsselwort     ⎪          ⎥
⎢               ⎩kompl-speicherref ⎭          ⎥
⎢                                             ⎥
⎢  ⎧%@⎫              ⎧datenname         ⎫     ⎥
⎨  ⎨  ⎬ ([•][qua•]   ⎨                  ⎬)    ⎬
⎢  ⎩%L⎭              ⎩kompl-speicherref ⎭     ⎥
⎢                                             ⎥
⎢  %L=(ausdruck)                              ⎥
⎢                                             ⎥
⎢  AID-literal                                ⎥
⎢                                             ⎥
⎣  vorschubsteuerung                          ⎦

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

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

qua
Eine oder mehrere Qualifikationen geben Sie nur an für Speicherobjekte, die nicht im aktuellen AID-Arbeitsbereich liegen.

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

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

datenname
ist der im Quellprogramm definierte Name von Konstanten, Datenfeldern, vordefinierten Mehrzweckregistern, Programmabschnitten, Pseudoabschnitten, externen Pseudoabschnitten, Pseudoregistern und gemeinsamen Hilfsabschnitten.
datenname ist der Namenseintrag einer DC-, DS-, EQU-, CSECT-, DSECT-, XDSEC-, DXD- und COM-Anweisung (siehe Kapitel 3).

L’name’
ist ein Anweisungsname und bezeichnet die Adresse einer ausführbaren Assembler-Instruktion oder eines Aufrufs für einen vordefinierten Makro.
name ist der Namenseintrag einer Assembler-Instruktion oder eines Aufrufs für einen vordefinierten Makro (@-Makro).

Geben Sie *L’name’* ohne Pointer-Operator an, so wird die entsprechende Adresse in sedezimaler Darstellung ausgegeben. Mit Pointer-Operator, also mit %DISPLAY L’name’->, gibt AID 4 Bytes des an der entsprechenden Adresse stehenden Maschinencodes aus.

S'stmt-nr'

ist eine Source-Referenz über die jede ausführbare Assembler-Instruktion mit Namen und jeder Aufruf eines vordefinierten Makro angesprochen werden kann. *stmt-nr* ist die Statementnummer aus dem Übersetzungsprotokoll; siehe Spalte STMNT.

Geben Sie *S'stmt-nr'* ohne Pointer-Operator an, so wird die entsprechende Adresse in sedezimaler Darstellung ausgegeben. Mit Pointer-Operator, also mit %DISPLAY S'stmt-nr'->, gibt AID 4 Bytes des an der entsprechenden Adresse stehenden Maschinencodes aus.

schlüsselwort

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

```
%n                    Mehrzweckregister, 0 ≤ 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 Gleitpunkt-Register mit doppelter Genauigkeit
                      in Tabellenform aufbereitet

%PC                   Befehlszähler (Program Counter)
%CC                   Condition Code
%PCB                  Process Control Block
%PCBLST               Liste aller Process Control Blocks
%SORTEDMAP            Liste aller CSECTs des Benutzerprogramms
                      (namen- und adreßsortiert)
%IFR                  Interrupt Flag Register
%IMR                  Interrupt Mask Register
%ISR                  Interrupt Status Register
%PM                   Program Mask
%AMODE                Adressierungsmodus des Testobjekts
%AUD1                 P1-Audit-Tabelle, wenn vorhanden auch SAVE-Tabelle

%•subkdoname          Durchlaufzähler
%•                    Durchlaufzähler des gerade aktiven Subkommandos

%HLLOC(speicherref)   Lokalisierungsinformation auf symbolischer Ebene
                      für eine Speicherreferenz im ausführbaren Teil
                      des Programms (High-Level-Location)
%LOC(speicherref)     Lokalisierungsinformation auf Maschinencode-Ebene
                      für eine Speicherreferenz im ausführbaren Teil
                      des Programms (Low-Level-Location)
```

kompl-speicherref

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

- Adreßversatz (•)
- indirekte Adressierung (->)
- Typmodifikation (%T(datenname), %X, %C, %P, %D, %F, %A)
- Längenmodifikation (%L(...), %L=(ausdruck), %Ln)
- Adreßselektion (%@(...))

Nach Adreßversatz oder indirekter Adressierung gibt AID den Speicherinhalt an der errechneten Adresse standardmäßig im Dump-Format und der Länge 4 aus (%XL4). Mit der Typmodifikation können Sie sich *daten* in jeder gewünschten Aufbereitung ausgeben lassen, vorausgesetzt der Inhalt von *daten* paßt zum angegebenen Speichertyp. Mit %X können Sie sich jedes Datenfeld stets in sedezimaler Darstellung ausgeben lassen, unabhängig davon, wie es im Quellprogramm definiert ist und welchen Inhalt es hat.
Mit der Längenmodifikation können Sie die Ausgabelänge selbst bestimmen, z.B. wenn Sie nur Teile eines Datenfeldes oder ein Datenfeld in der Länge eines anderen Datenfeldes ausgeben lassen wollen.

%@(...)

Mit dem Adreßselektor können Sie sich die Adresse eines Datenfeldes oder von *kompl-speicherref* ausgeben lassen.
Der Adreßselektor läßt sich nicht auf symbolische Konstanten anwenden, dazu gehören auch die Anweisungsnamen *L'name'* und die Source-Referenzen *S'stmt-nr'*.

Beispiel

```
%DISPLAY %@(AFELD)
```

Die Adresse von AFELD wird ausgegeben.

%L(...)

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

Beispiel

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

Die Länge von AFELD wird ausgegeben.

%L=(ausdruck)

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

Beispiel

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

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

AID-literal

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

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

vorschubsteuerung

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

%NP bewirkt einen Seitenvorschub

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

```
medium-u-menge
```

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

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

⎧ T  ⎫   ⎧ MAX ⎫
⎪ H  ⎪ = ⎨     ⎬
⎪ Fn ⎪   ⎩ MIN ⎭
⎩ P  ⎭

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

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

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

MAX Ausgabe mit Zusatzinformationen

MIN Ausgabe ohne Zusatzinformationen

Beispiele

1.

```
/%DISPLAY M2,EINGABE,PACKE,_R5

SRC_REF:   212 SOURCE: SUMME     PROC: SUMME ******************************
M2              = |SUMME:|
EINGABE         = 00060000 F9F9
I375  SYMBOL    PACKE NOT FOUND
_R5             = 0000000B
```

Für den Operanden *medium-u-menge* gilt standardmäßig T=MAX.
Jede Ausgabe hat eine AID-Kopfzeile. Sie gibt an, auf welcher Zeile im Quellprogramm das Programm zum Zeitpunkt der Ausgabe unterbrochen ist. Durch einen Schreibfehler kommt es zur Fehlermeldung I375, nach der AID aber noch den letzten Namen _R5 bearbeitet.

Die Definitionen im Assembler-Quellprogramm sahen folgendermaßen aus:

```
M2      DC    C'SUMME:'
EINGABE DC    XL6'00'
PACK    DC    PL2'0'
```

2.

```
/%DISPLAY E=D1.EINGABE,'LETZTER WERT'

** D1: DUMP.NAME.2069.00001 ***********************************************
EINGABE         = 00060000 F2F9
LETZTER WERT
```

3.

```
/%DISPLAY _R10 -> DS

 DS
 DS1             = |ABCDE|
 DS3             =       1234
 DS4             = -.1300000 E+003
```

Für dieses Beispiel wurde das folgende Assembler-Quellprogramm verwendet:

```
CS      START
        USING *,15
        LA    10,DAT1
        TERM
DAT1    DC    CL5'ABCDE'
        DC    CL6'XXXXXX'
        DC    FL4'1234'
        DC    EL4'-1.3E2'
DS      DSECT
DS1     DS    CL5
        DS    CL6
DS3     DS    FL4
DS4     DS    EL4
        END
```

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

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

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

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

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

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

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

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

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

Im Trefferfall wird die Trefferadresse im AID-Register %0G und die Fortsetzungsadresse (Trefferadresse + Suchstringlänge) im AID-Register %1G abgespeichert. Bei der Angabe von *ALL* wird die Adresse des letzten Treffers in %0G und die Fortsetzungadresse des letzten Treffers in %1G abgespeichert. Falls *suchbegriff* nicht gefunden wurde, setzt AID

%0G auf -1; %1G bleibt unverändert.
Die beiden Registerinhalte ermöglichen es Ihnen, das %FIND-Kommando auch in Prozeduren oder Subkommandos einzusetzen und mit den Ergebnissen weiterzuarbeiten.

%FIND verändert den Programmzustand nicht.

`suchbegriff`

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

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

⎧C'x...x' | 'x...x'C | 'x...x'⎫
⎩X'f...f' | 'f...f'X         ⎭

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

`{C'x...x' | 'x...x'C | 'x...x'}`
: Character-Literal
mit einer maximalen Länge von 80 Zeichen. Kleinbuchstaben können nur nach Eingabe von %AID LOW [=ON] als Character-Literal gesucht werden.

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

`{X'f...f' | 'f...f'X}`
: Sedezimal-Literal
mit einer maximalen Länge von 80 Sedezimal-Stellen bzw. 40 Zeichen. Ein Literal mit ungerader Stellenzahl wird rechts mit X'0' ergänzt.

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

`find-bereich`

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

Ist kein *find-bereich* angegeben, so setzt AID den Standardwert %CLASS6 ein (siehe

AID-Basishandbuch, Kapitel 9), d.h. es wird der Klasse-6-Speicher zur aktuell eingestellten Basis-Qualifikation (siehe %BASE) durchsucht.

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

              ⌈datenname          ⌉
              |L'name'->          |
IN [•][qua•] {S'stmt-nr'->        }
              |                   |
              ⌊kompl-speicherref  ⌋

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

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

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

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

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

datenname
: ist der im Quellprogramm definierte Name von Konstanten, Datenfeldern, vordefinierten Mehrzweckregistern, Programmabschnitten, Pseudoabschnitten, externen Pseudoabschnitten, Pseudoregistern und gemeinsamen Hilfsabschnitten.
datenname ist der Namenseintrag einer DC-, DS-, EQU-, CSECT-, DSECT-, XDSEC-, DXD- und COM-Anweisung (siehe Kapitel 3).

L'name'->
: bezeichnet die Speicherstelle an der Adresse einer ausführbaren Assembler-Instruktion oder eines Aufrufs für einen vordefinierten Makro.
name ist der Namenseintrag einer Assembler-Instruktion oder eines Aufrufs für einen vordefinierten Makro (@-Makro).
Ohne Längenmodifikation werden 4 Bytes ab der Adresse durchsucht, die in der Adreßkonstanten *L'name'* hinterlegt ist.

S'stmt-nr'->
: bezeichnet die Speicherstelle über die jede ausführbare Assembler-Instruktion mit Namen und jeder Aufruf eines vordefinierten Makro angesprochen werden kann. *stmt-nr* ist die Statementnummer aus dem Übersetzungsprotokoll; siehe Spalte STMNT.

  Ohne Längenmodifikation werden 4 Bytes ab der Adresse durchsucht, die in der Adreßkonstanten *S'stmt-nr'* hinterlegt ist.

kompl-speicherref
: bezeichnet einen Bereich von 4 Bytes ab der errechneten Adresse. Soll eine andere Anzahl von Bytes durchsucht werden, muß Bei der Längenmodifikation von Datenfeldern müssen Sie die Bereichsgrenzen beachten oder mit %@(datenname)-> auf Maschinencode-Ebene wechseln. Folgende Operationen können in *kompl-speicherref* vorkommen (siehe AID-Basishandbuch, Kapitel 6):
  - Adreßversatz (•)
  - indirekte Adressierung (->)
  - Typmodifikation (%A)
  - Längenmodifikation (%L(...), %L=(ausdruck), %Ln)
  - Adreßselektion (%@(...))

`alignment`

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

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

suchbegriff wird gesucht an:

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 in der Variablen DATA gesucht. Ein Treffer wird auf SYSOUT ausgegeben.

2. `%F X’D2’ IN S’12’->%L=(S’13’-S’12’) ALIGN=2`
   Im für die Anweisung 12 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.

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

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

%HELP verändert den Programmzustand nicht.

`info-ziel`

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

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

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

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

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

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

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

```
medium-u-menge
```

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

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

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

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

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

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

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

%INSERT

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

- Mit *testpunkt* bezeichnen Sie die Adresse eines Befehls im Programm, vor deren Ausführung AID den Programmablauf unterbrechen soll, um *subkdo* zu bearbeiten.
- Mit *subkdo* vereinbaren Sie ein Kommando oder eine Kommandofolge und eventuell eine Bedingung. Wird *testpunkt* erreicht und ist die Bedingung erfüllt, wird *subkdo* ausgeführt.
- Mit *steuerung* vereinbaren Sie, ob *testpunkt* nach einer vorgegebenen Anzahl von Durchläufen gelöscht und ob danach das Programm angehalten werden soll.

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

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

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

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

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

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

%INSERT verändert den Programmzustand nicht.

`testpunkt`

muß die Adresse eines ausführbaren Maschinenbefehls sein, der für eine Assembler-Instruktion 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 - - - - - - - - - - - - - - - - - - - - - - - - - - - -

            ⌈program-name       ⌉
            |L'name'             |
 [•][qua•] {S'stmt-nr'           }
            |                    |
            ⌊kompl-speicherref  ⌋

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

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

qua
Eine oder mehrere Qualifikationen geben Sie nur an, wenn *testpunkt* nicht im aktuellen AID-Arbeitsbereich liegt.

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

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

program-name
Diese Angabe ist nur nach einer expliziten PROG-Qualifikation möglich:
`PROG=program-name•program-name`
Mit der Wiederholung von *program-name* setzen Sie *testpunkt* auf die erste Anweisung der bezeichneten Programmeinheit.

L’name’

ist ein Anweisungsname und bezeichnet die Adresse einer ausführbaren Assembler-Instruktion oder eines Aufrufs für einen vordefinierten Makro.
name ist der Namenseintrag einer Assembler-Instruktion oder eines Aufrufs für einen vordefinierten Makro (@-Makro).

Mit dieser Angabe legen Sie *testpunkt* auf den Maschinencode, der zu einer Assembler-Instruktion generiert wurde.

name darf nicht innerhalb eines Pseudoabschnitts oder eines Pseudoregisters liegen (siehe DSECT-, XDSEC- und DXD-Anweisung, ASSEMBH, Beschreibung [10]).

name kann auch ohne L’...’ angegeben werden, da in diesem Kommando eine Verwechslung mit einem Datennamen nicht möglich ist.

S’stmt-nr’

ist eine Source-Referenz über die jede ausführbare Assembler-Instruktion mit Namen und jeder Aufruf eines vordefinierten Makro angesprochen werden kann.
stmt-nr ist die Statementnummer aus dem Übersetzungsprotokoll; siehe Spalte STMNT.

Mit dieser Angabe legen Sie *testpunkt* auf den Maschinencode, der zu einer Assembler-Instruktion generiert wurde.

kompl-speicherref

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

- Adreßversatz (•)
- indirekte Adressierung (->)
- Typmodifikation (%A)
- Längenmodifikation (%Ln)
- Adreßselektion (%@(...))

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

`subkdo`

wird immer dann bearbeitet, wenn der Programmablauf an die mit *testpunkt* bezeichnete Adresse kommt.

Wird der *subkdo*-Operand nicht angegeben, so setzt AID ein <%STOP> ein.

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

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

                                  ⌈AID-kommando    ⌉
<[subkdoname:] [(bedingung):] [{                   } {;...}]>
                                  ⌊BS2000-kommando ⌋

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

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

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

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

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

`steuerung`

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

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

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

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

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

Sie gibt an, beim wievielten Durchlaufen von *testpunkt* die weiteren Vereinbarungen dieses *steuerung*-Operanden ausgeführt werden sollen.

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

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

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

Beispiele

1. `%IN S'118'`

   testpunkt wird mit einer *stmt-nr* angegeben. Sie bezeichnet die Assembler-Instruktion mit der Statementnummer 118 im Übersetzungsprotokoll.

2. `%IN PROG=PRO2.PRO2 <%DISPLAY NR> ONLY 10 S`

   testpunkt wird auf den Anfang des Moduls PRO2 gesetzt. Immer, wenn der Programmablauf zur ersten Instruktion im Modul PRO2 kommt, wird das %DISPLAY-Kommando aus *subkdo* ausgeführt.

   Wird *testpunkt* zum 10. Mal erreicht, versetzt AID das Programm in den Zustand STOP, löscht den Testpunkt und es können neue Kommandos eingegeben werden.

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

   Mit dem ersten %INSERT wird als *testpunkt* die Assembler-Instruktion mit dem Namen ST2 festgelegt. Kommt nach der Beendigung der Kommando-Eingabe der Programmablauf zu ST2, wird das Subkommando ausgeführt. Es besteht aus einem %DISPLAY für das Feld TEXTDAT und dem AID-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 zur Assembler-Instruktion ST3, wird das zugehörige *subkdo* ausgeführt, d.h. das %DISPLAY-Kommando für das AID-Literal 'INSERT1' und das Feld 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 Programmlauf 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 das Kommando %D 'INSERT2', I, J, K, ANZAHL ausgeführt. Außerdem enthält *subkdo* wieder ein %INSERT-Kommando, dessen *testpunkt* diesmal mit der *stmt-nr* S'172' bezeichnet ist.

Wird im weiteren Programmlauf die mit S'172' bezeichnete Stelle erreicht, führt AID den %DISPLAY für das AID-Literal 'INSERT3' und die Inhalte der Felder 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 *subkdo's* 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. %IN ST4  <%D TEXTDAT>
            .
            .
            .
   %IN ST4
```

An diesen beiden Kommandos soll gezeigt werden, welche Wirkung die Kettung in Verbindung mit dem von AID eingesetzten Default-Wert für *subkdo* hat.

Für das fehlende *subkdo* im zweiten %INSERT setzt AID folgendes *subkdo* ein:
`<%STOP>`
Da der zweite %INSERT denselben *testpunkt* bezeichnet, kommt es zur Kettung, bei der das *subkdo*
`<%STOP; %DISPLAY TEXTDAT>`
entsteht. Ein *subkdo* wird durch einen %STOP, %RESUME oder %TRACE abgebrochen.
Deshalb erzielen Sie mit den zwei aufeinanderfolgenden %INSERT dieselbe Wirkung, als ob Sie mit einem
`%REMOVE ST4`
den ersten %INSERT gelöscht und dann `%INSERT ST4` oder nur einmal `%INSERT ST4` geschrieben hätten.

%MOVE

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

- Mit *sender* bezeichnen Sie ein Datenfeld, einen Anweisungsnamen, eine Source-Referenz, eine Länge, eine Adresse, einen Durchlaufzähler, ein Register 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 oder ein Register, das überschrieben werden soll. *empfänger* kann nur im virtuellen Speicher des geladenen Programms liegen.
- Mit *REP* geben Sie an, ob AID zu einer durchgeführten Änderung einen REP-Satz erzeugen soll. Mit diesem Operanden setzen Sie eine mit dem Kommando %AID vereinbarte Voreinstellung für das aktuelle %MOVE-Kommando außer Kraft.

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

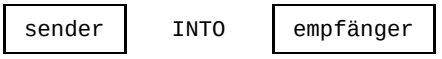
Im Gegensatz zu %SET überprüft AID beim %MOVE nicht die Verträglichkeit der Speichertypen von *sender* und *empfänger* und paßt *sender* nicht an den Speichertyp von *empfänger* an.

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

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.

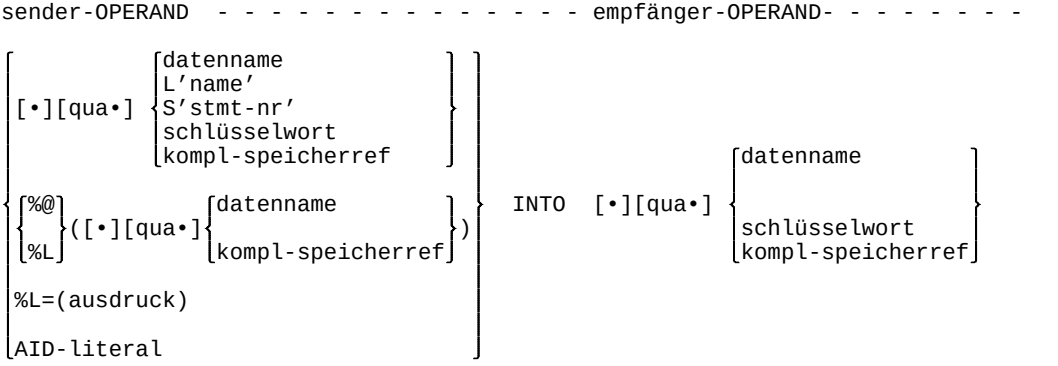


Für *sender* oder *empfänger* können Sie ein Datenfeld oder eine komplexe Speicherreferenz, einen Durchlaufzähler oder ein Register angeben. Anweisungsnamen und Source-

Referenzen, 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; *empfänger* kann dagegen nur im virtuellen Speicherbereich des geladenen Programms liegen.

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



- •

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

qua

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

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

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

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

datenname
: ist der im Quellprogramm definierte Name von Konstanten, Datenfeldern, vordefinierten Mehrzweckregistern, Programmabschnitten, Pseudoabschnitten, externen Pseudoabschnitten, Pseudoregistern und gemeinsamen Hilfsabschnitten.
datenname ist der Namenseintrag einer DC-, DS-, EQU-, CSECT-, DSECT-, XDSEC-, DXD- und COM-Anweisung (siehe Kapitel 3).

L'name'
: ist ein Anweisungsname und bezeichnet die Adresse einer ausführbaren Assembler-Instruktion oder eines Aufrufs für einen vordefinierten Makro.
name ist der Namenseintrag einer Assembler-Instruktion oder eines Aufrufs für einen vordefinierten Makro (@-Makro).

S'stmt-nr'
: ist eine Source-Referenz über die jede ausführbare Assembler-Instruktion mit Namen und jeder Aufruf eines vordefinierten Makro angesprochen werden kann.
stmt-nr ist die Statementnummer aus dem Übersetzungsprotokoll; siehe Spalte STMNT.

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

Beispiel

```
%MOVE S'5' INTO %0G
```

Die Adresse der Anweisung mit der Statementnummer 5 wird in das AID-Register %0G geschrieben.

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

Beispiel

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

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

schlüsselwort

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

```
%•subkdoname       Durchlaufzähler
%•                 Durchlaufzähler des gerade aktiven Subkommandos
%PC                Befehlszähler (Program Counter)
%n                 Mehrzweckregister, 0 ≤ n ≤ 15
%nD|E              Gleitpunktregister, n = 0,2,4,6
%nQ                Gleitpunktregister, n = 0,4
%nG                AID-Mehrzweckregister, 0 ≤ n ≤ 15
%nDG               AID-Gleitpunktregister, n = 0,2,4,6
```

kompl-speicherref

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

- Adreßversatz (•)
- indirekte Adressierung (->)
- Längenmodifikation (%L(...), %L=(ausdruck), %Ln)
- Adreßselektion (%@(...))

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

Beispiel

```
%0G.2%AL2->
```

Die letzten beiden Bytes des AID-Registers %0G sollen als Adresse benutzt werden.

Nach Adreßversatz (•) oder Pointer-Operation (->) gehen impliziter Speichertyp und implizite Länge der Ausgangsadresse verloren. An der errechneten Adresse gilt der Speichertyp %X in der Länge 4, falls Sie nicht Typ und Länge explizit angeben.

Für jeden Operanden in einer komplexen Speicherreferenz darf der zugeordnete Speicherbereich durch einen Adreßversatz oder eine Längenmodifikation nicht überschritten werden, sonst führt AID das Kommando nicht aus und gibt eine Fehlermeldung aus. Durch die Verbindung von Adreßselektion (%@) mit Pointer-Operator (->) verlassen Sie die symbolische Ebene. Nun können Sie die Adresse eines Datenfeldes verwenden, ohne auf dessen Bereichsgrenzen achten zu müssen.

Beispiel

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. Das folgende Kommando würde AID wegen Bereichsverletzung von CFELD ablehnen:

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

Richtig müßte es dagegen heißen:

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

%@(...)

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

%L(...)

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

Beispiel

```
%MOVE %L(FELD1) INTO %0G
```

Die Länge von FELD1 wird übertragen.

%L=(ausdruck)

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

Beispiel

```
%MOVE %L=(FELD1) INTO %0G
```

Der Inhalt von FELD1 wird übertragen.

AID-literal

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

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

REP

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

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

REP = {Y[ES] | NO}

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

REP=Y[ES]

Zu der durch das aktuelle %MOVE-Kommando durchgeführten Änderung werden LMS-UPDR-Korrektursätze (REPs) erstellt. Wenn die Objekt-Strukturliste nicht zur Verfügung steht, erstellt AID keine Korrektursätze und gibt eine Fehlermeldung aus.
Auch wenn *empfänger* nicht vollständig innerhalb einer CSECT liegt, gibt AID eine Fehlermeldung aus und schreibt keinen REP. Um dennoch REP-Sätze zu erhalten, müssen Sie die Übertragung auf mehrere %MOVE-Kommandos verteilen, bei denen Sie die CSECT-Grenzen beachten
(siehe [2]).

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

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

REP=NO

Zum aktuellen %MOVE-Kommando werden keine REPs erstellt.

Beispiele

In einem Programm sind die folgenden Konstanten und Felder definiert:

```
IFELD  DC   F’123,456’
       DS   0F
JFELD  DS   10F
CVAR   DC   X’F0F0F0F0’
```

1. `%MOVE IFELD INTO JFELD`
   AID überträgt den Inhalt von IFELD sedezimal linksbündig an die symbolische Adresse JFELD.

2. `%MOVE 20 INTO JFELD`
   In das Feld JFELD schreibt AID ein Wort, das die Ganzzahl mit dem Wert 20 enthält.

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

%ON

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

- Mit *ereignis* beschreiben Sie eine normale oder abnormale Programmbeendigung, einen Supervisor-Call (SVC), einen Programmfehler oder ein anderes Ereignis, 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	ereignis [<subkdo>]

Wird ein *ereignis* nicht gelöscht, gilt es bis Programmende.

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.

Mit %REMOVE löschen Sie ein Ereignis, eine Ereignisgruppe oder alle aktiven Ereignisse.

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

%ON verändert den Programmzustand nicht.

`ereignis`

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

Wenn mehrere %ON-Kommandos mit unterschiedlichen *ereignis*-Vereinbarungen gleichzeitig aktiv sind und auch zutreffen, bearbeitet AID die zugehörigen Subkommandos in der Reihenfolge, in der die Schlüsselwörter in der folgenden Tabelle aufgeführt sind. Treffen verschiedene %TERM-Ereignisse zu, werden die zugehörigen Subkommandos entgegen der Reihenfolge abgearbeitet, in der die %TERM-Ereignisse erklärt wurden (LIFO-Prinzip wie bei der Verkettung der Subkommandos).
Zur Auswahl der SVC-Nummern siehe Makroaufrufe an den Ablaufteil [7].

ereignis	*subkdo* wird bearbeitet:
%ERRFLG (zzz)	nach Auftreten eines Fehlers mit dem Fehlergewicht 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, Exponenten-Überlaufs oder einer Mantisse Null und vor Abbruch des Programms.
%ABNORM	nach Auftreten eines der Fehler, die mit den vorher beschriebenen Ereignissen erfaßt werden
%ERRFLG	nach Auftreten eines Fehlers mit beliebigem Fehlergewicht.
%SVC(zzz)	vor Ausführung des Systemaufrufs (SVC) mit der angegebenen Nummer.
%LPOV(xxxxxxxx)	nach dem Laden des Segments mit dem angegebenen Namen xxxxxxxx (bis zu 8 alphanum. Zeichen)
%LPOV	nach dem Laden eines beliebigen Segments.
%TERM(N[ORMAL])	vor normaler Beendigung eines Programms
%TERM(A[BNORMAL])	vor abnormaler Beendigung eines Programms, jedoch nach der Ausgabe eines Speicherabzugs
%TERM	vor Beendigung eines Programms durch alle vorher beschriebenen %TERM-Ereignisse
%ANY	vor der Beendigung eines Programms mit %TERM
%SVC	vor Ausführung eines beliebigen Systemaufrufs (SVC).

zzz kann in zwei verschiedenen Formaten angegeben werden:

n maximal dreistellige vorzeichenlose Dezimalzahl

#'ff' zweistellige Sedezimalzahl

Für den Wert von *zzz* gilt: $1 \leq zzz \leq 255$

Es wird nicht überprüft, ob die angegebene Nummer des Fehlergewichts 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 Basishandbuch, Kapitel 5.

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

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

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

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

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

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

Beispiele

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

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

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

4. `%ON %ARTHCHK < %SD EINSATZ,MATRIX; %STOP >` Falls ein Datenfehler/Divisionsfehler/Exponenten-Überlauf oder eine Mantisse gleich Null auftritt, werden durch %SDUMP die Datenfelder EINSATZ und MATRIX ausgegeben. Durch das %STOP-Kommando wird der Programmlauf unterbrochen und Sie können mit weiteren Kommandos die Fehlersituation näher untersuchen.

5. `%ON %LPOV (MONA12) <%D INDEX, HAUPTSUMME> ONLY 37 S` Jedesmal, nachdem das Segment MONA12 geladen wurde, gibt AID mit %D die Datenfelder INDEX und HAUPTSUMME aus. Nachdem das *ereignis* %LPOV (MONA12) zum 37. Mal aufgetreten ist (und die Ausgabe erfolgte), wird das *ereignis* gelöscht und der Programmlauf unterbrochen.

%OUT

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

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

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

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

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

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

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

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

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

%OUT verändert den Programmzustand nicht.

`ziel-kommando`

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

```
⎧%D[IS]A[SSEMBLE]⎫
⎪%D[ISPLAY]      ⎪
⎨%H[ELP]         ⎬
⎪%SD[UMP]        ⎪
⎩%T[RACE]        ⎭
```

```
medium-u-menge
```

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

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

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

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

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

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

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

MAX Ausgabe mit Zusatzinformationen

MIN Ausgabe ohne Zusatzinformationen

Beispiele

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

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

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

%OUTFILE

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

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

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

Kommando	Operand
%OUTFILE	[link [= datei]]

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

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

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

%OUTFILE verändert den Programmzustand nicht.

`link`

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

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

`datei`

gibt den vollqualifizierten Dateinamen an, mit dem AID die Ausgabedatei katalogisiert und 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 Adreß-Operanden eines anderen Kommandos durch Voranstellen eines Punktes beziehen können. Diese verkürzte Schreibweise ist immer dann sinnvoll, wenn Sie mehrfach Adressen ansprechen wollen, die nicht im aktuellen AID-Arbeitsbereich liegen.

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

Kommando	Operand
%Q[UALIFY]	[vorqualifikation]

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

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

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

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

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

%QUALIFY verändert den Programmzustand nicht.

```
vorqualifikation
```

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

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

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

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

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

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

PROG=program-name
: bezeichnet eine Programmeinheit.

Beispiele

1.
```
%Q E=D1.PROG=INITIAL
%D .TAB1
```
Durch die *vorqualifikation* hat der %DISPLAY die gleiche Wirkung wie das folgende, ausgeschriebene %DISPLAY-Kommando: `%D E=D1.PROG=INITIAL.TAB1`

2.
```
%Q PROG=VORLAUF
%SET .E_SATZ INTO .A_SATZ
```
Durch die Verwendung der *vorqualifikation* hat das %SET-Kommando dieselbe Bedeutung wie das folgenden %SET-Kommando:
`%SET PROG=VORLAUF.E_SATZ INTO PROG=VORLAUF.A_SATZ`

%REMOVE

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

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

Kommando	Operand
%REM[OVE]	ziel

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

%REMOVE verändert den Programmzustand nicht.

`ziel`

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

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

┌%C[ONTROL] | %C[ONTROL]n    ┐
│%IN[SERT] | testpunkt       │
{                             }
│%ON | ereignis              │
└%•[subkdoname]               ┘

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

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

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

%IN[SERT]
Alle eingetragenen Testpunkte werden gelöscht.

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

%ON
Alle eingetragenen Ereignisse werden gelöscht.

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

Für die Ereignisse %ERRFLG(zzz), %SVC(zzz) und %LPOV(zzz) gilt:
%REMOVE *ereignis(zzz)* löscht nur das Ereignis mit der angegebenen Nummer.
%REMOVE *ereignis* ohne Angabe einer Nummer löscht alle Ereignisse der entsprechenden Gruppe.

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

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

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

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

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

Beispiele

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

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

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

Wenn der Testpunkt S’58’ 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’58’ kommt, wird Subkommando SUB1 ausgeführt. Mit `%REM S’58’` wird später der Testpunkt gelöscht. Ein `%REM %.SUB1` würde dasselbe bewirken, denn zum Testpunkt S’58’ ist nur noch dieses Subkommando eingetragen.

%RESUME

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

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

Kommando	Operand
%R[ESUME]	

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

%RESUME verändert den Programmzustand.

%SDUMP

Mit %SDUMP geben Sie einen symbolischen Dump aus: einzelne Daten oder Datenbereiche, alle Datenbereiche der aktuellen Aufrufhierarchie oder die Programmnamen der aktuellen Aufrufhierarchie werden ausgegeben. Die aktuelle Aufrufhierarchie reicht von der Unterprogrammebene, auf der das Programm unterbrochen wurde, über die aufgerufenen Unterprogramme bis zum Hauptprogramm.

- Mit *dump-bereich* bezeichnen Sie die Daten oder Datenbereiche, 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][,...]]

Für ein strukturiertes Assembler-Programm gilt:

- Mit %SD %NEST erhalten Sie eine minimale Ausgabemenge. AID listet nur die aktuelle Aufrufhierarchie auf.
- Mit %SD ohne Operanden gibt AID die maximale Ausgabemenge aus. D.h., es werden die Daten mit einem Namen ausgegeben, die in der aktuellen Aufrufhierarchie definiert sind und alle weiteren Assembler-Instruktionen mit einem Namen im Namenseintrag. Dabei werden Daten, die mehrfach definiert sind, auch mehrfach ausgegeben. Eine Überschrift informiert Sie über die Programmeinheit, in dem die Daten definiert wurden.
- Werden ein oder mehrere Namen im Kommando explizit angegeben, so werden alle Daten mit diesem Namen ausgegeben, die in der aktuellen Aufruf-Hierarchie definiert sind. Gleichnamige Daten, die in verschiedenen Moduln definiert sind, werden auch mehrfach ausgegeben.

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

In Assembler-Programmen, die nicht strukturiert programmiert sind, können Sie nur die Daten der aktuellen Programmeinheit ansprechen:

– Mit %SD ohne Operanden werden der gesamte Datenbereich und alle weiteren Assembler-Instruktionen mit einem Namen ausgegeben.

– Werden ein oder mehrere Namen im Kommando explizit angegeben, so werden auch hier nur die betreffenden Daten der aktuellen Programmeinheit ausgegeben.

dump-bereich können Sie bis zu 7mal wiederholen.

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

%SDUMP verändert den Programmzustand nicht.

`dump-bereich`

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

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

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

⎧       ⎧VM ⎫      ⎧[PROG=program-name[•]] [datenname]⎫
⎨[•][E=⎨   ⎬[•]] ⎨                                  ⎬
⎩       ⎩Dn ⎭      ⎩%NEST                            ⎭

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

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

E ={VM | Dn}

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

PROG=program-name

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

datenname

ist der im Quellprogramm definierte Name von Konstanten, Datenfeldern, vordefinierten Mehrzweckregistern, Programmabschnitten, Pseudoabschnitten, externen Pseudoabschnitten, Pseudoregistern und gemeinsamen Hilfsabschnitten. *datenname* ist der Namenseintrag einer DC-, DS-, EQU-, CSECT-, DSECT-, XDSEC-, DXD- und COM-Anweisung (siehe Kapitel 3).

Für *datenname*, der in den LSD-Sätzen nicht verzeichnet ist, gibt AID eine Fehlermeldung aus. Nachfolgende *dump-bereiche* desselben Kommandos werden ordungsgemäß ausgegeben.

%NEST

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

```
medium-u-menge
```

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

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

⎧ T  ⎫
⎪ H  ⎪   = ⎧ MAX ⎫
⎨    ⎬     ⎨     ⎬
⎪ Fn ⎪     ⎩ MIN ⎭
⎩ P  ⎭

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

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

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

MAX Ausgabe mit Zusatzinformationen

MIN Ausgabe ohne Zusatzinformationen

Beispiel

Das Programm SUMME aus der PLAM-Bibliothek PLAMBIB soll zum Ablauf gebracht werden.

```
/START-PROG (PLAMBIB,SUMME),TEST-OPTION=AID

%  BLS0001 *** DBL VERSION 070 RUNNING ***
%  BLS0517 MODULE 'SUMME' LOADED

BITTE BIS ZU 10 2-STELLIGE ZAHLEN EINGEBEN. ENDE: 00
*29
*37
*00
*
/
```

Das Programm verzweigt wieder zur Eingabe, obwohl das Endekriterium '00' eingegeben wurde. Es liegt ein Programmfehler vor. Durch Drücken der K2-Taste wird das Programm unterbrochen und mit %SDUMP ein Speicherabzug angefordert. Mit diesem Kommando wird ein symbolischer Dump des gesamten Moduls angefordert. Der Wert für *medium-u-menge* ist T=MAX. Das Quellprogramm zu dieser %SDUMP-Ausgabe finden Sie im Kapitel 6. Den verschiedenen %SDUMP-Zeilen folgen jeweils unmittelbar beschreibende Texte.

```
/%SDUMP

** ITN: #'000B018F' *** TSN: 4J60 ****************************************
SRC_REF:    60  SOURCE: SUMME     PROC: SUMME  ******************************
```

SRC_REF-Zeile: sie enthält die Statementnummer, bei der das Programm unterbrochen wurde und den Modulnamen.

```
R0              =           0

R1              =           1

R2              =           2

R3              =           3

R4              =           4

R5              =           5
```

Namen der EQU-Anweisungen, denen ein konstanter Wert zugewiesen wurde.

```
SUMME           = 00000000

ANFANG          =    17680

SCHLEIFE        = 5A502126

LESEN           =    17680

VERGL           = D5052105 211E

ADD             = F2112107 2105

AUS             = 0A812115 2120

ENDE            =   169608960

FEHLER          =    17680
```

Assembler-Instruktionen mit einem Namen; AID gibt entsprechend dem Längenmerkmal den Speicherinhalt an der betreffenden Adresse aus.

```
MELD1           = 0039
M1              = |BITTE BIS ZU 10 2-STELLIGE ZAHLEN EINGEBEN! ENDE: 00|
EINGABE         = 00060000 F0F0   ....00
PACK            =     +0
MELD2           = 0012
M2              = |SUMME:|
ERGEB           = |       |
ZEHN            =          10
NULL            = |00|
GESAMT          =        +66
ZONE            = F0
MELD3           = 0034
M3              = |ES KOENNEN MAXIMAL 10 ZAHLEN VERARBEITET WERDEN|
```

Datenbereich des Moduls; ausgegeben werden die Namen der DC- und DS-Anweisungen mit ihrem jeweiligen Speicherinhalt.

```
_R0             = 00000000
_R1             = 9F00003C
_R2             = 00000002
_R3             = 00000000
_R4             = 00000000
_R5             = 00000005
_R6             = 00000000
_R7             = 00000000
_R8             = 00000000
_R9             = 00000000
_R10            = 00000000
_R11            = 00000000
_R12            = 00000000
_R13            = 00000000
_R14            = 00000000
_R15            = 00000000
```

Mehrzweckregister; AID gibt den vordefinierten Namen des Registers aus und den Inhalt.

%SET

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

– Mit *sender* bezeichnen Sie ein Datenfeld, einen Anweisungsnamen, eine Source-Referenz, eine Länge, eine Adresse, einen Durchlaufzähler, ein Register 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 oder ein Register, das überschrieben werden soll. *empfänger* kann nur im virtuellen Speicher des geladenen Programms liegen.

Kommando	Operand
%S[ET]	sender INTO empfänger

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

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

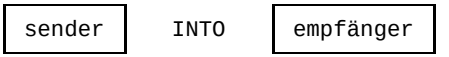
Welche Speichertypen miteinander verträglich sind und wie übertragen wird, können Sie der Tabelle am Ende der %SET-Beschreibung entnehmen.

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

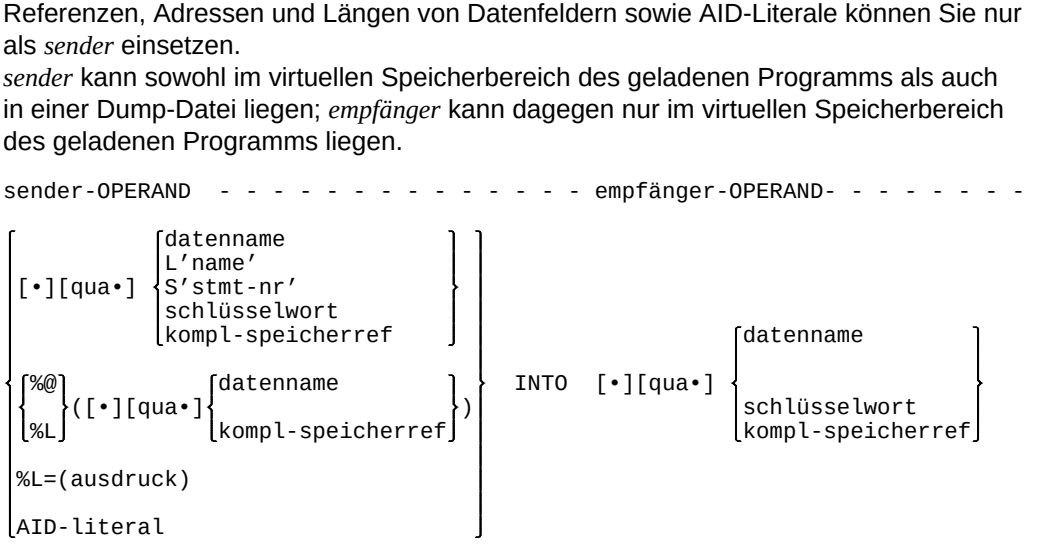
Neben den hier beschriebenen Operandenwerten können Sie auch die im Handbuch für das Testen auf Maschinencode-Ebene beschriebenen Operandenwerte einsetzen.

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.



Für *sender* oder *empfänger* können Sie Datenfelder oder eine komplexe Speicherreferenz, einen Durchlaufzähler oder ein Register angeben. Anweisungsnamen und Source-Referenzen, 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; *empfänger* kann dagegen nur im virtuellen Speicherbereich des geladenen Programms liegen.



- •

  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 nur an für Speicherobjekte, die nicht im aktuellen AID-Arbeitsbereich liegen.

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

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

empfänger kann dagegen nur im virtuellen Speicher liegen.

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

datenname
: ist der im Quellprogramm definierte Name von Konstanten, Datenfeldern, vordefinierten Mehrzweckregistern, Programmabschnitten, Pseudoabschnitten, externen Pseudoabschnitten, Pseudoregistern und gemeinsamen Hilfsabschnitten.
datenname ist der Namenseintrag einer DC-, DS-, EQU-, CSECT-, DSECT-, XDSEC-, DXD- und COM-Anweisung (siehe Kapitel 3).

L’name’
: ist ein Anweisungsname und bezeichnet die Adresse einer ausführbaren Assembler-Instruktion oder eines Aufrufs für einen vordefinierten Makro.
name ist der Namenseintrag einer Assembler-Instruktion oder eines Aufrufs für einen vordefinierten Makro (@-Makro).

S’stmt-nr’
: ist eine Source-Referenz über die jede ausführbare Assembler-Instruktion mit Namen und jeder Aufruf eines vordefinierten Makro angesprochen werden kann.
stmt-nr ist die Statementnummer aus dem Übersetzungsprotokoll; siehe Spalte STMNT.

Anweisungsnamen und Source-Referenzen sind Adreß-Konstanten und können daher nur als *sender* angegeben werden. Die mit *L’name’* bzw. *S’stmt-nr’* bezeichnete Adresse wird übertragen.

Beispiel

```
%SET S’5’ INTO %0G
```

Die Adresse der Anweisung mit der Statementnummer 5 wird in das AID-Register %0G geschrieben.

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

schlüsselwort

ist ein Durchlaufzähler, der Befehlszähler oder ein Register. Im AID-Basishandbuch, Kapitel 9 sind die impliziten Speichertypen der Schlüsselwörter angegeben.

Vor *schlüsselwort* können Sie nur eine Basis-Qualifikation 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, Kapitel 6):

- Adreßversatz (•)
- indirekte Adressierung (->)
- Typmodifikation (%T(datenname), %X, %C, %D, %P, %F, %A)
- Längenmodifikation (%L(...), %L=(ausdruck), %Ln)
- Adreßselektion (%@(...))

Mit einer expliziten Typ- oder Längenmodifikation können Sie die Speichertypen von *sender* und *empfänger* einander anpassen. Mit einem Speichertyp unvereinbare Speicherinhalte jedoch werden von AID auch bei der Typmodifikation abgelehnt (siehe AID-Basishandbuch, Abschnitt 6.8).
Nach Adreßversatz (•) oder Pointer-Operation (->) gehen impliziter Speichertyp und implizite Länge der Ausgangsadresse verloren. An der errechneten Adresse gilt der Speichertyp %X in der Länge 4, falls Sie nicht Typ und Länge explizit angeben. Für jeden Operanden in einer komplexen Speicherreferenz darf der zugeordnete Speicherbereich durch einen Adreßversatz oder eine Längenmodifikation nicht überschritten werden, sonst führt AID das Kommando nicht aus und schreibt eine Fehlermeldung. Durch die Verbindung von Adreßselektion (%@) mit Pointer-Operator (->) verlassen Sie die symbolische Ebene. Nun können Sie die Adresse eines Datenfeldes verwenden, ohne auf dessen Bereichsgrenzen achten zu müssen.

Beispiel

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

```
%SET CFELD.3%CL5 INTO CFELD1
```

Richtig müßte es dagegen heißen:

```
%SET %@(CFELD)->.3%CL5 INTO CFELD1
```

%@(...)

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

%L(...)

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

Beispiel

```
%SET %L(FELD1) INTO %0G
```

Die Länge von FELD1 wird übertragen.

%L=(ausdruck)

Mit der Längenfunktion können Sie den Wert von *ausdruck* berechnen und in *empfänger* abspeichern lassen (siehe AID-Basishandbuch, Abschnitte 6.9 und B6.10). In *ausdruck* können Sie Speicherreferenzen und ganze Zahlen mit den arithmetischen Operatoren (+,-,*,/) verknüpfen.
Die Längenfunktion liefert als Ergebnis eine Ganzzahl.

Beispiel

```
%SET %L=(FELD1) INTO %0G
```

Der Inhalt von FELD1 wird übertragen.

AID-literal

Alle im AID-Basishandbuch, Kapitel 8 beschriebenen AID-Literale können Sie angeben. Bitte beachten Sie die dort beschriebenen Konvertierungen der AID-Literale an den jeweiligen *empfänger*:

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

%SET-Tabelle

Sendefeld	Empfangsfeld Daten des Typs: A	B	C	D	E	F	H	L	P	Q	S	V	X	Y	Z	Reg. _Rn
Daten des Typs: A	2	-	-	-	-	-	-	-	-	-	-	2	-	2	-	2
B	-	1	-	-	-	-	-	-	-	1	1	-	1	-	-	-
C	-	-	2a	-	-	-	-	-	-	-	-	-	-	-	-	-
D	-	-	-	2	2	4a*	4a*	2	-	-	-	-	-	-	4c*	-
E	-	-	-	2	2	4a*	4a*	2	-	-	-	-	-	-	4c*	-
F	-	-	-	4b	4b	1	1	4b	-	-	-	-	-	-	4c	-
H	-	-	-	4b	4b	1	1	4b	-	-	-	-	-	-	4c	-
L	-	-	-	2	2	4a*	4a*	2	-	-	-	-	-	-	4c*	-
P	-	-	-	4b	4b	4a	4a	4b	-	-	-	-	-	-	-	-
Q	-	1	-	-	-	-	-	-	-	1	1	-	1	-	-	-
S	-	1	-	-	-	-	-	-	-	1	1	-	1	-	-	-
V	2	-	-	-	-	-	-	-	-	-	-	2	-	2	-	2
X	-	1	-	-	-	-	-	-	-	1	1	-	1	-	-	-
Y	2	-	-	-	-	-	-	-	-	-	-	2	-	2	-	2
Z	-	-	-	4b	4b	4a	4a	4b	-	-	-	-	-	-	3	-
Register:_Rn	2	-	-	-	-	-	-	-	-	-	-	2	-	2	-	2
AID-Literale alphanum.	-	-	2a	-	-	-	-	-	-	-	-	-	-	-	-	-
alphanum. sedezimal	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
Bit	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
numerisch	-	-	-	4a	1	1	4a	4a	-	-	-	-	-	-	4c	-
numerisch sedezimal	1	-	-	2b	2b	1	1	2b	-	-	-	1	-	1	-	1
Gleitpunkt	-	-	-	2	2	4a*	4a*	2	-	-	-	-	-	-	4c	-

Obige Tabelle gibt eine Übersicht über die zulässigen Kombinationen von Sende- und Empfangsfeld-Typen (siehe auch ASSEMBH, Beschreibung [10], "DC- und DS-Anweisung, Konstantentypen").

Bedeutung von 1, 2, 2a, 2b, 3, 4a-c, *, -

1 Das Empfangsfeld wird rechtsbündig mit dem Inhalt des Sendefeldes überschrieben. Bei unterschiedlicher Länge wird links mit X'00' aufgefüllt, bzw. links abgeschnitten.

2 Das Empfangsfeld wird linksbündig mit dem Inhalt des Sendefeldes überschrieben. Bei unterschiedlicher Länge wird rechts mit X'00' aufgefüllt, bzw. rechts abgeschnitten.

2a
Das Empfangsfeld wird linksbündig mit dem Inhalt des Sendefeldes überschrieben. Bei unterschiedlicher Länge wird rechts mit X'40' aufgefüllt, bzw. rechts abgeschnitten.

2b
Betrifft nur die Mantisse:
Das Empfangsfeld wird linksbündig mit dem Inhalt des Sendefeldes überschrieben. Bei unterschiedlicher Länge wird rechts mit X'00' aufgefüllt, bzw. rechts abgeschnitten.

3 Das Empfangsfeld wird rechtsbündig mit dem Inhalt des Sendefeldes überschrieben. Bei unterschiedlicher Länge wird links mit X'F0' aufgefüllt, bzw. links abgeschnitten.

4 Der Typ des Senders wird in die interne Darstellung des Empfänger-Typs überführt. Bei unterschiedlicher Länge von Sender und umgewandeltem Empfänger wird

- 4a links abgeschnitten, bzw. links mit X'00' aufgefüllt
- 4b rechts abgeschnitten, bzw. rechts mit X'00' aufgefüllt
- 4c links abgeschnitten, bzw. links mit X'F0' aufgefüllt.

* Die Typen F, H, Z dürfen nicht als Empfänger verwendet werden, wenn der Sender vom Typ E, D oder L nicht ganzzahlig ist.

– Übertragung nicht möglich

Beispiele

1. `%SET PROG=PRO1.MELD1 INTO PROG=PRO1.MELD2`
   MELD1 und MELD2 sind in der Programmeinheit PRO1 definiert. Der Inhalt von MELD1 wird nach MELD2 übertragen.

2. `%QUALIFY PROG=PRO1`
   `%SET .MELD1 INTO .MELD2`
   Dieses %SET-Kommando bewirkt dieselbe Übertragung wie in Beispiel 1: Vor den führenden Punkt wird die im %QUALIFY definierte Vorqualifikation übernommen.

3. `%SET 'ABCDEF' INTO MELDUNG`
   Das Datenfeld MELDUNG ist 8 Stellen lang. Nach der Übertragung steht in ihm:
   `ABCDEF␣␣`

4. `%SET #'0' INTO _R5`
   `%SET _R5 INTO _R10`
   Mit dem ersten %SET wird die Sedezimalzahl #'0' in Register 5 übertragen, d.h. Register 5 wird gelöscht. Mit dem zweiten %SET wird auch Register 10 gelöscht.

%STOP

Mit %STOP veranlassen Sie AID, das Programm anzuhalten, in den Kommandomodus zu gehen und eine STOP-Meldung auszugeben. Dieser Meldung können Sie entnehmen, an welcher Anweisung und in welcher Programmeinheit das Programm unterbrochen wurde.

Wird das Kommando am Terminal oder aus einer Prozedurdatei eingegeben, so wird der Programmzustand nicht verändert, da das Programm ja bereits steht. In diesen Fällen können Sie das Kommando anwenden, um mit der STOP-Meldung Lokalisierungsinformation über die Programmunterbrechungsstelle zu erhalten.

Kommando	Operand
%STOP	

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

Wenn 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.

Beispiel

```
/%INSERT S'214' <%DISPLAY M2,GESAMT; %STOP>
/%RESUME

M2               = |SUMME:|
GESAMT           = +.11000000000000000000000000000000 E+002
USTOPPED AT LABEL: ADD , SRC_REF: 214, SOURCE: SUMME ,PROC: SUMME
```

Mit %INSERT wird ein Testpunkt auf die Anweisung mit der Statementnummer 214 gesetzt. Das Subkommando enthält die Kommandos %DISPLAY und %STOP. Nach der Ausgabe von M2 und GESAMT hält AID das Programm an und schreibt eine STOP-Meldung mit Statementnummer und Programmeinheit der aktuellen Unterbrechungsstelle.

%SYMLIB

Mit %SYMLIB veranlassen Sie AID, PLAM-Bibliotheken zu öffnen oder zu schließen. Auf geöffnete PLAM-Bibliotheken greift AID zu, wenn Sie in einem Kommando symbolische Speicherreferenzen ansprechen, die in einer Programmeinheit liegen, zu der AID keine LSD-Sätze geladen hat.

– Mit *qualifikation-u-lib* melden Sie eine oder mehrere Bibliotheken an oder ab, in denen Bindemoduln mit den zugehörigen LSD-Sätzen abgespeichert sind. Sie können jede Bibliothek dem aktuellen Programm oder einer Dump-Datei zum Nachladen der LSD-Sätze zuordnen, indem Sie die entsprechende Basis-Qualifikation dazu angeben.

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

Bei Ausführung dieses Kommandos stellt AID nur fest, ob die angegebene Bibliothek geöffnet werden kann; es überprüft nicht, ob der Inhalt einer Bibliothek zu dem Programm paßt, das gerade bearbeitet wird. Vorbereitend können Sie also die Bibliotheken anmelden, die Sie während eines Testlaufs benötigen. Erst beim Zugriff auf die nachgeladenen LSD-Sätze überprüft AID, ob der Bindemodul des angesprochenen Programms mit dem der PLAM-Bibliothek übereinstimmt.
Sind zu einer Basis-Qualifikation mehrere Bibliotheken angemeldet, so durchsucht AID sie in der Reihenfolge, in der sie im %SYMLIB-Kommando angegeben wurden.
Verläuft die Suche von AID nicht erfolgreich oder ist keine Bibliothek angemeldet, so können Sie nach der entsprechenden Meldung mit einem neuen %SYMLIB-Kommando die richtige Bibliothek zuweisen und dann das Kommando wiederholen, zu dessen Ausführung die LSD-Sätze fehlten.

Eine Bibliothek bleibt solange angemeldet, bis sie durch ein neues %SYMLIB-Kommando zu derselben Basis-Qualifikation oder durch ein %SYMLIB ohne Operand abgemeldet wird oder bis /LOGOFF. Enthält ein neues Kommando neue Dateinamen, dann werden diese Bibliotheken angemeldet und geöffnet.

%SYMLIB verändert den Programmzustand nicht.

`qualifikation-u-lib`

ist eine Basis-Qualifikation und/oder der Dateiname einer PLAM-Bibliothek.

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

AID kann maximal 15 Bibliotheks-Anmeldungen verwalten. Dabei zählt eine Bibliothek, die gleichzeitig mit verschiedenen Basis-Qualifikationen angemeldet ist, so oft, wie sie angegeben wird.

```
qualifikation-u-lib-OPERAND - - - - - - - - - - - - - - - - - - - - - - - -

                ⎧VM ⎫
[•][E=          ⎨   ⎬•][dateiname]
                ⎩Dn ⎭

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

•
: Steht der Punkt an führender Stelle, ist er das Kennzeichen für eine *vorqualifikation*. Sie muß mit einem vorhergehenden %QUALIFY definiert worden sein und kann nur für eine Basis-Qualifikation stehen.

E=VM
: %SYMLIB gilt für das geladene Programm (siehe %BASE).

E=Dn
: %SYMLIB gilt für einen Speicherabzug in einer Dump-Datei mit dem Linknamen *Dn* (siehe %BASE).

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

Beispiel

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

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 ASSOUTPUT wird zur aktuell eingestellten Basis-Qualifikation angemeldet. Wurde bisher kein %BASE gegeben, verwendet AID diese Bibliothek zum Nachladen von LSD-Sätzen zum geladenen Programm.

%TITLE

Mit %TITLE definieren Sie einen eigenen Seitenkopf-Text. Diesen verwendet AID, wenn die Kommandos %DISASSEMBLE, %DISPLAY, %HELP, %SDUMP und %TRACE in die System-Datei SYSLST schreiben.

– Mit *seitenkopf* geben Sie den Text der Kopfzeile an, veranlassen AID, den Seitenzähler auf 1 zu setzen und vor der nächsten Druckzeile SYSLST auf Seitenanfang zu positionieren.

Kommando	Operand
%TITLE	[seitenkopf]

Mit einem %TITLE ohne *seitenkopf*-Operanden wechseln Sie wieder zur AID-Standard-Überschrift. AID setzt den Seitenzähler wieder auf 1 und positioniert SYSLST vor der nächsten Druckzeile auf Seitenanfang.

Eine mit %TITLE vereinbarte Seitenüberschrift gilt bis zu einem neuen %TITLE oder bis Programmende.

%TITLE verändert den Programmzustand nicht.

`seitenkopf`

gibt den variablen Teil der Seitenüberschrift an. Er wird von AID mit der Uhrzeit, dem Datum und dem Seitenzähler ergänzt.

seitenkopf

ist ein Character-Literal in der Form {C'x...x' | 'x...x'C | 'x...x'} und kann maximal 80 Zeichen lang sein. Ein längeres Literal wird mit einer Fehlermeldung abgewiesen, in der aber nur die ersten 52 Stellen des Literals protokolliert werden.

Auf eine Druckseite werden außer der Seitenüberschrift bis zu 58 Zeilen gedruckt.

%TRACE

Mit %TRACE schalten Sie die AID-Ablaufverfolgung ein und starten das Programm oder setzen es an der unterbrochenen Stelle fort.

%TRACE kann nur verwendet werden für strukturierte Assembler-Programme mit Aufrufen vordefinierter Makros. Zusätzlich dürfen diese Programme nur aus einem Programmabschnitt (CSECT) bestehen. Für Assembler-Programme, die nicht mit vordefinierten Makros erstellt sind und/oder die mehr als einen Programmabschnitt enthalten, ist %TRACE nicht zulässig. Für solche Assembler-Programme muß das %TRACE-Kommando auf Maschinencode-Ebene verwendet werden (siehe AID, Testen auf Maschinencode-Ebene [2]).

– Mit *anzahl* legen Sie fest, wieviele Assembler-Instruktionen maximal verfolgt, d.h. ausgeführt und protokolliert werden sollen.
– Mit *kriterium* wählen Sie verschiedene Typen von Assembler-Instruktionen aus, die AID protokollieren soll. Die Protokollierung erfolgt vor der Ausführung der ausgewählten Instruktionen.
– Mit *trace-bereich* legen Sie den Programmbereich fest, in dem *kriterium* berücksichtigt werden soll.

Kommando	Operand
%T[RACE]	[anzahl] [kriterium][,...] [IN trace-bereich]

Ein %TRACE wird durch fünf verschiedene Ereignisse im Testverlauf beendet:
1. Die maximale Anzahl der zu überwachenden Instruktionen wurde erreicht.
2. Ein Subkommando wurde ausgeführt, weil eine Überwachungsbedingung aus einem %CONTROLn, %INSERT, %ON zutraf, und in diesem Subkommando ist ein %RESUME, %STOP oder %TRACE enthalten.
3. Ein %INSERT endet mit einer Programmunterbrechung, weil der *steuerung*-Operand K oder S lautet.
4. Die K2-Taste wurde gedrückt. Dazu muß am Terminal die SDF-Option `OVERFLOW-CONTROL = USER-ACKNOWLEDGE` eingestellt sein (Kommando `/MODIFY-TERMINAL-OPTIONS`).
5. Das Programm wurde durch den Aufruf des Makro BKPT angehalten.

Ein noch aktiver %TRACE, der durch ein Ereignis, wie unter 2.- 5. beschrieben, unterbrochen wurde, kann mit %CONTINUE fortgesetzt werden.

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 die Programmeinheit, in der die aktuelle Unterbrechungsstelle liegt, für *trace-bereich* ein.

Mit %OUT können Sie steuern, welche Informationen eine Protokollzeile enthält und auf welches Ausgabemedium das Protokoll ausgegeben werden soll.

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

trace-bereich kann nur im geladenen Programm liegen, deshalb muß die Basis-Qualifikation E=VM eingestellt sein (siehe %BASE) oder explizit angegeben werden.

%TRACE verändert den Programmzustand.

`anzahl`

gibt an, wieviele Assembler-Instruktionen vom Typ *kriterium* maximal ausgeführt und protokolliert werden sollen.

anzahl

ist eine Ganzzahl mit $1 \leq anzahl \leq 2^{31}-1$. Standardwert ist 10. Er wird von AID in ein %TRACE-Kommando ohne *anzahl*-Operanden eingesetzt, wenn es keinen Wert aus einem vorhergehenden %TRACE gibt.

Nachdem die vorgegebene *anzahl* von Instruktionen 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 Instruktion und in welcher Programmeinheit das Programm angehalten wurde.

`kriterium`

ist ein Schlüsselwort, das den Typ der Instruktionen 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	Ablaufprotokollierung vor der Ausführung von
%CALL	dem vordefinierten Makro @PASS (Assembler-Prozedur)
%COND	den vordefinierten Makros für Auswahl-Strukturblöcke @IF, @THEN, @ELSE, @CASE, @BEGI, @CAS2, @OF, @OFRE
%GOTO	den vordefinierten Makros @BREA und @EXIT
%STMT	jedem vordefinierten Makro, der ausgeführt wird.

`trace-bereich`

legt den Programmbereich fest, in dem die Ablaufverfolgung stattfinden soll. Nur innerhalb dieses Bereiches werden die mit *kriterium* ausgewählten Anweisungen überwacht und protokolliert. Außerhalb dieses Bereiches ist der %TRACE inaktiv und wird erst bei Rückkehr in den Bereich wieder aktiv.

Eine *trace-bereich*-Definition ist wirksam bis zu einem neuen %TRACE mit eigenem *trace-bereich*-Operanden, einem %TRACE, der außerhalb dieses Bereiches eingegeben wird oder bis zum Programmende. Wird *trace-bereich* nicht angegeben, wird die Bereichsdefinition aus einem vorhergehenden %TRACE übernommen, wenn die aktuelle Unterbrechungsstelle in diesem Bereich liegt. Sonst setzt AID den Standardwert ein, das ist die Programmeinheit, in der die aktuelle Unterbrechungsstelle liegt.

Die Fortsetzungsadresse für den Programmablauf kann mit %TRACE nicht beeinflußt werden.

```
trace-bereich-OPERAND - - - - - - - - - - - - - - - - - - - - - - - - - -

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

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

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

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

PROG=program-name
: *program-name* ist der Name einer Programmeinheit.
  Diese Programmeinheit muß zum Zeitpunkt der Eingabe des %TRACE bzw. bei der Abarbeitung des Subkommandos, in dem der %TRACE enthalten ist, geladen sein.

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

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

(S'stmt-nr':S'stmt-nr')
: *stmt-nr* ist die Statementnummer aus dem Übersetzungsprotokoll; siehe Spalte STMNT.

  trace-bereich wird festgelegt durch die Angabe einer Anfangs- und einer Ende-*stmt-nr* und umfaßt damit einen Teil des Quellprogramms.

  Die Anfangs-*stmt-nr* muß kleiner sein als die Ende-*stmt-nr*.

  Soll *trace-bereich* nur eine Zeile umfassen, müssen Anfangs- und Ende-*stmt-nr* gleich sein.

Ausgabe des %TRACE-Protokolls

Das %TRACE-Protokoll wird standardmäßig in ausführlicher Form (%OUT-Operandenwert T=MAX) über SYSOUT ausgegeben. Mit %OUT können Sie die Ausgabe-Medien und den Informationsumfang für die Ausgabe festlegen (siehe AID-Basishandbuch, Kapitel 7).

Ein %TRACE-Protokoll mit Zusatzinformationen (T=MAX) enthält die Statementnummer und den Typ der Assembler-Instruktion, die ausgeführt wurde. Ist ein Namenseintrag vorhanden, wird auch dieser ausgegeben.

In einem %TRACE-Protokoll mit verkürzter Information (T=MIN) wird der Typ der Instruktion nicht ausgegeben.

Beispiele

```
/%OUT %TRACE    T=MAX
/%T 3

     7  RAHMEN                EXT.PRO
  1037  TEST1                 CALL
  1281  TEST2                 JUMP
STOPPED AT LABEL: TEST2 , SRC_REF:  1281,  SOURCE: RAHMEN , PROC: RAHMEN
```

Mit %OUT wurde die Ausgabe auf Terminal zurückgeschaltet und festgelegt, daß der maximale Informationsumfang ausgegeben werden soll.
Das %TRACE-Kommando soll drei Assembler-Instruktionen verfolgen. Nach der dritten Instruktion kommt die Abschlußmeldung für diesen %TRACE: Der Programmablauf wurde bei der Instruktion TEST2 mit der Statementnummer 1281 unterbrochen, diese steht in der Programmeinheit RAHMEN, der Lademodul hat denselben Namen.

```
/%OUT %T T=MIN
/%T 3

     7  RAHMEN
  1037  TEST1
  1281  TEST2
STOPPED AT SRC_REF:  1281,  SOURCE: RAHMEN, PROC: RAHMEN
```

Mit dem %OUT-Kommando wird der Informationsumfang für das Kommando %TRACE reduziert. Der danach eingebene %TRACE gibt das Protokoll mit verkürztem Informationsumfang aus.

6 Anwendungsbeispiel

In diesem Kapitel wird eine AID-Testsitzung für ein kleines Assembler-Programm gezeigt. Anhand dieser Testsitzung können Sie die Anwendung und Wirkung einiger AID-Kommandos nachvollziehen, die Vorgehensweise ist bewußt einfach gehalten. Das Assembler-Programm ist in Abschnitt 6.1 dargestellt, der Testablauf in Abschnitt 6.2. Zur besseren Lesbarkeit sind Eingaben fettgedruckt.

6.1 Assembler-Programm

Aufgabenbeschreibung

Das Programm SUMME soll maximal 10 zweistellige Zahlen einlesen und ihre Summe ausgeben. Als Endekennzeichen wird die Zahl 00 eingegeben.
Bei mehr als 10 Zahlen wird eine Meldung und die errechnete Summe ausgegeben.

Quellprogrammliste

```
BERECHNUNG DER SUMME VON N ZAHLEN (N <= 10)                                                                    11:09:30  91-11-05
 LOCTN  OBJECT CODE    ADDR1    ADDR2     STMNT  M  SOURCE STATEMENT
 000000                                       1     SUMME    START
                                              2              TITLE 'BERECHNUNG DER SUMME VON N ZAHLEN (N <= 10)'
                                              3              PRINT NOGEN
 000000                 00000000              4     R0       EQU   0
 000000                 00000001              5     R1       EQU   1
 000000                 00000002              6     R2       EQU   2
 000000                 00000003              7     R3       EQU   3
 000000                 00000004              8     R4       EQU   4
 000000                 00000005              9     R5       EQU   5
                                             10     SUMME    AMODE ANY
                                             11     SUMME    RMODE ANY
                                             12              GPARMOD 31
                                             14   2                *,VERSION 010
 000000 0D 20                                15              BASR  R2,R0
 000002                 00000002              16              USING *,R2
                                             17     ANFANG   WROUT MELD1,ENDE
                                             24   2                *,FHDR VERSION 105 / 1988-06-13
                                             48   2                *,@DCEO      952    900503   53531004
                                             51   1                *,WROUT      005    910215   53121058
 000026 58 50 2176      00000178             52              L     R5,=F'1'
 00002A 5A 50 2176      00000178             53     SCHLEIFE A     R5,=F'1'
 00002E 49 50 2138      0000013A             54              CH    R5,ZEHN
 000032 47 20 20BE      000000C0             55              BH    FEHLER
                                             56     LESEN    RDATA EINGABE,ENDE
                                             63   2                *,FHDR VERSION 105 / 1988-06-13
                                             92   2                *,@DCEI      920    881104   53531002
                                             95   1                *,RDATA      006    910215   53121057
 000062 D5 05 2121213A 00000123 0000013C     96     VERGL    CLC   EINGABE+4,NULL
 000068 47 80 207A      0000007C             97              BE    AUS
 00006C F2 11 21232121 00000125 00000123     98     ADD      PACK  PACK,EINGABE+4(2)
 000072 FA 31 213C2123 0000013E 00000125     99              AP    GESAMT,PACK
 000078 47 F0 2028      0000002A            100              B     SCHLEIFE
 00007C F3 63 2131213C 00000133 0000013E    101     AUS      UNPK  ERGEB,GESAMT
 000082 D3 00 21372140 00000139 00000142    102              MVZ   ERGEB+6(1),ZONE
                                            103              WROUT MELD2,ENDE
                                            109   2                *,FHDR VERSION 105 / 1988-06-13
                                            133   2                *,@DCEO      952    900503   53531004
                                            136   1                *,WROUT      005    910215   53121058
                                            137     ENDE     TERM  DUMP=Y
                                            140   2                *,VERSION 010
                                            152     FEHLER   WROUT MELD3,ENDE
                                            159   2                *,FHDR VERSION 105 / 1988-06-13
                                            183   2                *,@DCEO      952    900503   53531004
                                            186   1                *,WROUT      005    910215   53121058
 0000E2 47 F0 207A      0000007C            187              B     AUS
                                            188              EJECT
                                            189     *
                                            190     * DEFINITIONEN
                                            191     *
 0000E6 0039                                192     MELD1    DC    Y(L'M1+5)
 0000E8 404001                              193              DC    X'404001'
 0000EB C2C9E3E3C540C2C9                    194     M1       DC    C'BITTE BIS ZU 10 2-STELLIGE ZAHLEN EINGEBEN! ENDE:
 00011F 000000000000                        195     EINGABE  DC    XL6'00'
 000125 000C                                196     PACK     DC    PL2'0'
                                            197     *
 000128 0012                                198     MELD2    DC    Y(L'M2+L'ERGEB+5)
 00012A 404001                              199              DC    X'404001'
 00012D E2E4D4D4C57A                        200     M2       DC    C'SUMME:'
 000133 40404040404040                      201     ERGEB    DC    CL7' '
                                            202     *
 00013A 000A                                203     ZEHN     DC    H'10'
 00013C F0F0                                204     NULL     DC    C'00'
 00013E 0000000C                            205     GESAMT   DC    PL4'0'
 000142 F0                                  206     ZONE     DC    X'F0'
                                            207     *
 000144 0034                                208     MELD3    DC    Y(L'M3+5)
 000146 404001                              209              DC    X'404001'
 000149 C5E240D2D6C5D5D5                    210     M3       DC    C'ES KOENNEN MAXIMAL 10 ZAHLEN VERARBEITET WERDEN'
```

```
 000000                                  211              END   SUMME
 000178 00000001                         212                    =F'1'
 00017C 9101221427002852                 213                    =X'9101221427002852' CONSISTENCY CONSTANT FOR AID
FLAGS IN 00000 STATEMENTS, 000 PRIVILEGED FLAGS, 000 MNOTES
HIGHEST ERROR-WEIGHT : NO ERRORS
THIS PROGRAM WAS ASSEMBLED BY ASSEMBH                   V1.0B00     ON 1991-11-05 AT 11:07:54
```

6.2 Testablauf

1. Schritt

Das Assembler-Quellprogramm SUMME in der Datei SOURCE.TEST wird mit dem ASSEMBH-XT übersetzt. Auf Grund der Angabe TEST-SUPPORT=YES wird vom ASSEMBH-XT LSD-Information erzeugt und an den Bindemodul übergeben. Das Quellprogramm wird fehlerfrei übersetzt.

```
/DEL-SYS-FILE OMF
/START-PROG $ASSXT

%  BLS0500 PROGRAM 'ASSEMBH-XT', VERSION '1.XXXX' OF 'yy-mm-dd' LOADED.
%  BLS0552 COPYRIGHT (C) SIEMENS NIXDORF INFORMATIONSSYSTEME AG. 1990. ALL
   RIGHTS RESERVED
%  ASS6010 V 1.XXXX OF BS2000 ASSEMBH-XT READY

//COMPILE SOURCE=SOURCE.TEST,
          TEST-SUPPORT=YES

%  ASS6011 ASSEMBLY TIME: 80 MSEC
%  ASS6018 0 FLAGS, 0 PRIVILEGED FLAGS, 0 MNOTES
%  ASS6019 HIGHEST ERROR-WEIGHT: NO ERRORS
%  ASS6006 LISTING GENERATOR TIME: 102 MSEC

//END

%  ASS6012 END OF ASSEMBH-XT
```

2. Schritt

Das Programm SUMME soll zum Ablauf gebracht werden.

```
/START-PROG (*OMF)

%  BLS0001 *** DBL VERSION 070 RUNNING ***
%  BLS0517 MODULE 'SUMME' LOADED

BITTE BIS ZU 10 2-STELLIGE ZAHLEN EINGEBEN. ENDE: 00
*05
*16
*48
*00
*0
*00
*EN
/
```

Das Programm verzweigt immer wieder zur Eingabe, es liegt ein Programmfehler vor. Das Programm wird durch Drücken der K2-Taste unterbrochen.

3. Schritt

Um das Programm symbolisch testen zu können, wird es mit der Angabe TEST-OPTION=AID erneut geladen.

```
/LOAD-PROG (*OMF),TEST-OPTION=AID

%  BLS0001 *** DBL VERSION 070 RUNNING ***
%  BLS0517 MODULE 'SUMME' LOADED

/%IN S'96' <%D EINGABE;%STOP>
/%R
```

Mit dem %INSERT-Kommando wird ein Testpunkt auf die Zeile mit der Statementnummer 96 gesetzt, also auf den CLC-Befehl. Jedesmal, wenn das Programm an dieser Adresse angekommen ist, soll der Inhalt des Feldes EINGABE ausgegeben werden. Nach der Ausgabe soll das Programm in den Zustand STOP übergehen, damit neue Kommandos eingegeben werden können.

Mit %RESUME wird das geladene Programm gestartet.

```
BITTE BIS ZU 10 2-STELLIGE ZAHLEN EINGEBEN. ENDE: 00
*05

** ITN: #'004B012E' *** TSN: 2069 ****************************************
SRC_REF:    96 SOURCE: SUMME     PROC: SUMME ********************************
EINGABE         = 00060000 F0F5   ....05
STOPPED AT LABEL: VERGL , SRC_REF:  96, SOURCE: SUMME ,PROC: SUMME

/%R
*48
EINGABE         = 00060000 F4F8   ....48
STOPPED AT LABEL: VERGL , SRC_REF:  96, SOURCE: SUMME ,PROC: SUMME

/%R
*16
EINGABE         = 00060000 F1F6   ....16
STOPPED AT LABEL: VERGL , SRC_REF:  96, SOURCE: SUMME ,PROC: SUMME

/%R
*00
EINGABE         = 00060000 F0F0   ....00
STOPPED AT LABEL: VERGL , SRC_REF:  96, SOURCE: SUMME ,PROC: SUMME
```

Das Feld EINGABE enthält jeweils den richtigen Wert. Das Programm erkennt offensichtlich das Endekriterium nicht.

4. Schritt

Mit dem %DISASSEMBLE-Kommando sollen jetzt von der Zeile 96, also vom CLC-Befehl an, 5 Zeilen "rückübersetzt" ausgegeben werden.

```
/%DA 5 FROM S'96'

SUMME+62          CLC   121(6,R2),13A(R2)      D5 05 2121 213A
SUMME+68          BC    B'1000',7A(R0,R2)      47 80 207A
SUMME+6C          PACK  123(2,R2),121(2,R2)    F2 11 2123 2121
SUMME+72          AP    13C(4,R2),123(2,R2)    FA 31 213C 2123
SUMME+78          BC    B'1111',28(R0,R2)      47 F0 2028
```

Dabei stellt sich heraus, daß das Längenfeld des CLC-Befehls '6' statt '2' enthält. Deshalb wird das Endekriterium nicht erkannt.

Der Assemblerbefehl müßte korrekt lauten:

```
VERGL     CLC  EINGABE+4(2),NULL
```

5. Schritt

Dieser Fehler kann vorläufig mit dem %SET-Kommando behoben werden. Dazu wird das Programm erneut geladen.

```
/LOAD-PROG (*OMF),TEST-OPTION=AID

%  BLS0001 *** DBL VERSION 070 RUNNING ***
%  BLS0517 MODULE 'SUMME' LOADED

/%SET X'D5012121213A' INTO VERGL
/%DA 1 FROM VERGL

SUMME+62          CLC   121(2,R2),13A(R2)      D5 01 2121 213A

/%R
```

Mit dem %SET wird der Speicherinhalt an der Adresse VERGL geändert. Übertragen wird ein AID-Literal in der Länge des CLC-Befehls, das die Längenangabe '01' statt '05' enthält. Anschließend wird mit %DISASSEMBLE der CLC-Befehl kontrolliert und mit %RESUME wieder gestartet.

```
BITTE BIS ZU 10 2-STELLIGE ZAHLEN EINGEBEN. ENDE: 00
*05
*16
*48
*12
*10
*15
*17
*19
*29
ES KOENNEN MAXIMAL 10 ZAHLEN VERARBEITET WERDEN
SUMME:0000171

%  IDA0N51 PROGRAM INTERRUPT AT LOCATION '000000BE (SUMME), (CDUMP), EC=90'
%  IDA0N45 DUMP DESIRED? REPLY (Y=USER-/AREADUMP;Y,SYSTEM=SYSTEMDUMP;N=NO)?Y
%  IDA0N53 DUMP BEING PROCESSED. PLEASE HOLD ON
%  IDA0N54 USERDUMP WRITTEN TO FILE 'benutzerkennung.DUMP.name.2069.00001'
%  IDA0N55 TITLE: 'TSN-2069  UID-benutzerkennung  AC#-xxxxxxxx  USERDUMP
   PC-0000BE EC=90  VERS-100  DUMP-TIME  11:26:51  91-11-05'
```

Es liegt ein weiterer Programmfehler vor, da der Benutzer bisher nur 9 Zahlen eingegeben hat. Deshalb wird bei Beendigung des Programms ein Dump zur weiteren Diagnose erzeugt.

6. Schritt

Mit dem %DUMPFILE-Kommando wird die Dump-Datei eröffnet und ihr der LINK-Name D1 zugewiesen. Durch %BASE wird der AID-Arbeitsbereich auf die geöffnete Dump-Datei umgeschaltet. Von jetzt an wird mit einer Adresse ohne eigene Basisqualifikation immer auf die Daten der Dump-Datei zugegriffen.

```
/%DUMPFILE D1=DUMP.NAME.2069.00001
/%BASE E=D1

/%D EINGABE
** D1: DUMP.NAME.2069.00001 ************************************************
EINGABE         = 00060000 F2F9   ....29

/%D _R5
_R5             = 0000000B
```

Die zuletzt eingegebene Zahl des Feldes EINGABE soll ausgegeben werden. Die Eingabe stimmt mit dem Protokoll überein.

Da im Register 5 die Anzahl der Eingaben gezählt wird, wird jetzt der Stand des Registers 5 untersucht.

Register 5 enthält den Wert '11', obwohl erst 9 Zahlen eingegeben wurden. Ein Vergleich mit dem Übersetzungsprotokoll ergibt, daß Register 5 mit '1' vorbesetzt ist, statt mit '0'.

Der Assemblerbefehl müßte korrekt lauten: `L  R5,=F'0'`

7. Schritt

Dieser Fehler kann vorläufig mit Hilfe des %SET-Kommandos behoben werden. Dazu wird das Programm erneut geladen.

```
/LOAD-PROG (*OMF),TEST-OPTION=AID

%  BLS0001 *** DBL VERSION 070 RUNNING ***
%  BLS0517 MODULE 'SUMME' LOADED

/%BASE
/%SET X'D5012121213A' INTO VERGL
/%IN SCHLEIFE <%SET #'0' INTO _R5; %REM %.>

/MOD-TEST-OPT DUMP=NO
/%R
```

Als erstes muß mit %BASE wieder das geladene Programm als AID-Arbeitsbereich zugewiesen werden.

Durch das erneute Laden des Programms sind vorherige Korrektureingaben gelöscht. Um einen fehlerfreien Programmablauf zu gewährleisten, setzen wir deshalb das %SET-Kommando aus dem 5. Schritt hier nochmals ab.

Mit dem %INSERT-Kommando wird ein *testpunkt* auf den Assemblerbefehl mit dem Namenseintrag SCHLEIFE gesetzt. AID soll das folgende *subkdo* also vor dem Additionsbefehl ausführen.

Das %SET-Kommando, das Register 5 mit '0' vorbesetzt, steht im *subkdo* von %INSERT. Mit %REM %. wird dieses *subkdo* nach dem ersten Durchlauf gelöscht (da kein weiteres Subkommando zu diesem *testpunkt* eingetragen ist, wird auch der *testpunkt* gelöscht), und das Programm läuft dann weiter.

Da im Quellprogramm der Makroaufruf TERM mit dem Operanden DUMP=Y definiert ist, wird bei jedem Programmende ein Speicherabzug (Dump) angeboten. Dieses kann vor dem Start des Programms (%RESUME) mit folgendem Kommando verhindert werden: /MODIFY-TEST-OPTIONS DUMP=NO

```
BITTE BIS ZU 10 2-STELLIGE ZAHLEN EINGEBEN. ENDE: 00
*05
*16
*48
*12
*10
*15
*17
*19
*29
*11
ES KOENNEN MAXIMAL 10 ZAHLEN VERARBEITET WERDEN
SUMME:0000182

%  IDA0N51 PROGRAM INTERRUPT AT LOCATION '000000BE (SUMME), (CDUMP), EC=90'
%  IDA0N47 DUMP PROHIBITED BY /OPTION COMMAND
/
```

Nach dieser Korrektur läuft das Programm fehlerfrei ab. Die Fehler können endgültig im Quellprogramm behoben werden.

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.

Adreß-Operand

ist ein Operand, mit dem Sie eine Speicherstelle oder einen Speicherbereich adressieren. Sie können virtuelle Adressen, Datennamen, Anweisungsnamen, Source-Referenzen, Schlüsselwörter, komplexe Speicherreferenzen oder eine PROG-Qualifikation angeben. Die Speicherstelle oder der Speicherbereich liegen entweder im geladenen Programm oder in einem Speicherabzug in einer Dump-Datei.
Wenn Sie ein Datenfeld, einen Anweisungsnamen oder eine Source-Referenz ansprechen wollen, die nicht in der aktuellen Programmeinheit liegen, so können Sie die entsprechende Speicherstelle über eine Qualifikation adressieren.

Änderungsdialog

Mit dem Kommando %AID CHECK=ALL schalten Sie den Änderungsdialog ein. Er wird bei der Ausführung von %MOVE oder %SET wirksam. AID fragt im Dialog nach, ob die Änderung des Speicherinhalts wirklich durchgeführt werden soll. Wird als Antwort ein N eingegeben, unterbleibt die Änderung; wird ein Y eingegeben, führt AID die Übertragung aus.

AID-Arbeitsbereich

ist der Adreßraum, in dem Sie Adressen ohne Angabe von Qualifikationen ansprechen können.
Beim symbolischen Testen ist der AID-Arbeitsbereich die aktuelle Programmeinheit. Nur die Daten- und Anweisungsnamen und die Source-Referenzen, die innerhalb der aktuellen Programmeinheit liegen, sind ohne Angabe einer Qualifikation ansprechbar. Im geladenen Programm ist die aktuelle Programmeinheit die, in der sich gerade der Programmablauf befindet. In einem Speicherabzug ist die aktuelle Programmeinheit die, in der sich der Programmablauf befand, als der Speicherabzug erstellt wurde.

In einem Kommando können Sie vom AID-Arbeitsbereich abweichen, indem Sie im Adreß-Operanden eine Qualifikation angeben. Mit dem Kommando %BASE können Sie den AID-Arbeitsbereich vom geladenen Programm in einen Speicherabzug verlegen oder umgekehrt.

AID-Ausgabedateien

sind die Dateien, in die Sie sich die Ausgaben der Kommandos %DISASSEMBLE, %DISPLAY, %HELP, %SDUMP und %TRACE schreiben lassen können. Die Dateien werden in den Ausgabekommandos über ihre Linknamen F0 bis F7 angesprochen (siehe %OUT und %OUTFILE).
In die Datei, die dem Linknamen F6 zugewiesen wurde, werden die REP-Sätze geschrieben (siehe %AID REP=YES und %MOVE).

Es gibt drei Wege, eine Ausgabedatei anzulegen:

1. /%OUTFILE-Kommando mit dem Link- und Dateinamen
2. /FILE-Kommando mit dem Link- und Dateinamen
3. Für einen Linknamen, dem noch kein Dateiname zugewiesen ist, setzt AID einen FILE-Makro mit dem Dateinamen AID.OUTFILE.Fn ab.

Eine AID-Ausgabedatei hat stets das Format FCBTYPE=SAM, RECFORM=V und OPEN=EXTEND.

AID-Eingabedateien

sind Dateien, die AID zur Ausführung von AID-Funktionen benötigt, im Unterschied zu Eingabedateien, die das Programm benutzt. AID verarbeitet nur Platten-Dateien.
AID-Eingabedateien sind:

1. Dump-Dateien, in denen sich Speicherabzüge befinden (%DUMPFILE)
2. PLAM-Bibliotheken, in denen sich Bindemodule befinden. Wird die Bibliothek mit %SYMLIB zugewiesen, kann AID die LSD-Sätze nachladen.

AID-Konstante

Zu den AID-Konstanten zählen 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. Sie besitzen kein Adreßattribut.
Eine Adreßkonstante repräsentiert eine Adresse. Das sind Anweisungsnamen, Source-Referenzen und das Ergebnis einer Adreßselektion. In Verbindung mit einem Pointer-Operator (->) können Sie damit die entsprechende Speicherstelle adressieren.

AID-Literale

AID stellt Ihnen Zeichen-Literale und numerische Literale zur Verfügung (siehe AID-Basishandbuch, Kapitel 8):

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

AID-Standard-Arbeitsbereich

Beim Testen auf Maschinencode-Ebene ist das der nicht-privilegierte Teil des virtuellen Speichers in Ihrer Task, der vom Programm samt allen seinen konnektierten Subsystemen belegt ist.

Beim symbolischen Testen ist es die aktuelle Programmeinheit des geladenen Programms. Ohne Vereinbarung mit %BASE und ohne Angabe einer Basis-Qualifikation gilt der AID-Standard-Arbeitsbereich.

Aktuelle Aufrufhierarchie

ist der Stand der Unterprogrammverschachtelung an der Unterbrechungsstelle. Sie reicht von der Unterprogrammebene, auf der das Programm unterbrochen wurde, über die verlassenen Unterprogramme bis zum Hauptprogramm. Beim symbolischen Testen von Assembler-Programmen kann die Aufrufhierarchie nur verfolgt werden, wenn das Programm aus Assembler-Prozeduren aufgebaut ist, d.h., wenn es sich um ein strukturiertes Assemblerprogramm handelt.

Die Aufrufhierarchie wird mit %SDUMP %NEST ausgegeben.

Aktuelles Programm

ist das Programm, das in der Task geladen ist, in der Sie AID-Kommandos eingeben.

Aktuelle Programmeinheit

ist die Programmeinheit (Übersetzungseinheit) in der das Programm unterbrochen wurde.

Für strukturierte Assemblerprogramme ist hierbei zu beachten:

Liegt die aktuelle Programm-Unterbrechung im Laufzeitmodul des ASSEMBH-XT, so ist die aktuelle Programmeinheit nicht mehr die Übersetzungseinheit des Benutzers. In AID-Kommandos muß in diesem Fall mit der entsprechenden PROG-Qualifikation gearbeitet werden.

Die AID-STOP-Meldung gibt den Namen der Programmeinheit aus.

Assembler-Prozedur

ist eine Programmeinheit in der Strukturierten Programmierung mit einem Eingang und einem Ausgang, die unter einem Namen, ggf. mit aktuellen Parameterwerten aufgerufen werden kann. Eine Assembler-Prozedur wird begrenzt von den vordefinierten Makros @ENTR (Prozeduranfang) und @END (statisches Prozedurende).

Anweisungsname

bezeichnet die Adresse einer ausführbaren Assembler-Instruktion oder eines Aufrufs für einen vordefinierten Makro (@-Makro).
Der Anweisungsname wird mit *L'name'* angegeben.
name ist der maximal 64-stellige Namenseintrag einer Assembler-Instruktion oder eines Aufrufs für einen vordefinierten Makro (@-Makro).
name wird von AID auf 32 Stellen verkürzt.
name kann auch ohne L'...' angegeben werden, wenn in einem Kommando eine Verwechslung mit einem Datennamen nicht möglich ist.

Attribute

Jedes Speicherobjekt hat bis zu sechs Attribute:
Adresse, Name (opt), Inhalt, Länge, Speichertyp, Ausgabetyp
Mit Selektoren können Sie auf Adresse, Länge und Speichertyp zugreifen. Über den Namen findet AID in den LSD-Sätzen alle zugehörigen Attribute, um damit zu arbeiten.

Adreßkonstanten und Konstanten aus dem Quellprogramm haben nur bis zu fünf Attribute:
Name (opt), Wert, Länge, Speichertyp, Ausgabetyp
Sie haben keine Adresse. Beim Ansprechen einer Konstanten greift AID nicht auf ein Speicherobjekt zu, sondern setzt nur den dafür vorgemerkten Wert ein.

Ausgabetyp

Attribut eines Speicherobjekts, das bestimmt, wie der Speicherinhalt von AID ausgegeben wird. Jedem Speichertyp ist ein Ausgabetyp zugeordnet. Im AID-Basishandbuch, Kapitel 9 sind die AID-spezifischen Speichertypen samt zugehörigen Ausgabetypen aufgelistet. Für die in ASSEMBH verwendeten Datentypen gilt eine entsprechende Zuordnung. Eine Typmodifikation bei %DISPLAY und %SDUMP bewirkt eine Änderung des Ausgabetyps.

Basis-Qualifikation

ist die Qualifikation, mit der Sie den AID-Arbeitsbereich in das geladene Programm oder in einen Speicherabzug in einer Dump-Datei legen. Sie wird mit E={VM | Dn} angegeben.
Die Basis-Qualifikation können Sie global mit %BASE vereinbaren oder im Adreß-Operanden für eine einzelne Speicherreferenz angeben.

Bereichsgrenzen

Jedem Speicherobjekt ist ein bestimmter Bereich zugeordnet, der bei Datennamen und Schlüsselwörtern durch die Attribute Adresse und Länge festgelegt ist. Bei virtuellen Adressen liegen die Bereichsgrenzen zwischen V'0' und der letzten Adresse des virtuellen Speichers (V'7FFFFFFF'). Bei PROG-Qualifikationen ergeben sich die Bereichsgrenzen aus Anfangs- und Endadresse der Programmeinheit (siehe AID-Basishandbuch, Abschnitt 6).

Bereichsüberprüfung
AID überprüft bei Adreßversatz, Längenmodifikation und bei *empfänger* in einem %MOVE, ob die Bereichsgrenzen der angesprochenen Speicherobjekte überschritten werden und gibt im Fehlerfall eine entsprechende Meldung aus.

Benutzerbereich
ist der Bereich des virtuellen Speichers, der vom geladenen Programm samt allen seinen konnektierten Subsystemen belegt ist. Er entspricht dem Bereich, der durch das Schlüsselwort %CLASS6 bzw. %CLASS6ABOVE und %CLASS6BELOW repräsentiert wird.

CSECT-Informationen
stehen in der Objekt-Strukturliste.

datenname
steht für alle Namen, die im Quellprogramm für Daten vergeben wurden. Mit *datenname* sprechen Sie Konstanten, Datenfelder, vordefinierte Mehrzweckregister, Programmabschnitte, Pseudoabschnitte, externen Pseudoabschnitte, Pseudoregister und gemeinsame Hilfsabschnitte beim symbolischen Testen an (siehe Kapitel 3).

Datentyp
Gemäß dem im Quellprogramm deklarierten Datentyp ordnet AID allen Datenfeldern einen AID-Speichertyp zu:
- Binärstring (≙ %X)
- Character (≙ %C)
- numerisch (≙ %F, %D)

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.

Dump-Datei
ist eine Plattendatei, die einen Programm-Speicherabzug enthält.

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 Assembler 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).

Kommandofolge

Mehrere Kommandos werden mit Semikolon (;) zu einer Folge verbunden, die von links nach rechts abgearbeitet wird. Wie im Subkommando darf eine Kommandofolge AID- und BS2000-Kommandos enthalten. In Kommandofolgen nicht zugelassen sind die AID-Kommandos %AID, %BASE, %DUMPFILE, %HELP, %OUT, %QUALIFY und die im Anhang aufgelisteten BS2000-Kommandos.
Enthält eine Kommandofolge eines der Kommandos zur Ablaufsteuerung, wird die Kommandofolge an der Stelle abgebrochen und das Programm gestartet (%CONTINUE, %RESUME, %TRACE) oder angehalten (%STOP). Nachfolgende Kommandos aus der Kommandofolge werden nicht mehr ausgeführt.

Kommandomodus

Mit Kommandomodus wird in den AID-Handbüchern der EXPERT-Modus der SDF-Kommandosprache bezeichnet. Falls Sie gerade in einem anderen Modus (`GUIDANCE={MAXIMUM|MEDIUM|MINIMUM|NO}`) arbeiten, sollten Sie mit Kommando `MODIFY-SDF-OPTIONS GUIDANCE=EXPERT` in den EXPERT-Modus umschalten, wenn Sie AID-Kommandos eingeben wollen. AID-Kommandos verfügen nicht über eine SDF-Syntax:

- Operanden werden nicht über Menüs abgefragt.
- Im Fehlerfall gibt AID eine Fehlermeldung aus, führt aber keinen Korrekturdialog.

Im EXPERT-Modus fordert Sie das System mit "/" zur Kommandoeingabe auf.

Konstante (AID)

Zu den Konstanten zählen 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. Sie besitzen kein Adreßattribut.
Eine Adreßkonstante repräsentiert eine Adresse. Das sind Anweisungsnamen, Source-Referenzen und das Ergebnis einer Adreßselektion. In Verbindung mit einem Pointer-Operator (->) können Sie damit die entsprechende Speicherstelle adressieren.

LIFO

Last In First Out; Treffen an einem Testpunkt (%INSERT) oder bei Auftreten eines Ereignisses (%ON) Anweisungen aus verschiedenen Eingaben zusammen, so werden die zuletzt eingegebenen zuerst abgearbeitet (siehe AID-Basishandbuch, Abschnitt 5.4).

Lokalisierungsinformation

Mit %DISPLAY %HLLOC(speicherref) für die symbolische Ebene und mit %DISPLAY %LOC(speicherref) für die Maschinencode-Ebene gibt Ihnen AID die statische Programmverschachtelung zu der angegebenen Speicherstelle aus.
Im Gegensatz dazu erhalten Sie mit %SDUMP %NEST die dynamische Programmverschachtelung, die Aufrufhierarchie zur aktuellen Programmunterbrechungsstelle.

LSD

List for Symbolic Debugging ist ein Verzeichnis der im Modul definierten Daten- und Anweisungsnamen. Ebenso sind dort die vom Assembler erzeugten Source-Referenzen (Statementnummern) hinterlegt. Die LSD-Sätze werden vom Assembler erzeugt. AID holt sich hieraus die Informationen zur symbolischen Adressierung.

Namensraum

umfaßt alle zu einer Programmeinheit in den LSD-Sätzen verzeichneten Datennamen.

Objekt-Strukturliste

Auf Basis des ESD (External Symbol Dictionary) erstellt der Binder die Objekt-Strukturliste, wenn die Standardeinstellung SYMTEST=MAP gilt bzw. wenn Sie SYMTEST=ALL angegeben haben.

Programmeinheit

ist eine Übersetzungseinheit. Der ASSEMBH-XT erzeugt für jede Übersetzungseinheit einen Bindemodul (Objektmodul).
Der Name einer Übersetzungseinheit ist der Name des ersten benannten Programmabschnitts (START- oder CSECT-Anweisung), bzw. der Name der ersten Assembler-Prozedur (@ENTR-Makro).

Programmzustand

AID unterscheidet drei Programmzustände, in denen sich das zu testende Programm befinden kann:

1. Das Programm steht.

%STOP, K2-Taste und der Aufruf des Makro BKPT unterbrechen ein laufendes Programm. Außerdem wird das Programm unterbrochen, wenn ein %TRACE abgearbeitet ist. Die Task befindet sich im Kommandomodus. Sie können Kommandos eingeben.

2. Das Programm läuft ohne Ablaufverfolgung.

%RESUME startet ein Programm oder setzt es fort. %CONTINUE bewirkt dasselbe; ist allerdings ein %TRACE noch nicht ganz abgearbeitet, so setzt %CONTINUE das Programm mit Ablaufverfolgung fort.

3. Das Programm läuft mit Ablaufverfolgung.

%TRACE startet ein Programm oder setzt es fort. Der Programmablauf wird entsprechend der Vereinbarungen im %TRACE protokolliert. %CONTINUE bewirkt dasselbe, wenn noch ein %TRACE aktiv ist.

Qualifikation

Mit einer Qualifikation können Sie eine Adresse ansprechen, die nicht im aktuellen AID-
Arbeitsbereich liegt. Die Basis-Qualifikation gibt an, ob die Adresse im geladenen Programm oder in einem Speicherabzug liegt. Die PROG-Qualifikation gibt an, in welcher Programmeinheit 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 PROG-Qualifikation für ein Datenfeld der aktuellen Programmeinheit angegeben wird.

Source-Referenz

ist eine Statementnummer, die vom Assembler in der Übersetzungsliste in der Spalte STMNT vergeben wird. Angesprochen werden kann jede ausführbare Assembler-Instruktion mit Namen und jeder Auruf eines vordefinierten Makro (@-Makro).
Die Source-Referenz wird mit *S'stmt-nr'* angegeben.
stmt-nr ist eine Ganzzahl zwischen 1 und 2^{31}-1.

Speicherobjekt

ist eine bestimmte Anzahl von zusammenhängenden Bytes im Speicher. Auf Programmebene sind das die Daten des Programms, sofern ihnen ein Speicherbereich zugewiesen ist, und der Befehlscode. Außerdem gehören alle Register, der Befehlszähler sowie alle anderen Bereiche, die nur über Schlüsselwörter angesprochen werden können, ebenfalls zu den Speicherobjekten.
Keine Speicherobjekte hingegen sind alle im Programm definierten Konstanten, die Anweisungsnamen, Source-Referenzen, die Ergebnisse von Adreßselektion, Längenselektion und Längenfunktion und die AID-Literale. Sie repräsentieren einen Wert, der nicht verändert werden kann.

Speicherreferenz

Mit einer Speicherreferenz sprechen Sie ein Speicherobjekt an. Es gibt einfache und komplexe Speicherreferenzen. Einfache Speicherreferenzen sind virtuelle Adressen, Namen, zu denen AID sich die Adresse aus den LSD-Informationen holen kann, und Schlüsselwörter. Anweisungsnamen und Source-Referenzen sind in den AID-Kommandos %CONTROLn, %DISASSEMBLE, %INSERT und %REMOVE als Speicherreferenz erlaubt, obwohl es nur Adreßkonstanten sind.
Mit den komplexen Speicherreferenzen geben Sie AID eine Vorschrift an, wie die gewünschte Adresse errechnet werden soll und welcher Typ und welche Länge gelten sollen. Folgende Operationen können in einer komplexen Speicherreferenz vorkommen: Adreßversatz, indirekte Adressierung, Typmodifikation, Längenmodifikation und Adreßselektion.

Speichertyp
ist entweder der Datentyp, der im Quellprogramm festgelegt wurde oder der durch Typmodifikation gewählte. AID kennt die Speichertypen %X, %C, %P, %D, %F, %A (siehe %SET und AID-Basishandbuch, Kapitel 6 und 9).

Statementnummer
Im Assembler-Quellprogramm gilt jede einzelne Instruktion und jeder Kommentar als ein Statement und erhält im Übersetzungsprotokoll eine Statementnummer. Wird eine Assembler-Instruktion über mehrere Listingzeilen fortgesetzt, hat trotzdem die gesamte Instruktion nur eine Statementnummer. Die Statementnummer von Assembler-Instruktionen, die durch Makros generiert wurden, sind aus dem Übersetzungsprotokoll ersichtlich, wenn mit PRINT GEN übersetzt wurde.
Über die Statementnummer ansprechen können Sie beim symbolischen Testen alle Assembler-Instruktionen mit einem Namen im Namenseintrag und alle Aufrufe von vordefinierten Makros (@-Makros).

strukturierte Assemblerprogramme
entsprechen den Regeln der Strukturierten Programmierung mit ASSEMBH-XT. Sie sind mit Hilfe von vordefinierten Makros programmiert. Strukturierte Assemblerprogramme sind aus Assembler-Prozeduren aufgebaut.

Subkommando
ist ein Operand der Überwachungskommandos %CONTROLn, %INSERT oder %ON. Ein Subkommando kann einen Namen, eine Bedingung und einen Kommandoteil enthalten. Der Kommandoteil kann aus einem einzelnen Kommando oder aus einer Kommandofolge bestehen. Er kann AID- und BS2000-Kommandos enthalten. Jedes Subkommando hat einen Durchlaufzähler. Wie eine Ausführungsbedingung formuliert wird, wie Name und Durchlaufzähler vergeben und angesprochen werden, und welche Kommandos innerhalb von Subkommandos nicht erlaubt sind, ist im AID-Basishandbuch, Kapitel 5 beschrieben.
Der Kommandoteil des Subkommandos wird dann ausgeführt, wenn die Überwachungsbedingung des entsprechenden Kommandos (*kriterium*, *testpunkt*, *ereignis*) zutrifft und die eventuell definierte Ausführungsbedingung erfüllt ist.

Unterbrechungsstelle
Die Adresse, an der ein Programm unterbrochen wurde, wird Unterbrechungsstelle genannt. Aus der AID-STOP-Meldung können Sie die Anweisung und die Programmeinheit der Unterbrechungsstelle entnehmen. Dort wird das Programm fortgesetzt.

vordefinierte Makros
werden verwendet bei der strukturierten Programmierung mit ASSEMBH-XT. Das erste Zeichen des Aufrufs eines vordefinierten Makro ist immer ein "@".
Der ASSEMBH-XT stellt dem Anwender einen Satz von vordefinierten Makros zur Verfügung, mit denen sich das Konzept der strukturierten Programmierung realisieren läßt. Für die Übersetzung von Assemblerprogrammen, die Aufrufe dieser Makros enthalten, muß dem ASSEMBH-XT die zugehörige Makrobibliothek bekanntgemacht

werden. Für den Ablauf der Programme muß das Laufzeitsystem des ASSEMBH-XT zur Verfügung stehen.

Literatur

[1] **AID** (BS2000)
Advanced Interactive Debugger
Basishandbuch
Benutzerhandbuch

Zielgruppe
Programmierer im BS2000
Inhalt
- Überblick über AID
- Beschreibung der Sachverhalte und Operanden, die für alle Programmiersprachen gleich sind
- Meldungen
- Gegenüberstellung von AID-IDA

Einsatz
Testen von Programmen im Dialog- und Stapelbetrieb

[2] **AID** (BS2000)
Advanced Interactive Debugger
Testen auf Maschinencode-Ebene
Benutzerhandbuch

Zielgruppe
Programmierer im BS2000
Inhalt
- Beschreibung der AID-Kommandos für das Testen auf Maschinencode-Ebene
- Anwendungsbeispiel

Einsatz
Testen von Programmen im Dialog- und Stapelbetrieb

[3] **AID** (BS2000)
Advanced Interactive Debugger
Testen von COBOL-Programmen
Benutzerhandbuch

Zielgruppe
COBOL-Programmierer
Inhalt
– Beschreibung der AID-Kommandos für das symbolische Testen von COBOL-Programmen
– Anwendungsbeispiel
Einsatz
Testen von COBOL-Programmen im Dialog- und Stapelbetrieb

[4] **AID** (BS2000)
Advanced Interactive Debugger
Testen von FORTRAN-Programmen
Benutzerhandbuch

Zielgruppe
FORTRAN-Programmierer
Inhalt
– Beschreibung der AID-Kommandos für das symbolische Testen von FORTRAN-Programmen
– Anwendungsbeispiel
Einsatz
Testen von FORTRAN-Programmen im Dialog- und Stapelbetrieb

[5] **AID** (BS2000)
Advanced Interactive Debugger
Testen von PL/I-Programmen
Benutzerhandbuch

Zielgruppe
PL/I-Programmierer
Inhalt
– Beschreibung der AID-Kommandos für das symbolische Testen von PL/I-Programmen
– Anwendungsbeispiel
Einsatz
Testen von PL/I-Programmen im Dialog- und Stapelbetrieb

[6] **AID** (BS2000)
Advanced Interactive Debugger
Testen von C-Programmen
Benutzerhandbuch

Zielgruppe
C-Programmierer
Inhalt
– Beschreibung der AID-Kommandos für das symbolische Testen von C-Programmen
– Anwendungsbeispiel
Einsatz
Testen von C-Programmen im Dialog- und Stapelbetrieb

[7] BS2000
Makroaufrufe an den Ablaufteil
Benutzerhandbuch

Zielgruppe
– BS2000-Assembler-Programmierer (nicht privilegiert)
– Systemverwalter
Inhalt
– Alle Makroaufrufe an den Ablaufteil in lexikalischer Reihenfolge mit Hinweisen und Beispielen, einschließlich ausgewählter Makroaufrufe für das DVS und für TIAM
– Zusammenstellung der Makroaufrufe nach Anwendungsgebieten
– Ausführlicher Lernteil über Ereignissteuerung, Serialisation, Inter-Task-Kommunikation, Contingencies
Einsatz
BS2000-Anwendungsprogramme

[8] BS2000
Programmiersystem
Technische Beschreibung

Zielgruppe
- BS2000-Anwender und -Betreiber, die sich für den technischen Hintergrund ihres Systems interessieren (Softwareentwickler, Systemanalytiker, RZ-Leiter, Systemverwalter)
- Informatiker, die ein konkretes "General-Purpose"-Betriebssystem studieren wollen

Inhalt
Funktionen und Realisierungsprinzipien
- des Binders
- des Laders
- des Binde-Laders
- der Test- und Diagnosehilfen
- des Programmbibliothekssystems

Bestellnummer U3216-J-Z53-1

[9] **ASSEMBH** (BS2000)
Benutzerhandbuch

Zielgruppe
Assembler-Anwender im BS2000
Inhalt
- Aufruf und Steuerung des ASSEMBH
- Übersetzen, Binden, Laden und Starten
- Eingabequellen und Ausgaben des ASSEMBH
- Laufzeitsystem, Listenausgabe der strukturierten Programmierung
- Sprachverknüpfungen
- Assembler-Diagnoseprogramm ASSDIAG
- Dialogtesthilfe AID
- Meldungen des ASSEMBH
- Format der Assemblerbefehle

[10] **ASSEMBH** (BS2000)
Beschreibung

Zielgruppe
Assembler-Anwender im BS2000
Inhalt
- Beschreibung des Sprachumfangs des Assemblers ASSEMBH
- Struktur der Assemblersprache, Assembleranweisungen
- Struktur und Elemente sowie Instruktionen der Makrosprache
- Strukturierte Programmierung mit ASSEMBH-XT, vordefinierte Makros für die strukturierte Programmierung

Stichwörter

Inhalt

AID V2.0A (BS2000)

Testen von ASSEMBH-Programmen

Zielgruppe

Assembler-Programmierer

Inhalt

- Beschreibung der AID-Kommandos für das symbolische Testen von ASSEMBH-Programmen
- Anwendungsbeispiel

Einsatz

Testen von ASSEMBH-Programmen im Dialog- und Stapelbetrieb

Ausgabe: **Dezember 1991**

Datei: **AID_ASS.PDF**

Information on this document

On April 1, 2009, Fujitsu became the sole owner of Fujitsu Siemens Computers. This new subsidiary of Fujitsu has been renamed Fujitsu Technology Solutions.

This document from the document archive refers to a product version which was released a considerable time ago or which is no longer marketed.

Please note that all company references and copyrights in this document have been legally transferred to Fujitsu Technology Solutions.

Contact and support addresses will now be offered by Fujitsu Technology Solutions and have the format *...@ts.fujitsu.com*.

The Internet pages of Fujitsu Technology Solutions are available at *http://ts.fujitsu.com/*...
and the user documentation at *http://manuals.ts.fujitsu.com*.

Hinweise zum vorliegenden Dokument

Zum 1. April 2009 ist Fujitsu Siemens Computers in den alleinigen Besitz von Fujitsu übergegangen. Diese neue Tochtergesellschaft von Fujitsu trägt seitdem den Namen Fujitsu Technology Solutions.

Das vorliegende Dokument aus dem Dokumentenarchiv bezieht sich auf eine bereits vor längerer Zeit freigegebene oder nicht mehr im Vertrieb befindliche Produktversion.

Bitte beachten Sie, dass alle Firmenbezüge und Copyrights im vorliegenden Dokument rechtlich auf Fujitsu Technology Solutions übergegangen sind.

Kontakt- und Supportadressen werden nun von Fujitsu Technology Solutions angeboten und haben die Form *...@ts.fujitsu.com*.

Die Internetseiten von Fujitsu Technology Solutions finden Sie unter *http://de.ts.fujitsu.com/*..., und unter *http://manuals.ts.fujitsu.com* finden Sie die Benutzerdokumentation.